# A Fast Emulation-based NoC Prototyping Framework

Yana E. Krasteva, Francisco Criado, Eduardo de la Torre and Teresa Riesgo
Centro de Electronica Industrial
Universidad Politecnica de Madrid
Email: yana.ekrasteva,eduardo.delatorre,teresa.riesgo{@upm.es}

***Abstract*—This paper presents an FPGA emulation-based fast Network on Chip (NoC) prototyping framework, called Dynamic Reconfigurable NoC (DRNoC) Emulation Platform. The main, distinguishing, characteristic of this approach is that design exploration does not requires re-synthesis, accelerating the process. For this aim, partial reconfiguration capabilities of some state of the art FPGAs have been developed and applied. The paper describes all the building elements of the proposed solution: the used partial reconfiguration approach, the design space exploration framework itself, and the data measuring system. Results and a use case are shown.**

## I. Introduction

Future Systems on Chip will contain hundreds of heterogeneous cores, running at different speeds and voltage levels. Finding an optimal solution for interconnecting such complex systems is a great challenge and the required performance can not be covered by traditional bus-based approaches. Therefore Networks on Chip (NoCs) have been proposed as a scalable solution for on chip communication [1][2].

NoCs have rapidly evolved during the last years and a lot of research effort has been oriented to NoC based SoCs design and prototyping.

One of the main challenges in NoC design is to find the optimal NoC solution for a given application. Several methods and design flows that permit to perform design space exploration, at different abstraction levels, have been proposed. Higher abstraction levels permit to rapidly evaluate different mapping and NoC implementation options without paying the cost of long simulations. For instance, in [3], a framework for MPSoC NoC system modeling, simulation and evaluation, based on System C models is presented. Systems are generated matching application and platform models. Other frameworks are based on object-oriented languages, like the presented in [4], where a C++ library is built on top of SystemC, or based on the Matlab simulation environment, like [5]. Other approaches generate NoC topologies getting the application graph representations or application descriptions as starting point, using analytical and/or heuristic methods. Examples of these are SUNMAP [6], based on the Xpipes [7] NoC generator, which creates topologies modeled in System C, or in [8], where systems are specified in XML. These approaches are very suitable for system design early stages as they permit to have the fastest design space exploration. There are other HDL or mixed (System C and HDL) solutions for NoC modeling, like MAIA [9], where NoC parameters are defined by the user, and NoCGeN [10], where the NoC topology can be selected. Lower level, VHDL or RTL, permit to have accurate results, but are more time consuming. Usually, traffic generators and traffic consumer models that simulate real Core behavior are used in order to reduce simulation time.

On the other side, there are FPGA based emulation solutions proposed to drastically reduce the simulation and, therefore, system evaluation time. For instance, in [11], a HW-SW FPGA based emulation framework is presented and combined with the Xpipes environment. Four orders of magnitude of speedup are reported in that work. Emulation, depending on the FPGA available area, may permit to test NoCs using real application cores instead of traffic models. In [12], four real applications mapped into a NoC and prototyped in an FPGA are presented. Nevertheless, the main disadvantages of emulation based solutions are:

1) Synthesis time: every time a system parameter needs to be changed, the system has to be re-synthesized, replaced and re-routed (from here after this process will be refered simply as synthesis). This is not a real problem if a system has to be synthetized once, but if the goal is to come up with the optimal system implementation, a lot of combinations have to be synthetized and emulated. An approach to overcome the synthesis problem is to have all system options implemented in the FPGA and switch between them, but this consumes a considerable amount of FPGA area.
2) The available FPGA area permits only to emulate relatively small systems. In [13] a solution for this problem is proposed. There, sequentially, parts of a parallel system are loaded into an FPGA. Speedups of 80 to 300 in comparison with System C simulation are reported. Anyhow, each new FPGA generation has more logic available and thus permits to emulate bigger systems.
3) Data extraction, measured from the FPGA: the FPGA has much more limited resources in this sense in comparison with a SW based simulation.

