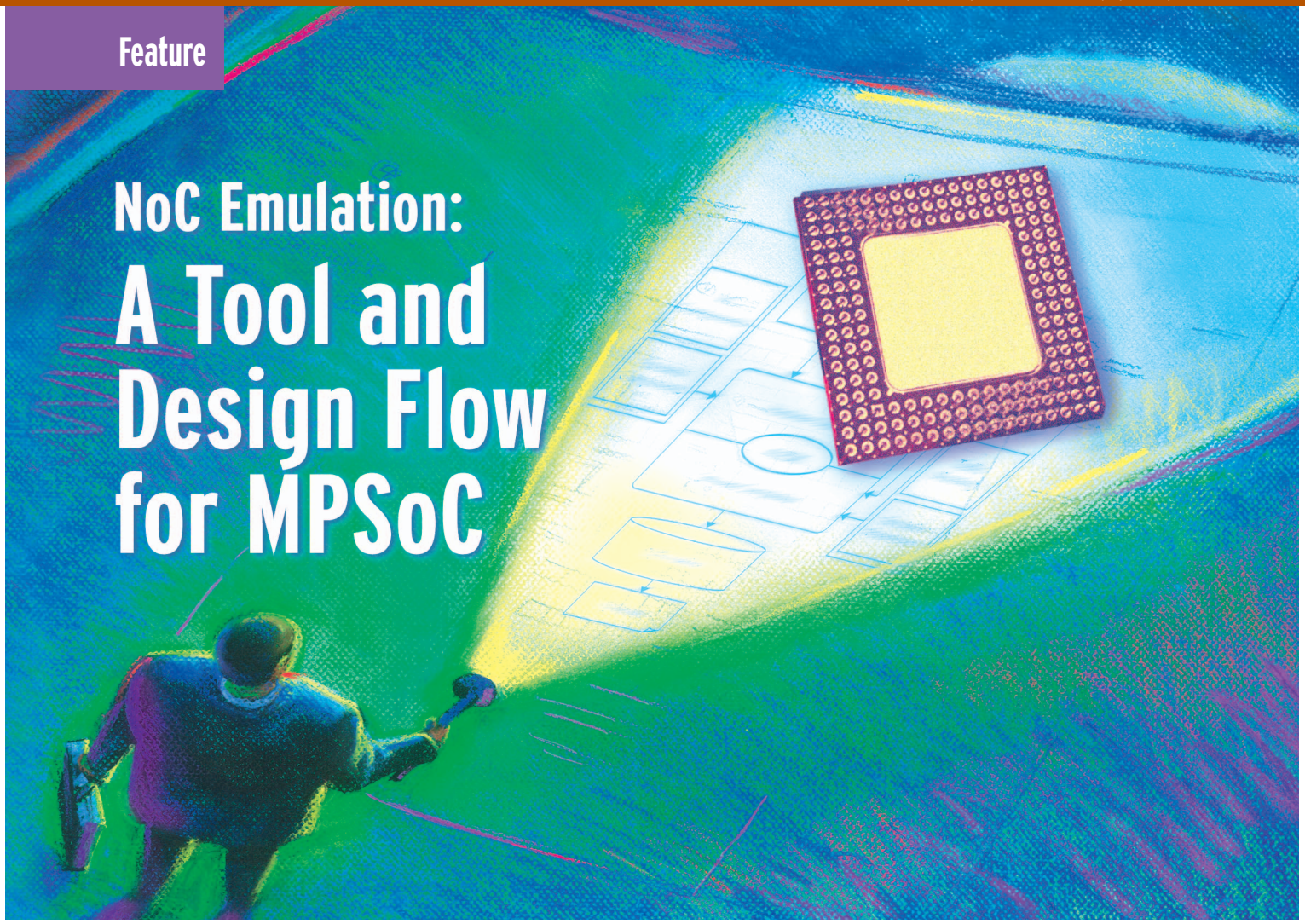
Feature

NoC Emulation: A Tool and Design Flow for MPSoC



Nicolas Genko, David Atienza,
Giovanni De Micheli, and Luca Benini

Abstract

Current Systems-On-Chip (SoC) execute applications that demand extensive parallel processing; thus, the amount of processors, memories and application-specific signal processing cores is rapidly increasing. In these new Multi-Processor SoCs, (MPSoCs) one of the most critical elements regarding overall efficiency is on-chip interconnections. *Network-On-Chip* (NoC) provides a structured way of realizing interconnections on silicon, and obviate the limitations of bus-based solutions. NoCs can have regular or *ad hoc* topologies and can be tuned by a large set of parameters. Simulation and functional validation are essential to assess the correctness and performance of MPSoC architectures. We present a flexible hardware-software emulation framework implemented on an FPGA that is specially designed to suitably explore, evaluate and compare a wide range of NoC solutions with a very limited effort. Our experimental results show a speed-up of four orders of magnitude with respect to cycle-accurate HDL simulation, while retaining cycle accuracy and flexibility of software simulators. Finally, we propose a validation flow for MPSoCs based on our flexible NoC emulation framework, which allows designers to explore and optimize a range of solutions, as well as quickly characterize performance figures and identify possible limitations in their on-chip interconnection architectures.

1. Introduction

With the growing complexity in consumer embedded products, new trends envisage heterogeneous *Multi-Processor System-On-Chip* (MPSoC) architectures consisting of complex integrated components communicating with each other at very high-speed rates. Nowadays, MPSoCs are increasingly complex, integrating a larger and larger amount of processing cores [1]. Thus, intercommunication requirements of complex MPSoCs will not be feasible using a single shared bus or a hierarchy of buses due to their poor scalability with system size.

