NoC-based Platform for Embedded Software Design: An Extension of the Hellfire Framework

Felipe G. Magalhães, Oliver Longhi, Sérgio J. Filho, Alexandra Aguiar, Fabiano Hessel
Faculty of Informatics – PUCRS – Av. Ipiranga 6681, Porto Alegre, Brazil
E-mail: {felipe.magalhaes, oliver.longhi, sergio.johann}@acad.pucrs.br,
{alexandra.aguiar, fabiano.hessel}@pucrs.br

Abstract— This paper presents an extension of the Hellfire Framework (HFFW), providing an intuitive and powerful web interface to build, test and debug a complete Multiprocessor System-on-Chip (MPSoC). Among the new functionalities presented, it is possible to highlight: **i)** the architecture builder tool, used to set up all the MPSoC architecture; **ii)** the possibility to use a Network-on-Chip (NoC) as the communication mean, and; **iii)** a new simulator, providing a fast and accurate high level Instructions Set Simulator (ISS) with a miss ratio less than 5%. In order to validate the new simulator accuracy several tests were taken, first using traffic generators and then an implementation of the Secure Hash Algorithm (SHA). The achieved results are discussed throughout the paper.

Keywords—MPSoC, NoC, Instruction Set Simulator

I. Introduction

Since MPSoC have been introduced as a common Embedded Systems' implementation alternative, one of the main design issues concerns in the way communication between internal components is done. As the amount of components grows, some older approaches, like buses, tend to be less adopted, especially because of the low scalability, resulting in the increasing design complexity [1].

Traditional bus-based systems have the communication of the system as a common bottleneck, affecting its overall performance [2]. To solve this issue, one of the most popular approaches consists of using NoC-based solutions.

In this context, NoC-based systems provide better communication performance [3], as routers are responsible for the communication management, by correctly directing packets exchanged over the network. Each point of the network consists of a router and a component attached to it. For instance, processors and memories can be attached to each router of the NoC. Still, NoCs usually have improved energy efficiency and reliability [4] and high reusability levels.

Another design issue related to the embedded systems project is that many restrictions are found, like area size, memory limitation and energy consumption, yet with a development time getting smaller and smaller due to time-to-market. Development platforms, like Le Moigne et al [5] and Yoo et al [6] try to reduce the development time, providing features that speed it up, such as simulators and debugging tools.

Thus, this paper introduces a new version of the Hellfire Framework [7] and all modifications made in order to provide the possibility to use a NoC as the MPSoC's communication mean, including a new ISS and a powerful web interface to build and debug a complete MPSoC.

The remainder of the paper is organized as it follows. Section **2** shows some related work. Section **3** presents the Hellfire System, used as the base architecture in this work. Following, a high-level model of a NoC that extends features of Hellfire is described in Section **4**. Results are presented in Section **5** and, finally, Section **6** concludes the paper besides presenting some future work.

II. Related Work

There are many tools for simulating and debugging embedded softwares. These tools enable designers to evaluate and verify systems in earlier stages of software and hardware development. In this section we present simulators that accomplish the task of evaluating both interconnection media and processor.

MP-ARM [8] is a simulation platform for MPSoCs which is based on SystemC to provide the simulation environment. Since SystemC is used, hardware and software can be described in the same language. So, a model of AMBA bus compliant communication architecture is described in SystemC, while an Instruction Set Simulator (ISS) of ARM Processors is used to debug applications. The problem of this approach is that it requires wrappers to integrate interconnection bus and processors, resulting in the system bottleneck.

MC-Sim [9] is a heterogeneous multi-core simulator framework which is capable of simulating a variety of processor, memory, NoC configurations and application specific coprocessors. MC-Sim offers a cycle-accurate and functional ISS based on SESC Simulator [10] to model processor cores. It also contains a detailed structural and cycle-accurate model for the NoC representation, beyond C-based models to simulate application specific coprocessors. This framework supports multi-tasking and simulate multi-threaded applications, but in order to ensure the tractable simulation of multiple cores, a full OS is not available. Although there is a lack of intelligibility in how the NoC is modeled, the domain of this work is similar to the proposal that will be presented in this article.