Some state of the art FPGAs permit to achieve higher flexibility by including partial reconfiguration capabilities, which allows modifying part of a system mapped in the FPGA while the rest, non reconfigured part, is kept active. Partial reconfiguration for FPGA NoC based systems is used in [14] and in [15]. These papers present two different solutions for

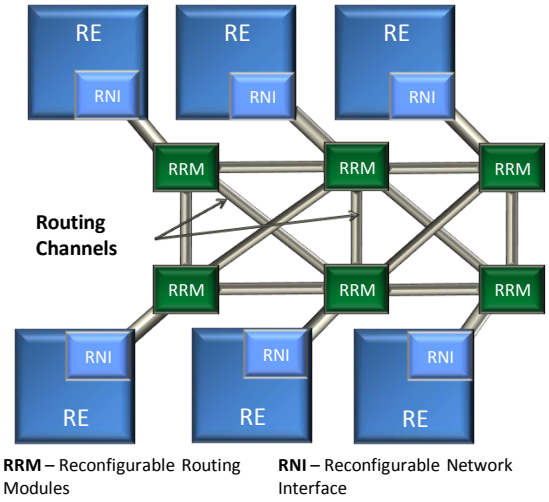


Fig. 1. DRNoC Architecture

dynamical insertion and removal of routers and cores on a MESH based NoC. Also, partial reconfiguration at communication level has been evaluated to be used for changing routers' FIFO size in [16].

This paper goes further and presents a complete fast NoC emulation framework based on different types of partial reconfiguration. The emulation framework permits to reduce design time as it permits to build systems using reusable hardware cores (placed and routed cores, partial configuration files) and therefore re-synthesis is not required and problems related to the extraction of data from the FPGA are attenuated. The rest of the paper is structured as follows: in section II, the partial reconfiguration approach is described. The work methodology is explained with a use case in section III, while the entire emulation framework is shown in section IV. Results and a use case are included in section V and finally conclusions can be found in section VI.

## II. Partial Reconfiguration

In order to create partially reconfigurable systems, *Virtual Architectures* (VAs) have to be defined. VAs are structural divisions of FPGA logic resources along with the internal communication of the different regions that permit to design and run partially reconfigurable systems. VAs structural divisions can be based on two different models: 1D models, where a single reconfigurable module is allocated in an area that spans the entire FPGA heigh, and 2D models, where two or more reconfigurable blocks can be allocated in the same FPGA column.

According to the Xilinx solution for partial reconfiguration [17] and [18], for Virtex II based FPGAs, these areas have to span the entire FPGA height or to be block based but surrounded by fixed logic. Therefore, only 1D models can be defined. Some examples of NoC based partially reconfigurable systems that follow the Xilinx approach are [19] and [20]. Differently 2D VAs are more naturally mapped in latest Virtex 4 and Virtex 5 FPGAs, where block partial reconfiguration is enabled (a block is composed of 256 CLBs).

The selected platform for the DRNoC emulation system is a Virtex II based proprietary board specially created for partially reconfigurable systems design. This board has been selected due to its flexibility and the availability of proprietary support software. On top of the FPGA, a 2D virtual architecture has been defined and mapped. This is possible due to the architecture design method, presented in [21] and the available bitstream manipulation tools presented in [22].

The proposed architecture for the reconfigurable system (the key element of the emulation platform), called Dynamic Reconfigurable NoC, can be seen in Figure 1. It is a MESH of Reconfigurable Elements (RE) that are connected through a *Reconfiguration Network Interface (RNI)* to a *Reconfigurable Routing Module (RRM)*. Each RRM is connected to all its neighboring RRMs with short, hard wired and position fixed *communication channels*, that are composed of an integer number of wires. Cores are allocated in REs, RNI allocate network interfaces (NIs), and routers are mapped to RRMs. Each router allocated in an RRM can use any amount of communication channels and also any amount of channel wires. Even more, if there is enough room, two independent routers can share the same RRM. This, along with the diagonal mesh-like channel interconnection network, permits to map different NoC topologies (star, mesh or a custom one, etc). IP cores, or traffic generator models, as well as routers occupy different areas, therefore REs can be grouped if it is necessary. In this case, the architecture hard wire connection links are kept.

