dcis2011
26th Conference on Design of Circuits and Integrated Systems
The next 25 years

# Automatic Communication Modeling for Early Exploration of HW/SW Allocation Based on Native Co-simulation

*Hector Posadas and Eugenio Villar*
*University of Cantabria, ETSIIT, Av. Los Castros sn, Santander, Spain*
*{posadash, villar}@teisa.unican.es*

**Abstract[1] - At the beginning of the design process, the allocation of each system component has to be decided. The exploration of all the possible allocation combinations requires a huge amount of performance evaluations. As a consequence, it is mandatory to find techniques capable of obtaining extremely fast evaluations with minimal designer effort. Among simulation techniques, native co-simulation is the most promising one for providing this kind of results. However, previous approaches require completely refined SW code to be explored. Addresses, protocol encapsulation and drivers are required in order to model communication mechanisms and evaluate the communication impact in system performance. This paper proposes an alternative solution capable of modeling initial, functional SW codes, evaluating communication impact without requiring any communication refinement.**

## Keywords
Resource allocation, Design Space Exploration, Electronic System Level, Embedded SW, Communication modeling

## I. Introduction

The evolution of electronic technology has enabled the design of increasingly complex systems. System functionality is usually composed of several components which are deployed on powerful HW platforms containing several processors. Additionally, some of these components can also be implemented as application-specific HW in order to optimize system performance. As a consequence, architectural mapping decisions are turning into a critical step in electronic design processes.

Once engineers receive the system models described in languages such as UML, MARTE, AADL, etc. and face the task of starting the implementation process, the first step is to decide the allocation of the different system components. These decisions condition the rest of the design process, so it is required to define adequate mechanisms to ensure optimal mappings. To do so, the resulting performance of all the different allocation possibilities has to be evaluated. Then these estimations can be used to lead the decision process. However, the initial code of the different components is limited to pure functional code, resulting from algorithmic development. In that context, codes has no the required refinement to deploy and communicate the SW components in different HW resources. Thus, the challenge is how to evaluate the different allocations with such initial codes minimizing exploration simulation times and designers effort.

In that context, the most exhaustive the exploration is, the most adequate the solution will be. However, the resulting amount of combinations considering all the possible permutations of functional components on the HW platform resources can be a really huge number. In order to obtain these estimations in a feasible time, fast tools capable of exploring the multiple combinations while requiring minimal design effort are required.

Several solutions to obtain performance estimations of electronic designs have been proposed in the literature. Static solutions [1,2] are usually applied for obtaining worst case execution times (WCET). However, those results are usually too pessimistic for initial evaluations. Furthermore, WCETs are not very useful at this level, since the use of initial, unrefined codes makes impossible to guarantee the correctness of the bounds. Additionally, the effort and time required to apply these techniques to very large, configurable systems can result extremely high.

Among dynamic techniques, virtual platforms based on instruction set simulators (ISS) [3,4] or in virtualization [5,6,7] can be used for evaluating the effect of executing a code in a certain processor or platform. However, these solutions require having the SW code completely adapted for the target platform. In these tools, complete binary target codes are needed for correct simulation.

Virtual platforms built with these technologies are based on the combination of models of all the HW components of the platform. Thus, SW components require knowing how to communicate with the rest of the system. This request can be easy to solve for memory transfers, but it becomes more complex to support in other cases, for example when trying to access peripherals, with their specific registers, addresses and operation modes, when accessing a component in other bus through bridges, when it is needed to use a DMA, or in networked communications, where it is mandatory to know protocols, addresses, package encapsulation, etc. Thus, when multiple resource allocations are evaluated, multiple code porting has to be done, wasting effort in developing implementations from which it will not be possible to take any advance in the future. Additionally, with these tools it is not possible to perform a high-level simulation where the communications among components are incompletely specified.