HVP (High-level Virtual Platform) [11] is a framework that supports developers at early MPSoC software development. HVP simulator is built on top of SystemC and it allows developers to integrate in-house processor simulator with third party ISSs. Until then, wrappers for ISSs generated by LISATek processor designer were available, but as MP-ARM, it requires wrappers to overcome the issue of executing different simulators simultaneously. The framework also provides a native simulator that does not require previous knowledge of the architecture and offers speedups of more than 2x compared to ISSs, but this approach is not efficient to evaluate real-time restrictions. Another issue of HVP is that in order to make simulator generic, details of the target platform are abstracted away and the communication of these models are performed through generic shared memories. It means that the impact of the communication media chosen may not be efficiently evaluated by the framework.

Schonwald et al [12] proposed a framework that enables the integration of IP-components through different NoC architectures is presented. Configurations such as NoC topology(mesh, torus or hypercube), number of incoming and outgoing ports for switches, routing algorithm, forwarding mechanism, packet- and flit-size as well as size of the input and output buffers of the switch can be made. As NoC architecture, the interface to integrate IPs is described in XML by an IP-XACT [13] interface description that is a standardized exchange format for the interface description of IP-components. Such approach enables the interconnection of application specific hardware component modeled in SystemC as well as ISSs. Finally, a modified SimpleScalar [14] ISS can be connected to the SystemC simulation model by using shared memory and *Memory Mapped IO*. In spite of the wrapping problem, as [8] and [11], this work contains distinct characteristics like a model to inject faults and broken states to switches or links and also a good range of NoC configurations to explore architectures.

In Aguiar et al [15] and Johann et al, a methodology for software execution time and energy consumption estimations of homogeneous MPSoCs is presented. It is composed by a design flow and a simulation tool. The simulation tool is written in C and can form bus-based interconnections and also integrate until 128 in-house Plasma [16] processor simulators. This proposal extends the design flow of these works to simulate and evaluate different configurations of NoC-based architectures in earlier stages of software and hardware development.

III. Hellfire System

The Hellfire System is a set of tools focused on the development of embedded systems. It provides its own design flow that comprehends different abstraction levels, from C application development to FPGA prototyping. This design flow is supported by several tools and modules that compose the Hellfire Framework (HFFW). From a single processor point of view, the designer can develop the application C code and run it over the Hellfire Operating System (HFOS), which is a highly configurable real-time modular micro-kernel based OS. From the platform point of view, the designer can add processors to the system, configure each one of them and, by using the HFOS API, develop parallel embedded applications which are able to exchange data and even migrate tasks.

A. *HellfireOS*

The HellfireOS (HFOS) [15] is a real-time operating system (RTOS) developed in order to ensure maximum flexibility in its configuration thus allowing a high level platform customization. To enable such feature, the HFOS was implemented in a modular way, where each module corresponds to some specific functionality.

Figure 1 shows the kernel modular organization, where all hardware-specific functions are defined in the first layer, named HAL (Hardware Abstraction Layer). The uKernel lays just above this, along with communication, migration, memory management and mutual exclusion drivers, as well the API, which are placed over the uKernel layer. The user applications belongs to the top layer.

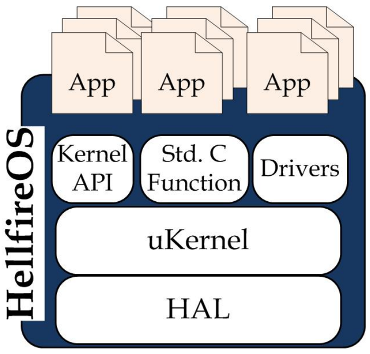
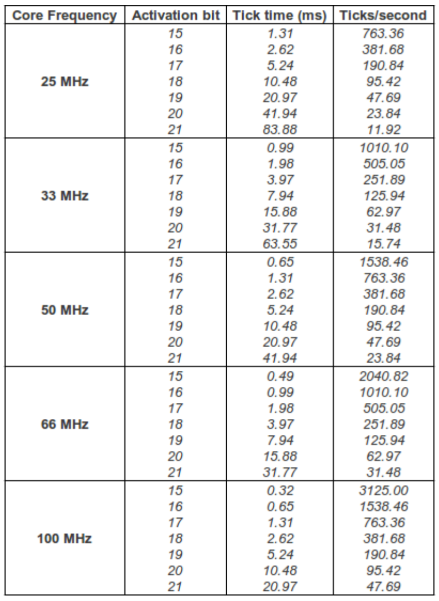


