

Automatic Generation of SystemC SMP Models for HW/SW Co-simulation

Patricia Botella, Pablo Sánchez, Héctor Posadas
Microelectronics Engineering Group
University of Cantabria
Santander, Spain
 {patrica, sánchez, posadash}@teisa.unican.es

Abstract – As the complexity of system-on-chip (SoC) design is increasing rapidly, design verification has been attracting more attention in recent years. Nowadays multiprocessor system-on-chips (MPSoCs) are becoming more commonly used in embedded systems and most of them have symmetric multiprocessor (SMP) architectures. In these architectures, two or more identical processors are connected to a single shared main memory and they are controlled by a single OS instance. Developing software for multiprocessor architectures is known to be complex and tedious, as it requires a combination of high-level programming environments with low level software design. Several verification techniques have been introduced to validate MPSoC-based embedded systems during the design process such as instruction set simulation (ISS), software emulation and native (host compiled) simulation.

In this paper a technique for generating a SystemC SMP model for native simulation is proposed. Native simulation is faster and less complex than ISS because the software source-code is directly compiled and annotated in the simulation host computer. This provides efficient and precise simulation models at the highest abstraction levels that are required to perform early design validations and architecture explorations. This work is based on SCoPE, a HW/SW native simulator.

Index Terms – symmetric multiprocessor, native simulation, co-design HW/SW, SystemC.

I. INTRODUCTION

Nowadays most embedded systems are composed of multiprocessors on chip (MPSoC). Several MPSoC architectures have been proposed, symmetric multiprocessors (SMP) being one of the most promising. It is expected that this type of architecture will have an important impact in the coming years.

Migrating one task among different processors is regarded as a key function of the operating system for symmetric multiprocessor based systems. This ability can balance the workload among different cores to reduce idle time and achieve higher overall performance.

In modern complex system on chip designs, software is gaining more and more importance as an integral part of the system. Given that embedded systems are composed of software and hardware components, HW/SW co-simulation

[1, 2] is a good alternative, and a powerful methodology for developing efficient software.

HW/SW co-simulation refers to verifying that the hardware and software work correctly together. This has traditionally been a task performed after the prototype hardware is available using different techniques. Nowadays, it is increasingly necessary to verify correct functionality before hardware is built.

There are several methodologies for HW/SW co-simulation. The most popular is based on an Instruction Set Simulator (ISS) [3], where the software application and the RTOS are compiled for the target platform. The binary code is executed on the ISS at target-processor instruction level. Despite the recent progresses in improving the speed of functional ISS cycle-accurate simulation it is still too slow. This limits the applicability of cycle-accurate simulators for performance evaluation of complex embedded applications.

To reduce ISSs simulation times, software emulation can be used. A virtual machine provides a complete platform model which supports the execution of the entire system. QEMU [4] is a fast machine emulator using an original portable dynamic translator that converts target to host instructions. It emulates several CPUs (x86, PowerPC, ARM and Sparc) on several hosts (x86, PowerPC, ARM, Sparc, Alpha and MIPS). It achieves good performance by using dynamic translation but this is not enough to verify real systems and the hardware integration is very complex.

The fastest methodology for HW/SW co-simulation is native simulation [1, 5, 6, 7], that consists on compiling the SW application for the simulation host instead of the target processor. The binary code is directly executed on the host processor. To obtain precise performance and energy estimations it is necessary to use a high level model of the HW platform.

During recent years, there has been a strong interest in the development of methodologies based on reuse of intellectual property cores (IP cores) [9]. An IP could be for example a SMP structure. These methodologies provide a way to enable reuse of parts from previous designs, in a changing highly competitive environment with very short time to market requirements.

