A Spectrum of MPSoC Models for Design Space Exploration and Its Use

Carlos A. Petry, Eduardo W. Wächter, Guilherme M. de Castilhos, Fernando G. Moraes and Ney L. V. Calazans

FACIN-Faculdade de Informática, PUCRS-Pontificia Universidade Católica
Porto Alegre, RS - Brazil
{carlos.petry,eduardo.wachter,guilherme.castilhos}@acad.pucrs.br,
{fernando.moraes,ney.calazans}@pucrs.br

***Abstract*— Implementing on-chip multiprocessors is enabled by the use of deep submicron technologies and constitutes today a daunting task, due to the complexity of their design and verification. The development of such devices can be facilitated by the use of a carefully crafted set of models for each implementation step. This paper proposes the use of such a set of abstract models at several levels. These improve simulation speed and observability in one sense and level of detail and precision in the opposite sense. Initial development tasks such as software development and functionality specification refinement can evolve fast with very abstract models, while confidence in the final implementation can be achieved with lower level models. Our basic multi-processor system on a chip is configurable in several parameters, including number of processors, type of employed communication architecture and combination of abstraction levels used in the description of the distinct modules that compose the model.**

***Keywords* — MPSoC, Verification, Abstract Model, NoCs.**

I. INTRODUCTION

Multi-Processor Systems on a Chip (MPSoCs) are complex architectures, composed by processors, memories and other specialized intellectual property cores (IP cores, or just IPs). Here, we call each of these isolated modules a processing element or PE. PEs need to be interconnected by an intrachip communication architecture, usually today in the form of a network on chip or NoC [1][2]. Electronic design automation (EDA) tools and frameworks to customize such architectures are mandatorily used, given the huge design space to explore.

MPSoCs are increasingly popular in embedded systems. Due to their complexity, a framework targeting MPSoC customization must provide abstract models to enable fast design space exploration, flexible application mapping strategies, all coupled to features to evaluate the performance of running applications.

While the number of PEs in an MPSoC is kept small, say below a dozen, each PE may be individually designed or chosen to fit performance and power requirements. However, the cardinality of PEs is expected to grow fast in the next few years and we already see commercial systems counting as much as 100 PEs [3]. Thus, scalable MPSoC design techniques are expected to be increasingly used. Homogeneous multiprocessing, or at least locally homogenous multiprocessing is expected to find larger acceptance, encompassing embedded systems domains. Processors are becoming parameterizable commodities, and often a processor choice supersedes processor design during product development. Each processor comes with a set of models describing its behavior and performance figures at several abstraction levels and system development is expected to be able to use these models during design and verification activities.

978-1-4673-2789-3/12/$31.00 ©2012 IEEE

This work approaches the use of a set of models that describe a whole MPSoC at several abstraction levels. Models of processors, memories, network interfaces, NoCs and software applications can be combined using an integration framework, which is capable generating a complete executable MPSoC model. Design elaboration, simulation of real and synthetic applications and debugging may be achieved directly from the integrating environment in most cases. The spectrum of model choices lead to simulation times that vary as much as several thousand times from the low end, high precision models to the high end high speed models.

The rest of this work is divided into seven Sections. Section II briefly explores some related work. Section III discusses the reference MPSoC and its generation environment, while Sections IV, V and VI cover the modeling of NoCs, PEs and the whole MPSoC. Section VII discusses simulation results for a subset of selected MPSoC models built using specific PE components and NoC modules models. Finally, Section 0 presents a set of conclusions and direction for future work.

II. RELATED WORK

The amount of references in current literature on MPSoC hardware and software modeling is far too large to allow an encompassing view of the subject area here. Thus we keep the focus here in some works that explore representative fields where much research and development work is currently produced. These fields are efficient software and hardware modeling for achieving high performance executable descriptions, generation of abstract transaction level models of MPSoCs and techniques for accelerating the simulation of executable MPSoC models.

Pétrot et al. [4] explore a set of techniques to build executable models able to run software applications in transaction level models (TLM) of MPSoCs with variable degrees of efficiency. The explored techniques are mostly based on simulation of software and how to execute the

software on a given TLM model. The techniques are classified in three main groups: instruction-accurate interpretation (the slowest), dynamic binary translation and native code execution (the fastest). The faster the technique the higher is the overhead for producing the software execution environment. An interesting point of contact with the work described here is that our highest abstraction level descriptions employ the slowest technique (instruction-accurate interpretation using instruction set simulators). This ensures that our results may be enhanced according to these authors by orders of magnitude, if more sophisticated software execution schemes are employed.

