Simulating Reconfigurable Multiprocessor Systems-on-Chip with MPSoCSim

PHILIPP WEHNER, JENS RETTKOWSKI, TOBIAS KALB, and DIANA GÖHRINGER,
Ruhr-University Bochum

Upcoming reconfigurable Multiprocessor Systems-on-Chip (MPSoCs) present new challenges for the design and early estimation of technology requirements due to their runtime adaptive hardware architecture. The usage of simulators offers capabilities to overcome these issues. In this article, MPSoCSim, a SystemC simulator for Network-on-Chip (NoC) based MPSoCs is extended to support the simulation of reconfigurable MPSoCs. Processors, such as ARM and MicroBlaze, and peripheral models used within the virtual platform are provided by Imperas/OVP and attached to the NoC. Moreover, traffic generators are available to analyze the system. The virtual platform currently supports mesh topology with wormhole switching and several routing algorithms such as XY-, a minimal West-First algorithm, and an adaptive West-First algorithm. Amongst the impact of routing algorithms regarding performance, reconfiguration processes can be examined using the presented simulator. A mechanism for dynamic partial reconfiguration is implemented that is oriented towards the reconfiguration scheme on real FPGA platforms. It includes the simulation of the undefined behavior of the hardware region during reconfiguration and allows the adjustment of parameters. During runtime, dynamic partial reconfiguration interfaces are used to connect the Network-on-Chip infrastructure with reconfigurable regions. The configuration access ports can be modeled by the controller for the dynamic partial reconfiguration in form of an application programming interface. An additional SystemC component enables the readout of simulation time from within the application. For evaluation of the simulator timing and power consumption of the simulated hardware are estimated and compared with a real hardware implementation on a Xilinx Zynq FPGA. The comparison shows that the simulator improves the development of reconfigurable MPSoCs by early estimation of system requirements. The power estimations show a maximum deviation of 9mW at 1.9W total power consumption.

CCS Concepts: • **Computer systems organization** → **Multicore architectures**; **Reconfigurable computing**; **System on a chip**; • **Hardware** → *Platform power issues*; • **Computing methodologies** → Simulation evaluation

Additional Key Words and Phrases: MPSoC simulator, network-on-chip, dynamic partial reconfiguration, heterogeneous MPSoC, reconfigurable computing, routing algorithms, OVP, virtual platform

ACM Reference Format:

Philipp Wehner, Jens Rettowski, Tobias Kalb, and Diana Göhringer. 2016. Simulating reconfigurable multiprocessor systems-on-chip with MPSoCSim. *ACM Trans. Embed. Comput. Syst.* 16, 1, Article 4 (October 2016), 24 pages.

DOI: <http://dx.doi.org/10.1145/2972952>

Authors' addresses: Ruhr-University Bochum, MCA: Application-specific Multi-Core Architectures Group, Building ID, Floor 1, Room 347, Universitätsstr. 150, 44780 Bochum, Germany; emails: {philipp.wehner, jens.rettowski,tobias.kalb, diana.goehringer}@rub.de.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

2016 Copyright is held by the owner/author(s). Publication rights licensed to ACM.

1539-9087/2016/10-ART4 \$15.00

DOI: <http://dx.doi.org/10.1145/2972952>

1. INTRODUCTION

Due to the rising advances in Very Large Scale Integration (VLSI) the number of transistors on a single chip increases significantly. For this reason, a high number of processing elements (PEs) can also be placed on a single chip. Recently, Multi-Processor Systems-on-Chip (MPSoCs) became a popular solution for embedded computing [Ceng et al. 2008]. They provide parallel computation power. The growing number of PEs enlarges the communication overhead, which cannot be fulfilled by bus systems.

Networks-on-Chip (NoCs) [Agarwal et al. 2009] are the most promising solution for multicore processors to overcome this issue. In comparison to bus-based on-chip interconnection, NoCs enhance the throughput and scalability. However, the design space exploration of such systems is large. NoCs have a great diversity referring to topology, buffer, or routing algorithms. Hence, a simulation of NoCs is useful to analyze properties and the system behavior. Traffic patterns generated by traffic generators enable the analysis of the NoC. The traffic patterns can be adjusted to meet the application requirements in terms of performance and power consumption.

Partial reconfiguration enables the designer to modify blocks of the FPGA logic without interrupting the remaining parts [Xilinx 2015a]. It is the most promising solution to enhance the flexibility offered by FPGAs as there is no need for a full reconfiguration and re-establishment of links. As a result, smaller devices can be used that consume less energy with lower costs. Naturally, the process of dynamic partial reconfiguration contributes to the power consumption of the overall system. However, dynamic partial reconfiguration has the potential to reduce the power consumption depending on the embedded application. For instance, a core running with a high frequency can be exchanged by a smaller core with a lower frequency, as soon as the high performance of the high-frequency IP core is not required anymore [Xilinx 2012]. In the lifecycle of modern embedded systems, several upgrades may become necessary. Besides upgrading software layers, the exchange of hardware often becomes necessary. Partial reconfiguration is a promising approach to solve this issue.

In case of heterogeneous architectures, application-specific PEs improve the performance and power consumption of the overall system. Open Virtual Platform (OVP) provides a various number of open-source processor models such as ARM, MIPS, open-Cores OR1K, PowerPC, Xilinx MicroBlaze, and Altera NIOS II [OVP]. MPSoCSim [Wehner et al. 2015] is an extension of OVP with a Network-on-Chip. Traffic generators and processor models communicate through the network via high-level communication mechanisms based on transaction level modeling (TLM). The simulator currently uses a scalable mesh-based NoC supporting wormhole switching. The use case for the evaluation of MPSoCSim is based on a NoC with a mesh topology. However, MPSoCSim is not limited to the already implemented NoC. The NoC can be exchanged with other SystemC-based models. The minimal requirement is a TLM-based interface that is compatible with the network interfaces of MPSoCSim. Any other NoC implementations that are not based on TLM need to be wrapped to be compatible with the TLM standard. Three different routing algorithms are currently supported: XY, a minimal West-First algorithm, and an adaptive West-First algorithm. The router has a modular structure to allow the easy integration of new algorithms. To simplify the programming of the simulator, an application programming interface (API) for bare-metal and Linux-based programs is provided.

The main contribution of this article is the extension of MPSoCSim to allow the simulation of reconfigurable multi-processor Systems-on-Chip. The extension is based on a SystemC NoC that enabled the flexible evaluation of the proposed simulation techniques. The goal of the present approach is to provide a NoC-based MPSoC simulator that takes dynamic partial reconfiguration into account and can be compared with an

implementation on real hardware. The user of MPSoCSim can furthermore replace the NoC with his own implementations. MPSoCSim is well structured and therefore supports the user by modifying existing models, such as routers, network interfaces, and processors. During the reconfiguration process on the real hardware, the functionality of the reconfigurable region is ordinarily unspecified. The simulation technique presented in this article takes this into consideration by simulating the undefined behavior of the hardware region during reconfiguration for a specific period of time. MPSoCSim therefore comes close to the behavior of real FPGA platforms. The flexibility of the framework in this article is also increased by the adjustment of parameters. Several settings, such as the frequency of the processors and the NoC and the delay for the routing, can be configured. They improve and adapt the simulation to specific hardware counterparts. During runtime, dynamic partial reconfiguration (DPR) interfaces enable the exchange of simulated IP cores and processor models by connecting the simulated Network-on-Chip infrastructure with the reconfigurable regions. The interfaces improve usability and ease of operation. Since dynamic partial reconfiguration becomes more and more important in today's embedded systems development [Mentens et al. 2015], MPSoCSim allows a holistic view on modern hardware/software co-design techniques. An additional feature is the extension of the local group that connects OVP processor models and the Network-on-Chip. A DPR RAM is added that is useful to separate reconfiguration specific parameters from ordinary network data, shared between the network nodes. Following the philosophy of MPSoCSim, simulation results are once more compared with a real hardware implementation. The NoC is implemented on a Xilinx Zynq System-on-Chip (SoC) which contains an ARM processor and several MicroBlaze processors located inside the programmable logic. Since MPSoCSim features the access of simulation statistics of the network interfaces, a detailed analysis of the dynamic partial reconfiguration processes is enabled.