To map the DRNoC model to the selected FPGA first, the regularity of the internal FPGA logic distribution needs to be taken into account for an optimized FPGA resource partition. Those FPGA areas that perturb the structure regularity are reserved for fixed (non reconfigurable) area of the FPGA. The remaining FPGA regular area is divided into slots that are used to map REs, RNIs and RRMs.

The target board has an XC2V3000 that has 56 CLB columns and 64 rows and the amount of slots that can be defined for resulting a reasonable slot size is 2x4, see the upper part of Figure 2. A direct mapping of the model is to assign one slot to each DRNoC component, thus three slots will be needed, i.e. one for RRM , one for RNI and one for hard cores allocation, resulting in a one column implementation. Therefore RNIs and hard cores have been grouped in one slot marked with S in the bottom part of Figure 2 and RRM use the next slot, marked with R in the bottom part of Figure 2. The resulting DRNoC is 2x2, where each slot/RRM size is 24x12 CLB and the implemented channel size is 40 bits in each direction (80 wires in total).

Finally, it is important to remark the scalability of the solution, by the fact that a 4x4 DRNoC with the same channels size has been successfully mapped to a XC2V8000 FPGA.

Regarding the supported reconfigurability, both intra-core and inter-core partial reconfiguration schemes have been applied. On one side, intra-core reconfiguration permits changing only a certain parameter of a core, NI or router. For

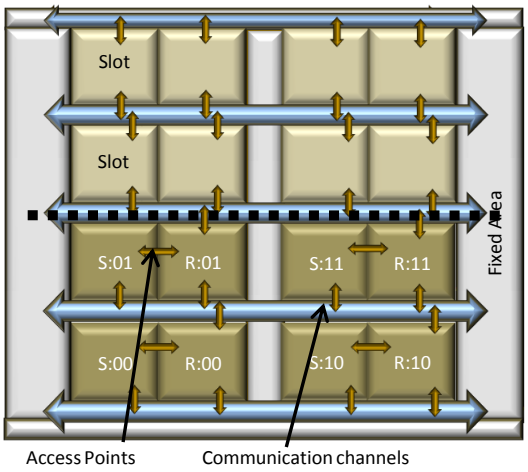


Fig. 2. DRNoC Structure mapped on an XC2V3000

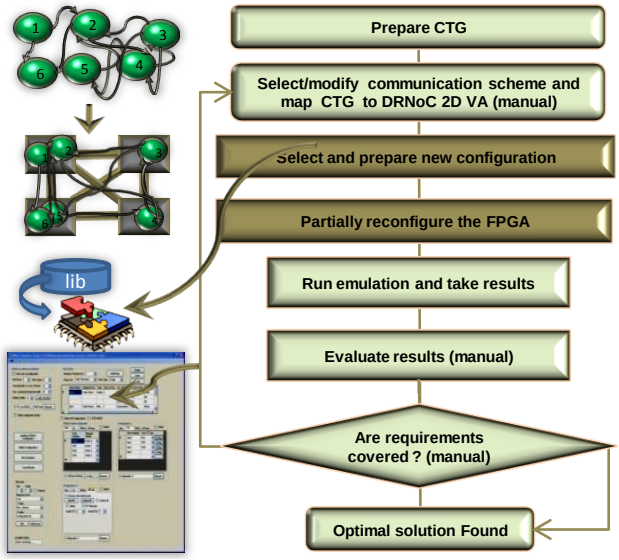


Fig. 3. Emulation workflow diagram

instance, the system supports changes in core target/source node addresses, to change routers routing strategy, or modify router buffers size. On the other side, inter-core reconfiguration is used to define the communication strategy, which permits setting NoC routers type, RE feed-through and/or NoC phit size.
These reconfiguration schemes are the core of the emulation workflow presented in the next section.