Some approaches based on native execution of annotated code have been proposed [8,9] to obtain faster system evaluations. Additionally, as they are based on native simulation it is not strictly required to dispose of a set of

[1] This work has been supported by Artemis JU 100029 SCALOPES and Spanish MICyT TEC2008-04107 projects.

completely refined ports of the system for the different allocations. Additionally, some works on obtaining accurate performance annotations of the SW components has been proposed [10,11]. Side effects as bus or peripheral accesses are modeled apart. Thus, performance of the system functionality and communications are analyzed in a partially separated way.

Applying that, the proposal described in this paper presents a solution to simulate different allocations (HW and SW) of the system components using initial, functional codes, without any communication refinement. The solution is based on the use of annotated functional models running in parallel with a model of the system architecture centered on communications ("communications model"). The functional model is implemented using a previous tool called SCoPE. Thus, this work focuses on communication modeling and integration between the two models.

Using the two combined models, each time two components of the functional models want to communicate, the initiator sends a request to the communication model. The communication model analyses the impact of the transfer, and it returns the results obtained to the annotated functional model. As a result communication times and performance can be integrated in the system simulation.

The way the communication model extract communication paths, how overheads are obtained and the mechanisms used to connect both models are described in next sections. After the state of the art, the paper analyzes the SCoPE tool [9] and presents all the modifications required to transform it into a tool capable of automatically modeling different component allocations. Then, it is described how communication paths among components are extracted and how communications performance are evaluated. A description of the mechanism for connecting both models follows. Finally, some results are provided to demonstrate the validity of the approach.

## II. State of the art

Simulation environments for design space exploration have to overcome several challenges. Mainly, these simulations require achieving very fast speeds, considering the large amount of points to be simulated. Thus, modeling techniques have to be able of evaluating all the configurations selected by the DSE tools without provoking additional delays.

While HW simulation can be performed at different abstraction levels using appropriate languages such as VHDL, Verilog, System-Verilog and SystemC [12], efficient, sufficiently accurate SW simulation required of additional research. The three main methodologies used for building virtual platforms on top of which simulate the embedded SW are Instruction-Set Simulation (ISS), virtualization with binary translation and native co-simulation.

The first solution, ISS-based HW/SW co-simulation, is the main industrial platform simulation technology supported by mature commercial tools offered by all the major vendors [3, 4]. The commercial modeling and simulation tools currently available are based on previous research activity in academia. In [13] a generic design environment for multiprocessor system modeling was proposed. The environment enables transparent integration of ISSs and prototyping boards. As an evolution of this work, in [14] a SystemC infrastructure was developed to model architectures with multiple ARM cores. However, ISS-based simulations results in very large simulation times, which makes its use not recommended for design space exploration.

Another technology proposed for SW simulation is using virtualization together with binary translation. The binary code of the target processor is dynamically translated to the host executing the same functionality. Qemu [5], Open Virtual Platforms [6] and Simics [7] are three of the most representative examples of virtual platforms. As a comparison, simulations performed by QEMU are about 5 to 20 times slower than native simulation [15] and obtaining performance and power consumption estimations using virtualization is still an open line research.

Native co-simulation is based on the direct execution of the source code on the host. Simulation speed can be improved avoiding the use of processor models at the expense of some estimation accuracy in order to enable an efficient system evaluation. Considering the ITRS'07, estimation errors of about 30-40% can be accepted for initial system assessments [16]. Using either static [1, 2] or dynamic techniques [11, 17, 18] the SW execution times are estimated and annotated to obtain timed simulations of the application SW. A simple version of this technique has been implemented in a commercial tool [19].

An integral part of the embedded SW is the RTOS used. Thus, native SW simulation is based on using an abstract OS model. As the RTOS functionality is abstracted, this approach is faster than the HAL-based one. Several alternatives have been proposed, from generic [20, 21, 22] to real OS models [23, 24]. The latter approach has the advantage that, when efficiently exploited, a more accurate model of the underlying RTOS can be achieved. This additional efficiency is obtained by combining the source-code execution time estimation with the OS model providing more accurate results. The work in [25] provides an analysis of the impact of including the OS time in the overall system estimation.

