A Novel Approach for Network on Chip Emulation

N. Genko⋆, D. Atienza†+, G. De Micheli⋆,
L. Benini‡, 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, hermida}@dacya.ucm.es
‡DEIS/Bologna University, Viale Risorgimento, 2 Bologna, Italy. {lbenini}@deis.unibo.it
+ IMEC vzw, Kapeldreef 75, Leuven, Belgium. {catthoor}@imec.be. Also professor at K.U. Leuven.

***Abstract*— Current Systems-On-Chip 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 solutions. 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 range of solutions, as well as characterize quickly performance figures.**

I. Introduction

In the near future, *System-On-Chip* consumer devices will contain billions of transistors thanks to nanoscale technologies, but will face additional constraints. 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 among all the attached cores.

Network-On-Chip (NoC) has been proposed as a promising replacement for buses and dedicated interconnections [1], [6] to solve the scalability and complexity problem. NoCs involve the design of network interfaces to access the on-chip network, the selection of suitable protocols and topologies of switches to transport the data. Hence, the NoC paradigm implies a new complex design and research topic for on-chip communication. Presently, concrete options for NoC topologies and interfaces have been proposed at different levels of abstraction [10], [7], [9] and some even implemented onto FPGAs for functional validation [8], [2]. Nevertheless, these different physical implementations onto *Field Programmable Gate Arrays* (FPGAs) are limited in flexibility and do not enable a full test of different actual realizations of NoC on silicon. Moreover, an architectural exploration of the involved design parameters such as packet size, number of switches or topologies implies a complete redesign of the physical implementation on the FPGA, which is a time-consuming effort.

In this paper, we present a complete mixed HW-SW NoC emulation framework where a wide range of NoC features (e.g. number of switches or connecting elements, topologies, etc.) 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 a HDL simulator).

The remainder of the paper is organized as follows. In Section II, we describe some related work. In Section III, we present the architecture of our emulation framework. In Section IV, we detail how the emulation process of NoC systems is performed with our platform. In Section V, we show with several experiments the speed of our emulation framework for functional validation of real-life NoC implementations. Finally, in Section VI, we draw our conclusions.

II. 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 [8] and [2], NoCs with a mesh-based topology and packet-switching as communication mechanism shows 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 [9], [15]. These previous approaches can validate several NoC implementations 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 infrastructures in VHDL, SystemC or combinations of both have appeared in the recent years. In [12], VHDL-based cycle-accurate models are employed to evaluate the latency, throughput and other features in mesh-based and hierarchical NoC topologies. In [5] a modeling environment is described for custom NoC topologies based on SystemC. The main difference with our approach is that their simulations have a much larger execution time compared to our physical NoC emulation environment.

Next, while trying to increase the simulation speed of VHDL environments but preserving its cycle-accurate behavior, several environments based on SystemC or custom language have been proposed. [7] proposes a SystemC-based simulation environment for several NoCs including a real-time operating system. Then, [3] presents a mixed VHDL/SystemC implementation and simulation methodology using a template router to support several interconnection networks. 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 relevant approaches improving the speed of cycle-accurate NoC simulation to get close to the speed of physical emulation have been proposed lately in high-level abstraction languages such as C or C++. In [4] a C-based interconnection network simulator to study power-performance trade-offs at the architectural level is described. Also, [6] presents a NoC design methodology guided by a parameterizable NoC architecture executed in a high-level C++-like event simulator. 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.

Finally, at high-level of abstraction, algorithms and analytical models have been proposed to achieve fast rough estimations of overall cost of NoCs using graphs representions [13], [11]. Such analytical approaches can be used in early stages of NoCs development, but do not enable accurate architectural exploration and functional evaluation as the emulation environment we propose in this paper.

III. NoC Emulation Architecture

As previously mentioned in the introduction, our emulation approach has been designed in a modular way to easily implement various custom NoC topologies and architectures. An overview of

0-7803-8834-8/05/$20.00 ©2005 IEEE.

