

# Native Co-Simulation of TCP/IP-Based Embedded Systems in SystemC

Héctor Posadas & Eugenio Villar

Microelectronics Engineering Group  
Univeristy of Cantabria  
Santander, Spain  
{posadash, villar}@teisa.unican.es

*Abstract*<sup>1</sup>— Fast, early estimation techniques are crucial to achieve optimal designs of large embedded systems while reducing the development time. However, modeling solutions at these early design steps still suffer of important lacks. Most of these embedded systems contains networks or are conceived to be connected to networks such as networks on chip or internet. These networks have an important impact in the systems in development. As a consequence, the modeling of the network effects must be included in early modeling environments. One of the most common techniques for early, fast modeling of large embedded systems is the native execution of annotated SW code on top of a SystemC-based virtual platform. However, SystemC language does not provide the components required for networked system modeling, such as TCP/IP stacks or network models. As a consequence, there are important limitations in both estimating the system performance and checking the integration with the other components in the network. To overcome this limitation, this paper proposes a complete infrastructure for early modeling of networked systems.

*Keywords*—Distributed Systems, system modeling, TCP/IP, design space exploration, native co-simulation

## I. INTRODUCTION

Distributed systems consist of multiple autonomous processing units interconnected through a communication network. The importance of distributed systems has been growing with the advent of computer networks of a wide spectrum: Networks of geographically distributed computers at one end, and tightly coupled systems built with a large number of inexpensive physical processors at the other end [1]. Both kinds of distributed systems are made available by the rapid progress in the technology of large-scale integrated circuits.

However, design tools for distributed systems have not achieved an equivalent evolution. The larger are the designs, the most important is early performance optimization. Large research work has been done in early, fast modeling techniques for mono-processor and multi-processor systems [18-21]. Virtual platforms can be created in order to model the behavior of HW and SW system components.

These platform models are capable of providing performance estimation of the systems at the initial development stages. As a consequence, early optimizations and fast design space exploration (DSE) processes can be made, proving the design processes.

These techniques obtain fast estimations of the system performance using high level solutions, such as system level languages (e.g. SystemC) and transaction level models (TLM). The large simulation times required by instruction set simulators (ISSs) are avoided using faster techniques, especially native co-simulation of annotated SW code.

In this native co-simulation, several solutions have been proposed for time and power modeling of the SW code, the RTOS and the HW platform. However, there is a lack of models capable of analyzing distributed systems at the same level.

Up to now, the methods usually applied for modeling distributed systems are based on creating cycle accurate models of the nodes, including ISS simulators and slow HW component models. Then, the SW code, including the operating system and the network communication SW, is cross-compiled and executed in the ISSs. As a consequence, a highly accurate simulation model is obtained. However, simulation times are extremely high. As a consequence, these techniques are not adequate for early design steps.

In order to allow the designers of distributed systems to obtain performance estimations feasible to be applied to DSE processes, the high-level modeling techniques must be improved with additional elements. Solutions for easily modeling the time and power effects of communication protocols and the behavior of the communication networks must be provided.

This paper proposes a complete solution to model TCP/IP based systems in a native co-simulation environment. The solution is built on top of a simulation framework oriented to native co-simulation [21], extending its functional capabilities. The framework includes a Linux OS model which lacks of networking capabilities. In order to solve this limitation, a well-known TCP/IP stack [22] has been adapted to work with the OS model. Furthermore, a simple network model, network interfaces to be connected to the system bus, and device drivers to be integrated in the OS model has been developed.

---

<sup>1</sup> This work was supported by the Spanish MICyT and the EC in Artemis JU 100029 SCALOPES and TEC2008-04107 projects.

This paper is structured as follows. First, the state of the art work is reviewed. Then, the complete methodology for modelling networked systems is presented. Next, the integration work done in a standard TCP/IP stack is described. After that, the other components required for system modelling are presented. Finally, results and conclusions are inferred.