Fig. 1. HellfireOS Structure

Due to its modular implementation, HellfireOS is easily portable to others architectures, requiring only the rewrite of the hardware-dependent functions, such as the interrupt service and its initialization and the context switch.

In order to optimize the kernel final size, allowing the HFOS usage even in architectures with area limitations, some parameters like the users tasks maximum number, the stack and heap size and the drivers usage, are configurable.

Another configurable parameter is the activation bit used by the timing register, which determines the system tick size. The tick size corresponds to the minimum temporal unit of the system and can be used in a predetermined interval, varying from 0.32 ms to 83.88 ms. Figure 2 shows the relation between the core frequency and the activation bit.



Core Frequency	Activation bit	Tick time (ms)	Ticks/second
25 MHz	15	1.31	763.36
	16	2.62	381.68
	17	5.24	190.84
	18	10.48	95.42
	19	20.97	47.69
	20	41.94	23.84
	21	83.88	11.92
33 MHz	15	0.99	1010.10
	16	1.98	505.05
	17	3.97	251.89
	18	7.94	125.94
	19	15.88	62.97
	20	31.77	31.48
	21	63.55	15.74
50 MHz	15	0.65	1538.46
	16	1.31	763.36
	17	2.62	381.68
	18	5.24	190.84
	19	10.48	95.42
	20	20.97	47.69
	21	41.94	23.84
66 MHz	15	0.49	2040.82
	16	0.99	1010.10
	17	1.98	505.05
	18	3.97	251.89
	19	7.94	125.94
	20	15.88	62.97
	21	31.77	31.48
100 MHz	15	0.32	3125.00
	16	0.65	1538.46
	17	1.31	763.36
	18	2.62	381.68
	19	5.24	190.84
	20	10.48	95.42
	21	20.97	47.69

Fig. 2. Tick Time x Core Frequency

B. *N-MIPS*

Gate-level simulations are useful to get accurate execution time and energy consumption. However, even for simple scenarios this technique represents prohibitive time to run. This leads designers to develop high-level simulation tools.

In Hellfire Framework, N-MIPS, an instruction set simulator of the MIPS-I, was implemented [15]. It runs native object code, executing it according to the hardware implementation of Plasma processor. On top of that, facilities like cycle counting and energy consumption measurement were implemented. A timing and functional emulation of the UART was also implemented. All these tools allow designers to evaluate energy, performance and behavior of different algorithm implementations with N-MIPS. Additionally, several reports are generated by simulator through output facilities of the operating system: (a) application type, (b) energy consumption, (c) load behavior, (d) deadline miss ratio, (e) migration monitoring and (f) communication traffic. If one of the previous listed requirements is not satisfied, the designer can come back to previous steps on the design flow and perform refinements on application and architecture.

Due to hardware limitations only 128 processors can be represented on a simulation. These processors communicate through a bus that, as processors, is modeled at high-level and has its behavior and energy consumption annotated. The high-level model of the MPSoC is based on a custom bus architecture that was described in VHDL, simulated and prototyped in FPGA.

Regarding the fact that N-MIPS can only represents homogeneous architectures no wrappers are needed to integrate bus with processors. The whole architecture is simulated at the same process without the need of interprocess communication techniques on the host. It conducts to better simulation times, and it represents a faster development environment.

C. *Hellfire Framework*

The Hellfire Framework (HFFW) [7] allows a complete deployment and test of parallel embedded applications, defining the HW/SW architecture to be employed by the designer. The HFFW is divided in three modules as it follows, and discussed along the remainder of this section.

