Flexible and Executable Hardware/Software Interface Modeling For Multiprocessor SoC Design Using SystemC

Patrice Gerin Hao Shen Alexandre Chureau Aimen Bouchhima Ahmed Amine Jerraya

System-Level Synthesis Group
TIMA Laboratory
46, Av Félix Viallet, 38031 Grenoble, France
e-mail : {patrice.gerin, hao.shen, alexandre.chureau, aimen.bouchhima, ahmed.jerraya}@imag.fr

Abstract – At high abstraction level, Multi-Processor System-On-Chip (SoC) designs are specified as assembling of IP's which can be Hardware or Software. The refinement of communication between these different IP's, known as hardware/software interfaces, is widely seen as the design bottleneck due to their complexity. In order to perform early design validation and architecture exploration, flexible executable models of these interfaces are needed at different abstraction levels.

In this paper, we define a unified methodology to implement executable models of the hardware/software interface based on SystemC. The proposed formalism based on the concept of services gives to this approach the flexibility needed for architecture exploration and the ability to be used in automatic generation tools. A case study of hardware/software interface modeling at the Transaction Accurate level is presented. Experimental results show that this method allows higher simulation speed with early performance estimation.

I Introduction

Multi-processor System-On-Chip architectures are made of processing nodes, which can be software or hardware, connected via a communication network. The hardware architecture of a software node is made of one or more identical processors and local processor specific subsystem (memory, DMA, interrupt controller, network interface, etc.). In this paper, the hardware/software interface refers to local hardware architecture of a software node and the *Hardware Dependent Software* (*HDS*). Fig. 1.b represents a low level implementation of MPSoC architecture called *Virtual Prototype*. At this level, the hardware/software interface is fully detailed, allowing time accuracy simulation at the expense of the simulation speed. At the opposite side, a *System Level* specification (Fig. 1.a) allows very fast simulation speed but with very low timing accuracy due to implicit hardware/software interfaces.

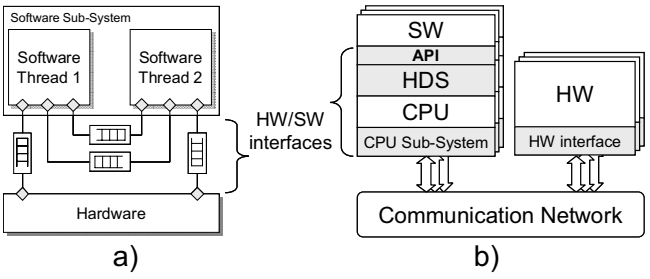

Fig. 1 Hardware/Software interfaces a) in MPSoC design specification b) in MPSoC Architecture

The gap in terms of abstraction level between these two models makes architecture exploration and validation ineffective. The solution to fill up the gap between these two models is to provide flexible and executable abstract hardware/software interface models which allow early validation and effective architecture exploration. Difficulties come from the heterogeneity of the target domain which makes modeling of such interfaces very complex.

The major contribution of this paper is to provide a unified semantic based on SystemC to describe executable models of the hardware/software interface. The proposed solution allows easy architecture exploration through fast simulation speed with performance estimation and flexible interface modeling.

The rest of the paper is organized as follows: We give a review of the related work in section 2. Section 3 describes concepts and details of our approach for hardware/software interface modeling. Experimentation in Section 4 shows a case study using the proposed solution and gives results while section 5 concludes the paper.

II. Related Works

Hardware/software interface design has been studied by different communities, each focusing on different components of the hardware/software interface. [1] and [2] propose automatic generation of wrappers to connect hardware components. In [3], a driver can be synthesized from a formal specification of the target device for a platform-independent virtual environment, and then the virtual environment is mapped to a specific platform to complete the driver implementation. CoWare presents an approach to co design the CPU and its hardware adaptation layer [4]. In Metropolis, designers can define communication primitives and execution rules suitable for design exploration of bus architectures and memory access during synthesis [5]. In [6], a component-based design approach generates an application-specific operating system including device drivers and network interface based on a fixed CPU subsystem architecture. In COSY, the specification is a set of tasks communicating through lossless blocking FIFOs, which are implemented using pre-defined schemes [7]. [8] proposes a service-based interface composition method as an alternative to the conventional layered software architecture of the OSI network protocol stack.

