HardNoC: A Platform to Validate Networks on Chip through FPGA Prototyping

Guilherme Heck, Ricardo Guazzelli, Fernando Moraes, Ney Calazans
Faculdade de Informática - PUCRS
Porto Alegre, Brazil
{guilherme.heck, ricardo.guazzelli}@acad.pucrs.br
{fernando.moraes, ney.calazans}@pucrs.br

Rafael Soares
CDTec - UFPel
Pelotas, Brazil
rafael.soares@inf.ufpel.edu.br

Abstract – **The use of intrachip buses is no longer a consensus to build interconnection architectures for complex integrated circuits. Networks on chip (NoCs) are a choice in several real designs. However, the distributed nature of NoCs, the huge amount of wires and interfaces of large NoCs can make system/interconnection architecture debugging a nightmare. This work accelerates the NoC validation process using FPGA prototyping. HardNoC is a platform based on simple modules to inject traffic and collect basic statistics of NoCs. It can be used to early validate NoC designs and to provide initial numerical results for NoC evaluation and design.**

Keywords – NoC, Prototyping, FPGA, GALS, Emulation.

I. INTRODUCTION

Networks on chip or NoCs are an emerging paradigm for communication architectures within large VLSI systems implemented on a single silicon chip, which are named Systems on Chip or SoCs. The proposition of NoCs for modern and future systems on chip capitalizes on the following features: (1) energy efficiency and reliability [1]; (2) scalability of bandwidth when compared to traditional bus architectures; (3) reusability; (4) distributed routing decisions [1] [2]. In a NoC-based SoC, modules such as processor cores, memories and other specialized IP blocks exchange packetized data using the NoC as a subsystem for data transport. Network interfaces, routers and point-to-point links define a NoC. Debugging NoCs is time-consuming and complex, especially when the NoC is large.

This paper presents HardNoC, a hardware platform to facilitate prototyping and evaluating NoCs in hardware. HardNoC allows traffic programming without hardware changes. The user feeds HardNoC with data, informing a set of (synthetic) traffic parameters. Next, the user initiates the HardNoC platform execution. After execution, the user may verify if all packets were correctly transmitted and analyze latency values (minimal, maximal and average).

The effects of specific traffic patterns in system performance can be greater than the effect of changing NoC structural parameters [3]. To account for this, HardNoC allows exercising NoCs with more than one type of traffic distribution. The available synthetic traffic distributions are *Constant Bit Rate* (CBR) and *Pareto on-off*, which allows evaluating NoCs under distinct traffic scenarios.

This paper is organized as follows. Section II describes related work. Section III describes some NoC assumptions built in the HardNoC platform, while Section IV describes the general HardNoC architecture. Section V discusses the platform traffic configuration and results evaluation. Section VI presents some experimental results of mapping two quite distinct NoCs to HardNoC. One is a well known synchronous NoC and another is a globally asynchronous locally synchronous (GALS) NoC designed for low power consumption. Finally, Section VII presents a set of conclusions and describes relevant ongoing work.

II. RELATED WORK

Several approaches have been developed to improve the performance of NoC evaluation. Some of these are based on hardware emulation, particularly through the use of FPGAs, to prevent the high costs in simulation time incurred by the use of HDL software simulators.

Wen et al. for example [4], developed a platform that generates online traffic for testing the performance of different NoC architectures. They suggest the use of IP modules called online configurable traffic generators (OCTGs) that emulate the local IP connected to each router in a 2D mesh NoC. According to the authors, the platform allows the user to directly set and reconfigure traffic generators through a JTAG port. This feature avoids stopping the prototyping session to restart a new configuration. Together with the NoC and OCTGs connected to it, the platform uses a dedicated configuration engine based on an Altera Nios II soft core processor. This interface acts as an intermediary between the JTAG and the OCTG modules. The implementation uses Altera Stratix II FPGAs. The paper describes an experiment prototype with a 3x3 NoC. Results compare injection rates to both packet latency and throughput, for different traffic scenarios produced by OCTGs. Also, the system supports several traffic modes and allows selecting the routing algorithm, both at runtime. But using embedded processors may lead to significant area overhead, reducing the maximum NoC dimension that is viable to emulate.