The Hellfire Framework modules are:

- HellfireOS;
- N-MIPS MPSoC Simulator, and;
- architecture builder.

Section **3**-A already covered HellfireOS and its characteristics, and the ISS was presented in Section **3**-B.

The third module, called architecture builder, is used to specify the target architecture and to configure all the HellfireOS parameters. All system configurations can be made through a web interface designed to favor the project development. The devisor can easily create and test a complete MPSoC using all available tools. Figure 3 shows the architecture builder main window, still without any processors or communication means.

It is possible to notice that the designer can choose between five different MIPS-like processors [17] and two communication means, either bus or NoC. To set up all system, the only thing the designer needs to do is to drag-and-drop the boxes representing the desired modules.

Figure 4 illustrates an MPSoC formed by nine processors interconnected by a 3x3 mesh NoC. The designer can now configure each processor individually, or all of them at once. It is

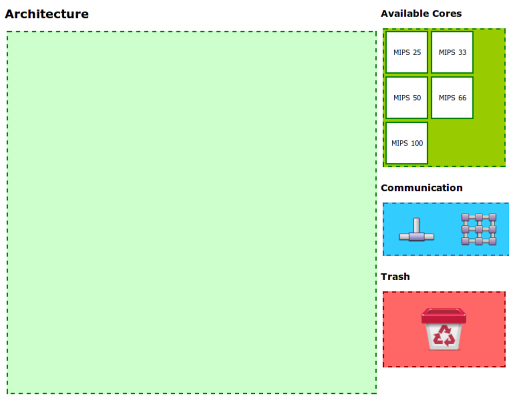


Fig. 3. Architecture Builder Window

possible to configure all HellfireOS images[1] as well, which is shown on Figure 5. All OS parameters can be defined by just checking or unchecking buttons on the web interface and the all system is automatically built on the background.

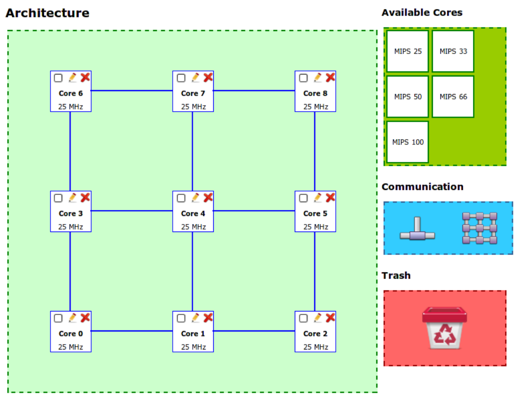
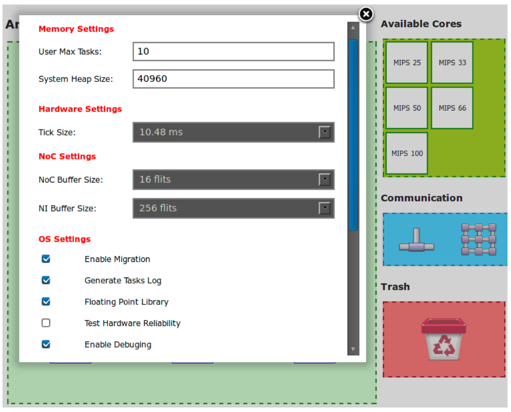


Fig. 4. Architecture Done



Memory Settings

User Max Tasks: 10

System Heap Size: 40960

Hardware Settings

Tick Size: 10.48 ms

NoC Settings

NoC Buffer Size: 16 flits

NI Buffer Size: 256 flits

OS Settings

- ✓ Enable Migration
- ✓ Generate Tasks Log
- ✓ Floating Point Library
- ☐ Test Hardware Reliability
- ✓ Enable Debuging

Fig. 5. HellfireOS Configuration Window

It is important to highlight that all design flow presented in [7] is still valid, only with a new communication possibility.

[1]in this context, image corresponds to the binary file that will be generated after compiling the system, and will be simulated on the ISS and/or prototyped directly on hardware

IV. High-Level NoC Model

To achieve better simulation times, a high-level of a 2D mesh architecture was implemented. Others topologies such as torus and hypercube can also be made with few modifications. However, other architectures are not presented on this work because hardware characterizations of them would be necessary to proof their efficiency.