However, none of these modeling methodologies are aimed at complex Multi-Processor systems on chip (MPSoC). Dynamic task mapping, drivers and interrupt management required in MPSoC modeling are not covered by the previous techniques with the simulation speed required for performance estimations and system dimensioning. To the best of out knowledge, only two works have been proposed for modeling such complex systems using native simulation [8,9]. Additionally, the second one has been initially adapted for design space exploration [9] so it has been selected as starting point for this work.

But previous approaches require the SW code to be explored to include complete information about communication mechanisms if communication impact has to to be considered. Addresses, protocol encapsulation and drivers are required for that goal. To solve that limitation at the beginning of the design process, this paper proposes a solution capable of evaluating that without adding additional information or design effort.

## III. Problem definition

Most design flows start defining separately the system functionality and the HW infrastructure. Then, co-design process begins by deciding the optimal resource allocation.

In this paper, it is considered that the method used to decide the allocation of the functional components is based on a design space exploration process. The exploration flow considered for obtaining this purpose is the following (figure 1). First, the code is annotated and compiled generating a simulator executable capable of modeling all the configurations. Then, a design space explorer tool is called. This tool starts a loop where the simulator executable is called as many times as points in the design space have to be evaluated. Finally, the explorer tool analyzes the data obtain and proposes the optimal solutions. In [26] the interfaces between the simulation infrastructure and explorer tools are presented, so the rest of the paper focuses only on the simulator itself.

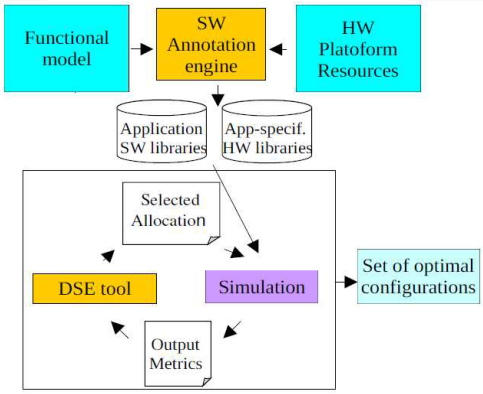


Figure 1: Allocation exploration flow

As a consequence of the point in the design flow, the inputs received by the simulator for the allocation evaluations have several characteristics that can be extracted.

First, a platform independent model of the system functionality or, at least, an executable code for all the system components is available. These codes are usually implemented using an underlying infrastructure capable of providing facilities for inter-component communication, synchronization and other services. SystemC or the native operating system are usually employed as infrastructures supporting the execution of the entire model in order to enable checking the functionality.

Taking advantage of that, the proposed solution starts from these executable models, adding performance annotations and HW resource allocation effects to the model. Then, the system performance resulting on the combination on the different component allocations can be estimated.

However, the effects caused by the communications among all system components are not considered. Functional model is platform independent, so resource allocation for each component is added during the exploration process, when the code is annotated. As a consequence, the functional model has no information at all about what the system has to do to perform these communications. In fact, for each simulation of the exploration loop, components are allocated to different resources, so how communications among tasks are made is something that changes on each simulation.

Thus, to model system communications, it is required to extract the path followed by the information from initiator to target on each simulation, model the performance and include the results in the simulation.

The infrastructure developed has been integrated in a design flow including UML/MARTE and Eclipse. The modeling capabilities of these elements have been used to improve the capabilities the designer has to drive the evaluation process, as is shown in following sections. Nevertheless, their use is not mandatory for the application of the technique itself, and they can be avoided it the proposed solution is integrated in another infrastructure.

## IV. Modeling communication paths

To perform accurate modeling of the system communications, the first step is to know the paths the information follows on each transfer. Usually, SW systems are built creating several layers. One of the lower system layer is the communication layer. Each communication in the functional model is not implemented in a completely different way. Communications usually take advantage of the services of this layer, to reduce the design effort.

As a consequence, similar communications make use of the same service in the communication layer, resulting in a similar communication type. For example, it is common to find that in a system, transfers sent from all processes in processor "a" to processes in processor "b" use the same lower layer service, being preformed in the same way.