## II. STATE OF THE ART

Accurate modeling of networked systems can be considered a mature area. Regarding the networks themselves, a large number of network models can be found [2-5].

Ns2 [2] is a discrete event network simulator. Ns is popularly used in the simulation of routing and multicast protocols, among others, and is heavily used in ad-hoc networking research.

OPNET [3] provides a library of open source, discrete event simulation models for the information technology industry. In a similar way, OMNeT++ [4] is a component-based, modular and open-architecture discrete event network simulator.

NetSim [5] is a network simulation tool used for teaching and network lab experimentation. Various technologies such as Aloha, Ethernet, Wireless LAN, Wi Max, TCP, IP, etc are covered in NetSim.

Another group of simulators is oriented not only to simulate the network but also allows its integration on complete system models. Several simulators integrate network models in SystemC environments [6-9].

Noxim [6] is a Network-on-Chip (NoC) simulator developed in SystemC and freely downloadable from "Sourceforge" under GPL license terms. NIRGAM [7] is a SystemC based discrete event, cycle accurate simulator for research in Network on Chip (NoC).

Grace++ [8] is a system-level design environment for Network-on-Chip (NoC) and Multi-Processor platform (MP-SoC) exploration.

NNSE (Nostrum NoC Simulation Environment in SystemC) [9] is a tool for simulating network-on-chip. It is aimed to fully simulate a whole NoC. The communication backbone is built in accordance to ISO's OSI seven-layer model.

In [10] a simulator for wireless sensor networks describing how to model this kind of systems on top of a system-level modelling language.

Finally, several system modelling environments have integrated network models to simulate communication effects together with the execution of the SW and HW components. These environments include ISSs to execute the SW and SystemC models for the other HW components.

Garnet [11] is a network-on-chip performance simulator which is compatible with the GEMS [12] multiprocessor framework.

EDALab [13] provides a flexible co-simulation environment in SystemC where ISS and SystemC models are connected. SystemC HW models at different abstraction levels can be used. In [14] SCNSL results are compared with the ns2 simulator.

MPARM [15] is a tool oriented to model multiprocessor systems based on ARM and some other

processors. The simulator integrates network models as the Xpipes NoC.

The Synopsys Platform Architect can be used for cycle accurate modelling of networked systems, It can be connected with the DesignWare System-Level Library [16], which provides TLM component models developed in SystemC, including wireless and Ethernet network models.

These tools have been used for exploring system designs [17]. However, for intensive HW architecture, SW and RTOS exploration, they can be too slow. Modeling of large systems at cycle accurate level requires extremely large times.

To solve this limitation, several solutions at a higher abstraction level have been proposed. In order to reduce the computation requirements for SW modelling co-simulation based on native simulation of annotated SW code is applied [18-21].

[18] and [19] employ native execution of embedded software with performance annotations based on the equivalent assembly code compiled for the target processor. Nevertheless, complex reverse compilation techniques must be applied in order to correlate assembly blocks with source blocks.

To avoid the correlation problem, an exhaustive work at intermediate level was developed in [20]. The work is mainly focused on the influence of compiler optimizations.

These modelling tools can be connected with network models in order to simulate distributed systems. However, these systems require another element that native co-simulation usually lacks: a complete SW infrastructure. To adequately handle communication, OS and communication libraries are required. In ISS-based environment, all these elements can be cross-compiled and executed as in the real platform. However, this solution is not valid in native co-simulation.

In [21] a simulation environment combining native execution of annotated code and a POSIX-based operating system model is provided. The environment works on top of SystemC and allows integrating HW component models. However, this model does not provide communication SW infrastructures.

In this paper a solution for modeling networked systems with native co-simulation is presented. To do so, a TCP/IP stack and a network model has been integrated co-simulation framework. The framework in [21] has been selected in order to take advantage of the OS features provided.

