Multi-level MPSoC Modeling for Reducing Software Development Cycle

Marcelo G. Mandelli[1,2], Felipe R. da Rosa[1,3], Luciano Ost[1], Gilles Sassatelli[1], Fernando G. Moraes[2]

[1] LIRMM – 161 rue Ada,
Cedex 05, 34095 Montpellier, France
{ost, sassatelli}@lirmm.fr

[2] FACIN - PUCRS – Av. Ipiranga 6681,
90619-900, Porto Alegre, Brazil
marcelo.mandelli@acad.pucrs.br, fernando.moraes@pucrs.br

[3] UFRGS – Av. Bento Gonçalves, 9500,
91501-970, Porto Alegre, Brazil
frdarosa@inf.ufrgs.br

Abstract - **Multiprocessor SoCs (MPSoCs) have rapidly evolved towards high-performance heterogeneous computing systems designed under performance, power efficiency and scalability concerns. Such systems accomplish billions of operations per second moving towards hundreds of processing elements that communicate through a network-on-chip. The hardware and software complexity of such systems is increasing dramatically, resulting in new design challenges, such as providing scalable modeling facilities and verification for both hardware and software. This work proposes a multi-level design approach for MPSoCs, targeting the reduction of software development cycle. The paper also presents different scenarios for exploration purposes, showing the benefits in term of design space exploration for to the proposed environment.**

Keywords: modeling, NoC-based MPSoCs, design space exploration of MPSoCs, dynamic mapping.

I. Introduction

Software development becomes an important issue in today's MPSoC design, requiring the development of application software in parallel with hardware enabling earlier testing and validation, which reduces time-to-market and development costs. Challenges in MPSoC software development comprise: (*i*) inter-processor communication protocol stacks definition; (*ii*) OS porting and analysis; (*iii*) exploration of better programming model facilities to address parallel programming [1]; (*iv*) drivers development [2]; (*v*) application software portability for heterogeneous multiprocessing hardware; (*vi*) simultaneous multithreading application debugging, among others.

Embedded software engineers must be able to verify the software development, while meeting the expected functionality and performance objectives of the platform. To design and verify large and MPSoCs, modeling and debugging facilities become crucial to guarantee the adequate design exploration support. Such facilities must be combined into a flexible and scalable framework where a designer can set up large scenarios that can be easily extrapolated, aiming at meeting system architecture and application requirements.

In this context, virtual platforms have been employed to achieve concomitant hardware and software development, while providing more flexibility and debugability. The *main contribution* of this paper lies in the modeling of a NoC-based MPSoC, previously validated in FPGA prototyping, into the open virtual platform (OVP) [3]. The considered MPSoC model complements a multi-level environment that unifies hardware and software development, covering design steps such as application development, platform configuration, and code generation in the same environment. Summarizing, this paper contributes in the following aspects: (*i*) integration of a clock-cycle accurate SystemC NoC model with instruction-accurate CPU models; (*ii*) debugging support software development; (*iii*) environment validation by using different scenarios, while comparing these to RTL MPSoC implementations.

II. State of the Art

State-of-the-art in software development and evaluation platforms includes full-system simulators. Full-system simulators are virtual platforms that emulate hardware behavior (e.g. CPU microarchitecture), making target software believe that it is running on a real physical hardware. While accelerating the software development of MPSoCs, such simulators usually offer a set of CPU models and memory system models, allowing the analyses of executing different application/OSs onto multiprocessor architectures without modifications. Examples of such simulators are Simics [4], PTLsim [5], SimpleScalar [6], GEM5 [7] and OVPSim [3]. Except PTLSim that only supports x86, all reviewed simulators offer at least five processor architectures. For instance, Simics from WindRiver supports Alpha, ARM, MIPS, PowerPC, SPARC and x86 models. While Simics and OVPSim are respectively functionally-accurate and instruction-accurate, the remaining simulators are quasi-cycle-accurate.

Cycle-accurate simulators target microarchitecture exploration since specific modeling details as the pipeline implementation and cache coherence protocols [7] are provided. However, these simulators are not scalable to a large number of processors, specifically when it comes to simulation speed and debugging usability, which is the main direction of our research.

This scenario points to the use of OVPSim since it covers our main requirements: (*i*) large number of processor architectures supported; (*ii*) scalable and acceptable simulation time (around hundred of MIPS); (*iii*) open source license; (*iv*) component-oriented infrastructure; (*v*) active development support. Nevertheless, OVPsim does not model cycle-accurate processors but rather instruction accurate processors, which provides inaccurate application execution time. Another limitation inherent to OVPSim is the fact that only bus-based architecture is available in the original distribution.