Abdi et al. [5] propose an environment able to generate TLMs from a task level specification of applications. Although automatic generation is interesting for specific contexts, we see this as less important than smoothly combining pre-existing models of various modules at distinct abstraction levels. Their TLM generation is naturally limited in what models it produces. The imposed restrictions are that only homogenous multiprocessing is supported and only static mapping of tasks is possible, which is quite restraining, even for current MPSoCs.

Chen et al. [6] focus on the use of multicore host systems to validate MPSoCs and propose to accelerate simulation of TLMs by optimizing simulation kernels to operate using multiple simulation threads. The use of such multiple threads requires the solution of complex TLM module synchronization issues. This complexity accounts for only moderate gains in simulation performance. For example, for a 4-core host the theoretical maximum speedup is 2.05 and in practical experiments the maximum achieved speedup was only 1.74.

III. The HeMPS MPSoC and its Generation Environment

Our reference MPSoC, called HeMPS [7], is currently a heterogeneous multiprocessing NoC-based MPSoC platform. Figure 1 presents a HeMPS homogeneous multiprocessing instance using a 2x3 2D mesh NoC. The main hardware components are an instance of a HERMES NoC family member router [8] and a PE built around the mostly-MIPS processor Plasma [10]. Plasma-IP is the name of the processing element, which wraps each Plasma processor and attaches it to the NoC. The PE also contains a private memory (RAM), a network interface (NI), and a DMA module.

Typical applications running in MPSoCs, such as multimedia and networking, often present a dynamic workload. This implies a varying number of tasks running simultaneously, and their number or load often exceeds the available resources. To tackle this issue, HeMPS assumes that: (*i*) applications are modeled using task graphs; (*ii*) only a subset of tasks is initially loaded into the system.

Remaining tasks are stored in an external memory, named task repository. This memory keeps all task codes necessary at any moment of the applications' execution.

The network interface (NI) adapts the processor communication protocol to the communication protocol expected by the NoC. For example, if we instantiate a NoC with 16-bit flits, since the Plasma processor deals with 32-bit words, it is necessary to serialize and de-serialize data passing between the processor and the NoC.

One of the processors is defined as Master, while all other are slaves. Application tasks run only on slaves. The Master performs task allocation (dynamic and/or static) and interacts with an external Task Repository and with peripherals (not shown in Figure 1).

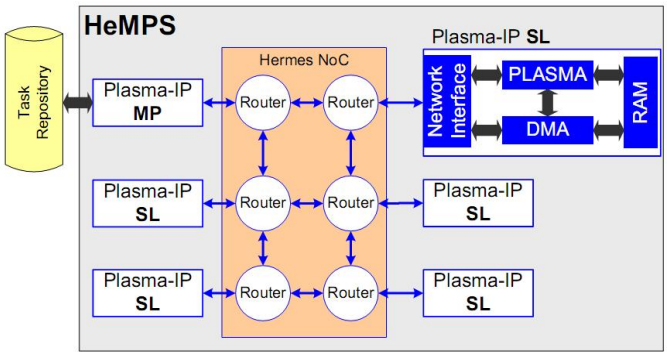

Figure 1 – MPSoC HeMPS platform topology [7].

To produce instance of HeMPs, we developed a tool called HeMPS Generator. The main GUI window of this tool appears in Figure 2. This tool allows performing a set of integrating tasks, including:

- Define the target MPSoC dimension by choosing the number and disposition of PEs;
- Select among the available processors one for each PE;
- Position of the Master Processor inside the topology;
- Select parameters of the PE internal memory, like page size and total memory size;
- Select a set of applications to run on the MPSoC. In Figure 2 two applications (MPEG4 and communication) are defined by multiple tasks;
- Select a static, dynamic or mixed mapping schedule for tasks. In Figure 2, drag and drop operations to specific PEs or to each of the processor external repositories define initial static and dynamically mapped tasks, respectively. Dynamically mapped tasks are loaded on demand, when some communication with them is requested by some task running on some PEs;
- Select among several NoC interconnects. In Figure 2 it is possible to see the current support to the Hermes NoC and two versions of the Hermes-GLP [9]. This is a low power version of Hermes, supporting the GALS paradigm and where communication flows may be attached priorities;
- Select the abstraction level of description for the processor and NoC.