In this paper we propose a technique that allows automatic generation of SystemC SMP models for native simulation. The resulting SMP model is integrated in the SCoPE framework [10], a HW/SW co-simulator. The aim of the work is to minimize design effort when integrating the SMP infrastructure. This enables performance and power consumption estimations to be obtained in the early stages of the design flow. Furthermore, an intuitive graphic user interface has been implemented. It enables all the parameters required to integrate the application SW in the SMP platform to be configured.

The rest of the paper is organized as follows. In Section 2 a brief introduction to the state of the art is presented. In Section 3 the symmetric multiprocessor model is explained. Section 4 presents the developed symmetric multiprocessor simulator SMPsim. In Section 5 the parametrization of the system components is explained. In Section 6 the automatic generation of the SystemC SMP model is described. Sections 7 and 8 present an example and state some conclusions.

II. STATE OF THE ART

The use of symmetric multiprocessor architectures is increasing as the complexity of the new MPSoCs and embedded software tends to inflate development times, contributing to a significant increase in the time to market. SMP systems allow any processor to work on any task no matter where the data for that task are located in memory. This enables tasks to be moved between processors in a simple way to balance the workload efficiently.

In [11], two SMP evaluation tools are described: Sled and Peppermint. *Sled* generates traces from applications running on IA-64 processors, while *Peppermint* models the complete system using cycle-accurate, trace-driven simulation.

MPARM [12] is a multi-processor cycle-accurate architectural simulator. Its purpose is the system-level analysis of architectures with different processors, interconnections, memory hierarchies and peripheral devices. This simulator works with ISS-based processor models and use SystemC to describe the system. No automatic generation of the multiprocessor system is done and the integration of new components must be done by hand.

QEMU is a fast and versatile emulation platform with support for numerous target architectures such as x86, ARM, SPARC, which has already been utilized in scientific experiments [13]. QEMU's performance is achieved with dynamic code translation. The general concept consists of translating every emulated target instruction into a set of native instructions which modify the emulated processor's data structure instead of real processor registers. The support for multiple target architectures is provided with a series of back ends for reading the target's instructions and emulating the results in the processor's state data structure. The unique feature of QEMU is the ability to emulate not only the whole system but also a single process. However, systems emulated by QEMU are about 5 to 20 times slower than native code execution [14].

PTLsim [15] simulates specifically x86-64 microprocessors at a configurable level of detail ranging from RTL-level models up to full-speed native execution on the host CPU. PTLsim obtains time estimations but not power consumption estimations.

Many of the recent simulation approaches [1, 5, 6, 7] are based on the native execution of the software to take advantage of considerable speed-ups. Two alternatives to use native simulation are described in [8]. The first one is the RTOS level approach where the simulator includes an abstract model of the RTOS, the host libraries and the communication libraries. Thus, only the application software and device drivers have to be linked to the simulator. The second one is the HAL level (Hardware Abstraction Layer) native simulation. Here all the layers (application, operating system, device drivers and libraries) are above the HAL Application Program Interface (API) that is implemented in the simulator. The software is encapsulated in a classic dynamic library and all the HAL APIs are exposed to OS and drivers. The results shown in [1] demonstrate that source-code estimation techniques constitute a sufficiently accurate technique that is much faster than ISS-based systems. This speed increase makes the technique optimal for fast system evaluation, satisfying the need to obtain effective metrics, which are necessary for efficient Design Space Exploration (DSE) frameworks.

SCoPE is a simulation framework based on SystemC and oriented to system modeling using native simulation. The execution of the application software is simulated in the host using abstract models of the processor and the RTOS. This approach provides timed modeling of the SW execution that can be integrated in a timed executable model of the entire HW/SW embedded system.

The work presented in this paper (SMPsim) enables the generation of a SystemC SMP model for native simulation. Using the SMPsim tool, all the system elements (processors, bus, memory, etc.) can be modeled easily. Moreover, a graphic interface has been implemented to facilitate the platform configuration.

III. MODELING SYMMETRIC MULTIPROCESSORS

The main goal of this work is to model symmetric multiprocessor architectures where two or more identical processors are connected to a single shared main memory and controlled by a single OS instance.