To overcome these problems of scalability and complexity, *Network-On-Chips* (NoCs) have been proposed as a promising replacement for buses and dedicated interconnections in forthcoming nanometer-scale technologies

Digital Object Identifier 10.1109/MCAS.2007.910029

because they possess better design predictability and more modularity than traditional bus-based systems [2, 3]. However, NoC-based MPSoCs involve new and critical design challenges, such as the design of network interfaces and protocols to provide reliable on-chip communication to transport the data of the cores. Also, the selection of suitable custom topologies of switches for the applications of the target MPSoC is critical to provide the needed low-latency at the physical interconnection layer to transport the data of the cores. All these challenges require a very time-consuming and error-prone design and tuning process of on-chip interconnects to design power-efficient and high-performance MPSoC.

In this article we present a combined hardware-software NoC emulation framework, which shows how flexible NoC emulation can be used as a powerful design tool for tuning and functional validation of on-chip interconnections for MPSoCs. This emulation framework is implemented onto a *Field Programmable Gate Array* (FPGA) platform and has as one of its main novelties the utilization of the FPGA as an active element in the emulation control layer to speed up functional validation and to add flexibility to the NoC configuration exploration, instead of merely being the platform where the circuit is prototyped, as emulation is typically used.

As an example of its possible application, we use our framework to tune and validate the X-pipes NoC architecture [4] for MPSoC designs. In this case, the emulation framework is able to test actual physical realizations of NoCs on silicon up to four orders of magnitude faster than *Hardware Description Language* (HDL) simulators (see Fig. 1), while preserving cycle accuracy. In addition, we exploit the flexibility of our emulation framework to define a procedure to rapidly validate and tune the X-pipes NoC physical implementation characteristics (e.g., buffer size, topology of switches, size of inter-switches links, etc.) for real-life traffic patterns of software applications that can be executed in the target MPSoCs or various software scenarios (e.g., bursts lengths, average on chip communication load, etc).

From the simulation viewpoint, to validate different architectural alternatives reducing the cost of synthesizable NoC design, several cycle-accurate simulation frameworks in VHDL or SystemC have been proposed. Goosens, Siguenza-Tortosa et al. [8, 9] use VHDL-based cycle-accurate models to evaluate the latency, throughput and other features in mesh-based and hierarchical NoC topologies. Bertozzi et al. [4] describe a cycle-accurate SystemC-based modeling environment for testing custom NoC topologies. These approaches are flexible to perform NoC design exploration, but their simulations are up to four orders of magnitude slower compared to our physical NoC emulation environment. Therefore, cycle-accurate simulators cannot use real-life traces to extensively evaluate the entire system under test. Other simulation approaches have been proposed to increase the speed of cycle-accurate simulation. Madsen et al. [10] propose a SystemC-based simulation environment that models the effect of a real-time operating sys-

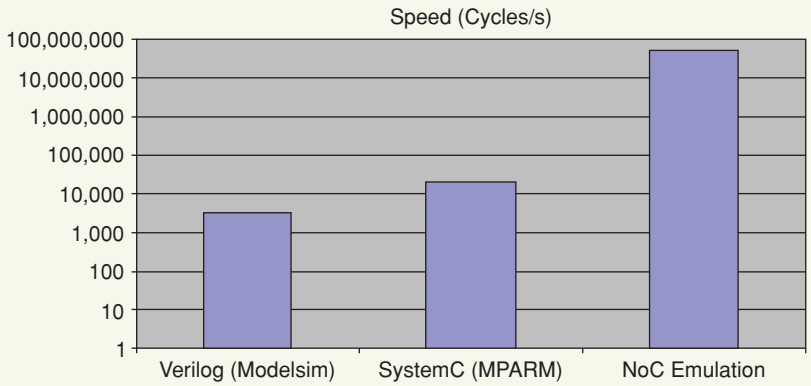


Figure 1. Speed comparison NoC-MPSoC simulation versus emulation.

NoC PROTOTYPING HISTORY

The benefits of FPGA-based prototyping for NoC-based design have been outlined by several authors. Brebner et al [5] studied mesh-based topologies and packet-switching communication mechanisms and provided circuit-level functional validation of their designs onto FPGAs. Also, Marescaux, and Moraes et al. [6, 7] ported their torus-based NoC architectures and designs of switches/routers to FPGAs to validate their choices of packet sizes and circuit-based switching modes based on HDL simulations. These previous approaches can validate several NoC implementations features but none of them has exploited the concept of FPGA emulation to propose an overall framework to enable both fast MPSoC design space pruning at cycle-accurate level and traffic effects validation with real-life inputs, as we propose in this paper.

Nicolas Genko, David Atienza, and Giovanni De Micheli are with LSI/EPFL, Lausanne, Switzerland, E-mail: {nicolas.genko, giovanni.demicheli, david.atienza}@epfl.ch. David Atienza is also with DACYA/UCM, Avda Complutense s/n, Madrid, Spain, E-mail: datienza@dacya.ucm.es. Luca Benini is with the Department of Electronics and Computer Science, University of Bologna, Bologna, Italy, E-mail: lbenini@deis.unibo.it.

