A Complete Network-On-Chip Emulation Framework

N. Genko*, D. Atienza†, G. De Micheli*,
J. M. Mendias†, R. Hermida†, F. Catthoor+
* Stanford University, Palo Alto, USA. {ngenko, nanni}@stanford.edu
†DACYA/UCM, Juan del Rosal 8, Madrid, Spain. {datienza, mendias, rhermida}@dacya.ucm.es
+ IMEC vzw, Kapeldreef 75, Leuven, Belgium. {catthoor}@imec.be
*

Abstract

Current Systems-On-Chip (SoC) execute applications that demand extensive parallel processing. Networks-On-Chip (NoC) provide a structured way of realizing interconnections on silicon, and obviate the limitations of bus-based solution. NoCs can have regular or ad hoc topologies, and functional validation is essential to assess their correctness and performance. In this paper, we present a flexible emulation environment implemented on an FPGA that is suitable to 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. With our emulation framework, designers can explore and optimize a various range of solutions, as well as characterize quickly performance figures.

1 Introduction

With the growing complexity in consumer embedded products, new tendencies envisage heterogeneous *System-On-Chip* (SoC) architectures consisting of complex integrated components communicating with each other at very high-speed rates. Intercommunication requirements of SoCs made of hundreds of cores will not be feasible using a single shared bus or a hierarchy of buses due to their poor scalability with system size and their shared bandwidth between all the cores attached to them.

To overcome these problems of scalability and complexity, *Network-On-Chip* (NoC) has been proposed as a promising replacement for buses and dedicated interconnections [1, 8]. NoCs involve the design of network interfaces to access the on-chip network and switches to provide the physical interconnection mechanisms to transport the data of the cores. Therefore, the definition and implementation of NoCs involve a complex design process, for instance, the selection of suitable protocols or topologies of switches to use.

Concrete options for NoC topologies and interfaces have been proposed at different levels of abstraction [13, 9, 10, 2] and some even implemented onto FPGAs for functional validation. Nevertheless, these different physical implementations onto FPGAs are limited in flexibility and do not allow a full test of different actual realizations of NoC on silicon.

In this paper, we present a complete mixed HW-SW NoC emulation framework where a wide range of NoC features can be easily instantiated and compared at the physical level. As a result, this emulation framework provides a consistent way to test the performance achieved by actual physical realizations of NoCs on silicon at a very high speed (16000 times faster than an HDL simulator). To this end, it is implemented onto an FPGA platform and supplies a wide range of statistics for the different traffic patterns that can be generated in NoCs. In addition, our framework implementation is very modular and the statistics reports are easily extensible for further testing of concrete effects (e.g. saturations effects in parts of NoCs) on a particular NoC instantiation.

The remainder of the paper is organized as follows. In Section 2, we describe some related work. In Section 3, we present the architecture of our emulation framework. In Section 4 we detail how the emulation process of NoC systems is performed with our platform. In Section 5, we introduce several case studies and present the experimental results. Finally, in Section 6 we draw our conclusions.

2 Related Work

In the last years, significant research has been done to evaluate the design and implementation features of NoC at its different levels of abstraction. To provide accurate functional validation (i.e. circuit level), several approaches have been implemented in FPGAs. In [10] and [2], NoCs with a mesh-based topology and packet-switching as communication mechanism have shown the effectiveness of NoC. Also, other NoC architectures (e.g. torus) and designs of switches/routers have been ported to FPGAs in order to validate their NoC features (e.g. packet sizes, switching-mode) based on additional HDL simulations [11, 18, 19, 4]. These previous approaches can validate several NoC implementa-

*This work is partially supported by NSF under contract CCR-0305718, by a grant from Jerry Yang and Akiko Yamazaki, a Fellowship of Institut Superieur d'Eletronique de Paris and by the Spanish Government Research Grant TIC2002/0750.

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)
1530-1591/05 $ 20.00 IEEE

tions features, but none of them is designed to exhaustively test the details of NoC topologies and traffics as ours.

To evaluate in detail different architectural alternatives reducing the cost of synthesizable NoC design, several cycle-accurate simulation environments have appeared. In [15], VHDL is employed to evaluate several features of virtual channels in mesh-based and hierarchical NoC topologies. In [13], XML and SystemC are used to specify the NoC components (e.g. routers, network interfaces) and to test mesh-based NoC design alternatives. The main difference with our approach is that their simulations have a much larger execution time compared to our physical NoC emulation environment.