1-4244-0630-7/07/$20.00 ©2007 IEEE.

As opposed to these approaches that focused separately either on the hardware part or the software part, we propose, in this paper, a unified flexible model that generalizes the service-based component approach to encompass the entire hardware/software interface.

III. Hardware/Software Interface Modeling

A. Hardware/Software interface concept

In order to allow concurrent hardware/software design, we need abstract models of both software and hardware components of the interface. An abstract model has to handle two different interfaces: one at the software side using API[1] and one at the hardware side using wires.

The hardware part abstracts the CPU subsystem, which stands for processors and all its peripherals. Hardware designers can rely on a hardware API[2] as shown in Fig. 2. The software part of the interface can depend of the considered software level. Most significant layers are Operating System (OS) and Hardware Abstraction Layer (HAL). Fig. 2 shows a particular model of the interface in which only the HAL layer is abstracted, upper software layers are supposed to be part of the application. This model is called Transaction Accurate Model of the hardware/software interface and provides to software designers a HAL API.

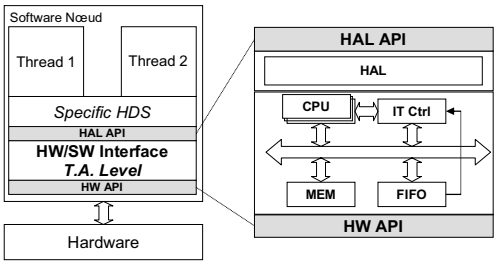

Fig. 2 Transaction Accurate HW/SW interface model

B. Service-based approach

A service-based approach for interface modeling allows separating the implementation from declaration. This separation is essential if a unified representation of hardware and software is needed. A service stands for a functionality which can be provided or required by an atomic unit called *Interface Element*. A hardware/software interface is build by assembling these elements as depict in Fig. 3.

- The software API of the interface is made of a set of software services (1).
- Software services can be required or provided through *Software Elements* (2).
- The hardware API of the interface is respectively made of a set of hardware services (5).
- Hardware services can be provided or required through *Hardware Elements* (4).

Between these two types of elements, we need a third type of element, able to provide or require the two kinds of services. Thus, we introduce the notion of ***Hybrid Element*** (3). The most typical hybrid element in a hardware/software interface is the processor, which can support software execution in one side and hardware interface in the other.

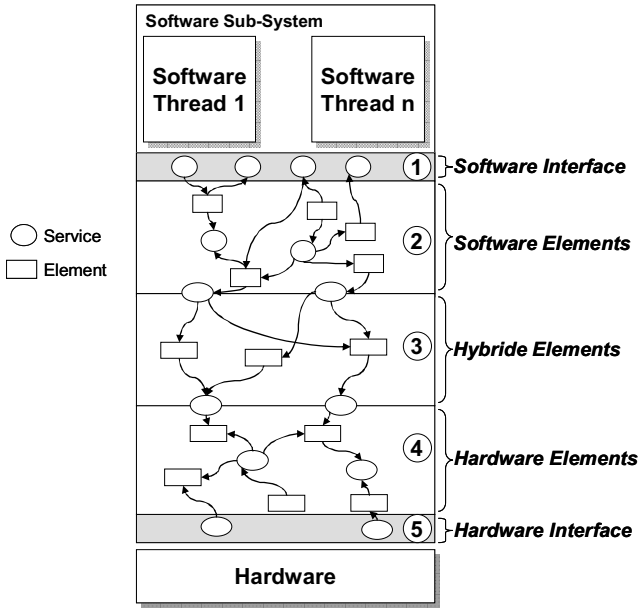

Fig. 3 HW/SW Interface model based on services and elements

C. SystemC as unified model

SystemC [10] has become the preferred development language for hardware/software designers to overcome the design complexity problem. However, SystemC is still hardware-oriented language and doesn't provide by default any construct method to implement and simulate sequential execution and time accuracy of software. Nevertheless, since SystemC is a C++ library, standard class methods can be implemented inside modules and provide software services using the SystemC interface mechanism. Our approach consists in defining a semantic based on SystemC "language" in order to formalize the implementation of the three types of interface element previously presented.