Lotlikar et al. [5] propose AcENoCs, a mixed software-hardware environment for NoC emulation. In the software part, the platform may use one or two Xilinx Microblaze soft core processors. The number of processors depends on the application for which to generate traffic. The hardware consists in a register bank that acts as interface between the software environment and the NOC under verification. AcENoCs allows selecting either the use of uniform traffic generation by the soft core or specific application traces stored in memory. According to the authors, it is possible to run simulations of NoCs with up to dimension 5x5, when using a Xilinx XC5VLX110 Virtex 5 FPGA. AcENoCs authors report speedups in the range of 10,000 to 12,000 times when comparing emulation and HDL simulation. Nonetheless the proposed architecture is subject to reduced

978-1-4673-0186-2/12/$31.00 ©2012 IEEE

emulation speed due to the centralized software-based traffic generation scheme. Because of this the authors suggest the use of a second processor to satisfy the throughput requirements for demanding traffic scenarios. A distinctive feature of AcENoCs is the support to GALS as well as synchronous NoCs. Clock generation relies on software which is flexible, but potentially slow. Also, the authors mention that clock generation is limited to mesochronous schemes, where all clock frequencies are the same, but phases vary. Unfortunately, this is not quite generic for GALS systems, since it does not support the use of arbitrary frequencies in the NoC.

Tan et al. [6] present a design flow to generate FPGA emulation platforms for NoCs. The flow generates a NoC on FPGA for traffic evaluation from parameterized synchronous NoC structures, given that the NoC HDL description is available and can be implemented on FPGAs. It is possible to choose the number of routers, buffer sizes, routing algorithms, and select which nodes will receive traffic generators and traffic receptors. Experiments use the Xilinx ML506 evaluation board, with a XC5VSX50 Virtex 5 Xilinx FPGA. Results are comparisons of area overhead and package latency using different routing algorithms for the Hermes NoC [7]. The goal of comparisons seems to be to decide the best routing algorithm for a specific application. Unfortunately, no detail is available on how traffic generators can be configured or on how to collect data from traffic receptors.

Hou et al. [8] show a performance evaluation system for virtual channel NoCs in real applications. The authors propose a module attached to routers that monitor data traveling through the router. It allows the user to evaluate performance parameters such as average latency and throughput. The data presented in these monitors can be accessed through a JTAG connection. Data are generated by six ARM cores (one at each node) which inject real traffic provided by a real application running on each core. To prototype a 2x3 NoC with two virtual channels the authors employ an Altera EP2S180 Stratix II FPGA device. Practical results executing multiple JPEG applications simultaneously demonstrate that the use of virtual channels improves throughput by 65%. Again high area overhead is caused by each ARM local IP. Accordingly, the NoC size is small and the approach is clearly costly to emulate large NoCs.

Unlike most reviewed emulation systems, HardNoC does not waste FPGA area with soft or hard processors. It uses small and simple dedicated traffic generators. Although this certainly reduces flexibility, the described evolution of the approach does not (See Section VII). Another advantage is the straightforward programming and results collection scheme. It employs a UART port connected to the NoC under emulation, coupled to the use of very simple data structures to interact with the external environment. This potentially allows that HardNoC operates at speeds closer to those of the final system, compared to other approaches.

III. NoC Assumptions

One of the goals of the HardNoC platform is to support different NoC architectures. Obviously, it is not possible to support any NoC structure. Some architectural assumptions define the class of NoCs supported: (*i*) 2D direct topologies such as 2D mesh or 2D torus, given the used traffic address generation scheme; (*ii*) 16-bit flit size, due to the format of traffic generation; (*iii*) credit-based control flow; (*iv*) priorities, the platform supports up to two flow priorities. All such assumptions are to be overcome in future versions of the platform, by developing a parameterizable platform generation tool, annex to the ATLAS NoC generation-validation environment [9] [10]. The NoC under prototyping may, on the other hand, use any routing algorithm, arbitration policy, buffering scheme and router control scheme (centralized or distributed). Thus, HardNoC is a good system to evaluate the effect of several parameters in state of the art NoCs.

IV. HardNoC Architecture

Figure 1 illustrates the HardNoC architecture, through an instance implementation. The instance has a 2x3 2D mesh NoC. It includes a *Serial IP* providing bidirectional communication between the system and a host, and several *Injector-Collector IPs*, for sending and receiving packets during the traffic execution phase. The number of Injector-Collector IPs is a function of the NoC size.