The high-level NoC model is implemented in C and simulates the behavior according to the Hermes architecture [18]. A description of the communication infrastructure is presented in Section **4**-A. Nevertheless, to efficiently represent the network environment, not only routers and links between them were implemented. In-house Network Interfaces (NI) were also described in VHDL and then modeled in high-level. So, a more accurate estimation of the entire network component can be performed. Details of Network Interface is described in Section **4**-B.

A. NoC

The Hermes specification of NoC implements a subset of the OSI reference model. In this architecture only Physical, Data Link, Network and Transport layers are present. However, the fourth layer, Transport, is implemented in IP cores connected to the NoC. So, models of the three first layers were implemented at high levels of abstraction.

The physical layer is fundamental to define aspects of transmission and physical characteristics of a network. In the NoC scenario, the physical layer is responsible for connecting network interfaces and switches to their neighbors. At the NoC simulation tool we used the concept of Channel to model input and output between switches. Each bi-directional router channel has two ports, one for input and other for output that united form a communicating channel when attached to other router channel. Figure 6 shows how channels are composed. The length of data signals is parameterizable.

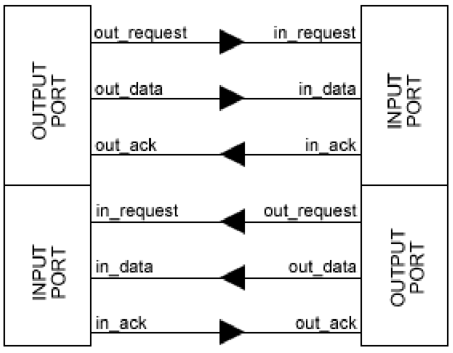


Fig. 6. Physical layer model

The next layer, Data Link, is responsible for creating the links between nodes. With these links, logical connections are built and transmissions are enabled. There, a handshake protocol is implemented based on hardware pinouts. The entity that has to transmit a flit signals bits in the *out_data* signal and asserts *out_request* pinout. A successful transmission is recognized when *out_ack* is active.

The Network layer provides a logical addressing scheme and delivering end-to-end packets. Hermes implements packet switching, which means that this layer also realizes the task of routing packets through the network. Figure 7 shows the logical structure of a Hermes switch. There are 5 bi-directional ports (**N**orth, **S**outh, **E**ast, **W**est and **L**ocal), given that, each of them is leashed to an incoming buffer with a parameterizable depth. Four ports are used to link neighbor routers, while the fifth (local port **L**) is used to link a third party processing element. This link will is detailed in Section **4**-B. With these five ports, five simultaneous routing connections can be established. Connections are realized by the Control Logic unit that is split into arbitration and routing. Routing controls whether a packet being received is addressed to router under concerned, in case positive, it is routed to local port, in case negative, this packet is forwarded to a neighbor router according to a routing algorithm. Although Hermes implements many routing algorithm, this work implements the XY Algorithm. As related to NoC topologies, other routing algorithms could be used, but further hardware specification would be necessary to proof the model's efficiency. Arbitration logic is used to pick a connection when there are more than one incoming packet that need to be forwarded by a common output port.

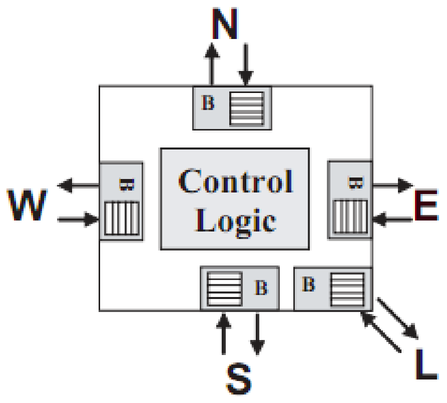


Fig. 7. Hermes Switches. Extracted from [18].