IV. HW/SW Interface Executable Model in SystemC

The following section will describe how SystemC can be used to implement software, hardware and hybrid elements of the hardware/software interface.

A. Software Modeling

In our approach, a pure Software element (Fig. 7.a) is a SystemC module providing only software functions through classical C++ methods. These methods are implemented

[1] Application Programming Interface.
[2] API is also used for the hardware interface.

using the *sc_interface* mechanism in SystemC. In this way, the implementation of a function is totally independent of its definition. This mechanism is mainly used in SystemC for *Transaction Level Modeling* [10] of hardware and needs to be restricted for our case in order to model the real software sequential execution.

Thus, we avoid all specific SystemC functions in the implementation of a function, so we can define a software element as:

- A software element is a SystemC module (*sc_module)*.
- This *sc_module* contains no *SC_THREAD*, *SC_CTHREAD* or *SC_METHOD*.
- A software service is declared as a C++ class derived from the SystemC interface class *sc_interface*.
- The prototype of the function that will implement this service is a method of the service class.
- An element provides a service by deriving from the declaration class and implementing the function. The provided function is made available using *sc_export.*
- A software function can call other functions through classical SystemC ports (*sc_port*).

In the source code detailed in Fig. 4, 2 functions f1 and f2 are declared and implemented by *E1* and *E2* respectively. *f2* is called inside *f1*.

```
class F1 : public sc_interface
{
  virtual int f1(int arg) = 0;
};
class F2 : public sc_interface
{
  virtual int f2() = 0;
};
```

```
SC_MODULE(E1),
  public F1
{
  sc_export<F1>   pE1;
  sc_port<F2>     pE2;

  int f1(int arg)
  {
    return(pE2.f2)
  };

  SC_CTOR(E1)
  :
    pE1("pE1"),
    pE2("pE2")
  {
    pE1(*this);
  }
};
```

```
SC_MODULE(E2),
  public F2
{
  sc_export<F2>   pE2;

  int f2(int arg)
  {
    return(0)
  };

  SC_CTOR(E2)
  :
    pE2("pE2")
  {
    pE2(*this);
  }
};
```

Fig. 4 Example of software elements implementation

Fig. 5 shows the sequential execution model of the previous example. Note that the x-coordinate represents this sequentiallity and not the simulation time which is null. The simulation time used by a software function will be managed by hybrid elements through annotations. This point is described in section III.*C*.

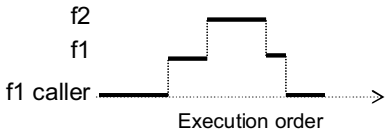

Fig. 5 Software execution model

B. Hardware modeling

Hardware elements are implemented using the *standard* SystemC methodology. In other words, a hardware element is a SystemC module in which one or more SystemC processes implement hardware services. The SystemC representation of a hardware element is shown in Fig. 7.c. At low abstraction level, a hardware service can be accessed through a set of SystemC ports (*sc_port*). For example, a hardware implementation of a *Read FIFO* service using a full handshake protocol could be accessible trough *Request, Data* and *Acknowledge* signals. As this kind of element corresponds exactly to the standard SystemC implementation style, no more details will be given for the hardware elements implementation.

C. Hybrid modeling

Hybrid element is represented in Fig. 7.b. A hybrid element can be seen as a mix of hardware and software elements in the sense that it can implement both software services as describe in section IV.*A* and hardware services described in the previous section.

Thus, hybrid elements can be connected to pure software elements in one side and pure hardware elements in the other. Finally, it can be seen as the synchronization point between software and hardware. The execution time of the entire application can be advanced only by hardware or hybrid elements since software elements should not call SystemC *wait* function. This notion is similar to the "*real world"* were the execution time of software is directly dependent on the underlying hardware and especially the processor on which it is executed.