The TCP/IP stack chosen to be integrated has been the lwIP package [22]. lwIP is a small independent implementation of the TCP/IP protocol suite that has been developed by the Swedish Institute of Computer Science (SICS). The focus of the lwIP TCP/IP implementation is to reduce resource usage while still having a full scale TCP. This makes lwIP suitable for use in embedded systems with tens of kilobytes of free RAM and room for around 40 kilobytes of code ROM.

The stack lwIP supports a wide set of network protocols IP, ICMP, UDP, TCP, DHCP, PPP and ARP. It has been integrated in several small embedded OS, such as eCos, FreeRTOS and uCos. It is open-source

code, distributed under BSD license.

### III. SYSTEM MODELING INFRASTRUCTURE

Native co-simulation consists on modeling HW/SW systems completely but with separate techniques for HW and SW components. SW is modeled by annotating the source code with performance information and executing it natively. The HW platform is modeled using approximately timed descriptions at transaction levels. SystemC is the language commonly used to create the HW virtual platform.

To support modeling of networked systems it is required to provide modeling capabilities both to handle network operation within the SystemC simulation and to connect the simulation itself to an external network, such as internet (Figure 1). That way, the system can be modeled in its real environment, considering the communication between the embedded SW and external nodes.

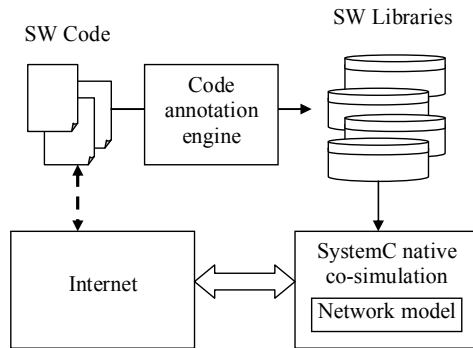


Figure 1: Networking in native co-simulation

To perform that modeling correctly, it is required to extend the SystemC infrastructure with all the components required to create a realistic system model (Figure 2).

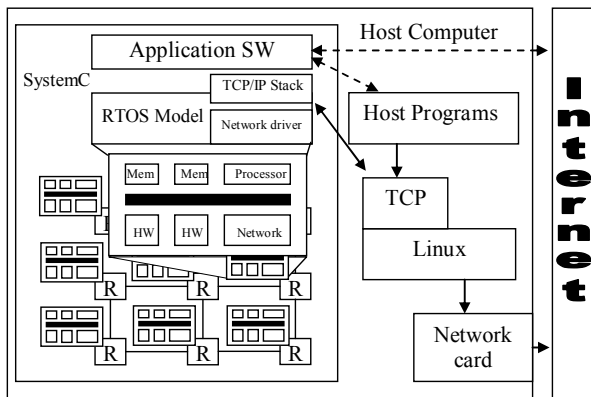


Figure 2: System components required for network modeling

To model communication among components inside the simulation, it is required to provide a TCP/IP stack model. This stack is placed on top of the OS model, and it is connected to a network driver. The network driver is in charge of controlling the access to the local network interface, which is also connected to the network model.

For external communication, the TCP/IP stack must be connected directly to the TCP stack of the host computer. To do so, the proposed solution is to use the

tunneling device driver (“/dev/net/tun”), with is usually included in many operating systems, such as Linux or UNIX. Sending Ethernet packages to this device, the internal components of the SystemC simulation can communicate with other programs in the host computer, or programs spread through the internet.

To create this entire infrastructure, the SCoPE tool has been selected as starting point. The tool provides capabilities to annotate the code, a Linux-like RTOS model and some SystemC components to create virtual platforms, such as buses and memories.

However, to model networked communication it is required to add a TCP/IP stack, a network device driver, a network interface and a network model.

### IV. INTEGRATING A TCP/IP STACK

To integrate the TCP/IP communication capabilities the lwIP stack has been integrated in a native co-simulation framework. The stack provides several ports, including a Linux/Unix one, with has been selected as starting point, as the framework proposes a Linux-based OS model. However, the OS model has no complete Linux functionality, so port modifications are required. Furthermore, the integration of a stand-alone package in a SystemC simulation requires additional effort.