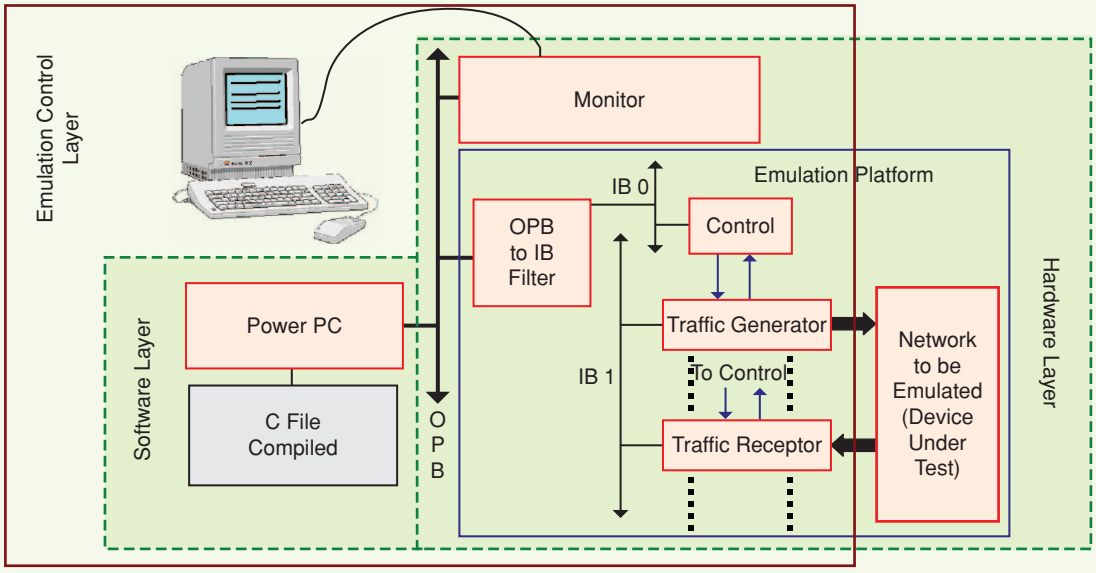


Figure 2. Overview of the proposed hardware/software NoC Emulation Framework.

tem on NoCs. Then Chan et al. [11] present a mixed VHDL/SystemC implementation and simulation methodology, which employs a template router to support several interconnection networks. These previous approaches enable a fast exploration of various features of NoCs, but their level of accuracy is limited. Furthermore, their simulation speed is still lower than our proposed emulation framework.

Finally, at high-level of abstraction, algorithms and analytical models [2, 12] have been proposed to achieve fast rough estimations of overall cost of NoCs using graphs representations. Such analytical approaches can be used in early stages of NoC development to perform a first pruning of the design space, but do not enable accurate architectural exploration and functional validation.

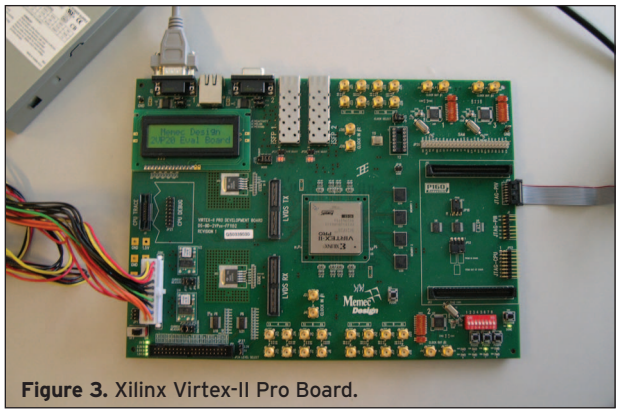


Figure 3. Xilinx Virtex-II Pro Board.

2. NoC Emulation Framework

Our emulation framework is a combined hardware-software platform (Fig. 2), which enables emulation of NoCs at different levels of abstraction. It can emulate a network of switches for internal validation of tuning of internal on-chip interconnection features or a complete NoC at the core interconnection protocol layer to explore the effects of real-life traffic patterns (e.g., congestion or effects of access bursts). It consists of two different layers. First, the emulation platform is a hardware layer that includes the network to be emulated, wraps it with the necessary hardware components to inject and receive on-chip requests (i.e., traffic generators and receptors) and provides a graphical interface to the host computer to monitor the emulation by the user. Second, the software layer is employed in our framework to initialize and control the emulation process of a physical NoC implementation. Consequently, as it is shown in Fig. 2, the emulation and design exploration control layer for our NoC emulation framework consists of both hardware and software elements. Thus, the additional hardware elements on the FPGA are used to speed-up the emulation, while the software elements are utilized to extend the versatility of the platform for NoC exploration purposes.

To achieve a correct interaction between hardware and software in the proposed hardware-software emulation framework, we have developed an HDL-based component that is accessible through a bus connection by a hard-core processor (i.e., a PowerPC on the Xilinx Virtex-II Pro board employed [13], as shown in Fig. 3), which executes a C program containing the concrete configuration of the

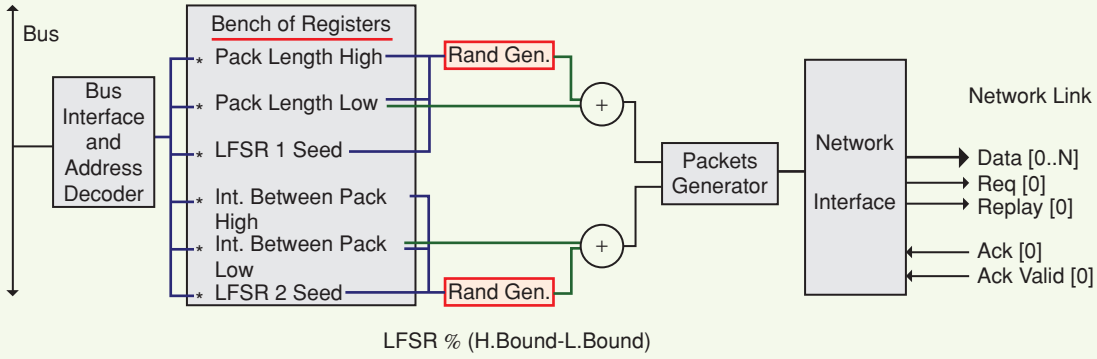


Figure 4. Example of stochastic traffic generator.

system. This program consists of a sequence of write and read operations into specific addresses of the system that are interpreted by the hardware components to behave according to the desired parameters. For instance, this mechanism is used to configure additional hardware elements to extract statistics during NoC emulation. Then the statistics are fetched by the processor and sent to a monitor module, which provides the interface to show the acquired statistics onto the screen of the host computer.

2.1 Hardware Architecture Overview

The hardware architecture of the emulation platform is designed in a modular way. It enables the implementation and emulation of multiple custom NoC topologies and architectures. An overview of the NoC programmable emulation architecture is depicted in Fig. 2 (*Hardware layer*).