In Fig. 6, the FIFO is a hybrid element providing the *WRITE* software service for the software side and a set of hardware ports implementing the *READ* service for the hardware side. Data structures inside this FIFO are both accessible from the function and the SystemC process. In order to introduce more accuracy in the model, the software function can call SystemC *wait* with appropriate value.

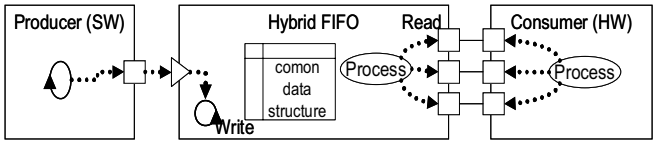

Fig. 6 Example of hybrid element

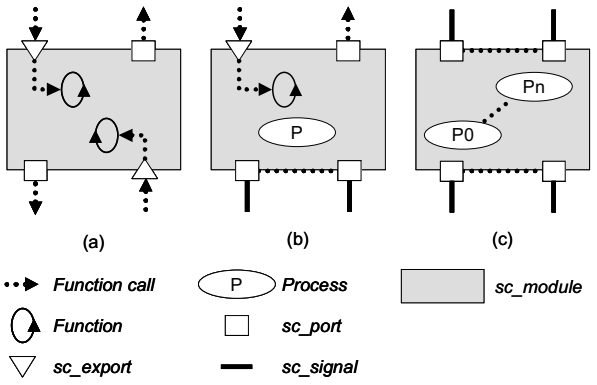

Fig. 7 SystemC representation of a) software b) hybrid and c) hardware elements

D. Flexible HW/SW interface modeling

Using this SystemC coding rules, several elements can implement identical services. A hardware/software Interface implementation relies on assembling these elements coming from libraries. The choice between elements implementing the same service can depend on several criteria like power consumption, area, code size, used memory, speed, etc.

The flexibility of the interface modeling is naturally provided by this approach and could be strengthen with automatic generation tools.

V. Experiments

A. Transaction Accurate Model of the Hardware/Software Interface

In this section, we show how to apply the proposed method to implement a Transaction Accurate model of the hardware/software interface proposed in [9]. In this model, the software API is based on the Hardware Abstraction Layer (HAL). The previous implementation allows validation of the entire application with early, fast and time accurate simulation of the global design but suffers from a lack of formalism and flexibility.

Fig. 8 represents a simplified view of the hardware/software abstract interface at the Transaction Accurate level.

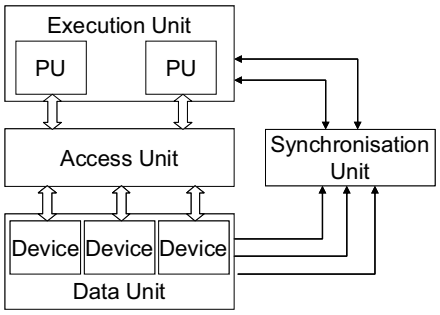

Fig. 8 HAL level abstraction of the CPU subsystem

This model allows parallel computations inside the *Execution Unit*. The *Access Unit* models communications and the data transfers in the CPU subsystem. The *Data Unit* encapsulates the model of physical devices that may hold relevant information from a user point of view. Finally, the *Synchronization Unit* modeled the interrupt controller and all interruption management mechanism.

Table I lists the main software services provided and required by the system programmers. In this list, 2 types of services have to be highlighted:

- The *OS_INIT* service is not provided but rather required by the software API and should be implemented by the Operating System executed on top of this model.
- The *CONSUME* service is provided for simulation purpose only, to allow timing estimation of software execution.