#### A. lwIP integration requirements

The lwIP stack is distributed with a port to run over Linux/Unix systems. Thus, its integration in SystemC on top of a POSIX based OS model, is a very suitable solution. More specifically, the POSIX functions required by the stack port from the OS model are listed in table 1.

'malloc'	'pthread_cond_broadcast'
'free'	'pthread_cond_destroy'
'memcpy'	'pthread_cond_init'
'memset'	'pthread_cond_timedwait'
'open'	'pthread_cond_wait'
'read'	'pthread_create'
'write'	'pthread_equal'
'ioctl'	'pthread_self'
'select'	'pthread_mutex_destroy'
'snprintf'	'pthread_mutex_init'
'system'	'pthread_mutex_lock'
'gettimeofday'	'pthread_mutex_unlock'

Figure 3: OS functions required by the lwIP port

Almost, all these functions are supported by the POSIX API provided by the OS model. Additionally, the “ioctl” and “select” functions has been added to the original library, and the “open”, “read” and “write” functions has been modified to access the HW drivers, as explained below.

The “system” function is not supported by the OS model because the goal of this function is to execute shell commands. As the shell modeling is not covered by the OS model, the function “system” cannot be executed. This function is used in the lwIP stack port to assign the IP address to the network device. Thus, a different solution has been applied for the SystemC porting.

### B. lwIP porting for SCoPE integration

The provided lwIP port to UNIX, is prepared to use the “tun” driver. However, for normal operation in SCoPE some modifications are required.

First, the lwIP port is not prepared to make loop-backs. When trying to access the same address of the node itself, or to the 127.0.0.1 address, the stack fails. To solve that problem the low-level transmission function has been modified. When the IP address of the outgoing package corresponds to the current processor, the package is not sent to the network; it is directly transferred to the lwIP low-level input function. Thus, the problem is solved. However, the use of the default loop-back address (127.0.0.1) is not allowed.

Secondly, the port does not get correctly the MAC address from the “tun” device. The port has been modified to obtain the address from the driver using the “ioctl” function with the adequate parameters.

The third modification done in the port is specific for SystemC. The “system” function is not implemented, so the IP address is assigned directly to the network driver using the configuration parameters. Thus, the system call done in the “tun” initialization is useless and has been eliminated.

This porting does not initialize the ARP table, so at the beginning of the connections, the corresponding ARP packages are sent, like in a real system. By default, only the local MAC address is stored, assigning the address reported by the “tun” device.

### C. lwIP dual porting support

In order to allow both internal and external communication a double port is required. lwIP can be configured to support several network interfaces at the same time. Each interface is defined by an IP address, a gateway address, a network mask and a function pointer to the initialization function for the network interface.

Using these facilities the stack has been prepared to support two interfaces. The first one calls the function of the original UNIX port. The second one calls the function in the modified port. The first function initializes the system calling the services and the “tun” driver of the host operating system, and the second one calls the OS model within SystemC.

It is important to note that the addresses assigned to the “tun” device in both cases are different. While for the internal communication address set is the local IP address, for external communication, the gateway address is applied. As a consequence, for the external port, the information in the ARP table is modified. The “tun” address is assigned to the gateway, and a random one to the local node.

To work properly, the addresses of both interfaces must be in different domains. Otherwise, the stack cannot select the correct interface for each communication.

### D. lwIP internal modifications

The lwIP stack is adequate to be integrated in a modeling system as it is developed as a stand-alone package that requires very few functions from the platform in the UNIX port. However, there are several

problems for the SystemC integration. The problems arise as a result of the multi-OS modeling capabilities of SCoPE. The stack is prepared to run only one copy for memory space, but it is not adapted for several instances. The stack uses global variables inside the code, so it is not possible to instantiate several copies working at the same time.

However, one stack IP has to be modeled for each node. As the entire SystemC simulation is a single host process, all stack instances share the memory space, and global variables collide.