This architecture consists of four elements to emulate realistic NoCs: *Traffic Generators* (TGs), *Traffic Receptors* (TRs), a control module, and a user-defined set of interconnections between the components of the emulated NoC.

- *Traffic Generators:* TGs have three interfaces. The first one connects them to the control processor of the emulation framework (i.e., a PowerPC processor of the Virtex-II FPGA [13]) via an *Internal Bus* (IB). The second one links them to the network to be emulated, which can be a simple network of switches or a complete NoC including an external public protocol and a set of traffic patterns of the applications that will be running on the target MPSoC, which need to be tested. Finally, the third one provides the interface to the control module for synchronization purposes. Our framework includes two types of TGs. The first type can generate different types of stochastic traffic and can be programmed in software by the control processor of the emulation. Fig. 4 shows an example of the internal functional blocks of a stochastic TG. It includes an interface in the bus of the control processor of the emulation, which can write in the memory-mapped register file to configure different parameters to shape the generated 2-state Markov chain stochastic traffic, such as, packet lengths, random seeds or transition probabilities through a *Linear Feedback Shift Register (LFSR)*. The second type generates traffic according to a trace (i.e., a collection of core transactions coming from a real-life application) sent to the TG in a continuous flow by the control processor. Therefore, the two kinds of TGs are radically different. On one hand, the stochastic TG is programmed before the start of the emulation and generates autonomously traffic according to its model. On the other hand, the trace-driven TG receives information in a continuous flow at run time with packets descriptors coming from the processor of the emulation platform. Moreover, both types of TGs can be combined to generate any variations of NoC traffic made of stochastic (i.e., Gaussian, Poisson…) and real-life patterns, just by using the software functions of the control processor.
  In case the emulation is carried out at the level of the network of switches, the TG is directly plugged to a link of a switch and generates flits according to the inner network protocol. In the case of a complete NoC emulation, the TG mimics the behavior of a core and generates traffic compliant to the public protocol provided by the emulated NoC.
- *Traffic Receptors:* Similarly to TGs, our TRs have the same three interfaces, one to the processor, a second one to the network to be emulated, and a third one to the control module. We also have two types of TRs. The first one generates overall statistics from the executed emulation in hardware. For example, it can compute the average latency

of packets through the network thanks to a time stamp embedded in the packet. We have used 13 bits for this time stamp, what means that we expect the latency of packets to be less than 4 KCycles, but it is configurable by the user. The second type of TR does not generate global statistics, but a continuous flow of packet descriptors with detailed statistics of a certain interval of the emulation, which can then be analyzed by the software running on the control processor to create subsets of statistics for different emulation time-frames. As with TGs, TRs can be included in a simple network of switches or into a complete NoC emulation. In the second case, the TRs behave as slave cores (e.g., private or shared memories) and can include user-configured latencies to generate the different reply packets to the current transactions.

- *Control Module:* The control module is in charge of performing global synchronization operations, such as reset, start or stopping the emulation. Thus, it includes an interface for each included TG and TR to communicate simultaneously with all the components of the platform. The control processor can assert the commands to the control module through a write operation onto its memory address range.

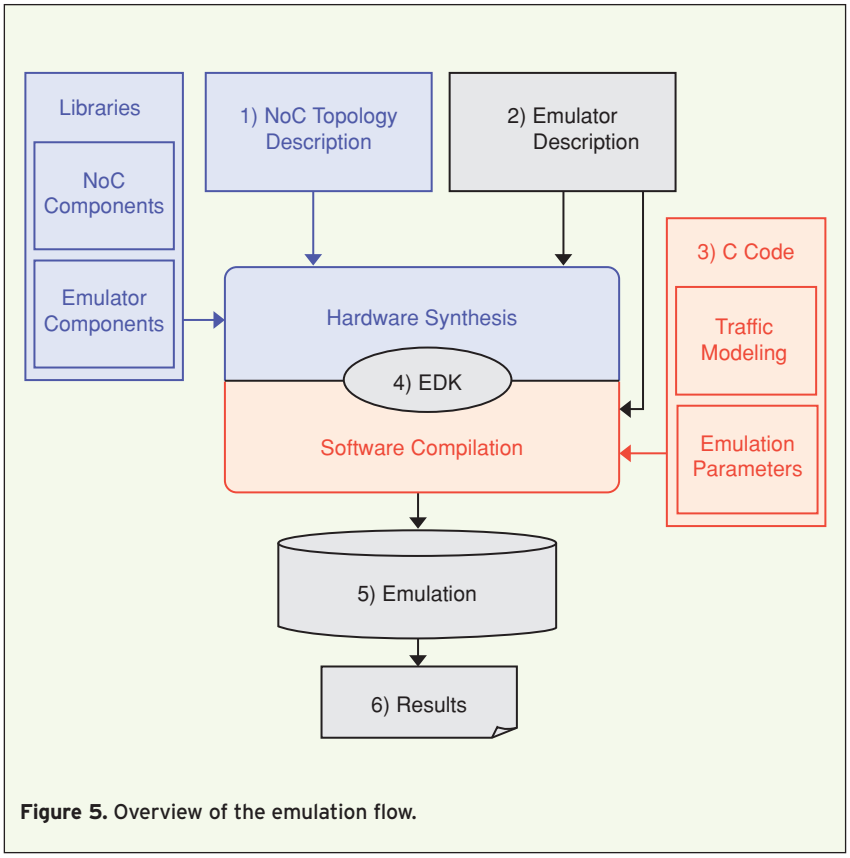


Figure 5. Overview of the emulation flow.

2.2 Software Initialization of Emulated NoC Hardware Architecture

A key element in the flexibility of the proposed framework for exploration purposes is the use of software to control the emulation parameters and process. For example, we can configure the routing tables of TGs by software. Also, the processor can read the generated statistics in the TRs (e.g., average latency of packets, congestion in each link, etc) and modify their response latencies by simply reading and writing in their memory address ranges. To make the software part as general as possible, we have implemented a set of functions to configure the emulation settings and to assert run-time commands (start, stop, reset, resume, etc.) in the framework via software.