Taking advantage of that, it can be accepted that it is not required to extract the communication path for any communication make in the system, but only extract the paths required to support communication among all the HW resources of the system: processor and application-specific HW. This simplification substantially reduces the computational effort required to model communications during system simulation. As a result, simulation overhead implied by the communication model is minimal, something which is critical if we want to explore large design spaces, with thousands of possible points.

To minimize the simulation time, the proposed approach is to obtain the resource-to-resource paths for each simulation just before it starts. Then, the transfer can be modeled only indicating the initiator and target.

To obtain these paths, two solutions has been implemented in this work. The first one extracts the paths automatically from the platform description used in [9] and described in [26]. Additionally, a solution has been proposed to enable the designer to fix some of these paths manually, instead of letting the tool to select one. To do that, an XML file format has been defined, where the path is described. An Eclipse plug-in has been developed to enable describing the paths in UML/MARTE, but these details are out of the scope of the paper, which is centered on the simulation itself.

The path extracting process is the following:

- First the user defined paths are read, if any.
- Second, if there is a path defined to connect resource "A" with resource "B", but not the opposite, it is analyze if the same path can be used in the opposite direction.

- Finally, if no paths are defined to communicate two resources, the automatic path mechanism is used.

- If no path is found to connect two elements, an error is only reported if it is requested during simulation.

### *IV.1 Automatic path extraction process*

To extract the communication path between two HW components, a branch and bound like algorithm has been implemented. Information about the HW platform is obtained from the XML file used as input by [9]. In this file all HW instances are listed, including their interconnections. Interconnection information also includes master/slave information. Summarizing, the path extractor knows who is connected to each HW component and in which direction.

The algorithm looks for the faster path capable of communicating initiator and target. Additionally it is defined an intermediate component where information can be exchanged (a memory or a network).

To extract the path, two trees are created containing all the elements with an already known connection path with the initiator or the target. Each node of the tree represents a HW component and it is linked in the tree to the components connected to it, as described in the XML file.

The procedure starts identifying initiator and target as leaf in their respective trees. Then it is checked if there are common leafs in both trees. If not, the HW components connected to the leaf components in the XML file are added as new leafs, and the checking is performed again. When a hit is find, a complete path has been found. The cost of the path is set as maximum cost and only faster solution are searched. To do that, all the branches which current cost is lager than this maximum are pruned. When all branches have been bounded, the minimal path is selected.

The radial structure of the algorithm helps finding a fast solution earlier, since usually these solutions has an smaller amount of components in the path.

### *IV.2 User defined paths*

In certain cases, the user can prefer to provide a path instead of letting the tool to automatically take the decision. When multiple possible paths can be found, it can be interesting to the user to have a way to provide it, in order to ensure that the simulation will follow designer decisions.

To do that, it has been enabled the possibility of using a UML/MARTE sequence, diagram to describe the communication path in Eclipse. Then, a plug-in generates a XML file indicating the paths selected. The resulting XML file has the following format:

```
<interconnections>
    <connection origin="processor1"
      target="processor10" link="memory1">
        <component name="comp1">
        <component name="comp2">
            ...
        <component name="memory1">
            ...
        <component name="last_comp">
    <connection/>
    <connection ... >
<interconnections>
```

### *IV.3 Communication model*

In order to create a model of the system capable of evaluating the impact of communications, the elements provided by [9] to create the platform models have been minimally modified. The tool provides models of buses, memories, DMA, bridges, networks and network interfaces, which are developed on top of a base class that handles the TLM2 protocol proposed by the OSCI.

First, TLM2 protocol is based on the use of structs for transferring the communication information. These structs include information about the type of communication (read/write), address, package size, data buffer, etc. Furthermore, the struct includes a void pointer for user-defined additional information.

To model only communications creating an infrastructure capable of working in parallel with the functional model, a new package type has been defined. This package contains all the members of the struct, but the data buffer, with is null.

The inherited class, has two operation modes, for communication channels (buses/networks) and for HW components. In the modification performed, the part inherited by the HW components has been extended to detect the new package type. When the package is detected, the HW component wait the reception time, indicated by the component parameters, and automatically returns, indicating that transfer has been completed. No functionality in the HW component is executed.