Given the user parameterization, the environment may then be used to produce an executable model of the MPSoC, compile all application tasks, insert an SO microkernel in each processor and elaborating the whole hardware-software structure of the executable model.

After compilation and elaboration, still inside the environment it is possible to run a simulation of the system and use processor windows to receive debugging information. Note that the environment assumes PEs, NoCs and their interfaces were already validated.

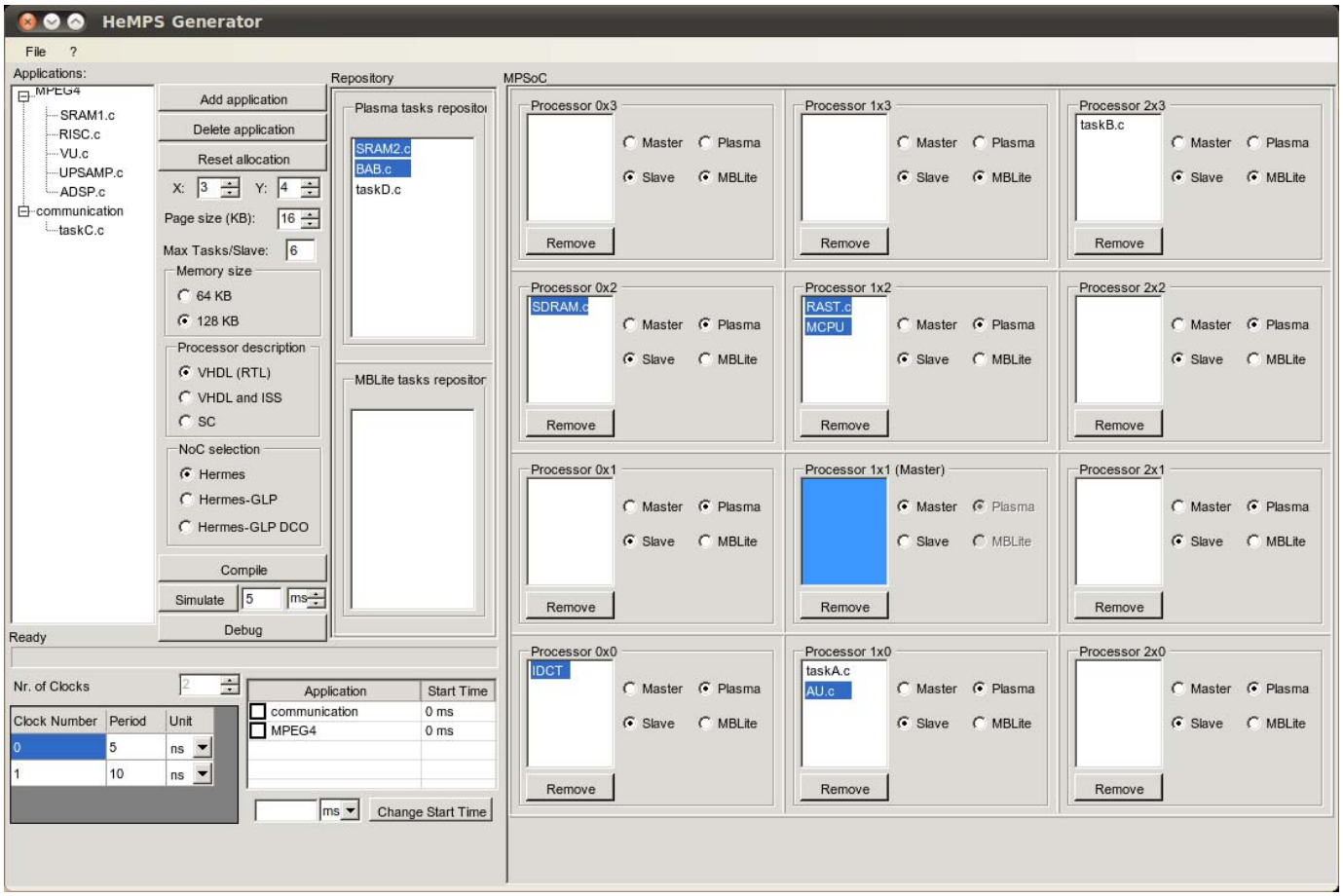

Figure 2 – The main GUI window of the HeMPS Generator integrating tool. Two applications are partially mapped in the 3x4, 12 processor platform. The top three PEs employ an MBLite processor, while the other PEs employ the Plasma processor. Highlighted tasks belong to the MPEG4 application, while the others belong to the communication application. Three tasks are selected for dynamic mapping in Plasma processors from both applications.