We distinguish three types of software functions:

- *Hardware initialization functions:* They are called before the emulation of the device under test, and are meant to initialize the value of the registers of the hardware components (i.e., TG/TR and control module). For instance, in the case of stochastic TGs, these functions initialize their registers according to the traffic pattern that the user wants to generate.
- *Emulation control functions:* These functions synchronize the whole emulation process, by sending global requests to the control module (e.g., start or stop emulation), and send packets descriptors to TGs at run-time.
- *Statistic collection functions:* This type of functions is dedicated to read the statistics generated by stochastic TRs from their register files by the control processor. In addition, functions exist to read the output traces coming from trace-driven TRs, which include the timing of each packet among other NoC-related information (e.g., number of packets receive, sizes of acknowledge flits, etc.).

Thanks to the software functions and the instantiation of a superset of physical on-chip networks, it is possible to explore efficiently many different NoC features and prototype the behavior of applications running into MPSoCs without any time-consuming re-synthesis process, as it is shown in the next sections.

3. Emulation Flow: Hardware/Software Interaction and Statistics Collection

Due to its mixed hardware/software structure, the global emulation flow used in our framework (Fig. 5) includes elements of the flows employed for both sides of the system design spectrum: the definition of hardware setting and synthesis, and the software coding and compilation. The main element that we exploit to combine both sides in an overall emulation system is the *Embedded Development Kit* (EDK) tool. EDK supports the hardware integration of the NoC that we use [4], the synthesis of our additional hardware components (see Section 2) and the software compilation of our emulation software layer. This tool is related to the FPGA used, which is a Xilinx VirtexII Pro vp30 [13], but similar commercial tools from other vendors exist if other boards are used instead.

As a first step, we define the hardware layer of the platform. To this end, the user defines the network to be emulated (step 1 in Fig. 5) and the types of TGs/TRs to utilize. Then a control module compliant to the specific TGs and TRs used is included. Since our library of hardware components is already included in the EDK libraries, gathering all the components or modifying the current hardware emulation architecture requires only a few minutes.

Next, we interconnect all hardware modules in the framework with a shared bus such that they are accessible by the control processor (step 2 in Fig. 5). Then the initialization software settings for the current set of emulations are set (step 3 in Fig. 5). For this part, the user needs to configure the kind of traffic and statistic collection to extract from the emulation, according to the hardware architecture selected in the previous phase. This software configuration phase is not a very time-consuming effort thanks to the defined functions (cf. Section 2) that we have included in the software libraries of EDK. Thus, we avoid re-synthesis of the emulation framework between different emulations by including ranges of memory addresses where the control processor can read or write to configure the hardware platform each emulation run. Thus, the framework can autonomously perform several successive emulations by defining a batch of configurations in the C source file of the control processor. Once the whole configuration of the hardware/software elements is finished, the next step of the emulation flow (number 4 in Fig. 5) is the synthesis of the system. Then the emulation of the whole system is performed (fully automatic or user-interactive) as a fifth step of the emulation flow. The current version runs cycle-accurate emulations of NoC designs at 50 MHz on the FPGA device. Finally, either at run-time or at the end of the batch of emulations (number 6 in Fig. 5), the control processor collects the data and processes it to show the acquired statistics on the terminal window of the host computer.

4. Automatic NoC Parameters Exploration

We present now an iterative emulation procedure, which enables exploration of NoC parameters in the context of MPSoC design. In Fig. 6, we present the main steps that compose the NoC tuning and validation process.

This flow starts by defining a set of parameters that can be tuned for the particular NoC family that is tuned. These parameters concern the instances of NoC hardware components of the platform under test and the types of traffic patterns that need to be tested in the software configuration of the emulation platform. Examples of the hardware features that can be tuned are the flit width, the number of switches or the NoC topology, while software elements that can be studied are the traffic load

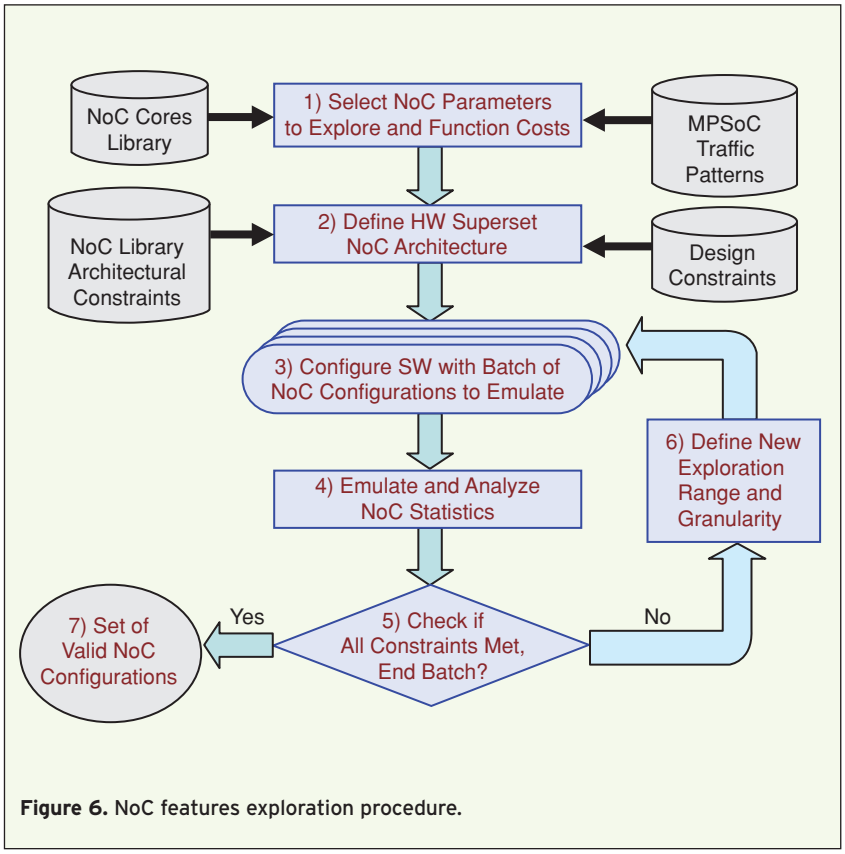


Figure 6. NoC features exploration procedure.