Second, system communication paths are supposed as a combination of two simple communication structures:

- HW component / Channel / HW Component, and
- HW Component / HW Component.

Thus, the internal modeling of a communication transfer extract all the simple communication structures in the path, and inject a package of the new type in all the initiators of the simple communications and gets the delay in all the structures. As a consequence, total communication delay is estimated. Additionally, as channels are used as in normal communications, contency and other side effects are considered.

## V. Functional/communication model link

As stated before, the functional model used in this work considers that services are accessed by clients through function calls. Thus, the execution flow of each task changes from one component to another, and usually from one HW resource to another, depending on the allocation of each component. The solution proposed to model this change of HW resource is to consider a global operating system and use the processors' mask to fix where the tasks is executing at each moment. However, the last problem we found is how to transform the original function calls in the functional model to do so, in an automatic manner where no additional recoding effort from the designer is required.

The basic idea to do that is to generate a Container/Component infrastructure for all system components. Here, the Component contains the part of the functional model related with the system component. The Container is a wrapper in charge of detecting inputs and outputs from the component (Figure 3).

Component and Container both provide the same interfaces, with the same function prototypes. Each time a Component requests a service from another Component, the call is redirected to the Container of the second component instead of the Component itself. Then, the Container calls the communication model indicating the initiator, target and size of information to transfer. Information size depends on the arguments of the function. "malloc" and "calloc" functions have been wrapped to save the size of each buffer created, to enable getting correct transfer sizes

After that, the same function in its Component is called to execute its functionality. When the function returns, the Container calls again the communication channel, to model how the service returns the obtained results, and continues the execution in the first Component. The modification of the function call can be done by changing the pointer to the interface called or by using "#define" clauses, avoiding manual recoding.

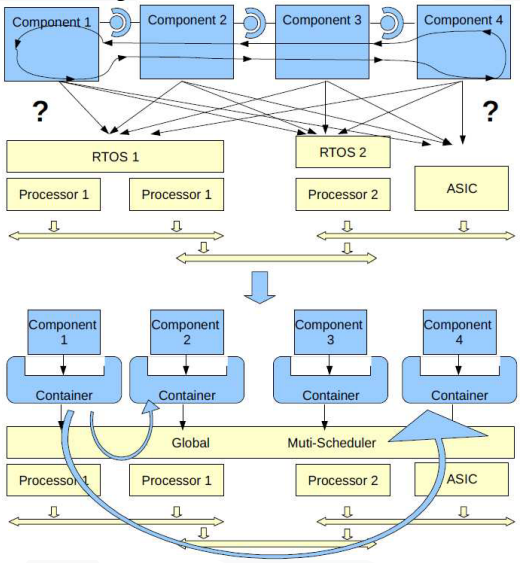


**Figure 3: Exploring resource allocation with containers**

Thus, the problem is limited to how to create automatically the Container for each component. For doing so, information in high level models such as UML/MARTE is required. In these models, information about the components in the system, the interfaces, the functions on each interface and from where they are called is present. From this information, the Container and the modification in the function calls can be done through specific generators. In our work, a generator from UML/MARTE has been developed in Eclipse to create the required code, but its internal details are far from the scope of this paper.

To generate the Container, the generator creates a code describing the functions provided. For example, in the following code, a container implementing an interface with three functions is shown:

```
CREATE_CONTAINER(ProcessingComponent):INHERIT_INTER
FACES(public ImageProcessor_PI){
  COMPONENT(ProcessingComponent);
   SPORADIC_FUNCTION(startOperation,m_ImageProcessor);
   SPORADIC_FUNCTION(stopOperation,m_ImageProcessor);
SPORADIC_FUNCTION(getState,m_ImageProcessor,state);
};
```

Internally, each function is implemented by a Macro as:

```
#define SPORADIC_FUNCTION(function, arg_type) \
    void function(arg_type arg1) {\
        int node = getCurrentProc();\
        if(node!=m_node){\
           chageCurrentProc(m_node);\
           modelCommunication(node, m_node, size);\
           component->function(arg1);\
           modelCommunication(m_node, node, size);\
           chageCurrentProc(node);\
        } else{\
            container->function(arg1);\
        } }
```

## VI. Examples and results

To demonstrate the possibilities the proposed approach can provide to system designers when deciding the resource allocations, an example based on a H264 coder has been applied. The coder algorithm used has 8 concurrent tasks, conforming a functional modeling running on top of a Linux system.