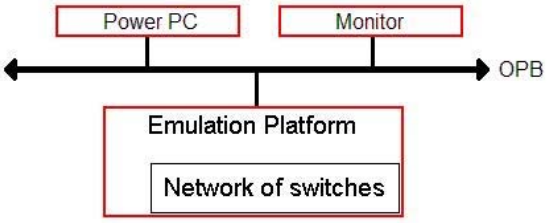

Fig. 1. NoC Emulation Framework

the architecture of our framework is depicted in Figure 1. It consists of three main elements, which are mapped onto an FPGA board with a hard-core processor. In this case, we have used a Xilinx Virtex 2 Pro v20 and a 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 to 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 realistic networks of switches: *Traffic Generators* (TGs), *Traffic Receptors* (TRs) and a user-defined set of interconnections between the switches of the network as depicted on Figure 2. Currently, the synthesizable switches are generated using the Xpipes compiler [5], but our proposed framework is directly applicable to any other type of NoC architecture. The previous modules communicate using a common bus available in our FPGA board called On-chip Peripheral Bus (i.e. OPB in Figure 1).

Our emulation platform (see Figure 2) consists of four types of components: a control module, several types of TGs/TRs and a network of switches. The control module and the TGs/TRs are fully addressable by the processor for configuration and statistics acquisition purposes. Also, the control component can communicate with all the components of the platform by sending broadcast control signals to all of them. Each TG generates different traffic (see Subsection III-A) and injects the packets into the network of switches through its dedicated connection (NoC Interface). Then, after passing through the network of switches, the traffic is received and analyzed by a set of TRs (see Subsection III-B). Finally, to enable an efficient scalability in the amount of TGs/TRs, we have included in the platform a set of independent busses to connect them. Hence, using our architecture it is possible to plug up to 1024 TR/TG, assuming that a larger number of traffic devices would not be fit on actual FPGAs. As a result, this emulation platform enables to instantiate and emulate real-life NoCs on current FPGAs.

In the following subsections we describe in detail the functionality of the main available components in our emulation platform (i.e. TGs/TRs and control module).

A. Traffic Generators

We define a Traffic Generator as a module which is programmable by a processor and controllable by a control module. In order to make it programmable, a bench of registers is addressable by the processor in each TG. Some control signals are used to communicate with its control module. Finally, a network interface is available to inject traffic into the network. As this component must be able to explore/analyze many characteristics of NoCs implementations, the traffic generated by each TG is a function of the content of its registers. This feature gives us the possibility to generate different types of traffic with a single type of TG, and as the configuration of the registers of the TGs is done by the processor, we do not need to resynthesize the platform to perform different emulations of NoC traffic. For our first instance of the emulation framework, we have developed two types of TGs.

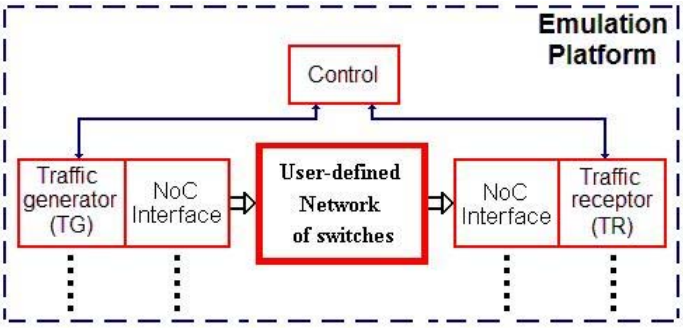

Fig. 2. Emulation Platform Architecture

The first one is able to generate traffic according to several stochastic models (e.g. normal distribution, burst-modes, etc.). The user can give as an input the wanted data rate and the packets characteristics. This type of traffic generator is useful for theoretical studies with stochastic traffic models.

The second type of traffic generator that we have developed is a trace-driven TG. In our case, a trace is an image of real NoC traffic coded with 3 pieces of information, namely a packet length, a destination and a relative time stamp that indicated when to inject the packet into the NoC. In this case the TG receives during the emulation a continuous flow of traces from the processor and the TG generates some traffic following the characteristics of the received trace. This type of TG can be used to emulate real NoC traffic streams/patterns fetched from any real application. As a result, these traces can include input traffic generated by TGs according to the traffic received by TRs if the input traces include such behavior.