The solution adopted is to create a global class where all global variables are grouped. Depending on the RTOS model running when the stack is called, the class instance associated to this node is called.

The lwIP stack is prepared to manage several interfaces, but they are supposed to be used in the same processor, choosing the best one depending on the target address. The stack has been modified to select the interface depending on the processor that generates the message.

Furthermore, in the stack integration, there is one thread to manage the IP packages for node. When a package is generated or received it is sent to the message box corresponding to the node and managed by the corresponding thread.

To know the current node to select the thread and the network interface, several structures and functions in the lwIP stack have been also modified.

## V. ADDITIONAL SCOPE COMPONENTS

### A. Tun driver for SCoPE

As said before, the original UNIX port uses the “tun” device driver for external connection. Thus, to obtain an equivalent SystemC model, a “tun” driver model has been developed to be inserted in the OS model. The driver is based on the Linux “tun” driver of the kernel 2.6.

The driver implements functions for initialization and close, and function to support the OS calls to “open”, “close”, “read”, “write” and “ioctl”. For the “select” function, the “poll” function has been implemented. Finally, an interrupt manager is assigned to the network interrupt to receive the requests from the network peripheral.

The initialization of the stack is done independently for each node in the booting sequence. There, the variables are initialized, the threads for message management started and the addresses assigned.

As it was stated before, the “system” function is not operational in the OS model. As a consequence, the IP address cannot be assigned in the standard way. To solve that, the “ioctl” function has been modified to receive that information from the IP stack. By default, the IP address is composed of three common numbers in the first tree positions, and the node number in the last one.

*IP Address: 192.168.0.node\_id*

The MAC Address is defined directly when the stack is initialized. Using the “ioctl” function the address is obtained by the stack to initialize the ARP table.

The address is:

MAC Address: 01:02:03:04:05:node\_id

### B. Network card model

The SystemC platform model consists of nodes interconnected by a network. The node's input/output of data is performed by a network interface. Thus, this module is connected on the one side to the node's bus and on the other side to the network. The network interface is based on the generic interface already presented. This interface handles read/write requests generated by the node's processors. To achieve this task the interface uses a module that connects the network model and the SystemC bus, transforming the received data to the network model structures.

### B. Network Model

The network model developed is a high-level one. It recreates a mesh network architecture. The designer can set the number of nodes fixing the size in "x, y and z" coordinates. Furthermore, the node assigned to each coordinate can be specified by the user.

The model is oriented to provide mainly functional results for package transfers, while obtaining performance information. The network model receives packages at the source nodes and delivers them at the specified target HW interfaces. However, it does not model all internal effects of the communication mechanisms. Collisions, routing protocols or internal buffers are not considered in the network performance estimation.

The performance estimation takes into account only the number of simple transfers the packet requires between consecutive nodes in the network within the network. As the mesh node is organized following the three spatial coordinates, the number of simple transfers can be estimated by obtaining the difference between the x, y and z coordinates of source and target nodes. The total number of simple transfers is the addition of the tree values. For example, if a packages is transferred from node 1,1,4 to node 2,1,2 the difference is  $x = 1$ ,  $y = 0$ ,  $z = 2$ , so the total number of simple transfers is  $1+0+2=3$ .

The total transfer time is obtained by multiplying the number of simple jumps by the time of each simple jump. For power consumption the estimation is similar.

As no internal network details are considered, the accuracy of the model is not as high as the obtained by cycle accurate models. In exchange, the simulation overhead caused by the model is negligible. Furthermore, the collision times not considered in the modelling proposed can be added applying mathematical formulas. When the network traffic is being obtained, the network overload can be estimated, adding the additional time a package requires to cross the network due to network overload. This solution also maintains the simulation overhead in minimal values. As a consequence, we can say that this kind of modelling is suitable to be integrated in a native co-simulation environment.