For the simulation of the above-mentioned bus- and NoC-based reconfigurable MP-SoCs, there is currently no other simulator available that accommodates the support of processor models and traffic generators as well as dedicated hardware accelerators [Göhringer 2014].

The article is organized in the following manner: Section 2 depicts related work of NoC and multicore simulators. In Section 3, a recap of the simulators structure is presented. Section 4 presents the modifications on MPSoCSim to enable the simulation of dynamic partial reconfiguration. While Section 5 introduces the application programming interface, Section 6 introduces the hardware (HW) implementation. Section 7 shows the evaluation by benchmarks and a comparison with the real HW. Finally, a conclusion and outlook is given in Section 8.

2. RELATED WORK

Table I gives an overview of the presented simulators in this section. The simulators with their respective modeling language, communication infrastructure, PEs, topology and the type of simulation results are listed. MPSoCSim is the only simulator which provides traffic generators and processor models to simulate a NoC. In addition, MP-SoCSim provides access to simulation results of the network interfaces. It therefore extends the functionality of the OVP simulator, which increases the flexibility especially for NoC-based simulations.

Several different simulators for the analyses of NoCs and MPSoCs are available. However, dynamic partial reconfiguration is only supported by a few. To the best of our knowledge, none of the available simulation frameworks features the support of NoC infrastructure, processor models and dynamic partial reconfiguration with power, area, and performance estimation. In this section, an overview of the existing simula-

Table I. Overview of Related Work

Simulator	Modeling Language	NoC support	Processing Elements	Simulation Results	Dynamic partial reconfiguration
Nirgam [Gaur et al. 2007]	SystemC	Yes	Traffic Generators, Application Core	Performance, Power	No
Noxim [Catania et al. 2015]	SystemC	Yes (Mesh)	Traffic Generators	Performance, Power	No
Booksim 2.0 [Jiang et al. 2013]	C++	Yes	Traffic Generators	Performance	No
HNOCS [Ben-Itzhak et al. 2012]	C++	Yes	Traffic Generators	Performance, Power	No
[Rosa et al. 2014]	SystemC	n/a	OVP Processor Models	Performance, Power	No
[Grüttner et al. 2014]	C/C++, SystemC	n/a	n/a	Performance, Power	No
MPARM [Benini et al. 2005]	SystemC	n/a	ARM	Performance, Power	No
SMASH [Cattaneo et al. 2013]	C++	n/a	n/a	Performance	Yes
[Schürmans et al. 2015]	SystemC	n/a	OVP Processor Models	Performance, Power	No
[Werner et al. 2015]	SystemC	n/a	OVP Processor Models	Performance	No
MPSoCBench [Duenha et al. 2014]	SystemC, ArchC	Yes (Mesh)	PowerPC, MIPS, SPARC, ARM	Performance, Power	No
MC-Sim [Cong et al. 2008]	C	Yes (Mesh)	MIPS	Performance	No
[Hansen et al. 2013]	VHDL	n/a	MIPS Soft Core	Performance	Yes
ReSim [Gong and Diessel 2011]	SystemVerilog	n/a	n/a	Performance	Yes
[Brito et al. 2007]	SystemC	n/a	MicroBlaze	Performance, Area	Yes
[Masing et al. 2013]	C	Yes	OVP Processor Models	Performance	Yes
MPSoCSim	SystemC	Yes (Mesh)	Traffic Generators, OVP Processor Models	Performance, Power, Area	Yes

tors is given. Moreover, the differences between MPSoCSim and these simulators are presented.

Amongst the simulators, several commercially available NoC implementations, such as iNoCs [2016] and the Spidergon STNoC [Dubois et al. 2011] exist. The iNoCs toolkit enables the interconnection of multiple intellectual property (IP) cores and allows their communication in an efficient manner. It includes a library with NoC component IPs that can be used for ASIC and FPGA designs. Protocols, such as the AMBA AXI, are

supported. Based on the high-level system description, arbitrary NoC topologies are implementable.

ST Microelectronics developed an interconnect called *Spidergon STNoC*. Among others, it includes network interfaces and router and physical links which handle physical communications [Dubois et al. 2011]. They are connected through the *Spidergon*, a regular point-to-point topology [Tatas et al. 2014].

The goal of the latter, iNoCs and Spidergon STNoC, is the hardware implementation of highly efficient Networks-on-Chip. In contrast, the focus on MPSoCSim relies on simulation. However, the features of commercial Networks-on-Chip, such as iNoCs and Spidergon STNoC, are very important for further extensions of the MPSoCSim simulation environment. During early design phases, MPSoCSim offers features to the developers of modern NoC-based embedded systems, helping to analyze the functionality of the targeted architecture. The above-mentioned commercial products provide the actual implementation as used for the final NoC.

2.1. NoC Simulators

NIRGAM [Gaur et al. 2007] is a SystemC-based NoC simulator. It provides cycle-accurate simulation for different topologies via configuration files. Additional configurable NoC parameters are the clock frequency, buffer depth, flit size, and virtual channels. In addition, the NoC can be tested with different applications in order to obtain performance analyses. Noxim [Catania et al. 2015] is a NoC simulator written in SystemC. Simulation results are calculated for a configurable NoC. Parameters such as the network size, buffer size, routing algorithm, packet-size distribution and selection strategy are customizable to evaluate the communication system. The evaluation in terms of several metrics is enabled by traffic generators. In addition, these generators support different traffic patterns. Another cycle-accurate NoC simulator is Booksim 2.0 [Jiang et al. 2013]. A traffic manager generates packets that are sent through the NoC. The topology, routing algorithm, flow control and additionally the router microarchitecture is configurable. This enables the evaluation of different performance metrics. An alternative to Booksim 2.0 is HNOCS [Ben-Itzhak et al. 2012]. HNOCS is a modular open-source simulator for heterogeneous NoCs with variable link capacities and a number of virtual channels (VC) per port. The simulator is based on OMNeT++ which is a framework for NoC modeling. The simulation results show statistical measurements, e.g., end-to-end latencies, throughput, VC acquisition latencies, and transfer latencies.

All the aforementioned simulators analyze the NoC with traffic patterns generated by traffic generators. This evaluation is essential to characterize the NoC properties. Compared to the work presented in this paper, MPSoCSim is convenient for NoC simulation as it supports a parameterizable mesh NoC that can be tested with traffic generators. In contrast to Jain et al. [2007], Catania et al. [2015], Jiang et al. [2013], and Ben-Itzhak et al. [2012], MPSoCSim uses processor models provided by OVP. Each processor is programmable with an arbitrary program. This enables the hardware/software co-design of MPSoCs, as the test of applications is already supported by the simulator.

2.2. Bus-Based MPSoC

Similar to MPSoCSim, the work presented by Rosa et al. [2014] is also based on OVP-Sim. Rosa et al. [2014] include an energy model in the OVPSim simulator. The simulation results are compared with a gate-level implementation of the simulated platform. MC-Sim [Cong et al. 2008] is a simulation platform for heterogeneous MPSoCs containing a NoC communication infrastructure. In addition, it supports coprocessors. The processor models are based on a modified version of SESC simulator [Renau

et al. 2015]. This framework facilitates the simulation of different processor models using the MIPS instruction set architecture (ISA). MC-Sim provides a performance evaluation. Since MIPS ISA is supported by OVP processor models, MPSoCSim is also able to integrate them. In addition, it includes further ISAs.

An electronic system level (ESL) framework for the rapid virtual system prototyping of heterogeneous SoCs regarding power and timing estimation is introduced by Grüttner et al. [2014]. The framework takes the entire system into consideration, including software, custom hardware, and third-party IP components. Based on a functional C/C++ description, virtual executable prototypes are generated that can be used for the design space exploration. Similar to MPSoCSim, Grüttner et al. [2014] combine a platform-based rapid prototyping approach with techniques for the timing and power estimation, but do not consider NoC communication systems nor dynamic partial reconfiguration.