To increase the simulation speed of cycle-accurate VHDL, several approaches have been proposed. [7] and [9] describe modeling environments for custom NoC topologies based on SystemC. [3] presents a mixed VHDL/SystemC implementation and simulation methodology using a template router to support several interconnection networks. In [5], a C++-based library of communication APIs is built on top of SystemC to explore NoCs topologies. Finally, [12] presents a fast transaction level modeling approach to explore bus-based communication architectures. While the previous approaches enable the fast exploration of the main features of NoC designs as our proposed emulation platform, their level of accuracy in the estimations and their simulation speed is more limited compared to our complete physical emulation of parameterizable NoCs.

Other proposed approaches improve the speed of cycle-accurate NoC simulations by using high-level abstraction languages, e.g. C or C++. [6] presents a C-based interconnection network simulator. Similarly, [17] proposes an event-driven C++ simulator. Also, [8] presents a NoC design methodology that uses a parameterizable NoC architecture executed in a high-level event simulator. At a higher-level of abstraction, several algorithms, analytical models and heuristics have been proposed to achieve very fast rough estimations of the cost of NoC topologies based on graphs represention [16, 14]. Although these approaches attain high simulation speeds (sometimes close to real hardware), they cannot obtain detailed statistics of final physical implementation systems as our emulation framework does.

3 NoC Emulation Architecture

Our emulation framework has been designed in a modular way to easily implement various custom NoC topologies and architectures. An overview of the architecture of our platform is depicted in Figure 1. It consists of three main elements, which are mapped onto an FPGA board with a hard-core processor. We have used a Xilinx Virtex 2 Pro v20 with an embedded Power PC. The hard-core processor of the FPGA is used to orchestrate the emulation process in a flexible way. Then, the monitor module provides the interface to communicate with the host PC and show the produced statistics onto its screen through the serial port. Finally, the main element of our NoC emulation framework is the NoC programmable emulation platform. It is a module that consists of the necessary elements to emulate a network of switches: *Traffic Generators* (TGs), *Traffic Receptors* (TRs) and a user-defined set of interconnections between the switches of the network. The originality of our platform is that TGs and TRs are addressable by the processor. By this way, the processor can set in those devices some parameters, which determines the behavior of the devices during emulation. The switches are generated using the Xpipes compiler [7]. The previous modules communicate using a common bus available in our FPGA board called On-chip Peripheral Bus (i.e. OPB in Figure 1). This emulation platform consumes a limited amount of logic and enables to instantiate and to emulate real NoCs on current FPGAs. For example, the instantiation of a NoC proposed in [7] including 6 switches and 4 TGs and 4 TRs uses 7387 Xilinx slices (79% of our device), and its emulation time for 16 million packets on our platform clocked at 50 Mhz takes 3.2 sec. Therefore, with present larger devices used in NoC (e.g. Virtex 2 Pro v40), large NoCs (i.e. More than 40 switches and TGs/TRs) can be emulated rapidly.

The first component of the emulation platform is a filter which drives *Internal Buses* (IBs). This unit manages the decoding of the address bus. In our design, we use a 12-bit address bus, which drives up to four IBs and up to 1024 devices per IB. We currently use only two IBs. The first IB is used to communicate with the Control unit. This component (also addressable by the processor) can communicate with all the other components of the platform by sending some control signals simultaneously to all of them. The second bus drives the present TGs and TRs. Each TG generates different traffic (see Subsection 3.1) that is injected to the network of switches through a dedicated connection for each TG. Then, after passing through the network of switches, the traffic is received and analyzed by a set of TRs.

In the following subsections we describe in detail the functionality of the different types of TGs/TRs that are available in our emulation platform.

3.1 Traffic Generators

In our platform the TG module offers two possible implementations or working modes. First, a TG can generate stochastic traffic. Second, it can use input traffic traces generated by real-life applications, thus emulating the behavior of real workloads for the NoCs. In our platform, both types of TG can be used at the same time by including a controller for each used type (Figure 1). These types of TGs are explained in detail, including their implementation figures, in the following subsections.