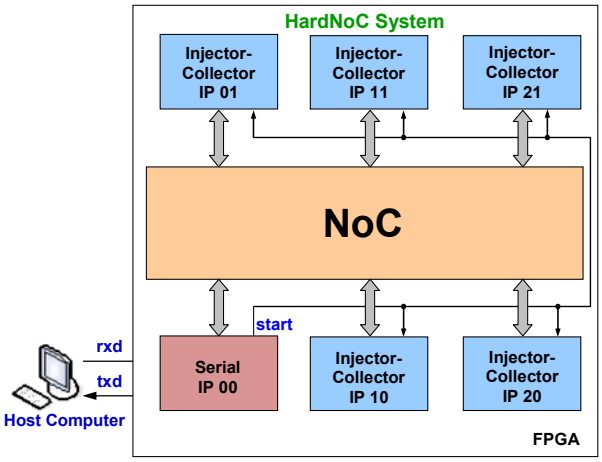

Figure 1. HardNoC system block diagram.

The physical external interface of HardNoC is very simple. It comprises only four wires: *reset* and *clock* for initialization and synchronization, and *rxd* and *txd* to provide an RS-232 bidirectional, serial interface to the host computer.

A. Serial IP

The Serial IP Core follows an RS-232 protocol using a standard serial interface. Figure 2 presents the Serial IP core two external interfaces. The signals at the top of the Figure connect the module with the host computer through FPGA pins. The signals at the bottom of the Figure, except the *start* one, connect the Serial IP Core with the NoC. Notice that this is just an example interface for a specific NoC. A different interface is available for each supported NoC. The *start* signal is a sideband signal that goes from the Serial IP Core to all Injector-Collectors IPs. The *start* signal synchronizes the traffic injection in all routers. This global signal ensures that all Injector-Collector IPs start their respective traffic generation processes at the same time. After the synchronization, interactions with the Injector-Collectors IPs must be avoided until the end of the simulation. This avoids that the Serial IP use the NoC architecture to provide communication between the user and the Injector-Collectors IPs. Not following this procedure may corrupt traffic evaluation during simulation. The basic function of the Serial IP is to assemble and disassemble NoC packets. When information comes from the host, the Serial IP may create a valid NoC packet. When the Serial IP receives a flit from the NoC it disassembles this flit and sends it serially to the host.

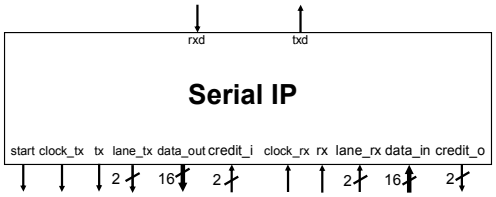

Figure 2. Serial IP two external interfaces.

Currently, the Serial IP accepts four command types. The host computer originates three of these: (1) *read,* a request for obtaining the contents of a specific memory region inside an Injector-Collector IP; (2) *write,* used for writing data to the Injector-Collector IP local memory; (3) *start,* which initializes the platform traffic execution. The fourth command accepted by the serial IP comes from the NoC and is called *read return*. This command contains the response to a *read* command generated by the host-Serial IP.

B. Injector-Collector IP

The Injector-Collector IP is a master/slave core that transmits and receives packets from the NoC. Its structure contains a Traffic Injector (TI) Module, a Traffic Collector (TC) Module and a Local Memory Block, as Figure 3 depicts. The TC module is responsible for receiving packets from other cores, computing packet latency values and storing theses values. The TI module is responsible for reading traffic parameters, generating packets and sending these packets to the NoC.

The Injector-Collector IP has a Clock Cycles Counter (CCC) used to compute latency. This counter works as a general reference clock to all Injector-Collector IPs in the HardNoC system. All CCCs receive the same global *start* signal so that they are always synchronized to each other, even if different routers and/or Injector-Collector IPs display any combination of clock frequencies and phases. This is essential to enable a generic capability to deal with GALS systems. The Local Memory Block stores the parameters of the traffic to be injected by the TI in each router during traffic execution. Figure 3 also shows the external interface of the Injector-Collector, including the global *start* signal, the *clock_ref* signal and the two router interfaces (TX/RX).