With the purpose of estimate execution times, the embedded microprocessor must be previously modeled and characterized. This model must be simple enough to allow fast simulations, in contrast to traditional ISS-based models that provide cycle or instruction accuracy.

In order to estimate the execution time, the framework integrates two techniques. The first technique associates an execution time to every source code statement and computes the total time in every segment. A segment is the code between two consecutive "wait" sentences. The estimation error of this approach may be high.

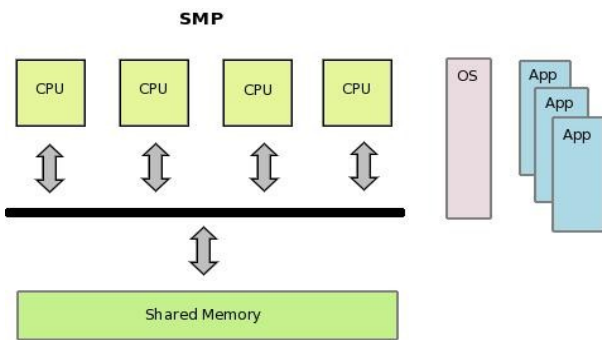


Figure 1. Symmetric multiprocessor architecture, SMP.

The second technique associates an execution time to every basic block [6]. In order to calculate the basic block execution time, an estimation of the number of target instruction is needed. This requires the target binary code to be obtained, but this can provide higher accuracy with a relatively low penalty in performance. This analysis is done in terms of number of instructions, so that execution time can be estimated multiplying by the average number of Cycles Per Instruction (CPI) of the processor. The information is back-annotated in the original source code in a similar way as in the previous technique based on statement cost.

The symmetric multiprocessors have a single memory space, which allows an operating system to distribute the tasks among multiple processors, or allows an application to obtain the required memory. However, this global memory contributes to one of the SMP's biggest problems: as the number of processors in the system increases, traffic on the memory bus became saturated. Adding cache to each processor can minimize this problem. In order to speed up the simulation, an instruction cache model that avoids search algorithms is used.

Using this model, no real caching of data is performed during execution, which makes the technique faster than other simulation-based approaches.

The cache is modeled as a two-dimensional array of pointers to cache line structures. The dimensions of the array depend on the physical characteristics of the cache to be considered. The caching mechanism consists of storing the addresses of the line structures in the array. An empty location in cache is modeled as a NULL value.

During simulation, when the execution flow reaches the end of a basic block, the annotation mechanism provides an extra functionality that determines if the cache lines associated to the instruction block are present in cache or not. The model also assigns energy costs for each cache access, allowing a fast power estimation of instructions cache misses.

The system bus is an intermediate element in the system communications. This model has to be able to control the system refinement process. The bus model that SCoPE uses is based on the TLM2 standard, which is described in [16]. In this bus model, communications are modeled considering complete payload transfers instead of word-by-word transfers. The bus

manages the simulation time considering bus bandwidth, packet priority and size. An arbiter is also defined to control the transfer scheduling. Transfers that are not scheduled are blocked in a queue. It is possible to replace the SCoPE bus model with another one described in SystemC.

Taking into account the processor and instruction cache models, we can generate a SystemC SMP description for native simulation. The SMP model will be part of a complete platform, in which new hardware and software IPs can be added easily.

The framework supports uCOS or POSIX-compliant operating systems. Thus, the application software could make use of POSIX or uCOS system calls. Using the SCoPE compiler [17] (scope-g++) every file will be compiled and annotated with the required data to obtain the performance estimations.

The complete system is shown in the Figure 2. It can be seen that the SMPsim is a separate component connected to a bus where other SystemC models, which describe components such as memory or I/O devices, are connected.

SMPsim includes multiple processors and the application software to be simulated. Those processors are modeled with the previously commented approaches, and each processor could use an instruction cache. Data caches are not supported in the actual version but they will be included on future releases.