Thus, the present work promotes a hybrid environment placed between SystemC-based only approaches and pure

OVPSim virtual platform. The resulting unification results into an efficient reuse of CPU models and debugging features inherited from the OVPSim, integrated with inter-CPU communication performance parameters obtained from SystemC-based NoC model modeling.

III. Multi-level Design Structure

The multi-level modeling environment, illustrated in Fig. 1, comprises three layers: (*i*) RTL VHDL implementation; (*ii*) SystemC-based ISS; (*iii*) the proposed SystemC/OVP model. The two bottom RTL platform models provide clock-cycle accurate results, but restrict the software development, as well as the evaluation of adaptive management techniques (as application mapping and migration). Thus, flexible and simple models, such as the proposed OVP-SystemC models are required to accelerate the *software development* due to the improved debugging facilities and fast simulation compared to lower level models.

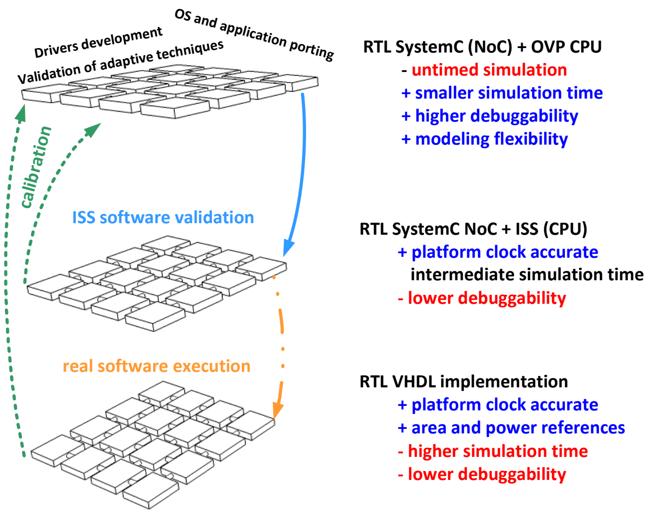
OS and application porting
Drivers development
Validation of adaptive techniques
calibration
RTL SystemC (NoC) + OVP CPU
- untimed simulation
+ smaller simulation time
+ higher debuggability
+ modeling flexibility
ISS software validation
RTL SystemC NoC + ISS (CPU)
+ platform clock accurate
intermediate simulation time
- lower debuggability
real software execution
RTL VHDL implementation
+ platform clock accurate
+ area and power references
- higher simulation time
- lower debuggability

Fig. 1 - Multi-level design structure.

In this direction, we claim that software engineers can easily implement/port C applications and operating systems, execute them until the point where no occurrence of errors is observed. Thus, applications are executed into the SystemC-based ISS to examine the detailed software behavior. Finally, they are executed into an RTL platform where performance and power estimation can be accurately obtained.

A. *Reference MPSoC Models*

Two clock-cycle accurate implementations of the NoC-based MPSoC platform are supported. These correspond to the RTL VHDL and RTL SystemC in Fig. 1. Both adopt a 2D-mesh NoC as interconnection communication infrastructure. Each PE connected to the NoC router includes: (*i*) NI (Network Interface), (*ii*) a 32-bit Plasma processor (MIPS-like architecture); (*iii*) a local memory; (*iv*) a DMA module.

Designers can define one or more PEs to act as manager(s) of the system. Such managers control the MPSoC behavior by executing, for instance, dynamic task mapping heuristics [8]. Each slave PE runs a *microkernel*, which is responsible for executing application tasks.

B. *SystemC/OVP model*

OVP [3] is composed of three main components: (*i*) APIs that enable modeling in C of hardware components; (*ii*) library of free open-source CPU and peripheral models; (*iii*) the OVPsim simulator. OVPsim is a dynamically linked library from Imperas, which supports the simulation of bus-based multiprocessor platforms. OVPSim relies on dynamic binary translation that increases simulation speed [3].

To overcome the restriction of using only bus-based platforms, a clock-cycle accurate SystemC NoC model was included in the OVP components' library. One could argue that a simple crossbar to interconnect CPUs at higher abstraction levels would be sufficient to develop and to validate applications and operating systems, since a crossbar supports parallel transactions between CPUs. This work advocates that integrating untimed CPU models with an accurate NoC model brings up the following benefits:

i. communication volume at each link – enabling to compute the power spent in the communication infrastructure as a function of the data volume and number of hops [9];
ii. mapping quality – using the hop number between tasks and the communication volume it is possible to evaluate different mapping heuristics;
iii. drivers development at higher abstraction levels: the NoC model use wires instead of TLM transactors, enabling to develop the required drivers.