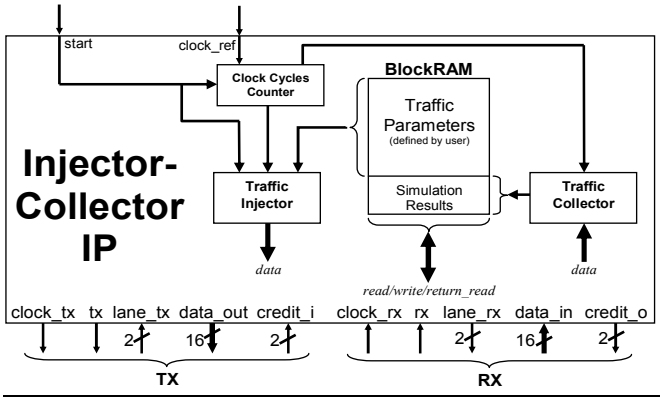

Figure 3. Injector-Collector IP external interface and internal structure.

The Injector-Collector IP accepts four command types, encapsulated in NoC packets:

- *write*: writes the local memory with data sent from the *Serial IP*.
- *data*: used at traffic execution time. The Injector-Collector IP may receive or transmit *data packets*.
- *read*: command issued by the *Serial IP* requesting contents from the local memory.
- *return read*: response to a *read* request by the Serial IP.

When a given router receives a data packet, the TC module in the *Injector-Collector IP* computes the packet latency according to Equation (1).

$$latency_{packet} = time_{reception} - time_{insertion} \qquad (1)$$

The insertion time corresponds to the time instant in which the packet is generated and is ready to enter the NoC. The source IP core includes this time in the data packet payload. The reception time corresponds to the CCC value when the last flit of the packet leaves the NoC at the destination Injector-Collector IP. The target IP of the packet captures this time and computes the received packet latency.

After calculating the packet latency, the Injector-Collector IP updates the values of accumulated, maximum and minimal latency and the number of received packets. These values are stored in four 64-bit word registers (located in the 16 lasts positions in the IP memory map).

V. TRAFFIC CONFIGURATION AND EVALUATION

The Injector-Collector IP may generate two traffic distributions: Constant Bit Rate (CBR) and Pareto on-off. The CBR model generates packets at a fixed rate. Applications such as digital non-compacted voice, audio and non-compacted video are typical examples of CBR traffic. The CBR traffic distribution is defined according to Figure 4, and the resulting *bandwidth* (in bits per second or *bps*) is obtained according to equation (2), where: (*i*) *packet_size* is the size of the packet in flits; (*ii*) *flit_width* is given in bits (16 bits for HardNoC); (*iii*) *fc* is the number of clock cycles to transmit one flit (1 in HardNoc); (*iv*) *idle_time*, interval between consecutive packets; (*v*) *T*, clock period.

$$bandwidth = \frac{(packet\_size * flit\_width)}{((packet\_size) * fc + idle\_time) * T} \qquad (2)$$

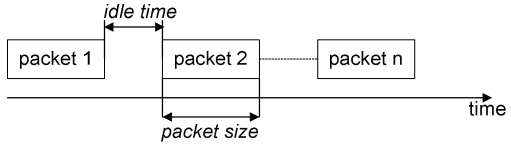

Figure 4. Definition of parameters describing CBR traffic generation.

The Pareto distribution is an on-off process, which alternate periods of traffic generation activity and inactivity, as Figure 5 depicts. An on-off model using Pareto distribution function is useful to characterize applications like MPEG-2 video and internet traffic. During activity periods, the traffic source produces fixed-length packets at regular intervals, while during inactive periods there is no packet generation. The packet size and the number of packets at each burst may be defined for each Injector-Collector IP, and the length of the (ON+OFF) period is parameterizable.

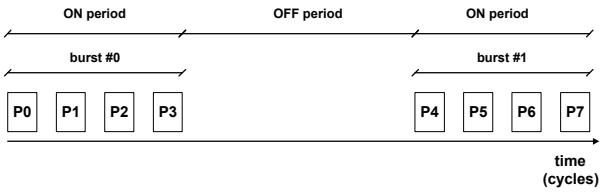

Figure 5. Definition of parameters describing the on-off traffic model.