B. Traffic Receptors

Similarly to the TGs, we have included two different implementations of TRs in our emulation platform. On the one hand, both possess as common functionality the acknowledgment of the received flits. In addition, both types enable two debug modes. 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 having an efficient implementation according to the required type of reports to generate and a suitable debug tool for the network.

On the other hand, the two types of TRs provide different kind of statistics to the user. The first type generates a histogram about the number of acknowledged flits. The second type generates a trace report for each received packet. The trace has the same format as the one used by the TGs. By this way the processor can compute a detailed analysis (e.g. latency, arrival time) for each delivered packet.

C. 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.

Since several types of TGs/TRs can be used, one specific control module has been developed for each kind to reduce the amount of spent logic (see Section V). Then, to be able to mix them in the same emulation, we have included the possibility to map several control modules simultaneously.

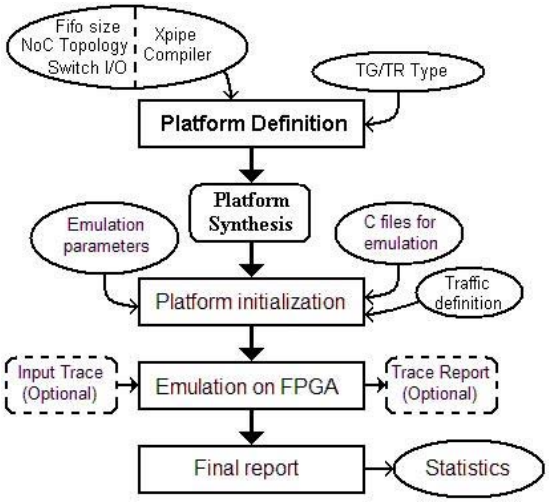

Fig. 3. Our NoC Emulation Flow

IV. Versatile Emulation Framework

The main feature of our emulation framework and its flow is the simple initialization and statistics acquisition of any emulation in the platform at circuit level without re-synthesizing and remapping the whole system. This is possible thanks to its mixed HW-SW structure. The regular FPGA flow would imply a new synthesis for each emulation. Because of the software flexibility, we have a way to perform many emulations without the inconvenience of synthesizing each time. The emulation flow in our FPGA environment varies slightly in case a stochastic emulation is performed or a trace-based one. 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, which are perfectly applicable to any kind of TGs/TRs and statistics to be attained from the NoC emulation.

A. Stochastic Emulation Flow

As Figure 3 indicates, from the hardware point of view we can emulate at the circuit level a wide range of switching configurations of a NoC. The specific topology used in the emulations is defined in the first phase (top box in Figure 3) by configuring several parameters in the Verilog code of our platform. For the sake of simplicity, currently we use hand-coded custom topologies. However, in future work we will develop a tool that automatically generates any type of physical topology (i.e. any number of switches and interconnections between them) based on the Xpipe NoC Compiler [5].

After the generation of the NoC topology, the initialization of the stochastic traffic is performed (Platform initialization box in Figure 3) using the hard-core processor included in the FPGA (i.e. Power PC). This processor runs a compiled C file that contains the information about the traffic the user wants to generate. As we have indicated in Section III, the traffic characteristics can be completely configured by defining some pointers in the C code executed by the processor, which will then write that information in the memory addresses of the TGs; thus, configuring the traffic they inject in the network of switches. This mechanism provides a high flexibility because no time-consuming resynthesis of the HW involved is needed to emulate and study a wide range of NoC implementation parameters. After configuring the TGs/TRs during the initialization phase, the processor is self-configured to wait and fetch the generated statistics at the end of the simulation.

After the configuration of emulation traffic, the system works autonomously and the TGs/TRs acquire the information to generate the statistics demanded by the user from the C configuration file.

Finally, at the end of the emulation, the stored statistics are read by 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 III) and its serial interface to the host PC. Regarding the statistics that can be obtained for NoC research and functional validation, the following type of information is provided: (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. Additional types of information for NoC studies (e.g. link congestion in the network of switches) can be easily added to our emulation framework if desired.