IV. PARAMETERIZATION

Platforms are composed of a set of configurable elements. Several configuration parameters allow these components to be defined. The next section shows the parameters that the user can define.

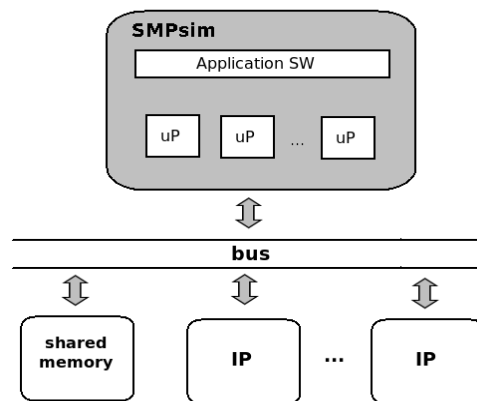


Figure 2. Complete multiprocessor system

SMPsim can be configured with a graphic user interface (GUI). This interface can interconnect several IPs and define their internal parameters. Two files are generated with the GUI. The first one is an XML file used to define the SMPsim components and their connections. The second file is a SystemC file where a system instantiation example is

described. This template description will enable the integration of other SystemC components.

A. SMPsim configuration parameter file.

The SMPsim will be composed of many processors and the application software. The user can configure many parameters for each processor such as:

- Type: it represents the processor type for estimation purposes. Several ARM processors are currently supported
- Frequency: processor frequency in MHz.
- Static_power: it is the power per instruction.

The application software will also be defined in this file. A list of all the processes or workloads is generated, where the user must specify the workload name, the main function, its arguments, the library that includes the compiled code and the processor where the workload is executed. Each process must be compiled with the SCoPE compiler to obtain the time and power estimations, as explained in section 3.

A simple example of the configuration parameter file is shown in Figure 3. The example describes a SMPsim model with three processors. Three different processes will be executed.

```
<Description>
<SMPsim_Parameters component_name="my_smp"number_of_processors="3"/>
<Processors>
  <Processor type="arm926t" frequency="200" static_power="2" />
  <Processor type="arm926t" frequency="200" static_power="2" />
  <Processor type="arm926t" frequency="200" static_power="2" />
</Processors>
<Application_SW>
  <Process name="proc1" main_function="proc1_main" args="proc1_args"
library="proc1.so" cpu_id="1" />
  <Process name="proc2" main_function="proc2_main" args="proc2_args"
library="proc2.so" cpu_id="2" />
  <Process name="proc3" main_function="proc3_main" args="proc3_args"
library="proc3.so" cpu_id="3" />
</Application_SW>
</Description>
```

Figure 3. SMPsim configurable parameters

B. System description file

The system description file is a SystemC description that includes an instantiation of the SMP component. This file will be automatically generated when the user inserts all the configuration parameters in the GUI. Not only SCoPE components can be connected, but also user application HW written in SystemC.

The peripherals will be automatically connected to the bus by the bind function of the bus defined in the UC_TLM_bus_class of SCoPE. The bus model uses the standard TLM library for fast data transmission.

In Figure 4, an example of easy and flexible system configuration is shown. The system is composed of a SMPsim component and a memory module connected to a single bus.

The bus and the memory used to verify this configuration system were the SCoPE [6] default models.

```
int sc_main(int argc, char **argv) {
  UC_TLM_bus_class *my_bus;
  UC_SMPsim_class *my_smp;
  my_bus = new
    UC_TLM_bus_class(sc_gen_unique_name("HAL"),100000000,0x80000000);
  my_smp = new UC_SMPsim_class(file_name);
  my_smp->bind(my_bus);
  UC_hw_memory *my_mem = new
    UC_hw_memory(sc_gen_unique_name("my_mem"),0x80000000,-1,100);
  my_bus->bind(my_mem);
  my_bus->generate_memory_map();
  sc_start(10,s);
  cout << "Main finish" << endl;
  cout << "Simulated time: " << sc_time_stamp() << endl;
  delete my_smp;
  return 0;
}
```

Figure 4. SystemC system description

First, the GUI allows system configuration and parameter checking. Possible errors will be reported to correct the parameterization failures. Secondly, the files that were described in the previous section are generated. The native simulator uses these files to configure the simulation.