A. *Platform Initialization*

The process of programming HardNoC has four steps. In the first, the reset signal is activated to initialize the platform. In the second step, the user adjusts the transmission/reception data rate through the host. To achieve this, the host sends one or more bytes with the hexadecimal value x"55" (binary pattern "01010101") to the Serial IP. This procedure is needed each time the HardNoC is initialized. In the third step, the host writes traffic parameters in the Injector-Collectors IP local memories using the *write* command. The first address (Address 0) defines the traffic distribution type and the second address (Address 1) informs the number of times to generate that traffic. The other parameters depend on the adopted traffic distribution (CBR or Pareto on-off). Refer to Figure 6 and Figure 7.

Each CBR traffic specification may contain up to *n* different flows, with *n* stored in the third position of memory (Address 2). The parameters that define each flow are:

- *Flow priority*: assume values 0 (low) or 1 (high);
- *Flow target*: target Injector-Collector address to which this Injector-Collector sends the flow;
- *Packet size*;
- *Number of packets* of the current flow;
- *Idle time (IT)*: number of clock cycles between the sending of two consecutive packets (see Figure 4).

Address	Data	Description
0	0	CBR distribution
1	5	Number of repetitions
2	2	Number of flows
3	0	Flow priority
4	11	Flow target
5	20	Packet size
6	20	Number of packets
7	0	Idle time
8	1	Flow priority
9	11	Flow target
10	50	Packet size
11	10	Number of packets
12	50	Idle time

1
2
5x

Figure 6. Example memory organization to specify a CBR traffic. The instance describes a traffic that repeats 5 times two distinct CBR flows, one of low priority and one with high priority.

The Pareto traffic comprises only one flow type. Figure 7 displays an example flow. In Pareto distributions, the insertion time is stored in memory and directly included in each data packet, instead of being calculated as in CBR traffic distributions. The relevant parameters are:

Address	Data	Description
0	1	Pareto distribution
1	5	Number of repetitions
2	10	Number of packets
3	1	Flow priority
4	11	Flow target
5	50	Packet size
6	6	Insertion time
7	106	Insertion time
8	206	Insertion time
9	306	Insertion time
10	406	Insertion time
11	506	Insertion time
12	606	Insertion time
13	706	Insertion time
14	2350	Insertion time
15	2450	Insertion time

10
5x

Figure 7. Example memory organization to specify a Pareto on-off traffic.

- *Number of packets* of the current flow;
- *Flow priority*: assume values 0 (low) or 1 (high);
- *Flow target*: target IP core address to which the flow will be sent;
- *Packet size*;
- *Insertion time*: time at which the packet was created.

The fourth step of using HardNoC consists initializing the traffic execution by Injector-Collector IPs. For this, the Serial IP must receive the *start* command from the host, which causes the activation of the start global signal. As a result, all Injector-Collectors begin to produce and send packets (traffic) to the NoC.

B. *Traffic Collection*

After the HardNoC traffic execution is done, the user may collect statistical data produced by the platform. As mentioned before, the accumulated, maximum and minimal latency, as well as the number of received packets are stored in the last 16 local memory of the Injector-Collector IP. In this way, the host requests this address range for each Injector-Collector IP. Figure 8 shows the low level graphic interface developed to work with the serial interface. The example shows the software requesting data from the Injector-Collector IP 11 and receiving latency data and the number of received packets.

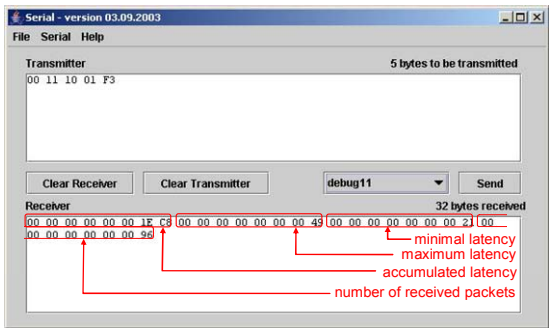

Figure 8. Serial software interface sending the debug file to the Injector-Collector IP 11 and receiving its response information.

VI. EXPERIMENTS

This Section shows two case studies of NoC evaluation with HardNoC. The first is Hermes-2VC [11] a synchronous NoC with two virtual channels per port, generated by the Atlas environment [10]. The second one is Hermes-GLP, a GALS NoC [12] designed for low power applications.