## VI. APPLICATION EXAMPLE

To show how the proposed solution can be used for

system modeling, a simple example is proposed. A server is placed in a SystemC node, and two clients, one in SystemC and another one as a native host process are connected. The clients send textual strings, and the server makes an echo. The code of the server and clients are a common TCP/IP code, calling the usual functions: "socket", "bind", "listen", "connect", "accept", "sendto" and "recvfrom". Thus, the code is not shown in the paper.

However, the code required to build the system is SystemC present several interesting points. It can be shown in the following figure:

```
int sc_main(int argc , char **argv) {

    UC_rtos_class *rtos;
    UC_TLM_bus_class *bus;
    struct ip4addr addr;
    UC_NoC_Interface *simulator;

    simulator=new UC_NoC_Interface("NoC",1,true);
    int location[1][2][2]={{0},{1}},{2},{3}};
    simulator->set_structure((int ***)location,2,2,1);

    for(int i=0;i<NUM_NODES;i++){
        rtos = new UC_rtos_class(CPUS_IN_RTOS);
        insmod(rtos, tun_init);
        IP4_ADDR(&addr.gw, 192,168,0,i);
        IP4_ADDR(&addr.netmask, 255,255,255,0);
        IP4_ADDR(&addr.ipaddr, 192,168,0,21);
        create_hwpi(rtos,&addr);

        if(i == 0){
            rtos->processor(0)->new_process(server, 0);
            IP4_ADDR(&addr.gw, 192,168,1,2);
            IP4_ADDR(&addr.netmask, 255,255,255,0);
            IP4_ADDR(&addr.ipaddr, 192,168,1,1);
            create_native_hwpi(rtos,&addr);
        }else
            rtos->processor(0)->new_process(client, 0);

        bus = new UC_TLM_bus_class("BUS", 100000000);
        rtos->processor(0)->bind(bus);
        bus->bind(new UC_hw_NoC("eth", NET_START,
            NET_START + 0x3F, NET_IRQ));
        bus->bind(new UC_hw_memory("mem",
            RAM_START, RAM_START + RAM_SIZE - 1));
    }
    sc_start(-1);
}
```

Figure 4: Example of sc\_main

As it can be shown, in the sc\_main function, the system is composed of a network and two nodes with its RTOS (and a processor), a bus, memory and a network interface. Furthermore, a "tun" device driver is inserted in each RTOS using the "insmod" function. Two IP stacks have been created, one for RTOS. The one in the client node has a local interface and the server one, receives two interfaces, for internal and external communication.

The second client is executed directly from the Linux shell and it is not included in the sc\_main.

Furthermore, in order to check the validity of the solution, a CORBA system has been simulated on top of the networking infrastructure [23]. The CORBA system is an audio input/output system (Figure 5). The system is composed of three cobra components, places in different nodes of the system. The Object Request Broker, which is in charge of controlling the CORBA

system is in one node. The I/O sound system is in a second node, and the user controller is in a third node.

Using the proposed infrastructure, the CORBA system was accurately modeled in SystemC, considering the effects of all its components. The CORBA model for SystemC and the example were provided by Thales Communication and TIMA.

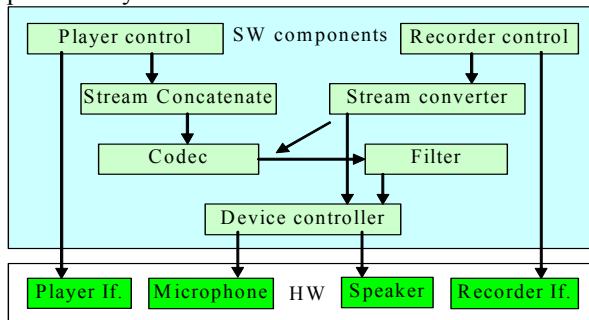


Figure 5: Architecture of the I/O sound system

As a result a timed simulation of the system where obtained. As a consequence, the functions developed to guarantee the quality of service of the CORBA functions were checked.

## VII. CONCLUSIONS

In the paper, a complete infrastructure for modeling TCP/IP based systems in a high-level environment is presented. The infrastructure has been built on top of a native co-simulation tool based on SystemC.