In the context of MPSoC simulators, MPARM [Benini et al. 2005], modeled in SystemC, must be introduced. MPARM contains a cycle-accurate ARM simulator called SWARM [Dales 2003]. To exchange data between the ARM processors, the communication is handled via an AMBA bus. Due to its cycle-accurate processor models, it is suited for power estimations. Additionally to the ARM processor, MPSoCSim also supports other processor models such as MicroBlaze. Furthermore, MPSoCSim utilizes a NoC as communication infrastructure instead of a bus.

A heuristic methodology for supporting the design of reconfigurable embedded systems, SMASH, is presented by Cattaneo et al. [2013]. The authors focus on the problem of manually determining the architecture's structure, taking dynamic partial reconfiguration into account. SMASH tries to improve the performance of architectures by combining design heuristics with heuristics for mapping and scheduling of partitioned applications. Synopsys Platform Architect [Synopsys 2016] is used for the validation, based on generated virtual platforms. SMASH is highly important to the context of this article as it combines dynamic partial reconfiguration techniques with design methodologies of MPSoCs.

Estimating the ESL power consumption of processor models provided as binary object code results in the challenges that are faced by Schürmans et al. [2015]. A black box approach based on OVP processor models is chosen that uses a calibration method for the system analysis. It is one of the latest techniques to enhance the flexibility of virtual prototyping regarding timing and power estimation. However, neither NoC communication nor dynamic partial reconfigurations are considered.

A Software-in-the-Loop approach based on the OVP processor models is presented by Werner et al. [2015]. It enables the communication with hardware devices to provide real-world data to the simulation environment. With this approach, new advances for the virtual prototyping of modern embedded systems occur. However, neither NoC support nor dynamic partial reconfiguration is taken into consideration. A similar approach combining real-world data with a simulation environment is presented by Wehner and Göhringer [2015].

2.3. NoC-Based MPSoC

To simulate processors attached to a NoC, MPSoCBench [Duenha et al. 2014] can be used. Four different processor models based on ArchC are supported: MIPS, PowerPC, SPARC, and ARM. In a high abstraction level, the MPSoC can be constructed with these processors and a communication system. The number of processors is scalable. MPSoCBench simulates the MPSoC executing a benchmark and shows the appropriate results in terms of performance and power. The simulation infrastructure is written in SystemC. Contrary to MPSoCSim, it is not possible to configure properties such as routing algorithms and flit size of the NoC. Furthermore, MPSoCSim supports

an operating system (e.g., Linux and FreeRTOS) running on the processors. Another difference is that MPSoCBench does not include traffic generators to analyze the NoC with general traffic patterns.

None of the above-mentioned frameworks covers the simulation of dynamic partial reconfiguration. Approaches that combine DPR techniques with simulation capabilities are presented in the following subsection.

2.4. Simulators for Reconfiguration

Several approaches, such as Hansen et al. [2013] and Gong and Diessel [2011], exist that focus on simulation of dynamic partial reconfiguration on the register transfer level (RTL). Especially for functional verification, frameworks that facilitate the modeling process are important to enable a look at the system under test in its entirety. An overview of design tools, including academic approaches for the simulation of reconfigurable systems is presented by Göhringer [2014], Gong and Diessel [2011], who use a reconfigurable system for simulation-based functional verification, is mentioned. A top-down methodology is used that supports system designers on several layers of abstraction, from the behavioral level to the RTL.

A technique for modeling dynamic partial reconfiguration based on a SystemC approach is presented by Brito et al. [2007]. It can be used at the transaction level as well as at the register transfer level. To demonstrate the accuracy of the technique, a comparison with a real implementation on a Xilinx Virtex-II FPGA is performed. Instead of a NoC, Brito et al. [2007] use a bus interconnect for the evaluation of the approach. An OVP-based platform for the virtual prototyping of heterogeneous dynamic systems is presented by Masing et al. [2013]. The target platform that is modeled by the simulator consists of two Virtex-6 FPGAs containing several general-purpose processors (GPP) and a reconfigurable fabric. Since DSPs and hardware accelerators can be instantiated at the latter, the simulation framework must take this into consideration. Similar to MPSoCSim, Masing et al. [2013] added peripherals to extend the high-level simulation of OVP. These peripherals also include an accelerator model that is reconfigurable. MPSoCSim is oriented towards the internal configuration access port (ICAP) and the processor configuration access port (PCAP) of Xilinx FPGAs. It includes the simulation of the undefined behavior of the hardware region during reconfiguration as well as a suitable interface to trigger the reconfiguration process.

Related work in this area illustrates the need for capable simulation frameworks that allow the observation of a variety of modern computation techniques, including dynamic partial reconfiguration. MPSoCSim hereby combines processor models, traffic generators, and Network-on-Chip infrastructure with dynamic partial reconfiguration in a way that none of the above-mentioned frameworks provide.

3. MPSOCSIM

The OVP technology allows the connection of the simulator to already existing SystemC platforms [Imperas 2015a]. Within the SystemC environment, the OVP simulator executes open source processors and peripheral models with hundreds of MIPS. Using a loosely timed “LT” model, the TLM2.0 interface is provided by a C++ header file, available for the appropriate peripheral. Here, a specific SystemC module is defined that instantiates the processor type by MPSoCSim. Amongst the execution of the processor model in a SystemC thread, SystemC instantiates a `tlmPlatform` object defining a quantum period. It is used to define a time delay between the runs of a processor model instance [Imperas 2015b]. In MPSoCSim, the quantum is adjustable and per default set to $10\mu\text{s}$. As shown in MicroBlazes, where the number of instructions per second (IPS) is defined as 100,000,000, the processor runs 1,000 instructions per quantum.

Fig. 1. Router with internal components (left) and mesh network (right).

Further low-level parameters of MPSoCSim are the frequencies of the processor models and the network; the flit time, which is the time a flit needs to be sent by a processor; and the network size. The latency to calculate the paths inside a router can also be configured. To the best of the authors' knowledge, the combination of these low-level parameters with the simulation of dynamic partial reconfiguration cannot be found in related projects.

Based on the existing system [Rettkowski and Göhringer 2014], the presented simulator supports mesh topology and wormhole switching. Currently, the implemented algorithms include, but are not limited to XY-, minimal West-First routing, and adaptive West-First routing. In this section, functionality and structure of major simulator components are described.

Since the presented NoC uses the Cartesian mesh topology, the router module provides five connectors, as shown in Figure 1. Each connector consists of one target and one initiator socket to enable TLM data transmission. A FIFO is located in the target socket and saves incoming flits. The FIFO depth is set to 1, since wormhole routing is used which allows minimal memory usage. In addition, an input and one output port is used for the transmission of the FIFO state. It uses a signal to inform the appropriate router about available memory in the buffer. A router sets the `FIFOFull` signal only when the buffer in the subsequent router is full. The local port connects the router to the network interface.

Receiving a flit is handled by callback functions in the target sockets. In case of a header flit, relevant information is requested from a routing manager. Here, the routing algorithm is executed. The router then calculates a simulation time offset for the routing. Afterwards, the request is forwarded to the arbiter. When the initiator socket has been reserved by the arbiter, each flit has to await the simulation time offset of the router until it is sent to the target. In case of a full-target FIFO, the flits have to wait until buffer memory is available. While routers are used for the transport of data within the network, the interconnection with the processing elements is handled by the network interfaces (Figure 2). The SystemC module `NetworkInterface` consists of two initiator sockets, two target sockets and one port for the `FIFOFull` signal of the router. Using TLM2.0, one initiator socket and one target socket are available for the connection with the router, while the remaining sockets are required for the processing element. In this case, the network interface acts like ordinary peripheral components and is therefore addressed via a local bus.