A. Synthesizable VHDL

The first version of the HeMPS MPSoC was prototyped in Xilinx FPGAs. Currently, FPGA and ASIC versions are available. The ASIC version has been implemented in STMicroelectronis 65nm technology and is currently being prepared for ASIC prototyping. Irrespective of the target implementation technology, all modules were described at the register transfer level (RTL) in VHDL. Memory blocks for example need to be personalized for the target technology. In Xilinx FPGAs we use multiples Block RAMs for efficiency, and for ASIC we employ automated memory generators.

Looking for faster simulation time, memory arrays can be modeled in a clock cycle accurate bit accurate (CABA) SystemC description. Processors were modeled using instruction cycle accurate instruction set simulators (ISSs) in C and embedded in SystemC wrappers to interact with the hardware models. Compared to an approach with all modules described in VHDL RTL, the speed up using ISS/memory models reduced the total simulation time up to 91%.

B. Validation

The validation process at the RTL level is performed using a commercial RTL simulator, such as Mentor Modelsim or Cadence Incisive coupled to adequate verifications models such as the U-model and/or assertion-based verification.

IV. NoC Modeling

A. Synthesizable VHDL

The Hermes NoC is a minimalist NoC using conventional 2D Mesh topology, packet switching, input buffering and wormhole mode. Centralized arbitration and routing allow the implementation of a low area overhead router in VHDL. The router overhead is dominated by the input buffers whose size has a strong impact on performance and power. More details on the structure of the Hermes router and NoC and its RTL implementation can be found in [8].

B. RTL SystemC

This model has the same structure as the VHDL RTL model. Each one of the VHDL modules is simply rewritten in RTL SystemC. This way, the two approaches (the VHDL and SystemC) are CABA models. This approach does not take advantage for example of SystemC language structures such as *sc_fifo*, for example. The code in Figure 3 shows one example code of the RTL SystemC.

```
void fifo::in_proc(){
    if(reset_n.read()==false){
        last.write(0);
        for(int i=0;i<FIFO_SIZE;i++) buffer_in[i]=0;
    }
    else{
        if((avail_space.read()==true) && (rx.read()==true)){
            buffer_in[last.read()] = data_in.read();
            if(last.read()==(FIFO_SIZE - 1))
                last.write(0);
            else
                last.write((last.read() + 1));
        }
    }
}
```

Figure 3 - Example of an RTL SystemC FIFO of the NoC buffer.

The debugging process at this level can proceed in two ways: (*i*) generating a pre-compiled executable file of HeMPS or (*ii*) using a commercial RTL simulator, such as Modelsim. Using Modelsim implies higher simulation times, compared to the executable file approach.

C. Untimed SystemC

This version of the NoC module was developed using the Synopsys System Studio high level modeling environment, as illustrated in Figure 4. This is simply a functionally equivalent description of the previous models, fully described in untimed SystemC. Routers are modeled by two types of external hierarchical channels: (*i*) those used among routers, interconnecting neighboring routers or a router to a PE and (*ii*) those inside the router, to perform arbitration and routing between ports within the router.

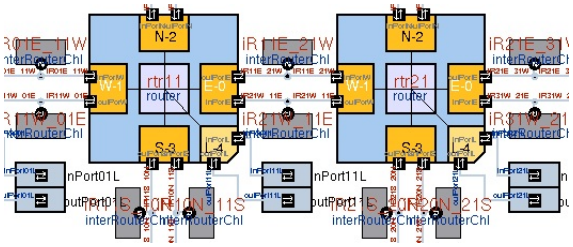

Figure 4 - Schematic view of the Hermes NoC router module.