3.1.1 Stochastic Traffic Generators

The TG includes tree different interfaces. The first one is linked to the Internal Bus 1 (IB1). Using this IB1 interface, some registers are initialized by the processor in each TG to

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)
1530-1591/05 $ 20.00 IEEE

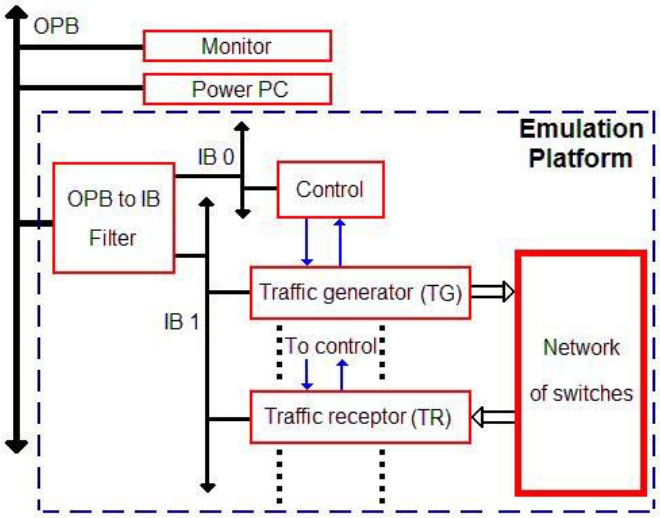

Figure 1. NoC Emulation Architecture

define the type of stochastic traffic to inject into the network of switches and the total emulation time. The second interface enables the communication of the TG with the Control module. Finally, the third one connects the TG with a network channel used to send the traffic of packets into the network of switches that simulate the internal physical connections of the emulated SoC.

To make the traffic generation as flexible as possible for NoC exploration, the TG can generate different types of stochastic traffic, which are the following:

- UNIFORM DISTRIBUTION: the length of packet, the latency between packets and the destination of packets are randomly chosen using initialized ratios that can be provided by the user and a *Linear Finite Shift Register* (LFSR) of 80 bits randomizes the process.

- BURST-MODE: this mode emulates typical bursts modes generated from real cores using a 2-state Markov chain. In the burst-mode state, all the packets are sent according to one latency indicated by the user. In the stable state, the TG is stopped for the amount of time indicated by the user. The probabilities of making a transition between the two states are set by the user at the initialization of the emulation. Then, the switching between two states is randomized by a LFSR.

Finally, note that any other stochastic mode can be incorporated to our emulation platform simply by making small modifications in that part of the TG code and adding it as an additional version of TG to our emulation environment. This process just takes a matter of minutes. Currently, this type of TGs consume on our platform 719 slices each on average (7.74% of the target FPGA) and this figure does not vary significantly (less than 3%) when other stochastic traffic generation functionality are implemented.

3.1.2 Trace-Driven Traffic Generators

Instead of working with traffic model, the trace-driven TG uses an image of some real traffic obtained from any actual

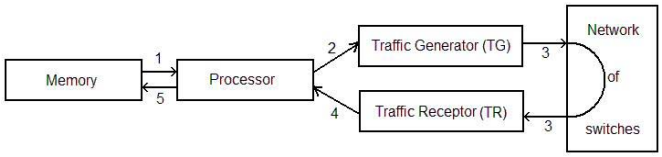

Figure 2. Data flow in trace-driven TGs

application. A trace is a set of traffic descriptors, which represents a packet. Each packet descriptor is made of three fields: a packet length, a destination and a time to inject the packet into the NoC. Before any simulation, the user must store a trace in the memory. The memory used is a regular DRAM available on the FPGA board. Some board can support up to 4GB of RAM, which could store a trace, which represents 4 billions packets.

As depicted on Figure 2, a continuous stream of packet descriptors are fetched from main memory by the processor and are sent to TGs (labeled as 1 and 2 in Figure 2). Then, the TGs store these descriptors in a buffer and inject the packets into the network of switches at the moment the traces indicate (labeled as number 3).

A very important feature of this type of TG is that it can provide an image of the congestion of the network at each moment in time of the emulation. In fact, each time a flit is not acknowledged by its receptor (i.e. switch or TR) and has to be resent, a readable counter by the processor is incremented.