The complex process of integrating distinct CPUs in NoC-based MPSoC platforms limits the implementation and the exploration of multiprocessor systems. For instance, the implementation of network interfaces is time consuming, which requires designer knowledge in terms of HW/SW implementation and protocols definition. In this sense, one important feature of the proposed modeling is the easiness of integrating different CPU models (as illustrated in Fig. 1 – modeling flexibility), allowing the development of heterogeneous MPSoCs. This paper puts focus on modeling and software development.

The main drawback of SystemC/OVP model is the fact that OVP is instruction accurate, thus packets are injected at inaccurate time instants compared to clock-cycle models.

Fig. 2 details the architecture of the SystemC/OVP PE. The numbers in the Figure correspond to a packet reception, and its processing by the OVP CPU model. In the first step, the NI receives a packet from the router (step 1). An event is triggered notifying the receiver module (block inside of the *SystemC-OVP interface*) about the incoming packet. The *receiver module* then reads the incoming packet, stores it into a buffer used to synchronize the communication between the untimed CPU and the clock-cycle NI (step 2). After storing a complete packet, the module informs the CPU that there is data stored in the buffer. As shown in step 3, there are two ways to inform a CPU about incoming data:

- for a slave CPU (CPU that executes user applications), an interrupt is raised and an ISR (Interrupt Service Routine) is called to read data.
- for a manager CPU, a memory mapped register is used to alert about the stored data. This CPU polls this register periodically. Once the CPU is ready, the data is read.

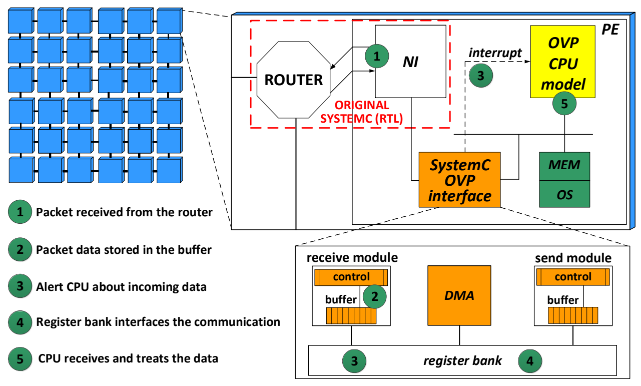
PE
ROUTER
NI
interrupt
OVP CPU model
ORIGINAL SYSTEMC (RTL)
SystemC OVP interface
MEM
OS
1 Packet received from the router
2 Packet data stored in the buffer
3 Alert CPU about incoming data
4 Register bank interfaces the communication
5 CPU receives and treats the data
receive module
control
buffer
DMA
send module
control
buffer
register bank

Fig. 2 - Integration of SystemC NoC model with OVP CPU model.

In both cases, the data embedded in the packet, is read through a DMA module using memory mapped registers (*register bank* in step 4). The register bank is implemented using an external memory, mapped in the processor address space, where each register has a pre-defined address. The CPU is connected to a bus to which all address-mapped components are connected. This bus connects the local memory and the register bank.

Fig. 3 shows the initialization of the register bank, with an address range from 0x00000000 to 0x0FFFFFFF (line 1). Once initialized, the *extMem* is connected to the processor bus in an address area (line 2), which is defined according to the adopted CPU model (in this case 0xF0000000 to 0xFFFFFFFF). This memory is also "connected" to the callback functions *regbankR* and *regbankW* (line 2).

```
1. extMem->init(0x00000000, 0x0fffffff);
2. proc->extMem(0xf0000000, 0xffffffff, regbankR, regbankW, extMem);
```

Fig. 3 - Example of register bank external memory initialization, and the connection to the processor bus

Callback functions are executed on every read (*regbankR*) or write (*regbankW*) access to the defined address area. In this case, when the processor accesses this area, the OVP simulator calls functions responsible to interconnect the SystemC to the OVP. Fig. 4 shows an example of a callback function related to a read memory access. The callback function name (*regbankR*) is defined as a parameter of the ICM_MEM_READ_FN macro.

```
1. ICM_MEM_READ_FN (regbankR)
2. {
3.      if(address = REG_ADDRESS1)
4.          value = system_c_signal1.read();
5. }
```

Fig. 4 - Example of a pseudo code for a read callback function.