## III. Emulation Workflow

A general view of the proposed methodology for NoCs design space exploration is presented in Figure 3. General aspects this flow are similar to other flows, like [23], but here partial reconfiguration is exploited and the NoC to be emulated is intended to be built entirely by reusable hard cores (already placed and routed partial configuration files) available in a hard core library. The flow begins with a mapped application communication task graph (CTG), where application tasks are assigned to system cores. For each node of the CTG, a suitable DRNoC hard core (traffic receiver or traffic generator) is assigned if they are available in the hard core library or generated if they are not available. This step is previous to the emulation process. After that, in the first step of the emulation flow, the CTG is mapped to the available emulation systems FPGA DRNoC architecture (CTG nodes are assigned to slots) and NoC parameters are defined (NIs and routers are mapped to slots/RNIs and RRMs). From here after, each CTG to DRNoC mapping with assigned NoC parameters will be referred as a *configuration*. Also, in this step, measuring points are defined in selecting the tracked nodes. Once all configurations have been setup, the emulation starts. For each DRNoC configuration, hard cores from the library are arranged in an FPGA bitstream and downloaded in the FPGA. All configurations are emulated consecutively, and the results related to each measuring point are saved. After this, results can be plotted, analyzed by a user and, if needed, new DRNoC configurations can be added with modified CTG to DRNoC mappings and/or new communication parameters assignment. This process continues until the best communication scheme and CTG mapping has been found.
Base on the presented working flow, an entire emulation platform have been created and is presented in the next section.

## IV. The DRNoC Emulation Platform

The Emulation platform is composed of three parts: i) the DRNoC design resources along with an associated SW tool for model generation, ii) the measuring system and iii) a SW tool for controlling the emulation process. Each element is described in the following subsections.

### A. DRNoC - Dynamically Reconfigurable NoC

The distinguishing characteristic of DRNoC is that it is prepared for being mapped to partially reconfigurable systems, like the one presented in section II. A set of models in VHDL at behavioral or RTL level created for building and testing different DRNoCs has been created. Available resources are:

1) Routers. Two types of routers have been designed. One based on the HERMES network routers [24], but implementing FIFOs in FPGA BRAMs and with different packet organization. These routers use XY routing and permit to build mesh NoCs. The second type of routers are based on table routing that permits to create irregular topologies of up to 9 connections.
2) Network interfaces. IP cores communicate with network interfaces using s simple AMBA APB protocol. AMBA has been selected because it requires less hard wires in comparison with other protocols, like OCP. Network interfaces are in charge of data packetizing. Two types of network packets can be generated depending on the NoC router: XY or table based.
3) Traffic generators(TGs). Two type of traffic generator have been implemented so far: i) simple traffic gen-

erators that generate regular traffic and ii) burst traffic generators based on Pareto ON/OFF scheme [25]. TGs can generate traffic to a single traffic receiver or to several traffic receivers.

4) Traffic receivers(TR). These modules implement the DRNoC platform measuring system. Two types of traffic receivers can be distinguished, depending on the measuring scheme: either to perform online measuring, or to keep only max/min values into the internal measuring registers. Each TR has one or several groups of measuring registers. Each register of a measuring group is related to a parameter to be measured: latency, the time a packet has been in the NoC and amount of transmitted data. Each group of measuring registers track data related to one TG, although there may be more than one register set. This permits to perform more detailed analysis of the system. Taking advantage of intra-core reconfiguration, it is possible to use TRs that track data from a single TG and then change the associated node without the need of having special logic for accessing these registers.

A new SW tool, called DRNoCGEN, has been designed. It uses a set of user defined parameters for automatically generating DRNoC models . The main difference of this tool compared to other solutions is that the generated models, apart from being synthesizable, are directly mapped on reconfigurable elements, like the described 2D architecture. The virtual architecture definition is used as a template where a user selects and maps a DRNoC. For instance a user can map a 2x2 DRNoC mesh based on XY routing or a star DRNoC based on table routing on a defined 8x8 2D VA. DRNoCGEN also permits to define new architecture templates. The output of DRNoCGEN is a set of VHDL files that include the selected NoC routers, NIs TGs and TRs, as well as a top design that includes the instantiation of all the needed communication macros (VA related user constraints file are not automatically generated till now).