It is interesting to observe that each of the layers are responsible for contentions on the real world. The handshake, e.g., uses 2 cycles to perform a flit transmission, but, if destination port does not signal the acknowledgment, then, following flits will remain in contention. The same happens to arbitration. There the arbitration logic in this architecture takes four clock cycles to treat a new routing request due to the routing algorithm delay. Another scenario of contention happens when a packet arrives in a router and the respective output port of the packet route is already in use. In this case, incoming buffers will help to receive flits, but they will soon get full and then contention will happen. This entire scenario shows how complex it is to simulate a system like that, and how important parameterizible variables like the buffer depth, flit length, packet size and NoC dimension are.

B. Network Interface

The network interface is a component that connects an IP to a router. This interface does not address any of the previous discussed layers of the communication protocol stack. Instead it is truly useful to decouple computational components from communication architecture and its implementation, in order to improve IP reuse flexibility and avoid the commitment of the whole architecture to a particular communicating infrastructure.

An in-house hardware module was developed to wrap the NoC Hermes network and the Plasma processors. The main objective of this component is simplifying drivers implementation. Figure 8 shows the lack of complexity of this hardware. This interface has two temporary storing buffers. The incoming buffer is used to manage packets that came from a switch connected by the local port, while the outgoing buffer receives packets originating from a processing unit. The size of these buffers is parameterizable but it was projected to be the same size of the packet, since all packets have the same size in the proposed architecture. Other tasks of this component are to signals interruptions when all flits of a particular packets are buffered and queue, dequeue or forward flits when memory mapped operations (read, write and status) are performed.

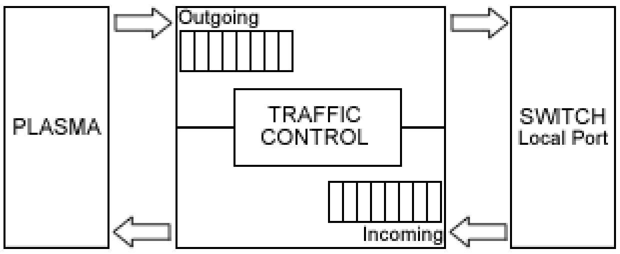


Fig. 8. Network interface representation

The complexity and accuracy of the proposed simulator lays on the cycles annotated to spend when situations like arbitration, hand-shake, buffers operations and routing occur. The proposed model respects values announced on the Hermes specification [18] and the times observed on network interface RTL simulation. Section **5** shows some comparing results of the proposed model and RTL simulations.

V. Results

To accurately verify and proof the efficiency of the proposed model, two different types of results were explored. The first realizes the simulation in RTL of the Hermes. It uses the Atlas Environment [19] to build both the architecture and some scenarios of traffic. Results are previously obtained with the ModelSim Simulator. Although in-house network interface was modeled in VHDL, it is not part of the Atlas Environment, so the simulation of the generated traffic can't be used to valid NI implementation. Network Interface is validated on the second type of simulation, where a whole system will be simulated using HellfireFW and prototyped on a FPGA. Both results are then compared.

A. Simulating the NoC Infrastructure

The first scenario simulates a 3x3 network. The five ports of each router have buffers with 16 positions of depth, and packets have a fixed size of 128 flits, given that, each flit has 16 bits. Routers where configured to operate in a frequency of 25Mhz. Each router forwards a packet to the centered router number 4. This scenario is useful to represent an application that was modeled using a master-slave topology. Figure 9 shows a chart where the clock cycles needed to receive each packet are shown. With the conformity of behavior of both TL and RTL and the small mean average error of 1.27050%, it is possible to notice the level of accuracy that the model provides.

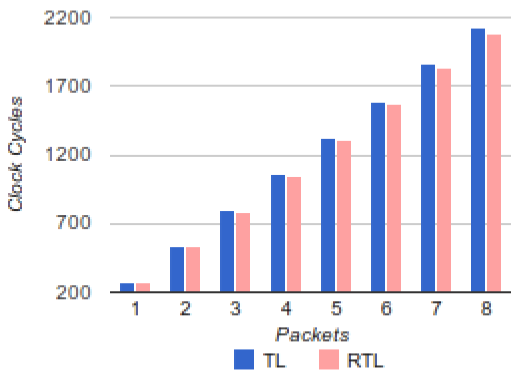


Fig. 9. First scenario simulated.