As with the previous type of TGs (Subsection 3.1.1), the implementation of the trace-driven TG tries to save as much logic as possible in the board. Thus, each instantiation consumes 652 slices (7.02%) on our FPGA device.

3.2 Traffic Receptors

Similarly to the TGs, we have included two different implementations of TRs in our emulation platform. They provide different kind of statistics to the user. In addition, both can be used in a debug mode, and in two corresponding submodes. First, it can perform an automatic check of the flits received via CRC check to guarantee that they are the correct ones sent by the TGs. Second, for manual checking, the content of the flits can be shown on the screen of the host PC to verify their content. The use of two different TRs implementations allows us to achieve an efficient implementation according to the required type of reports to generate and a suitable debug tool for the network. The two types of TRs are described in the following subsections.

3.2.1 Traffic-Activity-Analysis Receptor

This first type of TR has to be initialized by the processor to define the granularity of the analysis in the emulation. This granularity corresponds to the amount of clock cycles per taken sample of NoC traffic. During this specified time, the receptor counts the number of acknowledged flits. By this way, the TR can generate a histogram that represents the activity in the receptor. Then, the results can be monitored

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)
1530-1591/05 $ 20.00 IEEE

by the hard-core processor. This type of TR uses 371 slices (3.99%) on the FPGA.

3.2.2 Independent-Packet-Trace-Analysis Receptor

The second type of TR generates a trace report for each received packet. The trace has the same format as the one used by the TGs. Typically, the received trace would not be directly monitored, but stored in a memory accessible by the Power PC. Then, the processor can compute a detailed analysis (e.g. latency, arrival time) for each delivered packet that can be provided to the user. This TR is more complex than the one described previously and takes 690 slices (7.4%) on the FPGA.

3.3 Control Module

The Control module is addressable by the processor and takes care of the synchronization of all traffic devices in the platform (i.e. TGs and TRs). For instance, it makes sure that all devices start the emulation at the same time. Also, the controller has the ability to reset the whole platform or even stop it. This is useful if the emulation platform needs to be programmed to execute several consecutive emulations.

In our system, one controller module is required per set of types of TG/TR used in the system (e.g. stochastic, trace-driven). Therefore, since we have at least one kind of traffic device in a basic emulation platform (i.e. one type of TG and TR), we include at least one type of Control modules for each emulation. This is not a problem since the amount of slices used by each instantiation of the Control module is very small, i.e. 18 slices (0.19%) on the FPGA.

3.4 Implementation of Network Interfaces

In our platform the TGs/TRs are used to accurately control the traffic on the NoC. Additionally, we have introduced the option to use *Network Interfaces* (NIs) on our platform at each edge of the network of switches. This provides an additional way to interconnect new components to the emulated NoC (e.g. external memories, processors, etc.) and introduce more flexibility in our platform. The NIs implemented on the board are *Open Core Protocol* (OCP) compliant [20]. Note that in this case the routing of packets is statically implemented in NIs while it is dynamically provided when TGs are directly plugged into the network of switches. In this case, TGs/TRs are replaced by NIs and external cores.

In order to keep the programmability of our platform, we have introduced some programmable OCP Master and Slave devices. Like our TGs/TRs, those devices behave according to some registers addressable by the processor. Our results show that the traffic generated/received is different, showing that this way additional and more realistic studies of the influence of NIs in real-life NoCs can be performed in our emulation framework (as in transaction-level NoC simulators, but at much faster speed).

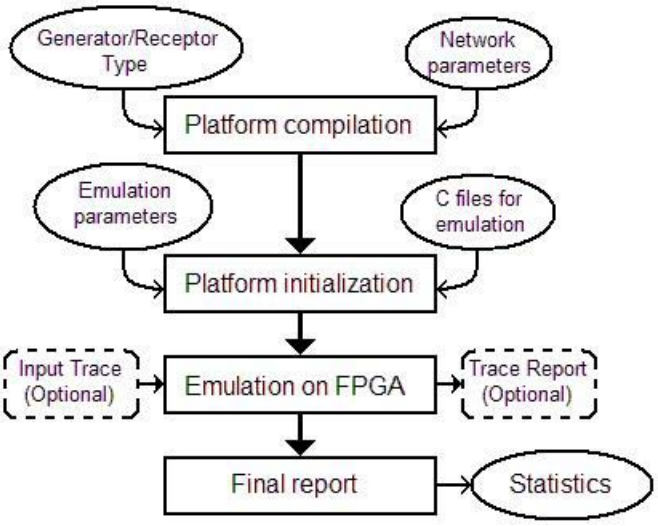

Figure 3. Our NoC Emulation Flow

Finally, adding cores with NIs on our platform still allows us to use TGs/TRs at the same time, which enables further research studies. For example, this kind of configuration can be used to study the behavior of a core when the NoC is artificially congested by TGs.