TABLE I
HAL API example

Services	Description
CXT_INIT	`Void init_context(cxt_type, task_h , …)` To initialize the context of a software task
CXT_SWITCH	`void switch_context(cxt_type old, cxt_type new)` Switch context between tasks
OS_INIT	`void os_init(void * args)` This service is *required* by the software interface, and *must* be provided by the operating system
READ	`int read(unsigned int addess)` Direct read access to a specific address
WRITE	`void write(int data, unsigned int addess)` Direct write access to a specific address
ATTACH_ISR	`Int attach_isr(int Id,ISR_h h, void * args)` Register a interrupt sub routine into the synchronization unit
CONSUME	`int consume(int cycles)` This service is not specific to HAL but allow annotated code to be simulated with time accuracy

B. Transaction Accurate model implementation

The implementation of the *Transaction Accurate* model using the notion of service and elements is detailed in Fig. 10. The main part of the software API is implemented by the *Processing Unit* hybrid element (PU). This can be explained by the fact that we are implementing a low level API and the element stands for a processor. Since the PU is a key element in this model, the rest of this section will focus on its implementation.

The role of the PU element is to allow the application and the Operating System on top of the HAL API to be executed sequentially taking into account the time aspect. To do this, the PU is modeled as a hybrid element containing one SystemC process. After design elaboration, this process will be called by the SystemC scheduler and will be executed until a call to the *wait* function. After low level initialization,

the PU element calls the software *OS_NIT* service provided by the Operation System which is the entry point of all software. In order to simulate the execution time of the software, this one has been annotated statically in the source code by calling the *CONSUME* service provided by PU. This consume will finally call the SystemC *wait* function. Since the software call graph is executed in the PU process context, all calls to SystemC *wait* are considered sequentially. This correctly models the sequential timing behavior of software execution.

This mechanism initially introduced in [9] allows performing performance estimation with more or less accuracy depending on the precision of the annotation (see Fig. 9). *Wait* function can also be called indirectly by software, for example during a *READ* or *WRITE* communication access to account for bus latency. Even if this *wait* is implemented in other elements, it will be executed in the context of the PU process.