The second scenario is very similar to the first one. But, in spite of each router forwards one packet, it forwards ten. It results in contention, because eighty packets pass through the architectures and they are all addressed to the same destiny. Figure 10 shows at which clock cycle each packet has arrived at router 4. The interesting of this chart, is that, although the number of packets increased 10x, the mean average error is yet low, 1.41943%.

B. Simulating the entire Architecture

In order to verify not only the ISS' communication capability but also its Plasma's accordance, in the last scenario a real application was used as a test case. A version of the Secure Hash Algorithm (SHA) was used. The SHA is a cryptographic hash function modeled by the National Security Agency (NSA) and an established pattern in a lot of security protocols and webservices, like: TLS, SSL, PGP and SSH, among others.

The SHA-I implemented algorithm produces an output with 160 bits and is limited to process messages with a maximum length of 2^{64} bits. It can be seen as a cipher block algorithm where the input is the previous iteration output of the algorithm and the key is the next chunk of the message being processed. Each computational iteration consists of eighty rounds which are divided into four stages of twenty rounds each. All rounds require only bitwise boolean operations with 32-bit registers [20].

To stress the system's communication, besides testing the core implementation, the SHA application was divided into five tasks, each one running on one different core. A 2x3 mesh NoC was used, which is illustrated in Figure 11. The system works as it follows:

1. **Core0** is initialized with four strings;
2. each string is sent to one different core. In this example **Core1**, **Core2** **Core3** and **Core4** will receive a string each;
3. the SHA algorithm is applied on the received string on each core, and;

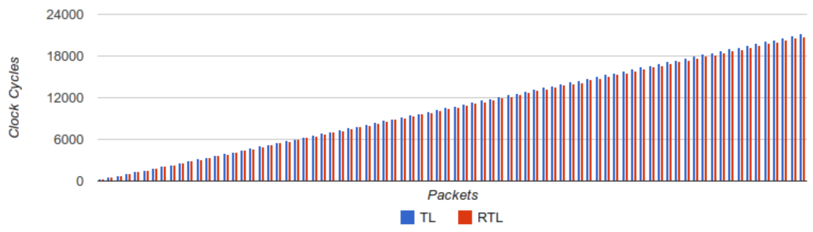


Fig. 10. Second scenario simulated.

4. all cores respond with one ACK signal to **Core0**, pointing the end of the execution.

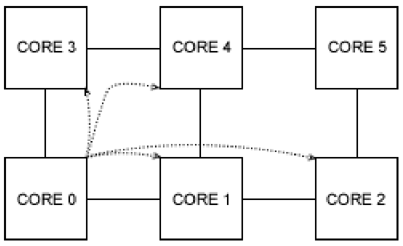


Fig. 11. Architecture used to run the application

The same architecture was simulated using the Hellfire Framework and after that using a hardware simulator. Table I shows the achieved results after four independent simulations, which proves the ISS' effectiveness.

TABLE I
MISS RATIO - ISS X HW

Cycles	HW/Cycles	Miss Ratio
1 056 013	1 010 604	4.30%
1 053 643	1 011 497	4.00%
1 056 013	1 012 716	4.10%
1 053 643	1 012 551	3.90%

VI. CONCLUDING REMARKS AND FUTURE WORK

Embedded systems have tight computational requirements that can be achieved using MPSoCs solutions. As new embedded communications architectures, like NoCs, begin to share an important piece of the market, new challenges in order to validate its performance arise.

This work presented an extension of Hellfire Framework and all modifications made in order to provide a high-level environment to create, simulate and debug an MPSoC.

Future works include the addition of a module to measure the communication mean power consumption on-the-fly, as well as tools to partition and map tasks to processors automatically.

ACKNOWLEDGMENT

The authors acknowledge the support granted by CNPq and FAPESP to the INCT-SEC (National Institute of Science and Technology - Embedded Critical Systems - Brazil), process 131954/2011-3.