A. *Hermes-2VC validation*

This validation uses a 2x3 instance of the Hermes-2VC NoC. Hermes is a parameterizable infrastructure used to implement low area overhead packet switching NoCs with 2D mesh topology, allowing the selection of flit size, buffer depth, and number of virtual channels (VCs), among other parameters [7]. Figure 9 shows practical data on the physical synthesis of the developed prototype.

```
Device Utilization Summary:
---------------------------

Selected Device: 2vp30ff896-7

Number of occupied Slices:       11,923 out of  13,696   87%
Total Number Slice Registers:     7,471 out of  27,392   27%
Number of 4 input LUTs:          20,291 out of  27,392   74%
  Number used as logic:          17,510
  Number used as a route-thru:    1,501
  Number used for Dual Port RAMs: 1,280
Number of bonded IOBs:                4 out of     556    1%
Number of Block RAMs:                10 out of     136    7%
Number of GCLKs:                      4 out of      16   25%
Number of DCMs:                       1 out of       8   12%
```

Figure 9. HardNoC physical synthesis report for a 2x3 Hermes-2VC.

The prototype used a Xilinx University Program Virtex-II Pro Development System (XUP-V2PRO) [13]. This board is built around one Xilinx XC2VP30 Virtex-II Pro FPGA. The HardNoC system used around 87% of the available slices and around 74% LUTs of the FPGA device, as depicts the physical synthesis area report of Figure 9.

One important step for achieving good prototyping results is to define the chip floorplan. Figure 10 illustrates the HardNoC layout after applying floorplan constraints and performing the physical synthesis. Each color corresponds to a given HardNoC IP.

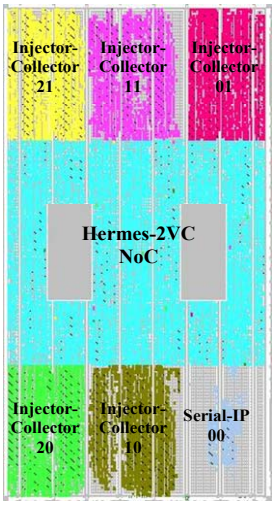

Figure 10. HardNoC final IP floorplanning after producing constraints and physically synthesizing the design. The drawing is illustrative of a typically correct synthesis process. The grey rectangles in the middle of the chip represent the (unused) PowerPC processors.

To test functionality the HardNoC was programmed with the simple traffic configuration presented in Figure 11: (1) Injector-Collector IP 01 transmits packets to Injector-Collector IP 11, which transmits packets to Injector-Collector IP 01; (2) Injector-Collector IP 10 transmits packets to Injector-Collector IP 20, which transmits packets to the Injector-Collector IP 10; (3) Injector-Collector IP 21 transmits packets to Injector-Collector IP 10.

All Injector-Collectors IPs were programmed with a CBR traffic that contains the parameters presented in Figure 6, only changing the flow target for each distinct Injector-Collector. The obtained performance results of the test appear in Table I. All results are correct, given the injected traffic. Note that to improve the predictability of the test, there is no traffic conflict present, except at the local port of router 10. We immediately notice the effect of such a conflict, a large increase in average and maximum latencies for packets arriving at router 10.

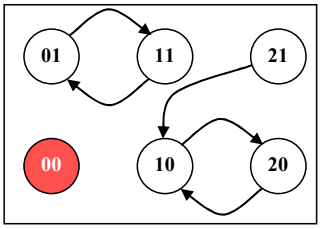

Figure 11. Example of traffic spatial distribution for Injector-Collector IPs in HardNoC.

TABLE I. RESULTS OBTAINED FROM THE APPLIED TRAFFIC. ALL RESULTS ARE CORRECT.

Tester IP	Description	Value
10	Number of Packets	**300**
	Minimal Latency	**32**
	Average Latency	**112**
	Maximum Latency	**789**
20	Number of Packets	150
	Minimal Latency	32
	Average Latency	53
	Maximum Latency	75
01	Number of Packets	150
	Minimal Latency	32
	Average Latency	52
	Maximum Latency	73
11	Number of Packets	150
	Minimal Latency	32
	Average Latency	53
	Maximum Latency	78
21	Number of Packets	0
	Minimal Latency	0
	Average Latency	0
	Maximum Latency	0

Using FPGA prototyping should improve execution time significantly. To measure this, a simulation of the same traffic distribution was run using Modelsim 10.0c in a 4-core, 3.2 GHz Xeon CPU with 32GB RAM, running Red-Hat 5.1. The speedup of the XUP-V2PRO prototype is around 2,200 times, compared to the HDL simulation time.