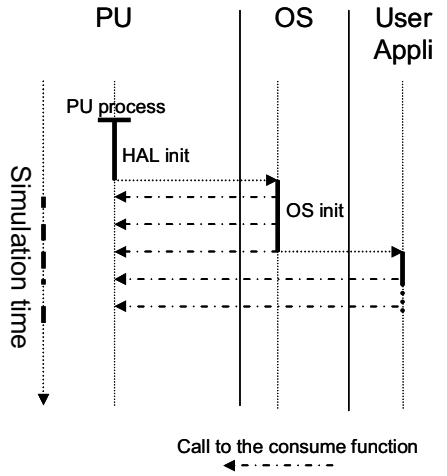

Fig. 9 Consume mechanism for software time estimation

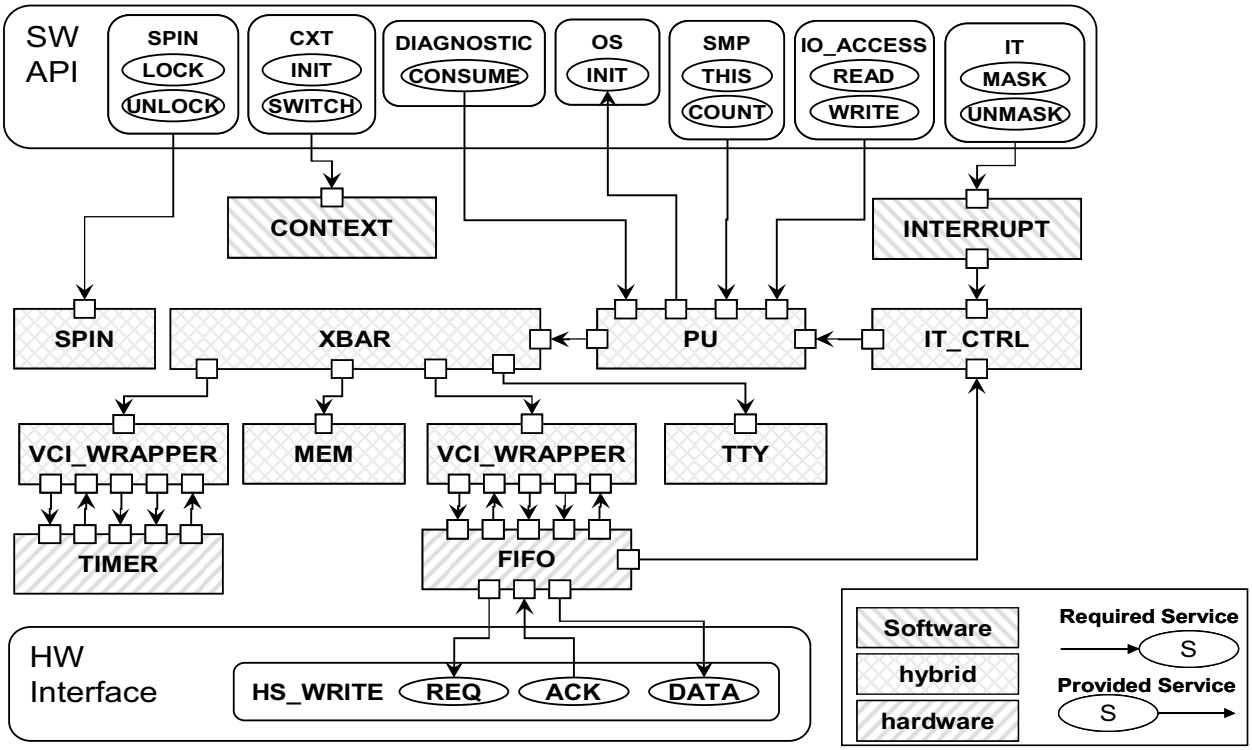

Fig. 10 Transaction Accurate level of the HW/SW interface mode

C. Application example

a. Overview : Application Specification

Fig. 11 represents the SystemC specification, also called *System Level Model* of the application used for the experimentation. This example is composed of two software threads and one hardware module communicating through FIFO. At this level, the interfaces between hardware and software are totally implicit and all modules are executed in a concurrent way. This model offers high simulation performance but suffers from a very low timing accuracy.

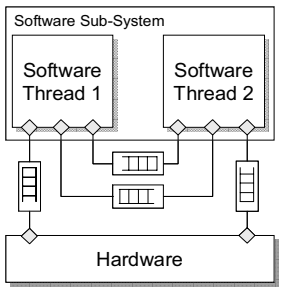

Fig. 11 Specification of the application example

b. *ISA/RTL model of the application*

At the opposite side, the Virtual Prototype using the ISA/RTL level offers a full detailed hardware/software interface description. In this model, software has been cross-compiled for a specific processor (MIPS-R3000 in our case) and is interpreted by an Instruction Set Simulator. This software is composed of a multi-threaded application, a POSIX compliant Operating System and a Hardware Abstraction Layer. The Hardware part is implemented in SystemC RTL.

The aim of this experimentation is to replace the hardware/software interface of the ISA/RTL model with the abstract executable model described in the previous section. As depicted in Fig. 12, the HAL software layer, the ISS and the processor sub-system will be removed and replaced by the model of the hardware/software interface at the Transaction Accurate Level.

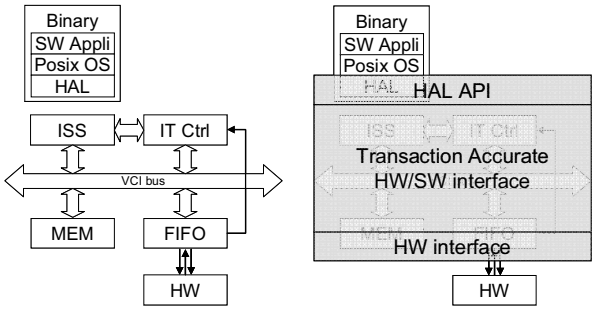

Fig. 12 Virtual Prototype and Transaction Accurate model of the application

c. *Results*

In this experiment, we have run the simulation at three abstraction levels (System Level, Transaction Accurate level and Virtual Prototype). TABLE II gives results in term of simulation speed and shows an interesting speed-up compared to the Virtual Prototype model of the application. Transaction Accurate model of the hardware/software interface also provides interesting capabilities in terms of validation facilities. These capabilities are related to both hardware and software. Fig. 13, shows hardware waveforms of communication port and software information of the current executed thread ID.

