Validation in a Component-Based Design Flow for Multicore SoCs

Gabriela Nicolescu Sungjoo Yoo Aimen Bouchhima Ahmed Amine Jerraya
SLS Group, TIMA Laboratory
46, Av. Felix Viallet, 30831, Grenoble, France
{Gabriela.Nicolescu, Sungjoo.Yoo, Aimen.Bouchhima, Ahmed.Jerraya}@imag.fr

ABSTRACT

Currently, since many SoCs include heterogeneous components such as CPUs, DSPs, ASICs, memories, buses, etc., system integration becomes a major step in the design flow. To enable this integration, we use a design approach called component based-design approach. In this approach, the validation of system integration takes most of design efforts. This paper presents an automatic method of SoCs design validation. Based on a generic simulation wrapper architecture, the presented method provides automatic generation of executable models throughout different stages of SoC design flow. A case study of validating a VDSL application shows the effectiveness of the method.

Categories and Subject Descriptors

J.6. [**Computer-Aided Engineering**]: Computer-Aided Design

General Terms

Design, Verification.

Keywords

SoC, Component-Based Design, Validation, Cosimulation, Abstraction Levels

1. INTRODUCTION

ITRS roadmap predicts that in 2004, 70% of ASICs will include at least one embedded instruction set processor [1]. SoCs already include several instruction-set processors in such applications as mobile terminals (e.g. GSM [2]), set-top boxes (e.g. pnx 8500 from Philips [3]), game processors (e.g. PlayStation2 from Sony [4]) and network processors [5]. Most system and semiconductor houses are developing and using design environments and target architecture platforms allowing the integration of multiple cores (CPU, DSP, MCU, co-processors, memory, etc.) [21] [22].

In SoC design with multiple cores, designers need to integrate **heterogeneous** cores. They are heterogeneous in terms of communication protocols, abstraction levels, and specification languages. Communication protocols of different cores (e.g. AMBA [22] and Open Core Protocol [8]) need to be adapted via protocol conversion, buffering, (de)multiplexing, etc. [24]. The designers need also to integrate cores described at different abstraction levels (e.g. an interconnection between a behavioral core and an RTL core). The specification languages of cores can be different (e.g. cores in VHDL, C, SDL, etc.).

In SoC design, most of design cycle (more than 60%) is devoted to validation [7]. Validation needs to be performed as many times as possible throughout the design cycle to obtain a reliable SoC implementation. However, for the designers, the validation of multicore SoCs is one of the most difficult design steps. It is mostly because the designers need to make simulatable the SoC specification with heterogeneous cores. To be more specific, for instance, the designers need to make simulatable the entire system specification, which consists of cores with different abstraction levels and languages. In current SoC design practice, since such a job is done manually, it is time consuming. **It needs to be automated to shorten the design cycle thereby allowing the designers to focus on the validation of system specification**.

In several commercial SoC design environments, the job of making simulatable the SoC specification with heterogeneous cores can be done mostly in a manual manner. For instance, Coware N2C enables mixed-level cosimulation for system specifications with behavioral and RT level cores [9]. In the case of SystemC, a method of mixed-level simulation based on the concept of **interface** is presented though it does not automate the generation of mixed-level interfaces [10].

Basically, for the simulation of SoCs with heterogeneous cores, commercial and academic tools are based on the concept of **simulation wrapper** to adapt different abstraction levels or languages. The wrapper concept is not new and has been widely used. However, the concept has not been used in a systematic way. In other words, there has been no systematic method of generating wrappers for the simulation of SoCs with heterogeneous components.

Such systematic methods of generating wrappers become more required as the abstraction levels of system description are getting higher and the number of abstraction levels is getting larger. For instance, in our SoC design flow called component-based SoC design flow, we use three abstraction levels for software cores, two for communication channels, and two for hardware cores. In such a case, the design of wrappers becomes much more difficult than the cases where only two abstraction levels, i.e. functional and RT levels, are used. To enable more extensive design space exploration and higher design productivity, the designers will use more and higher abstraction levels. Thus, systematic methods of generating wrappers will become more and more required.

In this paper, we present a systematic method of wrapper generation for multicore SoC design. To enable automatic generation of wrappers, we use a **generic wrapper architecture** and a **wrapper generation flow**. The generic wrapper architecture enables the adaptation of different communication protocols, abstraction levels, and specification languages as well as the reuse of wrapper components in the simulation of different SoCs. Since the same generic architecture is used for any SoC validation, wrappers can be generated automatically by instantiating components in the wrapper architecture.