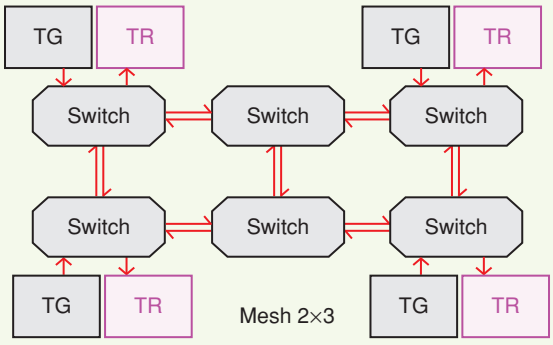


Figure 7. Topology for Application I.

and the response time of slave devices. Note that the large majority of these parameters are tunable by software in our emulation platform to avoid expensive re-synthesis. In the next step of our NoC tuning process (step 2), a parameter preprocessing and metric definition cost is performed to define the possible superset of hardware NoC architectures that cover the exploration design phase, while respecting the possible implementation constraints of the set of cores of the NoC under study (e.g., maximum switch or flit width sizes allowed in the NoC family). At this point it is possible to apply a first static exploration to reduce the possible configurations of NoCs to be emulated, as proposed in [14], but this part is beyond the scope of this paper. After the hardware of the NoC superset to be emulated is set, in the next step of the flow the batch of software configurations to explore is defined (step 3).

In the following step of the flow the overall system is emulated (step 4), and statistics of NoC behavior are extracted and analyzed in the control processor (step 5). Once the analysis phase has finished, the designer can choose to repeat steps 3 and 4 if the desired constraints and optimization goals have not been met, based on the solutions found in the output set. For these iterations, the designer can specify different strategies to refine the solutions found in the batch of NoC configurations explored (step 6) such as selecting solutions from the top and the bottom of the design spectrum, and then emulating new solutions at regular intervals between them to refine the granularity of the exploration. Finally, the solutions of the various set of emulations are available to the designer to select the best performance-area-power solution for the target MPSoC (step 7).

To illustrate this flow, consider for the sake of simplicity the problem of tuning only one parameter, namely, the flit width of the network with a fixed congestion constraint of 10% in the worst case [15]. The flit width exploration can be easily parameterized by software, by taking as hardware superset the synthesis of a NoC with the maximum flit width that can be taken into consideration during the exploration, and configure the NoC to use only a subset of the synthesized wires for smaller tests.

Following the architectural constraints of the NoC and MPSoC we are emulating [4], the exploration granularity for the flit width can vary from 4 to 128 bits, since 4 is the minimum allowed by the internal protocol of the network and 128 is the largest size needed for a complete parallel transaction of the emulated processing cores [14]. Thus, the design space is large and it would be unfeasible to cover it with pure software simulation or iterative exhaustive hardware re-synthesis for the different configurations. Using our hardware superset synthesis of 128-bit flit width, we vary the flit width from 4 to 128 utilizing our currently default implemented search method, namely, a dichotomic search. This method exhibits good convergence and stability properties in our experiments in NoC tuning, as congestion decreases when the flit width increases [14, 15]. Therefore, we divide the range in two equal parts and configure in software the emulation of configurations with 4, 66 and 128 bits. After this first set of emulations, the control processor analyzes the results and determines new bounds to explore the best configuration by reducing the design space by half. Then it automatically generates the software file for the new batch of configurations of the emulation framework, which reduces again the design space by half. Therefore, after only 7 emulations (log2132), the flit width configuration with minimal area, while respecting the congestion constraint, is found.

In addition, we can apply this emulation-supported exploration flow to tune several architectural NoC parameters in a multi-dimensional context, where different metrics (e.g., power, energy or area) and user-defined search methods can be applied by modifying the software of the control processor, and considering the largest superset of on-chip hardware architecture. Similarly, the effect of different real-life traffic patterns in the NoC configuration can be easily evaluated by modifying the control software for the included TGs and TRs.

5. Applications and Experimental Results

In this section, we provide two examples of real-life application of the proposed emulation framework and design flow to tune on-chip interconnects of MPSoCs at different abstraction levels. The first illustrates how to evaluate the performance of a network of switches with different internal NoC protocols, and the second shows the emulation of a complete NoC to explore the convenient NI configuration with respect to the traffic pattern of the running application.

5.1 Application I: Emulation of a Network of Switches

The first application presented of the proposed NoC emulation framework is the emulation of a network of X-pipes switches [4, 15]. In this experiment we have used an NoC topology of 6 16-bit switches in a 2 × 3 mesh configuration (depicted in Fig. 7), and we have placed stochastic TGs/TRs at each corner of the topology. Table 1 outlines that this platform uses 53% of the employed FPGA board. We have evaluated network performance and congestion level with different burst traffic patterns implemented using a two-state Markov chain (i.e., On/Off states) and we swept the probability to transit from one state to another using the software of the control processor. This implementation of the stochastic traffic required only few minutes to be configured and can already provide systems designers with a clear overview of how the network reacts in different working conditions.