In the proposed example, each task can run on top or an ARM9 processor at 200MHz, another ARM9 at 100MHz, on a Leon processor or as application-specific HW. Thus, all the possible combinations of component allocations are 8= 4096. The two ARM9 processors are placed in separated buses, both with access to another bus with a memory connected. Additionally, this structure, the Leon processor and the AS-HW are in different nodes of a network.

Through the adequate "makefile", 4 dynamic libraries for each one of the 8 components have been created with the different annotations required to model the 4 possible allocations. As the annotations are done automatically, no designer effort has been required for that step.

The first result extracted is that the proposed infrastructure was capable of simulating the functionality and detecting the communication paths automatically without requiring any additional effort from the designer.

An external tool, the design space explorer tool M3Explorer [27], has been used to automatically collect the performance of the 4096 possibilities, using the SCoPE simulator combined with the infrastructure presented in this paper. As a result, the explorer tool has reported a Pareto set including the optimal configurations.

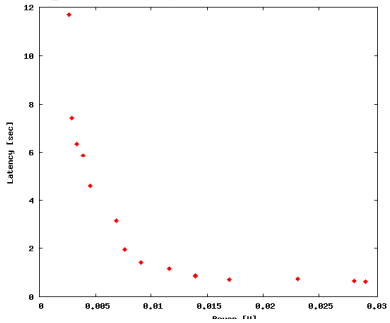


**Figure 5. Pareto set for latency/power consumption**

The overall time required to perform the complete exploration (4096 simulations) has been of about 5 hours without any manual intervention during them, which contrast with the almost 3 hours required to execute a single implementation in a ISS-based infrastructure developed for this model.

About estimation accuracy, as communication delays have not been involved in the presented model, no comparisons with real implementations can be done. Nevertheless, the accuracy completely depends on the annotation engine used, which in previous works has demonstrated to achieve estimation errors of about 20% at most [9].

## VII. Conclusions

The paper shows that efficient resource allocation exploration for large systems is possible in native simulation requiring minimal designer effort. Starting from a functional model and a graphical model such as UML/MARTE it is possible to transform the pure functional execution in a simulation where effects provoked by real HW resources can be evaluated, including communication effects.

The use of native co-simulation as basic infrastructure to model the system while considering the different possible allocations, enables obtaining fast performance estimation with a low designer effort, since it is not required to implement all possible system ports to enable component communications.

The annotation techniques used for native co-simulation enable modeling separately the performance of components functionality and communications. This solution enables modeling separately the performance of the functionality and the system communications.

Using an algorithm of branch and bound, feasible communication paths can be automatically found. Additionally, the designer can fix paths if preferred. Then, dividing the paths in minimal structures of HW Component, - Channel (bus or network) - HW Component, communication effects can be modeled by introducing packages of the required size in the first component and extracting them in the last one.

Finally, the use of high-level models, such as UML/MARTE models and generators, provides the information required to build the Component/Container infrastructure. This infrastructure wraps component codes, enabling the modeling of different allocations and their communications within the same infrastructure.

Summarizing, all the modifications required to evaluate the performance of the different allocation possibilities can be obtained automatically, minimizing the designer effort required for exploring the design space.

## VIII. References


[1] Balarin, F. et al: Hardware-Software Codesign of Embedded Systems: The POLIS Approach. Springer (1997).

[2] Hergenhan, A., Rosenstiel, W.: Static Timing Analysis of Embedded Software on Advanced Processor Architectures. Proc. of DATE, IEEE (2000).