B. Trace-based Emulation Flow

The trace-based emulation flow is similar to the previously explained flow in the way the configuration of the network parameters and the TGs/TRs types is concerned. However, several differences exist in the way the emulation is performed. In this case, instead of setting the stochastic parameters to describe a certain traffic model (e.g. uniform, bursts), a trace is used to represent realistic NoC traffic. In our model, a trace contains a collection of packet descriptors. Typically, the used traces are large since they model traffic generated by fully executing real-life applications. As a result, to enable a real-time emulation with a continuous injection rate of the packets through the emulated NoC, we have designed the packet descriptor to use only 32 bits and the whole trace can be loaded on the RAM placed on the FPGA board. For example, one million packets will consume 4MB of memory for the trace. Also, for this type of emulation, we have designed a special type of TR that is able to extract from a packet the latency of packets through the NoC. Then, the TR can report it.

This flow takes advantage of the software running in the processor of our HW/SW emulation framework to configure some features (e.g. injection rate) that are not defined in the NoC traces in the memory. Thus, performing several emulations does not imply the re-synthesis of any hardware part.

C. Generalization of our Emulation Flow

The strongest advantage of our approach is the possibility to establish a general HW/SW emulation framework to functionally validate and explore real-life NoC implementations. This is achieved by creating a link between a programmable processor and a fully HW emulation environment. To design this general approach we can distinguish two main phases.

The first phase is the design of the interconnection network of switches within the emulation environment. This is done in Verilog in our research, but it could be done in any HDL language. This environment has to be addressable (i.e. configurable by writing into memory addresses) by the processor. Thus, it has an interface compliant with the processor. In our case, we have used the OPB bus, which is the standard bus used to communicate with Xilinx hard-core processors (i.e. PowerPCs) but any other board and/or processor can be used instead since our HW emulation environment is platform- and processor-independent.

The second phase is how to reconfigure the HW environment using the processor, i.e. software reconfiguration instead of HW re-synthesis. To this end, we use a C file that is executed by the processor to read and write at the right addresses of the HW part of the emulation environment (e.g. TGs/TRs). We have assumed that for this step, the C language is appropriate because compiled C language models can control the bus access and can implement any complex algorithm. In fact, several inter-dependent emulations with different inputs for a certain algorithm can be sequentially simulated, which can be very useful to cover many cases in the emulation.

This previous HW/SW emulation approach is easily applicable to any type of NoC such as those proposed by [8], [2]. Moreover, the type of information (e.g. performance or latency statistics) that can be extracted from our framework is really extensive for a wide range of NoC research purposes.

TABLE I

IMPLEMENTATION FIGURES OF OUR EMULATION FRAMEWORK ON A VIRTEX-II PRO FPGA

Device	Number of slices	Board percentage (%)
TG stochastic	719	7.8
TG trace-driven	652	7
TR stochastic	371	4
TR trace-driven	690	7.4
Control module	18	0.2

TABLE II

COMPARISONS BETWEEN EMULATION/SIMULATION NOC ENVIRONMENTS (* EACH PACKET IS 10 CLOCK CYCLES ON AVERAGE)

Simulation Mode	Speed (cycles/sec)	Simulation Time (16×10^6 packets*)	(1×10^9 packets*)
Verilog (ModelSim)	3.2 K	13 hours 53 minutes	36 days 4 hours
System C (MPARM)	20 K	2 hours 13 minutes	5 days 19 hours
Our emulation	50 M	3.2 sec	3' 20 sec

V. EXPERIMENTAL RESULTS

We have tested our emulation approach with several NoC implementations to assess its speed and versatility for functional validation. In all the experiments our emulation was clocked at 50MHz. The choice of the clock speed is very sensitive and important for enabling emulation of real-life NoC traces with several billions of packets for a single application. In addition, the clock heavily influences the synthesis results on the FPGA. If the clock is slowed down, the FPGA synthesizer will be able to increase the critical path, thus reducing the area used by the NoC framework onto the FPGA and making easier the routing. The main caveat of such systems is the limited emulation speed. In our case, our approach is able to work at such frequency by using very optimized code for the TGs/TRs, which enables our FPGA (i.e. Virtex 2 Pro v20) to work at 50MHz or above.