TABLE II: speed up results

Abstraction level	Simulation time	Speed up
System Level	0.1 s	x 15594
Transaction Accurate	5.65 s	x 276
Virtual Prototype	1560.4 s	x 1

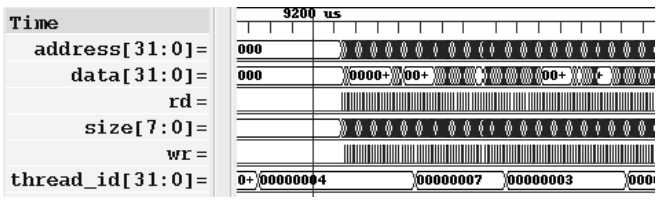

Fig. 13 Simulation waveforms at Transaction Accurate level

These debug information are helpful for the validation of the sensitive parts like Operating System and for performance estimations. Elements of the hardware software interface can easily be instrumented in order to highlight information like bus contention, software thread synchronization mechanisms, memory usage etc.

VII. Summary and Conclusions

In this paper we presented a unified model based on SystemC to implement the overall hardware/software interfaces. This approach has been used to implement an abstract model of the interface proposed in a previous work. This model has been applied to an application example.

Future work will consist to enrich the Transaction accurate model to extract key information for architecture exploration. Finally, next objectives are automatic generation of the hardware/software interfaces from a library of basic elements and to extend this approach to other abstraction levels.

References

[1] Smith, J., De Micheli, G., Automated Composition of Hardware Components, *Proc. 35th Design Automation Conf. (DAC'98)*, ACM Press, San Francisco, CA, 1998.

[2] Passerone, R., Rowson, J., Sangiovanni-Vincentelli, A., Automatic Synthesis of Interfaces between Incompatible Protocols, *Proc. 35th Design Automation Conf. (DAC 98)*, ACM Press, San Francisco, CA, 1998.

[3] Wang, W., Malik, S., and Bergamaschi, R.A., Modeling and Integration of Peripheral Devices in Embedded Systems, *Design, Automation and Test in Europe Conference and Exhibition (DATE'03)*, Munich, Germany, 2003.

[4] Vercauteren, S., Lin, B., De Man,.H., Constructing Application-Specific Heterogeneous Embedded Architectures from Custom HW/SW Applications, *Proc. 33rd Design Automation Conf. (DAC'96)*, ACM Press, Las Vegas, NV, 1996.

[5] Balarin F., Watanabe Y., Hsieh H., et al., Metropolis: An Integrated Electronic System Design Environment, *IEEE Computer*, April 2003.

[6] Cesário, W., Baghdadi, A., Gauthier, L., et al., Component-Based Design Approach for Multicore SoCs. . *Proc. 39th Design Automation Conference (DAC'02)*, ACM Press, New Orleans, LA, 2002.

[7] Brunel, J-Y., Kruijtzer, W. M., Kenter, H. J. H. N., et al. COSY Communication IP's, *Proc. 37th Design Automation Conf. (DAC'00)*, ACM Press, New York, NY, 2000.

[8] Zitterbart, M., Stiller, B., Tantawy, A.N., A Model for Flexible High-Performance Communication Subsystems, *Selected Areas in Communications, IEEE Journal on*, vol. 11, no. 4, May 1993

[9] A. Bouchhima, I. Bacivarov, W. Youssef, M. Bonacciu, A. A. Jerraya, "Using Abstract CPU Subsystem Simulation Model for High Level HW/SW Architecture Exploration", ASP-DAC 2005 proceedings,18-21 January 2005, Shanghai, China, 2005.

[10] SystemC, http://www.systemc.org.

[11] Adam Rose and Stuart Swan and John Pierce and Jean-Michel Fernandez and Cadence Design Systems, « *Transaction Level Modeling in SystemC* », Mentor Graphics; Cadence Design Systems.