In the example shown in Figure 2, the local initiator socket of the router is bound to the network target socket of the network interface. The network interface stores

Fig. 2. Extension of the OVP simulator.

received data in a memory which is accessible by the local elements, e.g., by the processing element. Data is stored at a known base address plus the offset defined by a header flit. Hence, the sender decides where data is placed in the memory. In comparison to the hardware implementation, this behavior would imply that the memory is part of the network interface. In simulation, it enables a higher flexibility for the network under test. The network interface uses an output FIFO to send flits. Here, only one flit per delta cycle is sent to the router, which allows the update of the signal indicating a full FIFO. In case of a full buffer, the sender has to wait until the memory is available. Accessing the network via network interfaces is enabled by an application programming interface. Here, the component itself generates the flits, including the header flit, and sends the data to the network interface. An address manager maintains the address space of the network and enables the definition of a specific address space for each network node.

As SystemC does not allow unbound sockets during simulation, a binding to dummy elements is done after elaboration. This feature provides a high flexibility of the simulator, as not every router needs to have a connection to other routers or processing elements. It is also possible to fill the network with traffic generators.

A traffic generator is implemented as an optional processing element. It periodically sends messages to the network and enables performance and functional analyses. The number of flits and the data rate can be adjusted. Start and stop functions are available to control the behavior of the traffic generator. For every default, the traffic generators send the messages randomly to network nodes; however, methods are available to define a specific target address. To create a network, the two-dimensional size of the network, its elementary period, and the routing algorithm have to be defined.

The traffic generators are therefore connected to the initiator and target sockets of the network interfaces. Finally, MPSoCSim combines the separated statistics, generated by the network interfaces.

The presented system uses round-robin arbitration to assign output sockets. Therefore, a waiting list includes the input sockets that request the output. Possible routing directions are forwarded to the arbiter. The input socket is registered in the waiting list of the requested output port, if the output port is not available. An event is triggered as soon as the appropriate output socket is released. As it is not possible to read the current simulation time from the application running on the processors, a peripheral component “timer” is added. A timer can be attached to a processing element. It provides a register where the requested value is accessible. Functions to start, stop, read,

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Target				Size of Payload								Memory Address																			

Fig. 3. Header flit.

Fig. 4. Partial Reconfiguration in FPGAs.

and reset the timer are available. While transmitting packets through the network, the first flit, as shown in Figure 3, contains necessary information for the routing. The router location in the mesh NoC is specified by X and Y coordinates.

The upper four bits precisely identify the target network address. This can be adjusted according to the size of the NoC. The size of the payload is specified by the following eight bits. They can be used to ascertain the complete transmission of the packet. As this information is sufficient to release the appropriate output port of the router, adding an additional tail flit is not required. However, the maximum payload size is limited to 255 bytes to prevent the system from getting blocked by messages that are too long. The remaining 20 bits specify a memory address that is needed by the network interface, as explained below.

To avoid bit operations in the router, a payload extension is added to each flit. It contains all the relevant data for the routing, such as the destination address, and stores additional information, e.g., to count hops and to measure the delay of a flit. As the payload extension does not affect the simulation results, further data can be attached.

The network interface contains a mechanism for the interpretation of collected data from incoming flits. It counts packets and included messages as well as sent and received data. A feature of the presented simulator is that these statistics can be accessed amongst the OVP specific results. Additionally, the network interfaces continuously determine the current and maximal data rate. The statistic can be used to calculate the mean number of hops of the packets. In case of a non-minimal adaptive routing algorithm, the number of hops can be different for each packet. In case of minimal deterministic routing algorithm, the number of hops is constant.

4. SIMULATION OF RECONFIGURABLE MPSOCS WITH MPSOCSIM

Dynamic partial reconfiguration enables the exchange of IP cores and processing elements inside an FPGA design during runtime [Xilinx 2014]. While a single element is reconfigured, the remaining elements are not influenced. Figure 4 illustrates this procedure. It shows a set A of partial bitstreams such as a processor or memory that are dedicated to a partial reconfigurable region (PRR). All the elements of set A are able to be placed in this region. In contrast to this, the static part (SP) cannot be reconfigured.

There are a lot of applications that benefit from partial reconfiguration. In case several elements are not needed at the same time, the resources of the FPGA can be reduced by dynamically exchanging elements. Furthermore, it provides flexibility in

Fig. 5. Concept of the DPR interface.

terms of algorithms and protocols, since they can be updated using reconfiguration [Xilinx, Inc. 2014]. Moreover, fault-tolerant systems can be built based on partial reconfiguration [Davis and Cheung 2014]. In case of a corrupt implementation of an IP core, dynamic partial reconfiguration allows the exchange of the defect elements inside the FPGA. To sum up, partial reconfiguration opens new and sophisticated techniques that can be applied in a wide range of applications.

The extension of MPSoCSim includes the design of an interface between the processing system and the simulated programmable logic to enable dynamic partial reconfiguration. By using configuration files in the `ini` format, parameters can be defined that support the setup of the SystemC simulation environment as well as application specific configurations. MPSoCSim allows the adjustment of the network dimensions during runtime. Simulated processors can access the `ini` files to be informed about the new network configuration. The concept of the dynamic partial reconfiguration in MPSoCSim allows the exchange of active processing elements at arbitrary network nodes during runtime. The functionality is therefore adjustable to the needs of the main application running on the master processing system. As the network infrastructure must remain untouched, the reconfiguration process uses only the local port of the router at the respective reconfigurable region.

As MPSoCSim is based on SystemC, two important fundamentals must be considered: Neither unbound ports and signals nor disconnecting modules during runtime are allowed. As a result, a dynamic partial reconfiguration, as performed on FPGAs, is initially not realizable. The first requirement of the interface is therefore that all reconfigurable modules are bound before starting the simulation. This can be done using a SystemC module that is connected to its respective router and has an adjustable number of connections on the side of the processing elements. In the following, this module is referenced as `DPR_I` (dynamic partial reconfiguration interface). Figure 5 presents the concept. Moreover, the ideal position of the `DPR_I` must be identified. There are two possibilities: The `DPR_I` can either be implemented between the processor and the network interface (Figure 6) or directly connected to the router (Figure 7).

A `DPR_I` located between the processor and the network interface benefits from lower memory requirements, as only one network interface is needed for each node. As each processor may have a specific interface and protocol for the bus communication, the `DPR_I` must then handle the communication with the network. The constellation of processors and network interfaces is left untouched by connecting the `DPR_I`s directly to the routers. In this case, flexibility is increased as different processors can be

Fig. 6. DPR between processor and network interface.

Fig. 7. DPR directly connected to the router.

connected to the DPR_I, using different network interfaces. The involved components are decoupled from each other and network interfaces can evolve over time by replacing the respective SystemC modules. Furthermore, it is possible to implement simulated hardware accelerators instead of processors and to change the network infrastructure and topology. DPR_Is are then not involved and a high flexibility is the result. The topology of MPSoCSim can be easily changed in one single file, implementing the structure of the entire platform.

The application programming interface of MPSoCSim simplifies the dynamic partial reconfiguration process using the DPR_I. Within the DPR_I, control mechanisms exist that allow the data exchange between the router and reconfigurable module. The master application can therefore switch between the modules, while handling the FIFO Full signals is performed by the DPR_I.

The development of the SystemC modules and their integration in MPSoCSim is described in the following.

The DPR interface initially consists of two initiator and two target sockets, enabling the interconnection of the router and the network interface. Figure 8 shows the required TLM sockets and the ports providing information about the FIFOs. Every socket needs to have a callback function, enabling the communication within the network. A router that wants to send data to the local processor must call the initiator socket.

As a result, a function is executed that uses the respective callback function to forward data to the processor via the local port.

Multiple processing elements can be connected to one network node to enable dynamic partial reconfiguration using the switching approach mentioned above. The

Fig. 8. Complete data path from the router to the processing element.

ini file therefore contains an additional entry for the number of processing elements (numberPE) on each node. Listing 1 presents the constructor including the setup of the required sockets, where lines 5 and 9 register the callback functions for the TLM communication.