4 Versatile Emulation Framework

The main feature of our emulation framework and its flow is the simple initialization and statistics acquisition of any circuit level emulation without re-synthesizing and remapping the whole system. This is possible thanks to its mixed HW-SW structure, which allows the processor to initialize some parameters in the hardware part of the platform. The emulation flow in our FPGA environment varies slightly in case either a stochastic or a trace-based emulation is performed. An overview of these two emulation flows is shown in Figure 3. In the following subsections we describe in detail the internal phases in each flow.

4.1 Stochastic Emulation Flow

As Figure 3 indicates, from the hardware point of view, thanks to the components explained in Section 3 we can emulate at the circuit level various switching configurations of a NoC. The precise configuration to use in our emulations is defined in the first phase (first square in Figure 3) by configuring the Verilog code of our platform, which is a matter of defining several parameters.

After that, in our flow the initialization of the stochastic traffic to generate (e.g. uniform, burst-mode, etc) is performed (second box in Figure 3) using the processor of the FPGA board (i.e. Power PC). It executes a file with C code that contains the software code to configure the system. This file contains information about the total emulation time, the sampling period for statistics generation, routing information, latencies to use between packets, flits per

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)
1530-1591/05 $ 20.00 IEEE

packet and stochastic traffic distribution. Thus, this enables a high flexibility because no time-consuming recompilation of the HW involved is needed to emulate and study a wide range of these parameters. Then, during the initialization phase, by reading and writing in the TGs/TRs, the processor transmits data to TGs/TRs and is configured to receive results of the specified analysis at the end of the simulation.

After that, the emulation works autonomously and the TGs/TRs acquire the information necessary to generate the statistics demanded by the user in its C configuration file.

Finally, at the end of the emulation, the stored statistics are sent back to the processor which displays a summary report about the behavior and congestion of the network on the screen of the user using the monitor module (see Section 3) and its serial interface to the host PC. The information contained in this report is related to (1) average latency in the emulation, (2) amount of packets sent/received in each TG/TR, (3) delivery time for each burst of flits, (4) total emulation time and (5) histogram of flits delivered according to the granularity defined by the user.

4.2 Trace-based Emulation Flow

The trace-based emulation flow is equivalent to the stochastic flow explained previously, as far as the configuration of the network parameters and the TGs/TRs types are concerned. Then, the main difference is related to the way the emulation is performed. In this case, an entire NoC trace can be loaded by software in a RAM located on the FPGA board. Then, the processor uses it to generate a continuous stream toward the emulation platform and receives another continuous stream of output traces, which can be used to generate detailed statistics (e.g latency, congestion) for each packet sent through the emulated NoC.

5 Case Studies and Experimental Results

We have applied our proposed emulation framework to several custom NoCs design to validate its emulation speed and versatility to change different NoC parameters and types of statistic measurements.

We first compared our emulation performances with traditional simulators. A HDL simulator (Modelsim) was able to compute 3.2k cycles/s while a SystemC simulator runs at 20k cycles/s. Our platform is driven by a 50MHz clock, which represent a significant improvement in speed. For example, 1 billon packets will be emulated in 3'20" in our platform but it would take about 6 days to simulate it in a SystemC simulator.