B. Hermes-GLP validation

As a second validation case study, a 3x3 Hermes-GLP NoC with 2 priority levels was prototyped using the HardNoC. Each router receives two different clock frequencies and operates at the minimum frequency allowed by the combination of packet priorities at its input ports. If no packet is present at the inputs, the router enters clock gating. If only low priority packets are entering the router chooses the lowest operating frequency to operate. Otherwise it uses the highest frequency. The concept is extendable to any number of clocks. Every Hermes-GLP router has bi-synchronous buffers at each of its ports. The HardNoC prototype employed this time was a Xilinx ML505 Virtex-5 Evaluation Platform. This board is built around a XC5VLX50T Virtex-5 FPGA. The HardNoC system used around 80% of the available slices and around 67% of FPGA LUTs, as Figure 12 details.

```
Device Utilization Summary:
---------------------------

Selected Device: xc5vlx50tffg1136-1

Number of BSCANs                          1 out of 4              25%
Number of BUFGs                           6 out of 32             18%
Number of BUFGCTRLs                       9 out of 32             28%
Number of DCM_ADVs                        1 out of 12             8%
Number of ILOGICs                         1 out of 560            1%
      Number of External IOBs             4 out of 480            1%
      Number of LOCed IOBs                4 out of 4              100%

Number of OLOGICs                         1 out of 560            1%
Number of RAMB18x2s                       9 out of 60             15%
Number of Slices                          5760 out 7200           80%
Number of Slices Registers                14125 out of 28800      49%
      Number used as Flip Flops           14124
      Number used as Latches              1
      Number used as LatchThrus           0

Number of Slices LUTS                     19439 out of 28800      67%
Number of Slices LUT-Flip Flop pairs      21068 out of 28800      73%
```

Figure 12. HardNoC physical synthesis report for the 3x3 Hermes-GLP.

As in the previous experiment, a floorplan was made to enable an efficient implementation. Figure 13 presents the employed floorplan. The main difference between the two floorplans is the NoC position. While in the first version the NoC is a single block, the Hermes-GLP prototype used a tile architecture with each router and local IP forming a module.

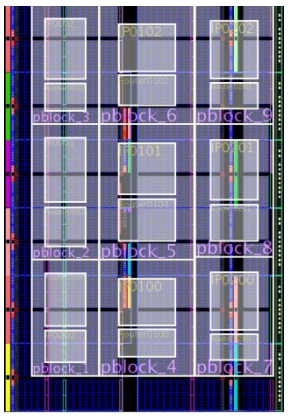

Figure 13. HardNoC final IP placement as a result of using floorplanning constraints. Each block contains one IP and its respective router.

For this test, a different traffic model was programmed. Figure 14 shows the corresponding spatial traffic distribution. Only two Injector-Collector IPs were programmed: (i) IP 12 transmits packets to IP 10 with low priority; (ii) IP 21 transmits packets to Injector-Collector IP 01 with high priority. Traffic from IP 12 starts first. Next, starts the traffic from IP 21. The distribution is such that at some moment both traffics pass simultaneously by router 11. Also, the traffic from IP 21 ends first. When router 11 receives packets only from IP 12, it will work at the lowest frequency, because it is dealing only with low priority packets. When router 11 receives packets from both IPs, the highest frequency is used. During the execution routers 00, 02, 22 and 20 are under clock gating, thus dissipating the least power.

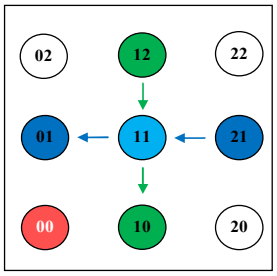

Figure 14. Sender/receiver relationship graph. The green (vertical) traffic is low priority, while the blue (horizontal) is high priority. IP 00 is the Serial IP.

The Injector-Collector IPs 12 and 21 have the following parameters: (1) Traffic distribution: uniform; (2) Repetitions: 10; (3) Packet size: 200; (4) Number of packets: 50; (5) Idle time: 200. Table II presents the expected and measured performance results of the test. Using the same simulation setup described in Section VI.A led to a speedup of around 7,000 times, with regard to HDL simulation.