LISTING 1: CREATION OF TLM SOCKETS WITHIN THE DPR INTERFACE

```

1  DprInterface::DprInterface(sc_core::sc_module_name module_name ,
2  unsigned int numberPE ) : sc_core::sc_module(module_name) {
3    for (unsigned int z = 0; z < numberPE ; z++) {
4      localInitiatorSocket[z] = new tlm_utils :: simple_initiator_socket < DprInterface >;
5      localInitiatorSocket[z]-> register_nb_transport_bw (this , &DprInterface ::
6      localNonBlockingTransportBackward);
7
8      localTargetSocket[z] = new tlm_utils :: simple_target_socket < DprInterface >;
9      localTargetSocket[z]-> register_nb_transport_fw (this , &DprInterface ::
10     localNonBlockingTransportForward);
11
12     localFIFOFull[z] = new sc_core :: sc_out <bool >;
13   }
14
15   networkInitiatorSocket . register_nb_transport_bw (this,
16   &DprInterface ::networkNonBlockingTransportBackward );
17   networkTargetSocket . register_nb_transport_fw (this,
18   &DprInterface::networkNonBlockingTransportForward );
19 }

```

Fig. 9. Reconfigurable modules in MPSoCSim.

The simulated reconfiguration is performed by redirecting data from the target socket to the local initiator socket at the network side. In the opposite direction, the activated reconfigurable module uses the local target socket to forward its answer to the initiator socket. The respective functions are presented in Listing 2. It gives an example on how the first socket can be accessed by the DPR interface. Lines 3 and 8 hereby return the callback functions of the network interfaces to incoming TLM messages.

LISTING 2: FUNCTION CALLS OF THE TLM SOCKETS IN THE DPR INTERFACE

```

1  tlm::tlm_sync_enum DprInterface::networkNonBlockingTransportForward(
2  tlm::tlm_generic_payload &payload, tlm::tlm_phase &phase, sc_core::sc_time &delay ) {
3  return (*localInitiatorSocket [1])->nb_transport_fw(payload, phase, delay);
4  }
5
6  tlm::tlm_sync_enum DprInterface::networkNonBlockingTransportBackward(
7  tlm::tlm_generic_payload &payload, tlm::tlm_phase &phase, sc_core::sc_time &delay ) {
8  return (*localTargetSocket [1])->nb_transport_bw(payload, phase, delay);
9  }

```

As a variable number of network interfaces exist for every node, a third dimension is added to the two-dimensional mesh-network. References to the network interfaces are stored in a `std::vector`. Figure 9 illustrates this extension. The final result is a simulated NoC with variable dimensions and a variable number of reconfigurable modules at each network node.

In MPSoCSim, two characteristics must be considered: The master node, in this article the ARM processor, does not have a DPR interface, as the ARM processor of the desired Zynq SoC is static and cannot be removed. It is directly connected to the router. Since this is accomplished by specific control structures, the position of the master node in the network can be changed flexibly. The second feature is the simulation of the unspecified behavior of the hardware region during reconfiguration on the real FPGA implementation. Initiator and target socket with ID 0 are therefore connected to an additional DummyPE that does not have any specific functionality. It does not send any data to the network by terminating the socket connections with TLM_COMPLETED and therefore releases an occupied TLM channel. Processing elements that must interact with other network nodes are thus connected to sockets with IDs 1 to numberPE-1. The reconfiguration process itself is handled by a master node that provides an additional SystemC module to access the DPR_I. It also influences the simulation time. The reconfiguration process is explained in the following section in detail.

Fig. 10. Interlaced API environment in MPSoCSim.

5. APPLICATION PROGRAMMING INTERFACE

The reconfiguration of specific network nodes by the master PE, e.g., the ARM Cortex A9, is one of the main tasks of the application programming interface. The underlying simulated hardware is capable of performing partial reconfiguration. The DPR interfaces therefore enable the activation of specific processing elements before starting the simulation. Basically any number of PEs can be bound to a node, but only one element is active at a specific point in time. Dynamic reconfiguration is thus enabled by the API. It hides any internal details to execute complex system functions from the user and performs the reconfiguration process in the background during simulation.

Figure 10 shows the interlaced environments. It is obvious that the main application running on the ARM processor can neither have direct access to the simulated NoC nor gather information regarding its parameters. The master node is therefore extended with additional interfaces as described in the following.

To facilitate dynamic partial reconfiguration, the interface function `reconfigureNode()` is provided that can be accessed by the application running on the master node (Listing 3):

LISTING 3: DECLARATION OF THE INTERFACE FUNCTION `RECONFIGURENODE()`

```
1 void reconfigureNode(unsigned int ID, unsigned int activePE);
```

The function is called by the ID of the DPR_I that needs to be reconfigured and the ID of the requested processing element. The DPR_I is identifiable as shown by the numeration of the DPR_Is corresponding with the network address of the respective node.

The function `reconfigureNode()` also writes the filled container to the memory address `DPR_CTRL_BASE`, which is located besides the processing element. The address is associated with the SystemC module DPR Controller, located at the local bus. It has the following memory addresses (Listing 4):

LISTING 4: MEMORY REGIONS OF THE DPR CONTROLLER

```
1 // DPR Controller address, address range starts from end of dpr ram top address
2 #define DPR_CTRL_BASE 0x03000000
3 #define DPR_CTRL_RANGE 0x00FFFFFFF
4 #define DPR_CTRL_TOP 0x03FFFFFFF
```

The DPR controller itself is a SystemC module with one initiator and target socket, shown in Figure 11. Writing the container to the memory address of the DPR controller effectively means that the required data for the partial reconfiguration is sent from the master application via the local bus to the target socket of the DPR controller.

Fig. 11. SystemC module of the DPR controller with sockets.

Fig. 12. Extension of the local group in MPSoCSim.

The DPR controller is instantiated by the network class and initialized with the relevant data for the reconfiguration process. This includes the network dimensions (`xSize`, `ySize`, `networkSize`, and `numberPE`) and the C++ maps `mapDprActivePE` and `mapDprInterface`. The key-value pairs of `mapDprActivePE` are the IDs of the DPR interfaces and the port numbers of active processor instances. In comparison, `mapDprInterface` includes the IDs of the DPR interfaces and pointers to their associated instances. Entries in the `DprController` are primarily used for supervisory purposes. The key element for the dynamic partial reconfiguration in MPSoCSim is provided by `mapDprInterface`. The pseudocode in Listing 5 demonstrates the reconfiguration process. It shows the function `reconfigureInterface()` that is called by the target socket of the DPR controller.

LISTING 5: RECONFIGURATION USING THE DPR CONTROLLER

```

1  bool reconfigureInterface(data) {
2      interface = 0th_byte_of(data);
3      port = 1st_byte_of(data);
4
5      if interface < 1 OR interface > networkSize then
6          throw_exception("DPR Interface out of range");
7      endif;
8
9      if port < 0 OR port >= numberPE then
10         throw_exception("Processing element out of range");
11     endif;
12
13     dprInterface = getMapDprInterface();
14     dprInterface->switch_to_dummy_PE();
15     wait(reconfiguration_time);
16     dprInterface->switch_to_desired_port_and_interface();
17
18     return(true);
19 }

```

Once the DPR controller receives the `dprData` container from `reconfigureNode()` in form of a TLM generic payload, the blocking function of the socket forwards the payload to `reconfigureInterface()`. This means that the master application blocks as long as `reconfigureInterface()` returns true. Not until then, the socket function returns the control back to the master application. This approach is equal to the partial reconfiguration using the PCAP or ICAP interface.

Within `reconfigureInterface()`, the ID of the DPR interface and the desired port are read from the data container (lines 2 and 3). To avoid abnormal behavior, `networkSize` and `numberPEM` are used to check if the reconfiguration is valid and can be performed (lines 5 to 11). At this point, the simulation terminates with an out-of-range exception (lines 6 and 10), in case of a missing interface ID or port number. Otherwise, `mapDprInterface` is obtained and readout in line 12. Finally, reconfiguration as performed on real hardware is modeled. The DPR interfaces changes the respective processing element.

The dummy processing element is activated and connected to the socket with id 0 (line 14). It simulates the circumstance that a reconfigurable partition of a network node is of no functionality during the reconfiguration process on real hardware. This configuration remains for an adjustable period of time, implemented using a `wait()` instruction (line 15). It can be adjusted to the real reconfiguration time of the PCAP or ICAP interface. MPSoCSim therefore can model the actual reconfiguration time of the real hardware implementation. The DPR interface explicitly activates the PE by switching to the desired port of the desired interface in line 16. Partial reconfiguration is hereby successfully finished and control is returned to the master application.