As a first experiment, we have used the stochastic TGs in the two different modes explained in Subsection 3.1.1 with a NoC of 6 switches. The instantiated network of switches is a flat topology 3x2 with 7 bidirectional link between switches. One TG and a TR have been placed at each edge of the network. In this case, we have compared the influence of the traffic model on the congestion of the network. The left graph of Figure 4 shows the relation between runtime of a traffic pattern and the number of delivered packets. The curves indicate the different results obtained with uniform-traffic and burst-traffic generation patterns. For the uniform traffic, we have used a packet length of 5 flits and latency between packet randomly chosen between 4 and 8 clock cycles. Those parameters have been decided to generate congestion on the network after evaluating a large range of values for them by configuring our emulation environment via the C file of the processor. With this configuration, the traffic load at the output of each TG is 45% of the maximum available bandwidth and the routing has been set in order to include some shared links on the network between several paths. For the burst-mode traffic, we have used several latencies for the input parameters to provide the same traffic load at the output of each TG with the same packet length.

As Figure 4 shows, the total delivery time with the same amount of packets for the burst-mode is higher than for the uniform traffic. This is because the probability of collisions between packets in the burst-mode is significantly higher. Also, as Figure 4 indicates, the probability of collisions initially increases fast when the number of packets to deliver grows, but at a certain level of congestion it stabilizes.

In our second set of experiments we have used the trace-based emulation devices (i.e. trace-driven TGs and TRs) to study the congestion produced with real traces on a network of 4 switches. The topology that is the focus of our study in this case is a flat 2x2 topology with one TG and one TR at each corner. In this case, we have measured the congestion rate compared to the number of delivered packets per burst in the burst mode. As in the previous case, we have set the NoC parameters to obtain a traffic load of 45% at the output of each TG. Since the traces have been captured in software by the processor, the traces from any real-life application can be easily used as input for the system. In this case, we have used a burst mode and the number of trace units (i.e. packets) to generate and inject into the network can be easily modified in the software of the processor. The results obtained are shown in the middle graph of Figure 4. They indicate that the congestion rate does not increase linearly with the number of delivered packets in a burst mode. In fact, the more packets are sent during a burst mode, the longer the inactivity between two burst modes is. Thus, the collision probability grows less than linearly. Also, the different curves show that the size of the packets is a relevant design constraint. The lower curve has been obtained with a packet length of 2 flits, the middle one with a length of 5 and the upper one with a length of 8. This indicates that when the number of packets in a burst mode increases significantly, the influence of the packets sizes becomes almost irrelevant. This is explained by the fact that the congestion tends to be the same when the burst mode length (i.e. number of packets $\times$ length of packets) is relatively large.

Finally, in the third set of experiments, we have measured with our trace-based emulation framework the latency of packets through the network. Thus, the graph on the right

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)
1530-1591/05 $ 20.00 IEEE

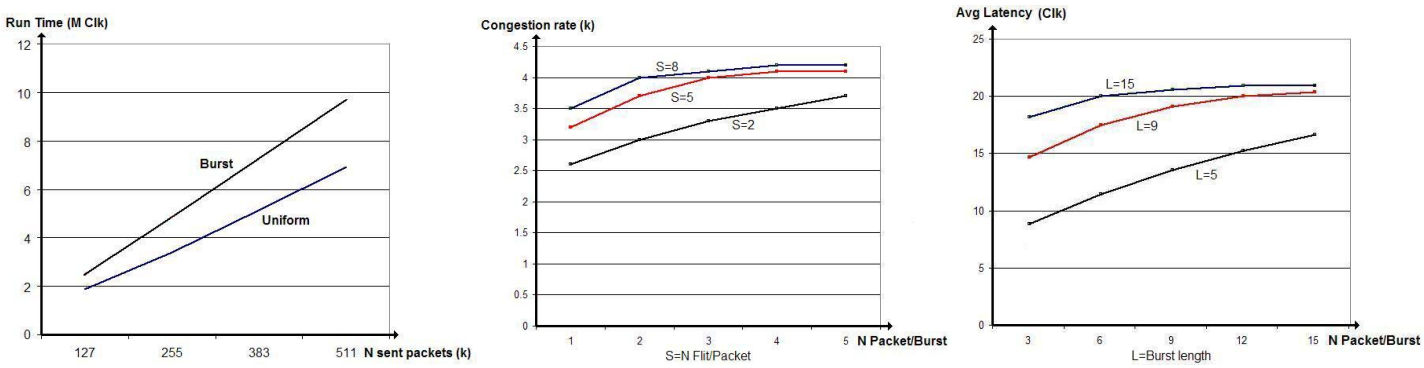

Figure 4. Emulation results for stochastic and trace-based experiments

side of Figure 4 shows the average packet latency through the network versus the packet size. Similarly to the other case studies, we have explored and defined the parameters to inject the same traffic ratio in the network by all TGs. As in the second set of experiments, the average latency of packets reaches a limit of congestion, which is the limit of the NoC in terms of latency.