Our first set of experiments has been devoted to evaluate how efficiently our code can be mapped with state-of-the-art synthesis tools (e.g. Xilinx EDK and ISE tools) onto an FPGA. In this case we have considered the instantiation of a NoC proposed in [5] including 6 switches and 4 TGs and 4 TRs. It uses 7387 Xilinx slices (79% of our device). The results for each main HW component of our emulation platform are shown in Table I. It shows that each TG takes about 700 slices (7.5% of the total space on the board) and a TR takes on average about 530 slices (6% of the total space). These results indicate that our code for generating both TGs/TRs can be efficiently mapped onto the available space of the FPGA. Therefore, using this small amount of logic for each pair of TG/TR (i.e. equivalent to one NoC core) makes feasible to instantiate many of them to emulate a NoC with many cores (i.e. more than 40 switches and TGs/TRs) in present larger FPGA devices (e.g. Virtex 2 Pro v40).

In our second set of experiments we have tested the feasibility of using NoC workloads (e.g. amount of packets, packet sizes, congestion rates, etc.) in different emulation/simulation environments to identify the speed up that could be obtained with our emulation approach. The results with different emulation/simulation approaches are shown in Table II. As Table II shows, emulations composed of 16 million packets traces (frequently considered in other simulation environments as realistic inputs) take just 3.2 seconds on our emulation platform clocked at 50 Mhz whereas several hours are needed at least in other approaches. Moreover, real-life workloads of NoCs with more than one billion packets emulations require just few minutes compared to other approaches were days or weeks are required (i.e. more than 4 orders of magnitude of final speedup on our side). Evidently, to validate a NoC structure, though traditional simulators could reduce the complexity of silicon design, only physical implementations using complete real-life workloads can provide the ultimate validation for such complicate systems, as our emulation approach enables.

Finally, to evaluate the reconfiguration effort of our approach in case of NoC features variations, we have varied the topology of the interconnections and number of NoC switches. The effort of modifying a physically implemented design with 4 switches to other configurations with 8 and 16 switches and completely different interconnections (e.g. meshes, torus, etc.) has taken us few hours, including the partial re-synthesis of the HW component of our emulation framework. In addition, note that changes in packet sizes, flit sizes, flits per packet, latencies, etc. take a matter of minutes instead of hours (or even days) as other custom-designed NoC functional verification approaches do. In fact, in our HW/SW emulation approach they only affect a C file that is executed by the processor. This implies a modification and recompilation of a C file, which is several orders of magnitude faster (i.e. few seconds) than a HW re-synthesis of many hours.

VI. 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 high emulation speed and low 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 configure the FPGA and its fast emulation speed.

ACKNOWLEDGMENT

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.

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] W. Hang-Sheng, et al. Orion: a power-performance simulator for interconnect. networks. In *Proc. MICRO*, 2002.
[5] A. Jalabert, et al. xpipesCompiler: A tool for instantiating application specific Networks on Chip. In *Proc. DATE*, 2004.
[6] S. Kolson, A. Jantsch, et al. A NoC architecture and design methodology. In *Proc. Annual Symp. VLSI*, 2002.
[7] J. Madsen, S. Mahadevan, et al. NoC modeling for system-level multiprocessor simulation. In *Proc. RTSS*, 2003.
[8] T. Marescaux, J.I. Mignolet, et al. NoC as hw components of an os for reconfigurable systems. In *Proc. FPL*, 2003.
[9] F. Moraes, et al. Hermes: an infrastructure for low area overhead packet-switch. NoC. *Integration-VLSI Journal*, 2004.
[10] S. Pestana, E. Rijpkema, et al. Cost-performance trade-offs in NoC: a simulation-based approach. In *Proc. DATE*, 2004.
[11] A. Pinto, et al. Efficient synth. NoC. In *Proc. ICCD*, 2003.
[12] D. Siguenza-Tortosa et al. Vhdl-based simulation environment for proteo noc. In *Proc. HLDVT Workshop*, 2002.
[13] L. Tang and S. Kumar. Algorithms and tools for NoC based system design. In *Proc. SBCCI*, 2003.
[14] D. Wiklund, S. Sathe, et al. NoC simulations for benchmarking. In *Proc. IWSoC for Real-Time Apps.*, 2004.
[15] C. Zeferino, M. Kreutz, et al. Rasoc: a router soft-core for NoC. In *Proc. DATE*, 2004.