Additionally, a DPR RAM is added to the local bus that can be used for the data exchange between the master application and the DPR controller. Listing 6 shows the memory structure.

LISTING 6: MEMORY REGIONS OF DPR RAM

```

1 // DPR RAM: extra memory, address range starts from end of shared top address
2 #define DPR_RAM_BASE 0x02000000
3 #define DPR_RAM_RANGE 0x00FFFFFF
4 #define DPR_RAM_TOP 0x02FFFFFF

```

The controller can store `mapDprActivePE` or similar data to provide additional information for the ARM processor, including network parameters, available modules and configuration data. The communication of the simulated ARM is performed via the network interface as usual, while dynamic partial reconfiguration is realized using the DPR controller.

6. HW IMPLEMENTATION

The MPSoC simulated with MPSoCSim is implemented on a Xilinx Zynq device [Xilinx 2015b] which provides an ARM processor connected directly to an FPGA. In this system, the FPGA contains three MicroBlazes and a NoC with the respective interfaces. The system was generated with the Xilinx tool PlanAhead 14.6 [Xilinx 2016].

The PEs are connected by RAR-NoC [Rettkowski and Göhringer 2014] to provide a flexible on-chip communication. Here, a packet can be sent either by XY or minimal West-First algorithm. The routers of RAR-NoC are constructed in a 2x2 mesh topology and use input buffers. To minimize the buffer depth of the routers, wormhole routing is used. As a result, the buffer depth can contain one flit. Flow control is realized with acknowledge signals between the input buffers. The flit size amounts to 32 bits, which is equal to the simulated NoC flit size. The delay of a flit forwarded through a router is 1 cycle when no resource conflict occurs. In case of multiple messages trying to occupy the

Fig. 13. Reconfigurable design to evaluate the simulated reconfiguration process.

same output channel, only one message gets access to the requested output channel. Hence, the remaining messages are delayed in addition to the 1 cycle.

Moreover, the ARM processor communicates via the high-performance port (HP) to the network. This provides a high-throughput interface between the ARM processor and the NoC. Since the Zynq devices contain an ARM processor with two cores, it is possible to attach also a second ARM core to the NoC. In the case of the presented simulation, one ARM core is sufficient. The maximum number of MicroBlazes which can be placed inside the FPGA depends on the configuration of the MicroBlazes. To compare the simulation results with the HW implementation, it is sufficient to use three MicroBlaze processors. The MicroBlaze is connected via the fast simplex link (FSL) to the NoC. The frequency of the FPGA is set to 100MHz and the ARM processor runs at 667MHz.

7. EVALUATION

Since this article targets the extension of MPSoCSim regarding dynamic partial reconfiguration, it is required to prove the implementation of the underlying NoC. The presented extensions are evaluated regarding the reconfiguration during runtime.

The Xilinx Zynq SoC is used to evaluate the simulated reconfiguration. In Figure 13, the reconfigurable design is shown. It contains the ARM and multiple MicroBlaze processors that are connected by a Network-on-Chip (NoC), as it is used in MPSoCSim. The NoC is static and cannot be reconfigured, since it represents the communication architecture which has to provide communication channels during runtime. While a single region is reconfigured, the remaining regions are still able to exchange data. Each partial reconfigurable region contains one MicroBlaze processor. Vivado 2014.4 is used to generate different partial bitstreams of MicroBlaze processors containing different programs. After reconfiguring a MicroBlaze processor, the functionality changes based on the program running inside it.

In this work, the ARM processor operates as a master core that controls the reconfiguration process. Therefore, the processor configuration access point (PCAP) interface, supported by the Xilinx Zynq SoC, reconfigures the partial regions using the respective partial bitstream. The implementation of this interface is presented in Figure 14. The PCAP bridge connects the PL configuration module with the ARM processor. The ARM processor initiates a reconfiguration by starting a DMA transfer inside the PCAP bridge of a partial bitstream from memory to the PL configuration module [Xilinx 2015c].

In order to control these devices, a device driver `xdevcfg` supported from Xilinx is used. This driver runs under Linaro that is installed on the ARM processor. A timing diagram of the appropriate FPGA configuration is shown in Figure 15.

Fig. 14. PCAP interface of the Xilinx Zynq-7000 SoC to reconfigure programmable logic.

Fig. 15. General timing diagram of a reconfiguration in a Zynq equipped with Linaro (adapted from Xilinx [2015c]).

Fig. 16. Execution time of a MicroBlaze reconfiguration.

After the supply voltage is stable, a first stage boot loader configures the FPGA with an initial bitstream that contains the design shown in Figure 13. The ARM processor can call the driver `xdecvfg` to reconfigure the FPGA with another MicroBlaze processor at runtime.

Figure 16 presents the execution time to reconfigure the partial region in the test design of this work. It takes 7ms to exchange a single MicroBlaze processor. The resources consumed by a MicroBlaze processor and of the appropriate region are given in Table II.

Based on these timing measurements, a worst-case reconfiguration time can be predicted. It can be used to adjust the simulation framework. Based on the resources of the reconfigurable regions, the size of the bitstream can be estimated. The reconfiguration time results from the 32-bit size of the PCAP port and its frequency of 100MHz. An additional overhead is generated by the DDR access induced by the ARM processor. Al Kadi et al. [2013] investigated a throughput of 62.8MB/s for the reconfiguration on a Xilinx Zynq SoC under Linux. This overhead can be added to MPSoCSim in order to consider the behavior of the physical system. The specific parameter directly influences the TLM-based simulation environment and therefore results in changes in simulated time. The following evaluation uses a worst-case reconfiguration time of 7ms. It performs the dynamic partial reconfiguration based on the DPR interface. Based on the size of the routers [Rettkowski and Göhringer 2014] and the MicroBlaze processors, MPSoCSim is capable of calculating the area of the entire design.

Table II. Resources of the MicroBlaze Processors and Partial Region for xc7z045ffg900-2

	MicroBlaze	Reconfigurable Region
LUTs	1850	6000
BRAM	10	30
DSPs	0	20

The bare-metal application for the dynamic partial reconfiguration implements the (inverse) discrete Fourier transform (DFT and IDFT) of a signal that is sent to a MicroBlaze processor. It shows the operation of MPSoCSim in the context of computationally intensive tasks. Amongst the ARM processor, an additional reconfigurable processing element is available, resulting in a NoC size of 1x2. For evaluation, this MicroBlaze processor is exchanged performing either DFT or IDFT.

The ARM processor at first reconfigures the reconfigurable region with a MicroBlaze processor performing the DFT and sends a sine wave to the processing element. After receiving the Fourier transformed signal, another reconfiguration process is triggered by the ARM processor. It changes the behavior of the reconfigurable region to have the IDFT available. The received data from the first MicroBlaze is sent to this new processing element that transforms the spectrum back to the original sine wave. In the master application, the timer module is used to measure the performance of the application, including the time for the DPR. The application takes 7ms for the reconfiguration process. In simulated time, the calculation takes 144.9s for the DFT and 140.5s for the IDFT with 360 samples. The implementation of the Fourier transform is based on Bourke [1993].

For the power estimation, it is assumed that processing elements in the form of MicroBlaze processors lead to a higher power consumption in comparison to real and modeled hardware accelerators. The power estimation of MPSoCSim is based on the maximum power dissipation. Additionally, the power consumption of the NoC without active processing elements can be estimated. As shown in Listing 7, the MPSoC is divided into separate components that are predefined for the estimation process.