Inside each router there are 4 ports, as Figure 5 shows, for communication with ports of neighbor routers and one for the local module. There is also the *intraRouter* module that performs the arbitration and routing between router ports.

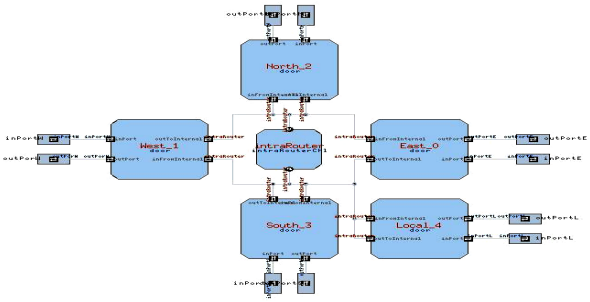

Figure 5 - Ports of communications inside of a Router module.

The debugging process can be made in two ways: (*i*) evaluating logs generated by the executable file of the NoC module or (*ii*) stepping the execution using simple debugger software, such as GNU *gdb*. With *gdb*, the source code can be traced step by step along the executable code, providing a way to evaluate the behavior and correctness of the NoC code.

V. PE Modeling

A. RTL VHDL

All modules are described in synthesizable VHDL.

B. SystemC ISS + RTL VHDL

Processors are modeled using cycle accurate instruction set simulators (ISSs) wrapped in SystemC modules. Compared to an approach with all modules described in VHDL RTL, the speed up using ISS/RAM models reduced the total simulation time up to 91% [7].

C. SystemC ISS + VHDL behavioral

Looking for faster simulation time, the PE memory was modeled using a behavioral SystemC description.

D. SystemC ISS + SystemC cycle accurate

In this model, the NI and the DMA are modeled as a SystemC clock cycle accurate description. The processor remains modeled by the ISS and the PE memory in behavioral SystemC.

E. Untimed SystemC

The PE is a module composed by: (*i*) Plasma processor (*MLite)*; (*ii*) *DMA* controller; (*iii*) *NI* network interface; (*iv*) *CP0* co-processor, which has registers used to control memory paging and system exceptions; (*v*) *UART* message handler - a serial interface used to handle messages sent by slave processors; (*vi*) *RAM* – the local memory and (*vii*) *MemExt* module that contains all tasks binary codes, statically allocated, present only at the master processor. Figure 6 shows the schematic view of the whole PE. All modules are written in untimed functional SystemC.

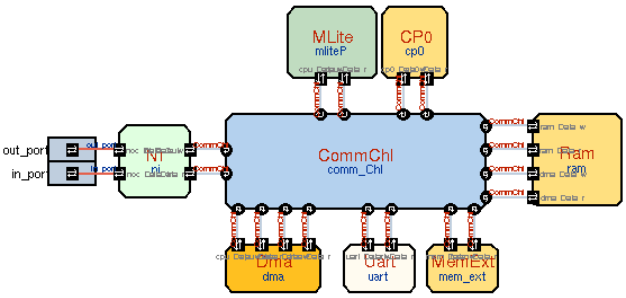

Figure 6 - Schematic view of Plasma PE module.

The communication between modules is provided by *CommChl*, a SystemC channel that acts as interface between all PE modules. Communication occurs by means of method calls. The communication mechanism uses memory-mapped addresses. Figure 7 shows part of the communication mechanism code between the *Plasma* e *NI* modules. The *NI* is accessible by address WRAPPER_SEND (0x20000130).

```
switch(address) {
    . . .
    case WRAPPER_SEND:
        cpu_Data_w->cpu_write_ni(address, value, &st_write);
```

Comm_Chl
comm_Chl

```
ni_Data_w->ni_request_write(&addr, &data_w);
    switch (addr) {
        . . .
        case WRAPPER_SEND:
            switch (state_write) {
            case TARGET_STATE:
                data_out.word=data_w;   sent=false;
                while(!sent) {                  //Stay trying to send
                    flit = data_out.flit[LOW];  //Sends the lower's word part
                    noc_Data_Out->try2SendFlit(flit, &sent);
                    if(!sent)
                        wait(1,SC_NS);
                }
```

Figure 7 - Example of communication mechanism between PEs.