### B. DRNoC Measuring System

The measuring system is distributed in two platforms: measuring points, included in the DRNoC FPGA and the measuring system buffers, based on an XUP board with an XC2VP30 FPGA.

Following [26], the system measuring points (a group of measuring registers) are allocated in TRs and in TRs NIs. Data is extracted from measuring points using the AMBA APB interfaces. In the current approach measuring in routers is not foreseen in order to save area.

The pulled data is buffered in a FIFO allocated in the FPGA area of the XC2VP30 and connected as a custom peripheral to the on-chip Power PC (PPC). The PPC is used to transmit data from the buffer FIFO and send it to a Host PC, through a serial port, and also, to control the DRNOC emulation process (run, stop, reset, reconfigure). Control commands are sent from the Host PC when a SW control is defined, or are automatically controlled by the HW when a HW control is defined. In the second case, when the system emulation is finished, an interrupt to the PPC is generated and this activates the Host PC SW, described next.

The XUP based platform is needed to isolate as much as possible the measuring system from the proper NoC and to leave the DRNoC FPGA as regular as possible.

### C. DRNoC Emulation SW

The SW tool running on the host PC is in charge of controlling the entire emulation process and the design space exploration. The SW includes a GUI and its main features are listed next:

1) To define DRNoC configurations and measuring points.
2) To control system configurations, reconfigure the FPGA and communicate with the DRNoC measuring system. Partial reconfiguration is used to pass from one configuration to other whenever it is possible. For this purpose, the hard core reallocation tool mentioned in section II has been integrated in the DRNoC emulation SW.
3) To collect, organize and plot measured data for each measuring point.

The tool works only with hard cores (FPGA partial configuration files) that are held in a configurations library. A hard core can be a NoC router, a NI, a TG or a TR. Additionally, smaller partial configuration files are automatically generated from hard cores for intra-core reconfiguration. The configuration library can be expanded with other hard cores, it is not limited to just one core per element type.

There is not an automated connection between the presented tool for DRNoCGEN and this one. If a new hard core is going to be added to the library, this process has to be done manually. For including a new router, for instance, first a DRNoC with the proper options has to be generated with the DRNoCGEN tool. The generated DRNoC includes more than a router, apart from the TGs, TRs and NIs. For having fast synthesis and since only one router is needed, a router is selected and isolated. The code generated by DRNoCGEN is modular and well structured, therefore this step is quite easy, only the top and the user constraints file have to be modified (all but the router and the communication macros have to be commented). Second, a partial configuration file containing only the router hard core has to be generated. This can be done using the conventional Xilinx design Flow and the tool BITPOS [22], or using the Plan Ahead tool provided by Xilinx included in its partial reconfigurable flow [18]. In both cases, synthesis times are drastically reduced in comparison to synthesizing the entire system.

The system permits to track the tested DRNoC configuration and the obtained results. The user can also select which configurations to be included in the emulation process.

## V. Results and Use Case

Area requirements of some TR implementations are presented in Table I, including TRs of both available types: one

TABLE I
TR AREA REQUIREMENTS

| *TR type* | Slice | FF | LUTs |
|---|---|---|---|
| *TR-Single-online* | 286 | 290 | 395 |
| *TR-Single* | 285 | 218 | 528 |
| *TR-2* | 489 | 350 | 781 |
| *TR-4* | 810 | 613 | 1293 |
| *TR-8* | 1571 | 1120 | 2522 |

TABLE II
ROUTER AREA REQUIREMENTS

| *Router type* | Num ports | Slice | FF | LUTs | BRAMs |
|---|---|---|---|---|---|
| *HERMES* | 3 | 876 | 882 | 929 | 0 |
| *HERMES* | 5 | 1452 | 1460 | 1547 | 0 |
| *DRNoC XY* | 3 | 261 | 183 | 505 | 3 |
| *DRNoC XY* | 5 | 385 | 296 | 808 | 5 |