TABLE II. EXPECTED RESULTS OF THE TEST.

Tester IP	Description	Value
01	Number of Packets	500
	Minimal Latency	232
	Average Latency	233
	Maximum Latency	234
10	Number of Packets	500
	Minimal Latency	234
	Average Latency	235
	Maximum Latency	242

VII. CONCLUSIONS AND FUTURE WORK

This paper presented HardNoC, a simple FPGA-based prototyping platform for NoCs. It was used to validate two very distinct NoC architectures. Avoiding the use of soft processors for traffic injection/collection allows incrementing the dimension of NoCs that can be emulated.

This first approach of NoC prototyping is evolving in several directions. First the next version of HardNoC will support the traffic files generated by the Atlas environment. It consists in changing Injector-Collector IPs from traffic injectors to mere injectors-collectors, which read packets annotated with injection timestamps and sends them to the NoC. This simplifies IPs further, but requires larger IP memories. Another ongoing change in HardNoC is substituting the communication interface from a slow serial interface to a TCP/IP Ethernet port, for improving host communication speed and enable access to HardNoC from the Internet. It would be interesting a non-invasive interface, which allows the user to set and collect information without interference on the simulation besides adding a new Injector-Collector IP (replacing the Serial IP). Automating the HardNoC instance generation process is another goal, to allow parameterizing values as flit size, topology range, and flow control.

ACKNOWLEDGEMENTS

This work is partially supported by the CNPq-PNM (under grant 134878/2010-8), and by the BPA-PUCRS Program. Professors Moraes and Calazans also acknowledge the CNPq support under grants 301599/2009-2 and 309255/2008-2, respectively. Authors would like to acknowledge the support granted by CNPq to the INCT-SEC (National Institute of Science and Technology – Critical Embedded Systems – Brazil), process no. 573963/2008-8.

REFERENCES

[1] Benini, L; De Micheli, G. "Networks on chip: a new SoC paradigm", IEEE Computer, 35(1), pp. 70-78. Jan. 2002.

[2] Guerrier, P.; Greiner, A. "A generic architecture for on-chip packet-switched interconnections", in DATE, pp.250-256. Mar. 2000.

[3] Duato, J.; Yalamanchili, S.; Ni, L. "Interconnection Networks", Elsevier Science, 600p., 2002.

[4] Wen, H.; Du, C.; Zhang, D.; Geng, L.; Gao, M.; Chen, Y.; Verhoeff, T. "Design of An On-Line Configurable Traffic Generator for NoC", in ASID, pp. 556-559, 2009.

[5] Lotlikar, S.; Pai, V.; Gratz, P. "AcENoCs: A configurable HW/SW Platform for FPGA Accelerated NoC Emulation", in VLSI Design, pp.147-152, 2011.

[6] Tan, J.; Fresse, V.; Rousseau, F. "Generation of emulation platforms for NoC exploration on FPGA", in RSP, pp. 186-192, 2011.

[7] Moraes, F. et al. "HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip", Integration, the VLSI Journal, vol. 38, no. 1. pp. 69-93, Oct. 2004.

[8] Hou, N.; Zhang, D.; Du, G.; Song, Y.; Wen, H. "Design and performance evaluation of virtual-channel based NoC", in ASID, pp. 294-298, 2009.

[9] Ost, L. et al. "MAIA - A Framework for Networks on Chip Generation and Verification". In: ASP-DAC, pp. 49-52, 2005.

[10] GAPH – Grupo de Apoio ao Projeto de Hardware. "Atlas – An environment for NoC Generation and Evaluation". Available at http://www.inf.pucrs.br/~gaph/AtlasHtml/AtlasIndex_us.html, captured in September, 2008.

[11] Mello, A. et al. "Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC". In: SBCCI, pp. 178-183, 2005.

[12] Pontes, J.; Moreira, M.; Soares, R.; Calazans, N. "Hermes-GLP: A GALS Network on Chip Router with Power Control Techniques". in ISVLSI, pp. 347-352, 2008.

[13] Xilinx, Inc. "Xilinx University Program Virtex-II Pro Development System". Hardware Reference Manual, UG069, V1.0, March, 2005, 138 p.