VI. MPSoC Modeling

Table 1 presents the different models available for describing each PE and NoC, together with the specification of readily available model combinations allowed by our platform. Note that PEs are in fact formed by four distinct modules, for which different combinations of abstract descriptions are available, producing the five rows of Table 1. The hard to obtain models derive from the use of a specific tool to produce untimed models (Synopsys System Studio), making models thus obtained not easily exportable to other vendor's environments.

Table 1 – Levels of abstraction implemented for NoC and PE. √ stands for readily available models, while Easy corresponds to combinations that are straightforward to produce and Hard stands for combinations that today require significant modeling effort.

NOC→ PE↓	VHDL	SC Cycle Accurate	SC Untimed
VHDL	√ (M1)	Easy	Hard
ISS + VHDL Synthesizable	√ (M2)	Easy	Hard
ISS + VHDL Behavioral	√ (M3)	Easy	Hard
ISS + Cycle Accurate SystemC	Easy	√ (M4)	Hard
SC ISS Untimed	Hard	Hard	√ (M5)

Table 2 presents a brief comparison of the available models. The main advantages of a VHDL model (M1) is that it is synthesizable and from it accurate measurements of area, operating frequency and power dissipation can be obtained. The main disadvantage is obviously its prohibitively long simulation times. The ISS+VHDL models (M2 and M3) reduce simulation time, showing the same results (for debugging and latency), without the results of area, frequency and power consumption. SystemC cycle accurate models (M4) produce improvements compared to the VHDL model. The main advantage of this model is the reduction of the simulation time, with the same results that VHDL+ISS model. The SystemC untimed model (M5) shows the best results in terms of simulation time.

Table 2 – Qualitative comparison of models. M*x* stands for a given model presented in Table 1.

Model→ Parameter↓	M1	M2/M3	M4	M5
Synthesizable	Yes	No	No	No
Precise Area, Frequency, Power	Yes	No	No	No
Simulation Time	VERY HIGH	HIGH	MEDIUM	FAST
Latency Values	Yes	Approximate	Approximate	None
Accuracy	Real, in clock cycles	Approximate	Real, in clock cycles	Functionality only
Precision	CABA	Locally CABA	CABA	Transaction Level

A. The SystemC untimed MPSoC modeling case study

This Section describes the integration of the NoC Hermes with Plasma PE to generate the MPSoC HeMPS using the SystemC untimed model (M5). Figure 8 shows the hierarchical organization of the modules that compose the HeMPS MPSoC.

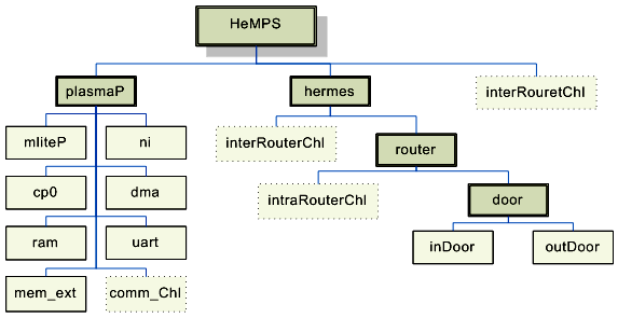

Figure 8 - Hierarchical organization of HeMPS.

Each of three main modules of Figure 8, *plasmaP* processor, *Hermes* NoC and *interRouterChl*, are used to generate the HeMPS MPSoC. Figure 9 shows the interconnection of these three modules, where the upper boxes of each module represent routers of the NoC.

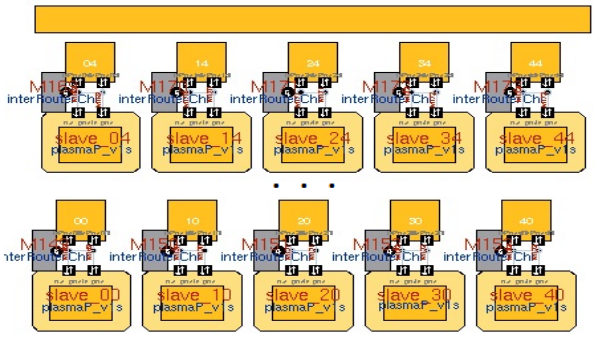

Figure 9 – Partial schematic view of the HeMPS MPSoC 5x5.

Inside the *plasmaP* module, the NI module is the interface responsible for communications between NoC and PE. The interconnection of *NI* and *Router* occur through SystemC ports using a FIFO queue for data coming from NoC and a register to send data to the NoC. Figure 10 shows the structure used to implement the communication between both modules. The depth of the FIFO is configurable.