The function parameter provides the memory address (*address*) accessed by the CPU, as well as the value (*value*) to be read from this address. Once a read memory access is triggered, the provided address is compared with the previously defined register address (line 3). The *value* is read by the processor (line 4) when the address is equal to *REG_ADDRESS1* (line 3). Note that the read value comes from a systemC signal (*system_c_signal1*), creating the communication between OVP and SystemC.

Fig. 5 gives an example of a write callback function. This function is specified using the ICM_MEM_WRITE_FN macro, and the callback function *regbankW* as a parameter. This parameter provides the memory address (*address*) accessed by a CPU, as well as the value (*value*) to be written in this address. When a write memory access is triggered, the provided address is compared to the previously defined register address (line 3). If the address matches REG_*ADDRESS1* (line 3), the *value* is written in a systemC signal, sending, for example, a read data request from the processor to the NI.

```
1.ICM_MEM_WRITE_FN (regbankR)
2. {
3.      if(address = REG_ADDRESS1)
4.          system_c_signal1.write(value);
5. }
```

Fig. 5 - Example of a pseudo code for a write callback function.

Finally, using the memory mapped registers as interface, the processor receives and processes data (step 5 in Fig. 2).

When the processor needs to send data through the NI these five steps are taken but using the *send module* (Fig. 2, inside *SystemC-OVP Interface*). First, the CPU uses the memory-mapped registers as an interface to the communication protocol with the NI. For each register access, a memory callback is triggered, generating a SystemC event. Then, the send module receives the packet and stores it in the buffer. When the packet is completely stored, it is sent through the NI.

IV. Results

This section evaluates quantitatively the simulation time of MPSoCs and the feasibility to explore the SystemC/OVP platform for software development. The software used as case study are mapping heuristics.

Qualitative measures include flexibility and debugability. Our approach benefits from the high debugability features supported in OVP, which provides a general view of each CPU model (e.g. registers, addressing, interrupts). Thus, software engineers can integrate the proposed OVP/SystemC with GDB or Eclipse, accessing their debugging functionalities. For instance, the engineer can execute an application in single step mode (i.e. step-by-step), insert code breakpoints, observe variables values, etc. It is also possible, to set watchdogs for accesses defined memory regions, obtaining the fetched instruction or the read/write values. Such important features for software development do not exist in the pure SystemC or VHDL platforms.

A. Simulation time

This section evaluates the simulation time for the SystemC and OVP/SystemC platforms. Results are evaluated varying: (*i*) platform size: 6x6 (36 PEs), 8x8 (64 PEs), 10x10 (100 PEs), 12x12 (144PEs), 14x14 (196 PEs) and 16x16 (256 PEs); (*ii*) resource occupation, i.e., number of PEs executing tasks: 30% and 50%

All scenarios execute one or more instances of a *Digital Time Warping* (DTW) application, with 10 tasks, which recognizes patterns measuring similarities between two sequences that may vary in time or speed.

Fig. 6 presents the simulation time for all scenarios. It is possible to observe a distinct behavior between the pure SystemC model compared to the SystemC/OVP model. Using SystemC only, the simulation time grows linearly for a load equal to 30% (r^2=0.998) and quadratically (r^2=0.999) for a load equal to 50%. On the other hand, for the SystemC/OVP model both graphs present a quadratically growing (r^2=0.998 for both graphs).

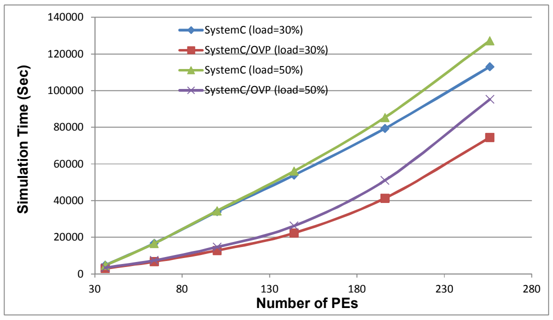

Fig. 6 - Simulation time, varying the platform modeling, number of PEs, and MPSoC load. Simulations setup – processor: Core 2 Duo E4400 2x2GHz; memory: 3GB; gcc version: 4.7.2; gcc flags: -mfpmath=sse -Ofast -flto -march=native -funroll-loops.

This behavior is due to the synchronization between the untimed OVP model with the cycle accurate SystemC. As the number of PEs increases, as well as the load applied to the system, the number of synchronization events also increases. To put in perspective the above results, Table 1 presents the speedup obtained using the SystemC/OVP model.

Table 1 –Normalized SystemC/OVP speedup vs pure SystemC.