LISTING 7: POWER CONSUMPTION OF THE SEPARATE COMPONENTS IN WATT

```

1 double powerARM = 1.532;
2 double powerARMPeriphery = 0.024;
3 double powerMicroblaze = 0.027;
4 double powerNoCStatic = 0.158;
5 double frequency = 100.0;

```

These values are based on the Xilinx Vivado Design Suite 2014.4. Additionally, the frequency must be defined (line 5). The following equations are modeled for the ARM and MicroBlaze processor, as well as the NoC:

$$P_{ARMtotal} = PE_{ARM} \cdot (P_{ARMsingle} + PE_{ARMperiphery} \cdot 0.01 \cdot f) \quad (1)$$

The calculation of the power estimation ($P_{ARMtotal}$) in Equation (1) is based on an addition of the power consumption of the ARM processor ($P_{ARMsingle}$) and the periphery ($PE_{ARMperiphery}$). PE_{ARM} specifies the number of ARM cores. The power consumption of the periphery components increases proportional with the frequency f . The ARM processor has a static frequency as it is provided by the ZedBoard.

$$P_{MBtotal} = PE_{MB} \cdot (P_{MBsingle} \cdot 0.01 \cdot f) \quad (2)$$

Table III. Total Power Consumption (in Watt)

Mesh Dimen- sions	Vivado 50 MHz	MPSoCSim 50 MHz	Δ 50 MHz	Vivado 75 MHz	MPSoCSim 75 MHz	Δ 75 MHz	Vivado 100 MHz	MpSocsim 100 MHz	Δ 100 MHz
1×2	1.719	1.721	0.002	1.736	1.736	0.000	1.751	1.752	0.001
1×3	1.737	1.739	0.002	1.764	1.764	0.000	1.787	1.788	0.001
2×2	1.756	1.757	0.001	1.789	1.791	0.002	1.822	1.825	0.003
2×3	1.791	1.792	0.001	1.842	1.843	0.001	1.887	1.894	0.007
3×3	1.845	1.841	0.004	1.925	1.916	0.009	1.989	1.992	0.003

The MicroBlaze power consumption ($P_{MBsingle}$) is multiplied with a frequency f and a proportional factor (0,01) that is estimated empirically. The number of MicroBlaze processors (PE_{MB}) influences the power consumption as shown in Equation (2),

$$decrFactor = 0.9^{Dimension} \quad (3)$$

$$incrFactor = (0.01 \cdot f) \cdot [PE_{MB} \cdot (0.012 + (0.001 \cdot PE_{MB}))] \quad (4)$$

$$P_{NoCtotal} = P_{NoC} + (incrFactor \cdot decrFactor) \quad (5)$$

The power consumption of the NoC is added in a non-linear manner since the resources of the routers depend on their position in the network: routers at the edges of the NoC have fewer ports than routers inside the NoC. Equations (3)–(5) are empirically determined.

$$P_{Total} = P_{ARMtotal} + P_{MBtotal} + P_{NoCtotal} \quad (6)$$

The total power consumption (P_{Total}) is the summation of the above-mentioned values.

Table III shows the comparison between the power estimations of Vivado with the results of MPSoCSim. The table also includes the difference between the simulated and the measured values (Δ). It can be seen that the maximum deviation amounts to 9 mW.

To evaluate the capability of MPSoCSim to run operating systems, a Linux OS, provided by OVP/Imperas, is installed on the ARM processor. A mesh network with a variable network size is used for the simulation. Each network node has a dummy processing element, and two MicroBlazes, running different programs. The results are compared with a bare metal application. The functionality of the bare metal application is identical to the Linux-based program. Initially, the dummy processing elements are reconfigured with the MicroBlazes. One request message with the size of one integer value plus the header flit is then sent to all processing elements. The ARM core waits for the acknowledgment of all MicroBlazes. The answer contains a value with the same size as the request message. It reconfigures the simulated reconfigurable regions, inserts the remaining MicroBlazes and repeats the sending process. Finally, the results of the bare metal application can be compared with the ones of the Linux-based program. Due to the negligible processing time of the MicroBlazes, the simulated time is mainly influenced by the reconfiguration of the modules. As mentioned above, the time for reconfiguration is set to 7ms. As each run of the simulation reconfigures three modules twice, an overall reconfiguration time of 42ms is the result. The program takes 0.042180s in case of the bare metal and 0.046s on the Linux OS. Hence, the bare metal application without dynamic partial reconfiguration consumes 180 μ s while the Linux application takes 4ms. The difference in simulated time is due to the overhead generated by the OS. Figure 17 shows the execution time of bare metal and Linux-based programs depending on the network size.

Fig. 17. Comparison of execution and reconfiguration time of bare metal and Linux-based applications.

8. CONCLUSIONS

Based on OVP, MPSoCSim benefits from several processor models and a flexible TLM2.0 communication infrastructure. The reconfiguration mechanism of MPSoCSim comes close to the behavior of real FPGA platforms, including reconfigurable regions and the simulation of the unspecified behavior of the real hardware regions during the reconfiguration process. Several settings can be configured to improve and adapt the simulation to specific hardware representations. Interfaces for the dynamic partial reconfiguration improve usability and ease of operation. A DPR RAM is added that is useful to separate reconfiguration specific parameters from ordinary network data, shared between the network nodes. MPSoCSim features the access of simulation statistics of the network interfaces and the support for operating systems running on the processors. The philosophy of MPSoCSim includes the evaluation of the simulation with real hardware implementations. In this article, the reconfiguration mechanism is evaluated and compared with a reconfigurable NoC implemented on a Xilinx Zynq SoC. In addition, MPSoCSim is capable of providing estimations for the area and power consumption of the simulated hardware. The power estimations shows a maximum deviation of 9mW at 1.9W (Δ 75MHz) total power consumption. In future work, MP-SoCSim will be extended with further NoC topologies. Focus will be set on the SystemC model of the network to increase flexibility and easy expandability. Also performance metrics and further traffic patterns will be taken into account.

ACKNOWLEDGMENT

The authors would like to thank Imperas Software Ltd. for providing tools and licenses and thereby enabling the work on MPSoCSim.

REFERENCES

- Ankur Agarwal, Cyril Iskander, and Ravi Shankar. 2009. Survey of network on chip (NoC) architectures & contributions. *Journal of Engineering, Computing and Architecture* 3, 1.
- Luca Benini, Davide Bertozzi, Alessandro Bogliolo, Francesco Menichelli, and Mauro Olivieri. 2005. MPARM: Exploring the multi-processor SoC design space with SystemC. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology* 41, 2 (Sept. 2005), 169–182.