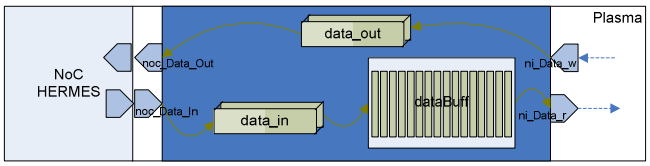

Figure 10 - Structure of the network interface (NI), used to implement communication between NoC and PE.

VII. EXPERIMENTS AND RESULTS

We build a test scenario that exercises and compares four model combinations. The abstraction characteristics of each of these models appear in Table 3.

Table 3 – Characteristics of the four chosen models.

Model→ Module↓	Synthesizable VHDL (M1)	ISS + VHDL (M2)	Cycle-accurate SystemC (M4)	Untimed SystemC (M5)
Router	RTL VHDL	RTL VHDL	RTL SystemC	Untimed SytemC
NI	RTL VHDL	RTL VHDL	RTL SystemC	Untimed SytemC
DMA	RTL VHDL	RTL VHDL	RTL SystemC	Untimed SytemC
RAM	RTL VHDL	RTL SystemC	RTL SystemC	Untimed SytemC
PE	RTL VHDL	SystemC+ISS	SystemC+ISS	SytemC+ISS

For several combinations of MPSoC sizes we run simulations of multiple combinations of a single synthetic application composed by six tasks, depicted in Figure 11.

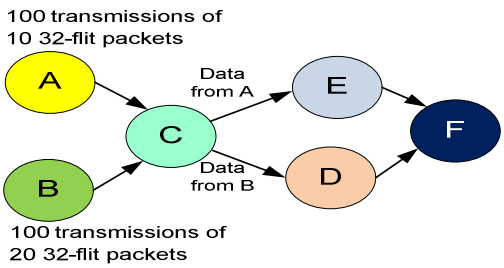

Figure 11- Group of 6 tasks used as application in the test scenario.

We run experiments in a HeMPS MPSoC with four distinct sizes: 4x4, 5x5, 7x7 and 10x10, on which the application shown in Figure 11 is statically mapped. Each PE is configured to have its internal memory divided into three pages, one containing the OS microkernel and the two other reserved for application tasks. The MPSoC was modeled in the four different abstract level combinations mentioned previously. Each line of Table 4 is identified by the size of the experiment and one suffix. When this suffix is _1, this means that only one instance of the application was present in the MPSoC during simulation. When the suffix is _25, _50, _75 or _100, this corresponds to a certain number of instances of the task such that near 25%, 50%, 75% or 100% of the MPSoC application pages of all processors are occupied by tasks of some instance of the Figure 11 application.

Table 4 shows the simulation time results obtained, in seconds, and the speedup obtained. The untimed model is not yet coupled to the environment capable to generate the random number of copies of the application. Thus, its runs are limited to versions of the MPSoC running only a single application.

Table 4 – Simulation time (in seconds), considering various levels of abstraction. Speedup compares the fastest untimed simulation to the slowest feasible simulation. NA=Not Available, due to excessive runtime, or no time to produce the results. Simulations run on a 6-core, 64 bits Xeon architecture with 12 Gbytes of RAM, running Linux OS.

Model Scenario	VHDL	ISS + VHDL	Cycle-accurate SystemC	Untimed SystemC	Untimed Speedup
4x4_1	3852.34	217.35	34.99	3.27	1178.1
4x4_25	4219.99	220.17	35.26	3.46	1219.7
4x4_50	4701.47	231.67	37.97	NA	NA
4x4_75	5162.13	245.78	40.56	NA	NA
4x4_100	7288.23	291.73	46.86	NA	NA
5x5_1	7742.04	336.25	53.22	3.85	2010.9
5x5_25	8134.28	359.32	60.83	4.18	1946.0
5x5_50	9087.88	423.85	65.76	NA	NA
5x5_75	12205.26	429.10	82.84	NA	NA
5x5_100	12460.31	466.63	85.45	NA	NA
7x7_1	19765.50	682.96	114.18	8.96	2206.0
7x7_25	21797.14	724.61	129.06	11.04	1974.4
7x7_50	32153.21	1049.32	179.42	NA	NA
7x7_75	41211.06	1272.44	226.46	NA	NA
7x7_100	50557.62	1538.20	274.07	NA	NA
10x10_1	50862.91	2344.64	289.52	39.92	1274.1
10x10_25	NA	4319.71	508.97	NA	NA
10x10_50	NA	5686.91	668.76	NA	NA
10x10_75	NA	7435.22	945.25	NA	NA
10x10_100	NA	9897.06	1122.85	NA	NA