Load/PEs	36	**64**	**100**	**144**	196	256
30%	1,59	**2,50**	**2,66**	**2,42**	1,92	1,52
50%	1,36	**2,25**	**2,34**	**2,14**	1,67	1,34

Speedup reported in Table 1 suggests that systems containing up to 144 PEs (12x12) benefit from a 2x speedup. Therefore, the OVP/SystemC combines advantages of OVP for software development with a cycle-accurate NoC model. The cycle-accurate NoC model enables to estimate the latency, throughput and the power consumed during communication [9].

B. *Software Development – Mapping Heuristic case study*

This section evaluates dynamic mapping heuristics, since it is the first action executed by the MPSoC manager when a given application is loaded into the system. The mapping heuristics evaluated are PREMAP-DN, LEC-DN, NN [8]. Six applications are used: MWD (12 tasks), AAV (8 tasks), MPEG4 (12 tasks), Synth (9 tasks), VOPD (12 tasks), SegImg (6 tasks). Applications are modeled synthetically, i.e., from the application graph it is obtained the communication volume between each communicating pair, and such behavior is modeled in C language, using send and receive MPI-like primitives.

This evaluation adopts a 6x6 MPSoC instance (1 manager PE and 35 slave PEs). Each slave PE may execute up to 2 tasks simultaneously. Therefore, the MPSoC can execute simultaneously up to 70 tasks. Three different scenarios are evaluated: (*i*) MWD, MPEG4 and AAV - 32 tasks; (*ii*) MWD, VOPD and Synth - 33 tasks; (*iii*) MPEG4, VOPD, MWD and SegImg - 42 tasks. Table 2 presents the communication volume transmitted through the NoC for each mapping heuristic, in thousands of flits.

Table 2 – Communication volume transmitted through the NoC for each mapping heuristic, in Kflits.

	PREMAP-DN		LEC-DN		NN	
	SC	SC/OVP	SC	SC/OVP	SC	SC/OVP
Scenario 1	214,7	203,0	262,4	255,5	236,6	310,9
Scenario 2	54,2	53,6	72,6	72,1	88,0	97,6
Scenario 3	140,3	145,5	97,5	105,0	108,7	143,2

The first two mapping heuristics, PREMAP-DN, LEC-DN, have in their cost function the communication volume, with NoC power consumption as main optimization objective. Comparing both models, the difference is less than 2%. Even though NN (nearest neighbor) heuristic reports a difference of around 25%, the ranking among the scenarios remains the same. The observed difference is due to a different injection rate in the network, since the abstract processor models cannot generate data with cycle-accuracy. Such results demonstrate that SystemC/OVP may be used at higher abstraction levels to develop MPSoC applications.

V. Conclusion

In this paper, we presented a multi-level design approach to provide modeling flexibility and debug-ability for MPSoCs. This approach consists in an integration of a RTL SystemC NoC in the OVP framework. The resulting approach offers faster and easier software development while enabling accurate communication analysis thanks to the integrated clock-accurate NoC.

The insights gained from the preliminary results call for improvements towards the following aspects: synchronization between OVP and SystemC model; adoption of lighter NoC models; inclusion of energy consumption models for both NoCs and CPUs; exploration of new techniques for online system management.

Acknowledgments

Fernando Moraes is supported by CNPq project 302625/2012-7, and CAPES project CAPES-COFECUB 708/11.

References

[1] Marongiu, A. and Benini, L.. *An OpenMP Compiler for Efficient Use of Distributed Scratchpad Memory in MPSoCs*. IEEE Transactions on Computers, vol. 62(1), 2012, 222-236.
[2] Gray, I.; Audsley, N.C., "Challenges in software development for multicore System-on-Chip development". In: RSP, 2012, pp.115-121.
[3] OVP 2013, Available at: www.ovpworld.org/technology_ovpsim.php
[4] Simics, Available at : www.windriver.com/products/simics
[5] Yourst M.T. "PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator". In: ISPASS, 2007, pp. 23–34.
[6] Austin T., Larson E. and Ernst D.; "SimpleScalar: An Infrastructure for Computer System Modeling". Computer, v.35(2), 2002, pp. 59–67.
[7] Binkert N.; at al. "The gem5 simulator". ACM SIGARCH Computer Architecture News, v.39 (2), 2011, 7p.
[8] Mandelli, M.; Amory, A.; Ost, L.; Moraes, F.; "Multi-task dynamic mapping onto NoC-based MPSoCs". In: SBCCI, 2011, pp. 191-196.
[9] Hu, J.; Marculescu, R. "Energy-aware mapping for tile-based NoC architectures under performance constraints". In: ASP-DAC, 2003, pp. 233-239.