REFERENCES

[1] Thuan Le and M. Khalid, "Noc prototyping on fpgas: A case study using an image processing benchmark," jun. 2009, pp. 441–445.
[2] C. Hilton and B. Nelson, "Pnoc: a flexible circuit-switched noc for fpga-based systems," *Computers and Digital Techniques, IEE Proceedings -*, vol. 153, no. 3, pp. 181 – 188, May 2006.
[3] S. Tota, M.R. Casu, M.R. Roch, and M. Zamboni, "A multiprocessor based packet-switch: performance analysis of the communication infrastructure," in *Signal Processing Systems Design and Implementation, 2005. IEEE Workshop on*, nov 2005, pp. 172 – 177.
[4] Luca Benini and Giovanni De Micheli, "Powering networks on chips: energy-efficient and reliable interconnect design for socs," in *Proceedings of the 14th international symposium on Systems synthesis*, New York, NY, USA, 2001, ISSS '01, pp. 33–38, ACM.
[5] R. Le Moigne, O. Pasquier, and J.-P. Calvez, "A generic rtos model for real-time systems simulation with systemc," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, Feb. 2004, vol. 3, pp. 82–87 Vol.3.
[6] S. Yoo, G. Nicolescu, L. Gauthier, and A. Jerraya, "Automatic generation of fast timed simulation models for operating systems in soc design," in *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, Washington, DC, USA, 2002, p. 620, IEEE Computer Society.
[7] A. Aguiar, S.J. Filho, F.G. Magalhaes, T.D. Casagrande, and F. Hessel, "Hellfire: A design framework for critical embedded systems' applications," in *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, mar. 2010, pp. 730–737.
[8] Luca Benini, Davide Bertozzi, Alessandro Bogliolo, Francesco Menichelli, and Mauro Olivieri, "Mparm: Exploring the multi-processor soc design space with systemc," *J. VLSI Signal Process. Syst.*, vol. 41, pp. 169–182, September 2005.
[9] Jason Cong, Karthik Gururaj, Guoling Han, Adam Kaplan, Mishali Naik, and Glenn Reinman, "Mc-sim: an efficient simulation tool for mpsoc designs," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, Piscataway, NJ, USA, 2008, ICCAD '08, pp. 364–371, IEEE Press.
[10] Jose Renau, Basilio Fraguela, James Tuck, Wei Liu, Milos Prvulovic, Luis Ceze, Smruti Sarangi, Paul Sack, Karin Strauss, and Pablo Montesinos, "SESC simulator," January 2005, http://sesc.sourceforge.net.
[11] Jianjiang Ceng, Weihua Sheng, Jeronimo Castrillon, Anastasia Stulova, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr, "A high-level virtual platform for early mpsoc software development," in *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, New York, NY, USA, 2009, CODES+ISSS '09, pp. 11–20, ACM.
[12] T. Schonwald, J. Zimmermann, O. Bringmann, and W. Rosenstiel, "Network-on-chip architecture exploration framework," in *Digital System Design, Architectures, Methods and Tools, 2009. DSD '09. 12th Euromicro Conference on*, aug. 2009, pp. 375 –382.
[13] SPIRIT Consortium, "IP-XACT," http://spiritconsortium.org.
[14] SimpleScalar LLC, "SimpleScalar Tool Suite," http://simplescalar.com.
[15] S.J. Filho, A. Aguiar, C.A. Marcon, and F.P. Hessel, "High-level estimation of execution time and energy consumption for fast homogeneous mpsocs prototyping," jun. 2008, pp. 27–33.
[16] Steve Rhoads, "Mips plasma," 09 2001.
[17] Gerry Kane, *MIPS RISC architecture*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.
[18] Fernando Moraes, Ney Calazans, Aline Mello, Leandro Möller, and Luciano Ost, "Hermes: an infrastructure for low area overhead packet-switching networks on chip," *Integr. VLSI J.*, vol. 38, no. 1, pp. 69–93, 2004.
[19] "Atlas," 2011, https://corfu.pucrs.br/redmine/projects/atlas. Último acesso em 07/06/2011.
[20] C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*, Springer-Verlag New York Inc, 2010.