- Yaniv Ben-Itzhak, Eitan Zahavi, Israel Cidon, and Avinoam Kolodny. 2012. HNOCS: Modular open-source simulator for heterogeneous NoCs. In *Proceedings of the 2012 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*.
- Alisson V. Brito, Matthias Kühnle, Michael Hübner, Jürgen Becker, and Elmar U. K. Melcher. 2007. Modelling and simulation of dynamic and partially reconfigurable systems using SystemC. In *Proceedings of the 2007 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. Porto Alegre, Brasil.
- Paul Bourke. 1993. DFT, FFT. Retrieved from <http://paulbourke.net/miscellaneous/dft/>.
- V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti. 2015. Noxim: An open, extensible and cycle-accurate network on chip simulator. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*.
- Riccardo Cattaneo, Christian Pilato, Gianluca C. Durelli, Marco D. Santambrogio, and Donatella Sciuto. 2013. SMASH: A heuristic methodology for designing partially reconfigurable MPSoCs. In *Proceedings of the 2013 International Symposium on Rapid System Prototyping (RSP)*.
- J. Ceng, J. Castrillón, W. Sheng, H. Scharwächter, R. Leupers, G. Ascheid, H. Meyr, T. Isshiki, and H. Kunieda. 2008. MAPS: An integrated framework for MPSoC application parallelization. In *Proceedings of the 45th ACM/IEEE Design Automation Conference (DAC)*. 754–759.
- Jason Cong, Karthik Gururaj, Guoling Han, Adam Kaplan, Mishali Naik, and Glenn Reinman. 2008. MC-Sim: An efficient simulation tool for MPSoC designs. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.
- Michael Dales. 2003. SWARM - SoftWare ARM. Retrieved from <http://www.cl.cam.ac.uk/~mwd24/phd/swarm.html>.
- James J. Davis and Peter Y. K. Cheung. 2014. Achieving low-overhead fault tolerance for parallel accelerators with dynamic partial reconfiguration. In *Proceedings of the 24th International Conference on Field Programmable Logic and Applications (FPL)*. 1–6.
- Florentine Dubois, Jose Cano, Marcello Coppola, Jose Flich, and Frederic Petro. 2011. Spidergon STNoC design flow. In *Proceedings of the 5th IEEE/ACM International Symposium on Networks on Chip (NoCS)*.
- Liana Duenha, Marcelo Guedes, Henrique Almeida, Matheus Boy, and Rodolfo Azevedo. 2014. MPSoCBench: A toolset for MPSoC system level evaluation. In *Proceedings of the 2014 International Conference on Embedded Computer Systems (ICECS)*.
- M. S. Gaur, B. M. Al-Hashimi, V. Laxmi, R. Navaneeth, Naveen Choudhary, Lavina Jain, Mushtaq Ahmed, K. K. Paliwal, Varsha, Rekha, and Vineetha. 2007. NIRGAM: A simulator for NoC interconnect routing and application modeling. In *Proceedings of the Workshop on Diagnostic Services in Network-on-Chips, Design, Automation and Test in Europe Conference (DATE)*.
- Diana Göhringer. 2014. Reconfigurable multiprocessor systems: Handling hydras heads – a survey, In *Proceedings of the 5th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*.
- Lingkan Gong and Oliver Diessel. 2011. ReSim: A reusable library for RTL simulation of dynamic partial reconfiguration. In *Proceedings of the 2011 International Conference on Field-Programmable Technology (FPT)*.
- Kim Grüttner, Philipp A. Hartmann, Tiemo Fandrey, Kai Hylla, Daniel Lorenz, Stefan Stettmann, Björn Sander, Oliver Bringmann, Wolfgang Nebel, and Wolfgang Rosenstiel. 2014. An ESL timing & power estimation and simulation framework for heterogeneous SoCs. In *Proceedings of the 2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*.
- Simen Gimle Hansen, Dirk Koch, and Jim Torresen. 2013. Simulation framework for cycle-accurate RTL modeling of partial run-time reconfiguration in VHDL. In *Proceedings of the 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*.
- Imperas Software Limited. 2015a. OVP & SystemC. Retrieved from http://www.ovpworld.org/technology_systemc.
- Imperas Software Limited. 2015b. Using OVP Models in SystemC TLM2.0 Platforms. Retrieved from http://www.ovpworld.org/documents/OVPsim_Using_OVP_Models_in_SystemC_TLM2.0_Platforms.pdf.
- Imperas Software Limited. 2015c. OVPsim and Imperas CpuManager User Guide. http://www.ovpworld.org/documents/OVPsim_and_CpuManager_User_Guide.pdf.
- iNoCs. 2016. Homepage. Retrieved from <http://www.inocs.com>.
- L. Jain, B. Al-Hashimi, M. Gaur, V. Laxmi, and A. Narayanan. 2007. NIRGAM: A simulator for NoC interconnect routing and application modeling. In *Workshop on Diagnostic Services in Network-on-Chips, Design, Automation and Test in Europe Conference (DATE)*. Nice, France. <http://nirgam.ecs.soton.ac.uk>.
- Nan Jiang, Daniel U. Becker, George Michelogiannakis, James Balfour, Brian Towles, John Kim, and William J. Dally. 2013. A detailed and flexible cycle-accurate network-on-chip simulator. In *Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software*.

- Muhammed Al Kadi, Patrick Rudolph, Diana Goehringer, and Michael Huebner. 2013. Dynamic and partial reconfiguration of zynq 7000 under linux. In *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. 1–5.
- Leonard Masing, Stephan Werner, and Jürgen Becker. 2013. Virtual prototyping of heterogeneous dynamic platforms using open virtual platforms. In *Proceedings of the 10th IEEE International Symposium on Industrial Embedded Systems (SIES)*.
- Nele Mentens, Jochen Vandompe, J. O. Vliegen, A. N. Braeken, Bruno D. A. Silva, Abdellah Touhafi, Stephan Knappmann, Alois Kern, Jens Rettkowski, Muhammed Soubhi A. L. Kadi, Diana Göhringer, and Michael Hübner. 2015. DynamIA: Dynamic hardware reconfiguration in industrial applications. In *Proceedings of the 11th International Symposium on Reconfigurable Computing: Architectures, Tools, and Applications (ARC)*.
- Open Virtual Platforms – the source of Fast Processor Models & Platforms. Retrieved from <http://www.ovpworld.org/>.
- Jose Renau, Basilio Fraguela, James Tuck, Wei Liu, Milos Prvulovic, Luis Ceze, Smruti Sarangi, Paul Sack, Karin Strauss, and Pablo Montesinos. 2015. SESC Simulator. Retrieved from <http://sesc.sourceforge.net>.
- Jens Rettkowski and Diana Göhringer. 2014. RAR-NoC: A reconfigurable and adaptive routable network-on-chip for FPGA-based multiprocessor systems. In *Proceedings of the 2014 International Conference on ReConfigurable Computing and FPGAs (ReConFig)*.
- Felipe Rosa, Luciano Ost, Thiago Raupp, Fernando Moraes, and Ricardo Reis. 2014. Fast energy evaluation of embedded applications for many core systems. In *Proceedings of the 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*.
- Stefan Schürmans, Gereon Onnebrink, Rainer Leupers, Gerd Ascheid, and Xiaotao Chen. 2015. ESL power estimation using virtual platforms with black box processor models. In *Proceedings of the 3rd Workshop on Virtual Prototyping of Parallel and Embedded Systems (ViPES) as part of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XV)*.
- Synopsys, Inc. 2016. Platform Architect. Retrieved from <http://www.synopsys.com/Prototyping/ArchitectureDesign/Pages/platform-architect.aspx>.
- Konstantinos Tatas, Kostas Siozios, Dimitrios Soudris, and Axel Jantsch. 2014. *Designing 2D and 3D Network-on-Chip Architectures*. Springer, New York.
- Philipp Wehner, Jens Rettkowski, Tobias Kleinschmidt, and Diana Göhringer. 2015. MPSoCSim: An extended OVP simulator for modeling and evaluation of network-on-chip based heterogeneous MPSoCs. In *Proceedings of the 3rd Workshop on Virtual Prototyping of Parallel and Embedded Systems (ViPES) as part of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XV)*.
- Philipp Wehner, and Diana Göhringer. 2015. Internet of things simulation using OMNeT++ and hardware in the loop. In *Proceedings of the 2015 International Workshop on Components and Services for IoT Platforms: Paving the Way for IoT Standards*.
- Stephan Werner, Leonard Masing, Fabian Lesniak, and Jürgen Becker. 2015. Software-in-the-loop simulation of embedded control applications based on virtual platforms: software-in-the-loop with open virtual platforms (OVP) using real world data and devices. In *Proceedings of the 25th International Conference on Field Programmable Logic and Applications (FPL)*.
- Xilinx, Inc. 2012. WP374 (v1.2): Partial Reconfiguration of Xilinx FPGAs Using ISE Design Suite. Retrieved from http://www.xilinx.com/support/documentation/white_papers/wp374_Partial_Reconfig_Xilinx_FPGAs.pdf.
- Xilinx, Inc. 2014. Vivado Design Suite User Guide – Partial Reconfiguration, UG909 (v2014.4). Retrieved from <http://www.xilinx.com>.
- Xilinx, Inc. 2015a. Partial Reconfiguration in the ISE Design Suite. Retrieved from <http://www.xilinx.com/tools/partial-reconfiguration.htm>.
- Xilinx Inc. 2015b. Zynq-7000 All Programmable SoC. <http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/>
- Xilinx Inc. 2015c. Zynq-7000 All Programmable SoC – Technical Reference Manual, UG585 (v1.10). Retrieved from <http://www.xilinx.com>.
- Xilinx Inc. 2016. PlanAhead Design and Analysis Tool. Retrieved from <http://www.xilinx.com/products/design-tools/planahead.html>.

Received October 2015; revised June 2016; accepted July 2016