With this experimental setup, we have extracted statistics about the amount of non-acknowledged flits in TGs (i.e., flits delayed because of network congestion), the average latency of packets, and the total runtime for different traffic patterns with a constant amount of packets. Additionally, we have measured the degree of congestion in the network with a global congestion counter. This congestion counter is the metric used in this example to apply the exploration procedure presented in the previous section. In this case, it is shown how our procedure can reach an optimum for a given congestion constraint, which is measured by the average latency of packets. We have fixed a latency constraint of 19 clock cycles, and explored with our flow different configurations of the number of flits per packet (L) and the number of packets per burst. Fig. 8 shows how the presented flow explores this design space. Our constraints from the underlying NoC architecture [4] initially determine that for $L > 15$ our topology cannot respect our latency constraint for all possible interconnections, and the use of less than 5 flits per packet would not be able to include the incoming data from the processing cores of the running multimedia application in one transaction. Thus, our framework explores the range $L = 5..15$, and gives as outcome to the designer the possible range of solutions between $L = 10..14$ packets per burst, as valid values for the desired latency constraint.

Additionally, the results shown in Fig. 8 give additional pieces of information about how the shape of the traffic affects the average latency of packets. It shows that the average latency of packets through the network of switches increases linearly with the number of packets bursts at the beginning, but after a certain amount of bursts it reaches a saturation value (i.e., $15 > L > 13$). This type of analyses is very relevant for on-chip designers to test their switches implementations with different NoC topologies, similarly as for the flit width parameter explored in the previous section.

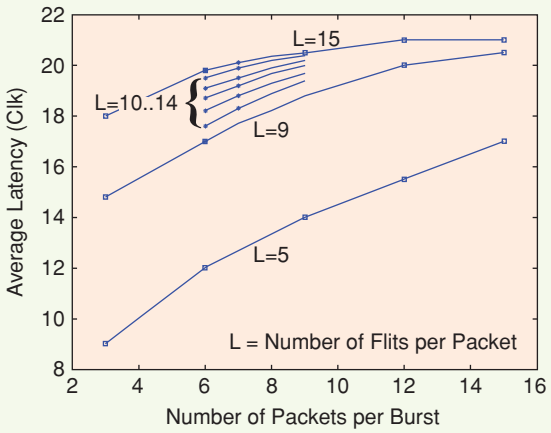


Figure 8. Experimental results obtained with Application I.

5.2 Application II: Emulation of Complete NoC Architectures

Our second application is the emulation of a full NoC, which includes a network of switches and the respective NIs implementing the public interface available to the external processing cores. In this case, the included TGs/TRs are *Open Core Protocol* (OCP)-compliant, according to the default interconnect standard supported by the emulated Xpipes NoC [4]. However, other internal/external protocols can be added in a few days using the available skeletons of TGs/TRs.

In this case, the internal NoC architecture is fixed and the main exploration and validation element is the behav-

Table 1.
Implementation figures of our emulation framework on a Virtex-II Pro VP30 FPGA.

Device Architecture	Number of Slices	FPGA (%)
Stochastic TG	719	7.8
Trace-Driven TG	652	7.0
Stochastic TR	371	4.0
Trace-Driven TR	690	7.4
Control Module	18	0.2
Network of Switches	7387	53
Complete NoC	7914	57

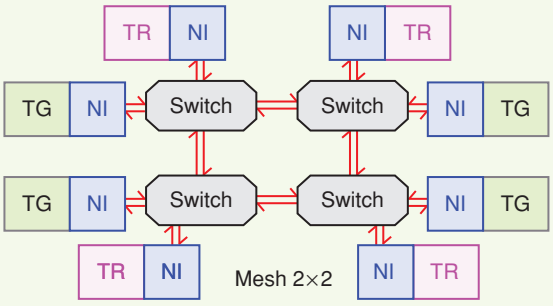


Figure 9. Topology for Application II.

ior at the external protocol level, from the traffic pattern viewpoint, of the MPSoC applications running on this architecture. This kind of analyses is very important to designer to anticipate over-designed or highly saturated NoCs in the presence of running applications different than the originally targeted. We have tested the behavior of a 2 × 2 mesh NoC interconnecting 8 cores (for 4 processing cores and 4 memory cores), which are emulated by including one TG and one TR connected to each switch, as shown in Fig. 9. This NoC-based architecture uses 57% of the FPGA board (see Table 1).

The master TGs receive memory transaction descriptors at run-time from the control processor, which were extracted from real-life traces of several multimedia applications [14]. The slave TRs analyze such traffic and reply to the master's requests. At the same time, they store a continuous stream of statistics in regular intervals of 1M cycles that can be read by the control processor.

The statistics extracted from this application of our NoC emulation framework are depicted in Table 2, where the main parameter is the instant OCP workload. Its variations are related to the features of multimedia applications running at each moment in time.

Table 2.
Experimental results for Application II.

OCP Load	Average Latency (Clk)			Acknowledgment
(%)	Read	Write	Overall	Ratio (%)
50.5	41	21	31	91
58.3	41	21	31	91
62.8	41	21	31	91
69.3	51	25	38	78
78.9	55	27	41	75
88.1	57	29	43	70

Our results show the effective latency of packets for the different OCP loads generated by the application, allowing system designers to evaluate if the acknowledgment ratio or packets correctly received within a certain fixed latency (an indication of the level of congestion in the network), which is defined as 14 clock cycles in this experiment, is sufficient for the running application. As Table 2 shows, for these NoC configurations with high OCP loads, the acknowledgment ratio decreases to 70% due to network congestion. Moreover, the continuous traces of statistics from TRs can provide a fine-grained analysis of latencies of different OCP transactions (read and write operations). This analysis shows that in this network implementation, congestion affects more read than write operations, as there is no support implemented in Xpipes for splitting read operations. Thus, the presented emulation framework can provide NoC designers indications of possible architectural improvements for different domains of MPSoC applications.