In addition to the generation of the platform model, the simulation will also be launched. The process simulation parameters are also defined with the GUI.

Regarding the software infrastructure, SCoPE provides an abstracted model of the RTOS. Currently POSIX and uCOS APIs are supported.

The hardware platform supports not only SCoPE based components (such as SMPsim), but also user-defined HW components. These components are described in SystemC. This means that two kind of components are supported: components modeled in SCoPE and specific HW peripherals.

SCoPE provides abstract models of the main elements of a MPSoC, such as processor, cache, bus, DMA and NoC models. Specific HW components can also be included in the HW platform. Since the components are obtained from different vendors or development teams, they must share a common interface at platform level. Thus, all the platform components have to integrate a specific bus interface.

Figure 5 shows a view of the graphic user interface. It is composed of four tabs. The first one allows the user to choose the number of processors and their parameters. It is possible to select the type of processor, the frequency and the static power. In the second tab the application software will be defined. It is possible to introduce as many processes or workloads as desired. For every workload, the user has to specify its main function, arguments, and shared library. To obtain the shared library the application software must be compiled with the scope-g++ compiler as explained before.

The third tab allows the user to add predefined peripherals to the system. All the specific parameters for each peripheral

can be defined in this tab. In the last tab the simulation time will be fixed and the execution will be started.

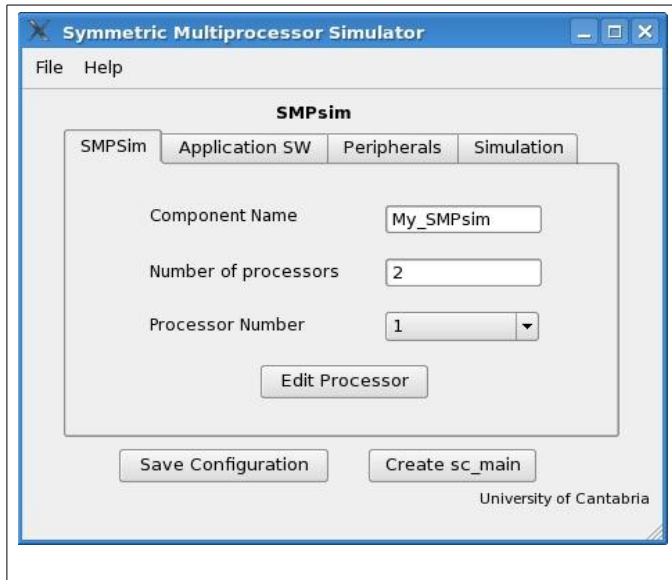


Figure 5. SMPsim graphic user interface

There are two main buttons in Figure 5. The first one will save the configuration of the SMP model, creating the Figure 3 XML file. The second button will allow the user to create the SystemC platform model.

VI. EXAMPLE

The proposed approach has been verified with an example that implements an edge detection algorithm.

The edge detection algorithm has three modules. The first module captures the image and sends it to an array in the shared memory (test producer function). Then the second module applies an edge detection algorithm to the image. It takes an image from an array of the share memory and stores the results into another array of the same memory. To finish the process, a test consumer process (third module) stores the results in a file.

Three alternatives have been implemented in order to compare the estimated performance of the example in three different architectures. These architectures are shown in figure 4.

The first proposal (Alternative 1) implements all the modules in software. In this alternative, only one processor is used.

The second alternative uses a platform with three processors. The first processor executes the test consumer process. The second processor executes the edge detection algorithm and the third executes the test producer function.