As a final test, we have increased the load generated by each TG to 55% in all the previous experiments. As a consequence, some links were overloaded and the limits extracted from the two previous experiments was largely exceeded. It shows the importance of the design of the network regarding expected traffic bandwidth between cores in a single SoC.

Note that the exploration results of these experiments were obtained in few hours thanks to the flexibility and speed of our proposed SW/HW emulation framework compared to similar approaches at the cycle-accurate level. Our platform enables changing the parameters of the traffic model and NoC characteristics (e.g. latency) in a matter of minutes. Also, it achieves the fast emulation speed of the system at physical level without the recompilation of a pure hardware platform, which can take about an hour. Thus, workloads of real NoCs (i.e. billions of packets) can be emulated in our HW/SW approach in few minutes, while in cycle-accurate simulators will take several days.

6 Conclusions

New consumer products have increasingly higher demands and complex SoCs are used to implement such systems under the tight time-to-market constraints. NoCs solutions have been proposed to reduce the complexity of integrating tens of cores on-chip, but none of them allows complete architectural studies of different NoC realizations on silicon. In this paper, we have presented a flexible HW-SW emulation environment implemented on an FPGA that is suitable to explore, evaluate and compare at the physical level various custom NoC solutions for these new consumer systems with a very limited implementation effort. Moreover, as we have shown, a large set of important implementation and design parameters for actual NoCs can be evaluated on this proposed emulation platform in a very short interval, thanks to its HW-SW framework design to avoid multiple hardware synthesis on the FPGA and its fast emulation speed.

References

[1] L. Benini and G. De Micheli. Networks on chip: a new soc paradigm. *IEEE Computer*, January, 2002.

[2] G. Brebner and D. Levi. Networking on chip with platform fpgas. In *Proc. FPT*, 2003.

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

[4] D. Ching, P. Schaumont, et al. Integrated modeling and generation of a reconfigurable NoC. In *Proc. IPDPS*, 2004.

[5] M. Coppola, et al. Occn: a NoC modeling and simulation framework. In *Proc. DATE*, 2004.

[6] W. Hang-Sheng, et al. Orion: a power-performance simulator for interconnect. networks. In *Proc. MICRO*, 2002.

[7] A. Jalabert, S. Murali, et al. xpipescompiler: A tool for instantiating application specific NoC. In *Proc. DATE*, 2004.

[8] S. Kolson, A. Jantsch, et al. A NoC architecture and design methodology. In *Proc. Annual Symp. VLSI*, 2002.

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

[10] T. Marescaux, J.I. Mignolet, et al. NoC as hw components of an os for reconfigurable systems. In *Proc. FPL*, 2003.

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

[12] S. Pasricha, et al. Fast exploration of bus-based communication architectures at the ccatb abstraction. In *DAC*, 2004.

[13] S. Pestana, E. Rijpkema, et al. Cost-performance trade-offs in NoC: a simulation-based approach. In *Proc. DATE*, 2004.

[14] A. Pinto, et al. Efficient synth. NoC. In *Proc. ICCD*, 2003.

[15] D. Siguenza-Tortosa et al. Vhdl-based simulation environment for proteo noc. In *Proc. HLDVT Workshop*, 2002.

[16] L. Tang and S. Kumar. Algorithms and tools for NoC based system design. In *Proc. SBCCI*, 2003.

[17] D. Wiklund, S. Sathe, et al. NoC simulations for benchmarking. In *Proc. IWSoC for Real-Time Apps.*, 2004.

[18] C. Zeferino, M. Kreutz, et al. Rasoc: a router soft-core for NoC. In *Proc. DATE*, 2004.

[19] C. Zeferino and A. Susin. Socin: a parametric and scalable NoC. In *Proc. SBCCI*, 2003.

[20] OCP International Partnership (OCP-IP). Open Core Protocol Standard. 2003. http://www.ocpip.org/home

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)
1530-1591/05 $ 20.00 IEEE