The co-simulation tool has been extended integrating a light TCP/IP stack, a network model, a network interface model and the corresponding driver for SW access.

As TCP/IP stack, the lwIP stack has been selected. It has been adapted for SystemC simulation. Furthermore, a double port solution has been applied in order to support, at the same time, communications with other elements of the SystemC simulation and with the host TCP/IP stack. This way, communication with process running on the host, or on other computers connected to internet can be accessed from the SystemC simulation.

## REFERENCES

[1] A. Yonezawa, C. Hewitt: "Modeling of Distributed Systems", 5<sup>th</sup> International Joint Conference on Artificial Intelligence, Cambridge, U.S.A, 1977  
 [2] NS2, <http://www.isi.edu/nsnam/ns/>  
 [3] OPNET, <http://www.opnet.com/>  
 [4] OMNET, <http://www.omnetpp.org/>  
 [5] Netsim, <http://ontwerpen1.khlim.be/projects/netsim>  
 [6] Noxim, <http://sourceforge.net/projects/nocsim/>  
 [7] NIRGAM, <http://nirgam.ecs.soton.ac.uk/>  
 [8] Grace++ Project. <http://www.iss.rwth-aachen.de/Projekte/grace/index.html>.  
 [9] Zhonghai Lu, Mingchen Zhong, and Axel Jantsch. "Evaluation of on-chip networks using deflection

routing". In Proceedings of GLSVLSI, 2006

[10] M. Rafiee, M. B. Ghaznavi-Ghoushchi, S. Kheirh, B. Seyfe: "Modeling and simulation of Wireless Sensor Network (WSN) with SpecC and SystemC", International Conference on Computer Engineering and Technology, 2009

[11] A. Kumar, N. Agarwal, Li-Shiuan Peh, N.K. Jha, "A system-level perspective for efficient NoC design," in Proceedings IEEE International Symposium on Parallel and Distributed Processing, (IPDPS) 2008.

[12] M. Martin, et al, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," SIGARCH Comput. Archit. News, vol. 33, 2005.

[13] F. Fummi, M. Loghi, G. Perbellini and M. Poncino: "SystemC Co-Simulation for Core-Based Embedded Systems", Journal of Design Automation for Embedded Systems, 2007.

[14] F. Fummi, D. Quaglia, F. Stefanni: "A SystemC-based Framework for Modeling and Simulation of Networked Embedded Systems", Forum on Specification and Design Languages, 2008

[15] 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, vol 45, 2005

[16] Synopsys Platform Architect tool <http://www.synopsys.com/Tools/SLD/VirtualPrototypin/g/Pages/PlatformArchitect.aspx>

[17] H. Jang, M. Kang, M. Lee, K. Chae, K. Lee, K. Shim: "High-Level System Modeling and Architecture Exploration with SystemC on a Network SoC: S3C2510 Case Study", DATE'04

[18] C.M. Kirchsteiger, H. Schweitzer, C. Trummer, C. Steger, R. Weiß, M. Pistauer. "A Software Performance Simulation Methodology for Rapid System Architecture Exploration". In proc. of ICECS, 2008.

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

[20] A. Bouchhima, P. Gerin, F. Pétrot. "Automatic Instrumentation of Embedded Software for High Level Hardware/Software Co-Simulation". ASP-DAC 2009.

[21] J. Castillo, V. Fernández, H. Posadas, D. Quijano, E. Villar: "SystemC Platform Modeling for Behavioral Simulation and Performance Estimation of Embedded Systems", in the book "Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation". IGI international ed.

[22] lwIP: Lighweigh TCP/IP stack. Web page <http://savannah.nongnu.org/projects/lwip/>

[23] L. Kriaa, A. Bouchhima, M. Gligor, A. Fouillart, F. Petrot & A. Jerraya : "Parallel programming of Multi-processor SoC : A HW-SW Interface perspective", International Journal of Parallel Programming, Feb, 2008