The third alternative includes two processors and one co-processor. The HW co-processor implements the edge detection algorithm. This HW component is described as a functional model, without specific timing information. Instead, a constant delay is applied on each SW access. The first

processor executes the test consumer function and the second one runs the test producer function. All the hardware co-processor and the microprocessors access to the same shared memory, what can produce an important number of bus collisions.

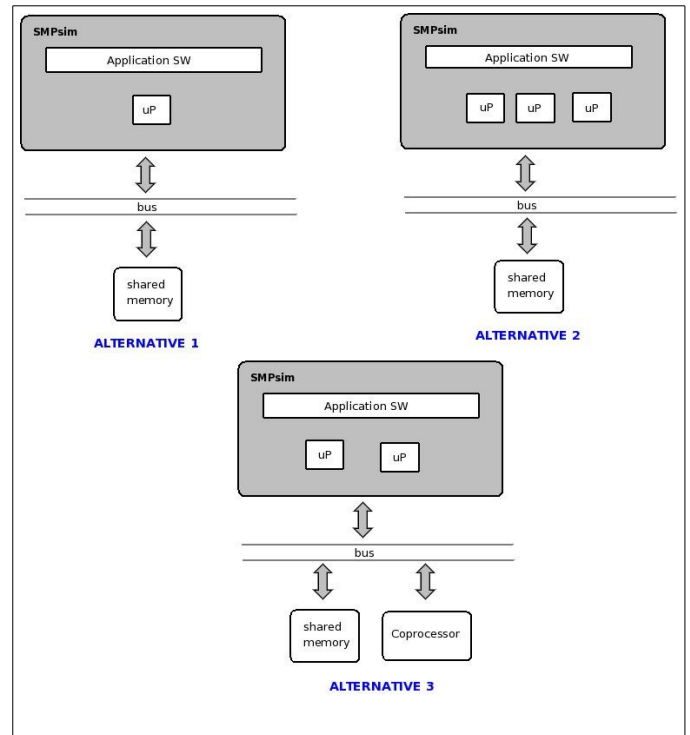


Figure 6. Different architectures for the edge detection example

The platform provides time, power and processor utilization estimations. Results show that source-code estimations at system level can provide valuable information during the first stages of development, when the system architecture has to be decided. The performance estimations of the edge detector example in the different architectures are shown in Table 1.

TABLE I. ESTIMATED PERFORMANCE OF THE EDGE DETECTOR EXAMPLE IN THE DIFFERENT ARCHITECTURES

		<i>Alternative 1</i>	<i>Alternative 2</i>	<i>Alternative 3</i>
Estimated time (sec)	Processor 1	7,21	0,758	0,758
	Processor 2	NA	6,45	148e-09
	Processor 3	NA	44e-09	NA
	Co-processor	NA	NA	✓
	Total	7,21	7,21	5,759
Estimated energy (nJ)	Processor 1	1,94e+09	2,048e+08	2,048e+08
	Processor 2	NA	1,74e+09	40
	Processor 3	NA	12	NA
	Co-processor	NA	NA	✓
Processor utilization	Processor 1	99,98%	10,51%	55,37%
	Processor 2	NA	89,48%	1,08e-05%
	Processor 3	NA	6,10e-07%	NA
	Co-processor	NA	NA	✓

NA = not applicable

✓ = hardware co-processor is used

As shown in Table 1, using SMPsim the performance of different platform configurations can be checked easily.

VII. CONCLUSIONS

A technique for automatic generation of SystemC SMP models for HW/SW co-simulation has been developed. Using MPSoCs with SMP architecture, tasks are optimally distributed among the processors, avoiding low or unbalanced microprocessor utilization. SMPsim allows the user to check different platform configurations rapidly.

Power and timing estimations of the application software running on the multiprocessing platform can be obtained from the system simulation.