Conclusions

One of the most critical elements in new complex MPSoC designs is the realization of the interconnections between the different cores of the system. To this end, the NoC paradigm has been proposed as a very recent and promising method to reduce the complexity of integrating on-chip tens of cores. However, NoC physical design for MPSoC creates a new set of challenging issues, such as, the definition of suitable topologies or low-latency protocols development, which are heavily influenced by the traffic generated by each type of MPSoC. Thus, it is necessary to develop complete frameworks to validate and effectively test different physical NoC implementations. In this article, we have presented a flexible hardware-software emulation framework implemented on an FPGA, which is targeted to implement and validate a wide range of NoC solutions with a very limited effort from the designer's viewpoint. Its application on an exploration flow of NoC characteristics at the physical level has illustrated that it enables a very fast tuning process of NoCs for target MPSoCs. Furthermore, it is a powerful tool for NoC designers to identify possible limitations of their on-chip interconnection architectures.

Acknowledgments

This work is partially supported by the Swiss FNS Research Grant 20021-109450/1 and the Spanish Government Research Grant TIN2005-5619.

References

[1] A. Jerraya and W. Wolf, *Multiprocessor Systems-on-Chips*. Morgan Kaufmann, Elsevier, 2005.

[2] G. De Micheli, L. Benini et al., *Networks on Chip*, Morgan Kauffman, 2006.

[3] W. Dally, B. Towles, *Principles and Practices of Interconnection Networks. NoC*. Morgan Kauffman, 2004.

[4] D. Bertozzi, et al., "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *Trans on Parallel and Distributed Systems*, 2005.

[5] G. Brebner and D. Levi, "Networking on chip with platform fpgas," in *Proc. FPT*, 2003.

[6] T. Marescaux et al., "Networks on chip as hardware components of an os for reconfigurable systems," in *Proc. of FPL*, 2003.

[7] F. Moraes et al., "Hermes: an infrastructure for low area overhead packet-switching NoC.," *Integration-VLSI Journal*, 2004.

[8] K. Goossens, et al., "The Aethereal network on chip: Concepts, architectures and implementations," in *IEEE Design and Test of Computers*, 2005.

[9] D. Siguenza-Tortosa et al., "VHDL-based simulation environment for Proteo noc.," in *Proc. HLDVT Workshop*, 2002.

[10] J. Madsen, S. Mahadevan, et al., "NoC modeling for system-level multiprocessor simulation," in *Proc. RTSS*, 2003.

[11] J. Chan et al., "Nocgen:a template based reuse methodology for NoC architecture," in *Proc. ICVLSI*, 2004.

[12] W. Hang-Sheng, et al., "Orion: a power-performance simulator for interconnect. Networks," in *Proc. MICRO*, 2002.

[13] Xilinx Corporation. Xilinx Virtex-II Pro FPGA and EDK. Available: http://www.xilinx.com, 2005.

[14] N. Genko, D. Atienza, et al., "A complete network-on-chip emulation framework," in *Proc. DATE*, 2005.

[15] S. Murali et al., "Designing application-specific networks on chips with floorplan information," in *Proc. ICCAD*, 2006.

Nicolas Genko is a Ph.D. candidate at Ecole Polytechnique Fédérale de Lausanne (EPFL). In 2004, he was visiting researcher at Stanford University and graduated from Institut Supérieur d'Electronique de Paris (ISEP). His research interests include emulation of complex systems on FPGA with emphasis on thermal issues and on-chip communication infrastructures.

David Atienza received the M.S. degree in Computer Science in 2001 and the European Ph.D. degree in Computer Science from Complutense University of Madrid (UCM), Spain, and Inter-University Micro-Electronics Center (IMEC), Belgium, in 2005. Currently he is Post-Doctoral Researcher at the Integrated Systems Laboratory (LSI) in EPFL, Switzerland. He also holds the position of Associate Professor at the Computer Architecture and Automation Department (DACYA) of UCM. His research interests focus on the definition of flexible emulation frameworks to explore thermal management techniques for MPSoCs, NoC interconnection design, dynamic memory management and low-power design of embedded systems. In these fields, he is author of more than 80 publications in prestigious journals and at international conferences, including, IEEE Micro, ACM TODAES, IEEE T-VLSI Systems, Integration—The VLSI Journal, IEEE/ACM DATE, IEEE/ACM DAC, etc. Also he is part of the Technical Program Committee of the IEEE/ACM DATE, IEEE ICCAD, IEEE GLSVLSI and IEEE VLSI-SoC conferences.

Giovanni De Micheli is Professor and Director of the Integrated Systems Centre at EPF Lausanne, Switzerland, and President of the Scientific Committee of CSEM, Neuchatel, Switzerland. Previously, he was Professor of Electrical Engineering at Stanford University. He is author of: Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994, co-author and/or co-editor of six other books and of over 300 technical articles. He is a Fellow of ACM and IEEE. Prof. De Micheli is the recipient of the 2003 IEEE Emanuel Piore Award for contributions to computer-aided synthesis of digital systems. He received the 1987 D. Pederson Award for the best paper on the IEEE Transactions on CAD/ICAS, two Best Paper Awards at the Design Automation Conference, in 1983 and in 1993, and a Best Paper Award at the DATE Conference in 2005. His research interests include several aspects of design technologies for integrated circuits and systems, such as synthesis, hardware/software codesign and low-power design, as well as systems on heterogeneous platforms including electrical, optical, micromechanical and biological components.

Luca Benini received the B.S. degree (summa cum laude) in electrical engineering from the University of Bologna, Italy, in 1991, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University in 1994 and 1997, respectively. He is currently a Full Professor in the Department of Electronics and Computer Science in the University of Bologna. He also holds a visiting professor position at EPFL. Dr. Benini's research interests are in all aspects of computer-aided design of digital circuits, with special emphasis on low-power applications, and in the design of portable systems. On these topics, he published more than 200 papers in international conferences and journals, and he is co-author of three books.

Dr. Benini is a member of the technical program committee for several technical conferences, including the Design Automation Conference (DAC), the International Symposium on Low Power Design (ISLPED) and the International Symposium on Hardware-Software Codesign (CODES/ISSS). He has been the Program Chair of the 2005 Design, Automation and Test in Europe (DATE) Conference.