[3] ARM Realview Development Suite (2005). Retreived form www.arm.com/products/DevTools/RealViewDevSuite.html

[4] SkyEye User Manual. From http://www.skyeye.org (2005).

[5] Qemu, http://wiki.qemu.org.

[6] Imperas, http://www.ovpworld.org.

[7] Wind River Simics - Embedded Systems Simulation Platform, Virtual Hardware for Embedded Software, http://www.virtutech.com/

[8] P Razaghi and A Gerstlauer: Host-compiled multicore rtos simulator for embedded real-time software development, DATE 2011

[9] Posadas, H., Castillo, J., Quijano, D., Villar, E., Ragot, D. & Martinez M.: SystemC Platform Modeling for Behavioral Simulation and Performance Estimation of Embedded Systems. In the book L. Gomes & J. M. Fernandes (Eds.): Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation, IGI Global (2009).

[10] Bouchhima, A. , Gerin, P. & F. Ptrot: Automatic Instrumentation of Embedded Software for High Level Hardware/Software Co-Simulation. Proc, of ASP-DAC. 2009

[11] J. Schnerr, O. Bringmann, A. Viehl, W. Rosenstiel. “High-Performance Timing Simulation of Embedded Software”. In proc. of DAC, 2008.

[12] SystemC, IEEE 1666 -2005 Standard LRM, http://www.systemc.org/downloads/lrm.

[13] Benini, L., Bertozzi, D., Bruni, D., Drago, N., Fummi, F., & Poncino, M.: SystemC cosimulation and emulation of multiprocessor SoC design, Computer, V.36, N.4, (2003).

[14] Benini, L., Bogliolo, A., Menichelli, F. & Oliveri, M.: MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. Journal of VLSI Signal Processing, V.41, N.2: Springer.

[15] M.Becker, T.Xie, W.Mueller, G. Di Guglielmo, G. Pravadelli and F.Fummi, “RTOS-Aware Refinement for TLM2.0-Based HW/SW Designs”, in DATE 2010.

[16] ITRS - Design. International Technology Roadmap for Semiconductors. Retrieved from http://www.itrs.net/Links/2007ITRS/Home2007.htm (2007).

[17] Brandolese, C., Fornaciari, W., Salice, F., & Sciuto, D.: Source-level execution time estimation of C programs. Proc. of CoDes, IEEE (2001).

[18] S. Stattelmann, O. Bringmann, W. Rosenstiel: Fast and accurate resource conflict simulation for performance analysis of multi-core systems, DATE'11

[19] InterDesign Technologies, FastVeri (SystemC-based High-Speed Simulator) Product Overview, http://www.interdesigntech.co.jp/english/.

[20] Gerstlauer, A., Yu, H. & Gajski, D. D.: RTOS Modeling for System Level Design, Proc. of DATE, IEEE (2003).

[21] Hassan, M.A., Sakanushi, K., Takeuchi, Y. & Imai, M.: Enabling RTOS Simulation Modeling in a System Level Design Language. Proc. of ASP-DAC, IEEE (2005).

[22] Yoo, S., Nicolescu, G., Gauthier L.G. & Jerraya, A.A.: Automatic generation of fast timed simulation models for operating systems in SoC design, DATE, IEEE (2002).

[23] Hassan, M. A., Yoshinori, S. K., Takeuchi, Y. & Imai, M. RTK-Spec TRON: A Simulation Model of an ITRON Based RTOS Kernel in SystemC. Proc of DATE, IEEE (2005).

[24] Posadas, H., Admez, J., Snchez, P., Villar, E., & Blasco, P.: POSIX modeling in SystemC. Proc. of ASP-DAC’06 (2006).

[25] Brandolese, C. & Fornaciari: Measurement, Analysis and Modeling of RTOS System Calls Timing,11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools, 2008.

[26] Posadas H., Villar, E.: “Automatic generation of modifiable platform models in SystemC for Automatic System Architecture Exploration”, DCIS’09, 2009

[27] M3Explorer, http://home.dei.polimi.it/zaccaria/ multicube_explorer/ Home.htm