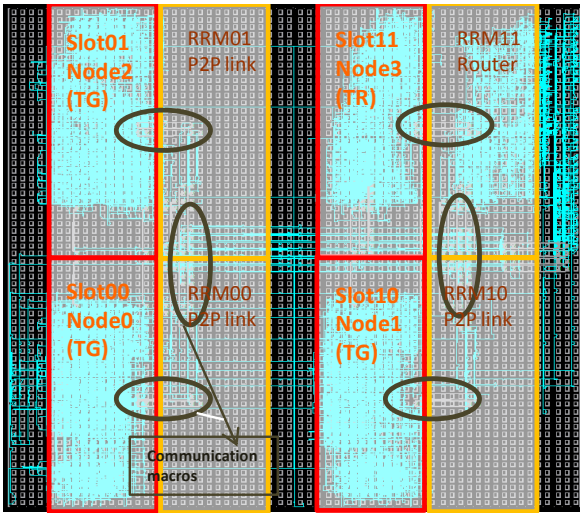


Fig. 4. STAR DRNoC

that follows the online measuring scheme, marked as TR-Single-online, and the remaining TRs, that follow a max/min value scheme, with different amount of measuring blocks (one measuring block for every associated TG). For instance, TR-4 tracks data from 4 TGs.

It can be noticed that the area needed for a TR increases linearly with the amount of TGs to be tracked. Therefore it is more suitable to have only simple TRs implemented and run the emulation as much times as needed, changing the tracked TG using intra-core reconfiguration.

Regarding routers area overhead, a comparison of the original HERMES routers with buffer depth of 32 and flit size of 8 bits, and the DRNoC XY routers with the buffers implemented in FPGA BRAMs, also with flit size 8 are presented in table II. Regarding router latency, additional logic has been included in the buffers control for maintaining the original router latency. For measuring the performance of the entire emulation system, an example DRNoC implementation on top of reconfigurable system has been defined. In the currently available FPGA an XC2V3000, a 2x2 DRNoC architecture has been defined as is the one used for the use cases presented in this section.

The use case supposes that there is an application where three sources try to access a common media (the application CTG has four nodes). Following the emulation flow, each node has been modelled with DRNoC design resources. Uniform traffic generator has been used for node0, node1 and node2, while the common media has been modelled as a traffic receiver node node3) that includes a measuring point. Two configurations have been defined for this application, both with the same mapping to the DRNoC architecture (node0 to slot00, node1 to slot01, node2 to slot10 and node3 to slot11), but with different NoC parameters. The first one is a 2x2 mesh composed of 4 DRNoC XY routers and the second is a star NoC, composed of 3 point to point (P2P) links and one router. As an example, the star topology mapping to DRNoC is presented in Figure 4. In the same figure, the used feed-throughs for the P2P connection allocated in RRM00, RRM01 and RRM10 can be seen.

Each configuration (mesh and star) transmits 100 packets of 320 bits each. For the mesh topology, emulation time is 1 ms for each desired measuring point tracked TG, while for the STAR it is 0,8 ms. This time is measured from the start command to run the emulation process until the end of the emulation in the FPGA. For simulating the same system with VHDL simulation, 10 minutes are needed for the mesh version and 2 minutes for the star. If results from all the possible TGs to be tracked are required, then three intra-core reconfigurations have to be done and emulation has to be run 3 times. In this case, the acceleration achieved is in the range on tens of thousands.

The main goal in this work was to try to solve one of the disadvantages of NoC emulation related to the system synthesis time. For instance, for the mesh implementation synthesis, that uses around 20 % of the XC2V3000 FPGA, it takes 16 minutes, while for the star it is 8 minutes. Differently, for building the star system from the mesh or vice versa, following the approach presented in this paper, 2 inter-core reconfigurations are needed, but no synthesis is required.

The required time for each partial reconfiguration is in the range of microseconds to milliseconds when using the FPGA internal confirmation port (ICAP) and in the range of seconds when using the JTAG interface. The achieved speedup in comparison with the synthesis approach (non reconfigurable) is in the range of hundreds of times for the worst case. Additionally, if for instance, the needed router is not available in the hard core emulation library, only one router is needed to be synthesized and this will take just 1 min, 8 times less than synthesizing the entire star system.