The results in Table 4 show how a wide spectrum of models may help during the development of complex MPSoCs. Roughly each time we follow the path to more abstract models we gain an order of magnitude improvement in simulation time. Let us take for example the 10x10_1 case, where a pure VHDL model needs around 14 hours to simulate. The ISS+VHDL model, reduces this duration 21 times. Going up to the Cycle-accurate SystemC model we get an additional speedup of 8, while the use of the Untimed SystemC model furnishes another speedup of 7.5. The overall speedup appears in the last column of Table 4 and amounts to exactly 1274.1.

VIII. Conclusion and Future Works

This paper presented an environment and set of abstract models for developing MPSoCs. The spectrum of models enables to trade-off efficiency of simulation and precision of results producing enhancements of more than 2,000 times in some cases. Further work comprises the integration of other models of processors and NoCs to the environment, the increase of choices in selecting different combinations of models and improvement of simulation performance for untimed models using e.g. some techniques proposed in [4].

Acknowledgments

Fernando Moraes is supported by CNPq, FAPERGS, and CAPES, projects 301599/2009-2, 10/0814-9, 708/11, respectively. Ney Calazans also acknowledges the support of the CNPq and FAPERGS under grants 310864/2011-9 and 11/1445-0, respectively. All authors 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] L. Benini, G. De Micheli. *Networks on chips: a new SoC paradigm*, IEEE Computer. 35 (1), 2002, pp. 70–78.

[2] W. Dally, B. Towles, *Route packets, not wires: on-chip interconnection networks*, in: 38th Design Automation Conference (DAC'01), June 2001, pp. 684–689.

[3] Tilera, Inc. *TILE-Gx Processor Family*. Captured at http://www.tilera.com/products/processors/TILE-Gx_Family, 2012.

[4] F. Petrot; M, Gligor; M.-M. Hamayun; Shen Hao; N. Fournel; P. Gerin. *On MPSoC Software Execution at the Transaction Level*, IEEE Design & Test of Computers, 28(3), May/June 2011, pp. 32-43.

[5] S. Abdi; G. Schirner; H. Yonghyun; D. D. Gajski; Y. Lochi. *Automatic TLM Generation for Early Validation of Multicore Systems*, IEEE Design & Test of Computers, 28(3), May/June 2011, pp. 10-19.

[6] W. Chen; X. Han; R. Dömer. *Multicore Simulation of Transaction-Level Models Using the SoC Environment*, IEEE Design & Test of Computers, 28(3), May/June 2011, pp. 20-31.

[7] Carara, E.; Oliveira, R.; Calazans, N. L. V., Moraes, F. G. *HeMPS - a Framework for NoC-based MPSoC Generation*. In: 2013 IEEE International Symposium on Circuits and Systems (ISCAS'09), 2009, pp.1345-1348.

[8] Moraes, F. G; Calazans, N. L. V.; Mello, A. V. de; Möller, L. H.; Ost, L. C. *HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip*. Integration the VLSI Journal, 38(1), Oct. 2004, pp. 69-93.

[9] J. J. H. Pontes, M. T. Moreira; R. I, Soares; N'. L. V. Calazans. *Hermes-GLP: A GALS Network on Chip Router with Power Control Techniques*, In: IEEE Computer Society Annual Symposium on VLSI Design (ISVLSI'08), Montpellier, April 2008, pp. 347-352.

[10] OpenCores. *Plasma-most MIPS I (TM) opcodes: overview*. Captured on: http://opencores.org/project,plasma, Jul. 2010.

[11] T. Kranenburg; R. van Leuken. *MB-LITE: A robust, light-weight soft-core implementation of the MicroBlaze architecture*. In: Design, Automation, and Test in Europe Conference (DATE'10), 2010, pp. 997-1000.