It is demonstrated that using native simulation techniques are sufficiently accurate and much faster than ISS-based systems [17]. This speed increase makes the technique optimal for fast system evaluation, satisfying the need to obtain effective metrics, which is necessary for efficient Design Space Exploration frameworks.

REFERENCES

- [1] H. Posadas, D. Quijano, J. Castillo, V. Fernández, E. Villar and M. Martínez: “SystemC Platform Modeling for Behavioral Simulation and Performance Estimation of Embedded System”, in L. Gomes and J.M. Fernandes (Eds.): “Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation”, IGI Global, 2009.
- [2] F. Fummi, M. Loghi, G. Perbellini, M. Poncino: “SystemC co-simulation for core-based embedded systems”, in Des Autom Embed Syst 11(2/3): 141-166, 2007.
- [3] A. Nohl, G. Braun, O. Schliebusch, R. Leupers, H. Meyr, and A. Hoffmann. “A universal technique for fast and flexible instruction-set architecture simulation”. In Proc. of ACM/IEEE DAC, pp. 22–27. 2002.
- [4] Qemu Home page: http://wiki.qemu.org/Main_Page
- [5] Y. Hwang, S. Abdi, and D. Gajski. “Cycle-approximate retargetable performance estimation at the transaction level”. In Proceedings of the conference on Design, automation and test in Europe, pages 3–8, Munich, Germany, mars 2008.
- [6] T. Kemp, K. Karuri, S. Wallentowitz, G. Ascheid, R. Leupers, and H. Meyr. “A SW Performance Estimation Framework for Early System-Level-Design Using Fine-Grained Instrumentation”. In Proceedings of the conference on Design, Automation and Test in Europe, pages 468-473, Munich, Germany, 2006.
- [7] W. Tibboel, V. Reyes, M. Klompstra and D. Alders: “System-Level Design Flow Based on a Functional Reference for HW and SW”. In Proceedings of the Design Automation Conference, pages 23-28, San Diego, CA, 2007.
- [8] F. Pétrot, E. Villar: “High Speed Multi-Processor System-On-Chip Simulation Platforms for Hardware Dependent Software Development”. SoftSoC Open Workshop, October 14th 2009
- [9] M. V. Carvalho da Silva, N. Nedjah and L. de Macedo Mourelle: “Evolutionary IP Assignment for Efficient NoC-based System Design using Multi-objetive Optimization”. IEEE Congress on Evolutionary Computation, 2009. CEC '09.
- [10] SCoPE Home page: www.teisa.unican.es/scope
- [11] S. Basu, S. Roy, R. Kumar, T. Fisher and B. Blaho: “Peppermint and Sled: Tools for Evaluating SMP Systems based on IA-64 (IPF) Processors”. Parallel and Distributed Processing Symposium, IPDPS 2002.
- [12] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli and M. Olivieri: “MPARM: Exploring the Multi-Processor SoC Design Space with SystemC”. Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology 41(2), pp. 169-182
- [13] K. Onoue, Y. Oyam, and A. Yonezawa, “A virtual machine migration system based on a cpu emulator”, pp. 3-3, 2006.
- [14] M. Becker, G. Di Guglielmo, F. Fummi, W. Mueller, G. Pravadelli and T. Xie: “RTOS-Aware Refinement for TLM2.0-based HW/SW Designs”. In Proceedings of the conference on Design, Automation and Test in Europe, 2010. DATE'10.
- [15] M. T. Yourst: “PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator”. IEEE International Symposium on Performance Analysis of Systems & Software, 2007.
- [16] H. Posadas, E. Villar and M. Martínez: “Protocol Bus Modeling using inheritance with TLM 2.0”
- [17] J. Castillo, H. Posadas, E. Villar and M. Martínez: “Fast Instruction Cache Modeling for Approximate Timed HW/SW Co-simulation”. 20th Great Lakes Symposium on VLSI (GLSVLSI'10), Providence, USA. 2010-05