Results of the online measuring of the traffic received from node0 (TG) are presented for the mesh and for the star in Figure 5 as example. The main advantage of the online measuring is the possibility of tracking the network dynamics as it is shown in figure 5, where latency for each received packet is plotted.

The main drawback of the presented system resides in the inherited restrictions of current partial reconfiguration tech-

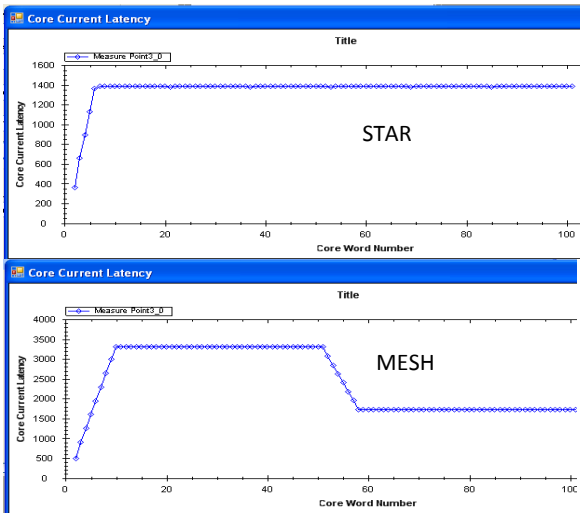


Fig. 5. MESH and START NoC latency, measured in clock cycles, for each received word. Plots correspond to an online measuring point, allocated in node3 and tracking data received from node0.

niques. Although the used method for VAs definition tries to reduce the area overhead due to partial reconfiguration, it is still high. Anyway a tendency of improving the partial reconfiguration capabilities in the newest FPGAs can be noticed. The presented systems can be retargeted to other FPGAs with the exception of the hard core libraries and the related hard core manipulation SW that support Virtex II and Virtex II Pro FPGAs.

## VI. Conclusion

A method for overcoming emulation systems drawback derived from long synthesis times has been presented. The core of the method is the exploitation of state of the art partial reconfiguration capabilities of some FPGAs. A work flow, based on partial reconfiguration where the system to be emulated is built using hard cores (partial configuration files) has been proposed. As a demonstration of the approach, a use case based on a NoC reconfigurable system, mapped on a Virtex II FPGA, has been defined and presented. Speedups of hundreds of time have been achieved in the presented use case compared with a non reconfigurable approach (synthesis based).

## Acknowledgment

The authors wish to express their gratitude to the Departamento de Fundamentos da Computacao, Pontificia Universidade Catolica do Rio Grande do Sul, specially to Ney Calazans, for providing the Atlas environment and the HERMES network.

## References

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

[2] F. Karim, A. Nguyen, S. Dey, and R. Rao, "On-chip communication architecture for oc-768 network processors," in *DAC*, 2001, pp. 678–683.

[3] S. Mahadevan, K. Virk, and J. Madsen, "Arts: A systemc-based framework for modelling multiprocessor systems-on-chip," *Design Automation of Embedded Systems*, 2006.

[4] M. Coppola, S. Curaba, M. D. Grammatikakis, R. Locatelli, G. Maruccia, and F. Papariello, "Occn: a noc modeling framework for design exploration," *Journal of Systems Architecture*, vol. 50, no. 2-3, pp. 129–163, 2004.

[5] S. S. K. A. Mehran and Armin, "Smap: An intelligent mapping tool for network on chip," *Signals, Circuits and Systems ISSCS 2007*, pp. 1–4, 13-14 July 2007.

[6] S. Murali and G. D. Micheli, "Sunmap: a tool for automatic topology selection and generation for nocs," in *DAC '04: Proceedings of the 41st annual conference on Design automation*. New York, NY, USA: ACM, 2004, pp. 914–919.

[7] A. Jalabert, S. Murali, L. Benini, and G. D. Micheli, "xpipesCompiler: A tool for instantiating application specific Networks on Chip," in *Design, Automation and Test in Europe (DATE)*, Paris, France, Feb. 2004.

[8] J. Joven, O. Font-Bach, D. Castells-Rufas, R. Martinez, L. Teres, and J. Carrabina, "xenoc - an experimental network-on-chip environment for parallel distributed computing on noc-based mpsoc architectures." in *PDP*. IEEE Computer Society, 2008, pp. 141–148.

[9] L. Ost, A. Mello, J. Palma, F. G. Moraes, and N. Calazans, "Maia: a framework for networks on chip generation and verification," in *ASP-DAC*, 2005, pp. 49–52.

[10] J. Chan and S. Parameswaran, "Nocgen: A template based reuse methodology for networks on chip architecture," *vlsid*, vol. 00, p. 717, 2004.

[11] N. Genko, D. Atienza, G. D. Micheli, J. M. Mendias, R. Hermida, and F. Catthoor, "A complete network-on-chip emulation framework," in *DATE*. IEEE Computer Society, 2005, pp. 246–251.

[12] Ü. Y. Ogras, R. Marculescu, H. G. Lee, P. Choudhary, D. Marculescu, M. Kaufman, and P. Nelson, "Challenges and promising results in noc prototyping using fpgas," *IEEE Micro*, vol. 27, no. 5, pp. 86–95, 2007.

[13] P. T. Wolkotte, P. K. F. Holzenspies, and G. J. M. Smit, "Fast, accurate and detailed NoC simulations," May 2007.

[14] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. P. Fekete, and J. van der Veen, "Dynoc: A dynamic infrastructure for communication in dynamically reconfigurable devices," in *FPL*, 2005, pp. 153–158.

[15] T. Pionteck, R. Koch, and C. Albrecht, "Applying partial reconfiguration to networks-on-chips." in *FPL*. IEEE, 2006, pp. 1–6.

[16] R.Benmouhoub and O.Hammami, "Noc monitoring harwdare support for fast noc design space exploration and potential noc partial dynamic reconfiguration," in *IEEE IES*, 18-20 oct 2006.

[17] M. P. Davin Lim, *XAPP 29 (v1.0)0: Two Flows for Partial Reconfigurtion: Module Based or Small Bit Manipulation*. Xilinx, 2004.

[18] *Partial Reconfiguration Software User's Guide*. Xilinx, 2006.

[19] T. Marescaux and et al., "Run-time support for heterogeneous multitasking on reconfigurable socs," *Integr. VLSI J.*, vol. 38, no. 1, pp. 107–130, 2004.

[20] L. Moller, I. Grehs, E. Carvalho, R. Soares, N. Calazans, and F. Moraes, "A noc-based infrastructure to enable dynamic self reconfigurable systems," in *ReCoSoC*, 2007, pp. 23–30.

[21] Y. E. Krasteva, E. de la Torre, and T. Riesgo, "Virtual architectures for partial runtime reconfigurable systems. application to network on chip based soc emulation," in *Annual Conference of the IEEE Industrial Electronics Society*, 2008, p. accepted for publication.

[22] Y. E. K. et al., "Virtex ii fpga bitstream manipulation: Application to reconfiguration control systems," in *FPL*, 2006, pp. 1–4.

[23] N. Genko, D. Atienza, G. De Micheli, and L. Benini, "Feature - noc emulation: a tool and design flow for mpsoc," in *Volume 7, Issue 4, Circuits and Systems Magazine, IEEE*, Fourth Quarter 2007, pp. 42 – 51.

[24] F. G. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "Hermes: an infrastructure for low area overhead packet-switching networks on chip," *Integration*, vol. 38, no. 1, pp. 69–93, 2004.

[25] L. Tedesco, A. Mello, D. Garibotti, N. Calazans, and F. Moraes, "Traffic generation and performance evaluation for mesh-based nocs," in *SBCCI*, 2005, pp. 184–189.

[26] C. G. et al., "Towards open network-on-chip benchmarks," in *NOCS*, 2007, p. 205.