This paper is organized as follows. Section 2 gives a review of simulation methods for systems with heterogeneous cores. Section 3 introduces our SoC design flow. Section 4 explains the generic wrapper architecture and wrapper generation. Section 5 gives a case study of applying our method to the design of an industrial SoC. Section 6 concludes the paper.

2. RELATED WORK

For the simulation of cores with different abstraction levels, the bus functional model (BFM) [14] is widely used. It is a conventional method to connect functional simulation models and cycle–accurate simulation models. It performs transformation between functional memory access and cycle-accurate memory access. SystemC provides a concept of interface for mixed abstraction level simulation [10]. Mixed-level interfaces have to be designed manually, which can be a source of errors and of lost of design time. Moreover, designers need also to design wrappers for multi-language simulation. SystemC^{SV} extends SystemC and makes the simulation of all of its abstraction levels of interfaces [23]. Thus, the interconnection of interfaces with different abstraction levels is possible. However, automatic generation of mixed-level simulation interface is not supported.

As a study of multi-language simulation methodology, Valderrama *et. al.* presents a tool called VCI for the automatic generation of cosimulation interface in C-VHDL cosimulation [17]. Coste *et. al.* presents a tool called MCI that enables multi-language cosimulation [18]. These solutions are very efficient for multi-language simulation, but they don't treat in a systematic way the multi-level cosimulation.

VSIA [26] proposes VCI (Virtual Component Interface) and SLI (System Level Interface) for heterogeneous components interconnection, where several abstraction levels are taken into account. VCI enables only point-to-point and unidirectional connections. VSIA standard presents a design guide (not an automatic design method) to enable multi-level simulation consisting of VCI and SLI by using the wrapper concept. Our work shares the same approach and our contribution is to enable automatic design of such wrappers.

3. COMPONENT-BASED SoC DESIGN FLOW

Figure 1 illustrates our component-based SoC design flow. The RT level architecture consists of processors connected to the communication network via wrappers. Wrappers are constructed in the form of software (e.g. operating system) as well as in the form of hardware. Our design flow starts from a system specification called **virtual architecture specification** and gives automatically an RTL implementation by generating hardware and software wrappers.

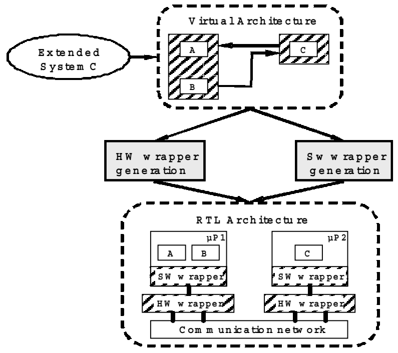


Figure 1. Component-based SoC design flow.

3.1 Virtual Architecture Specification

The virtual architecture specification represents a system as a hierarchical network of modules and communication channels (see Figure 2). Modules and communication channels can have different abstraction levels, communication protocols or they can be described in different specification languages. To enable the interconnection specification of such heterogeneous modules and communication channels, we use the concepts of virtual component, virtual port and virtual channel.

A **virtual component** consists of a module and its wrapper. The wrapper is composed of **internal** and **external ports**. The internal ports are the ports of the module and the external ports enable interconnections with external communication channels. The internal and external ports are grouped in **virtual ports** (see Figure 2). In a virtual port, there can be an *n* to *m* (*n* and *m* are natural numbers) correspondence between internal and external ports. A **virtual channel** is a set of several channels having a logical relationship (e.g. multiples nets belonging to the same communication protocol of AMBA [22]).

Figure 3 exemplifies the concepts of virtual component, virtual port and virtual channel. The module in the figure is connected to the communication channels through a virtual port. This module is described at a high abstraction level and it uses high-level communication services (in the figure, it calls a communication service *send* on the port P1) to communicate with other modules via communication channels. The module has to be connected to RT level signals with handshake protocol. Thus, the virtual port used for this connection has an internal port (the port P1 of the module) at a high abstraction level (called OS architecture level to be presented in Figure 4) and three external at RT level. Internal and external ports are parameterized for refinement and validation purposes. Examples of some parameters are illustrated in Figure 3.

In our design flow, the virtual architecture is described using an extension of SystemC, where the presented new concepts are introduced.

3.2 Abstraction Levels in SoC Component-Based Design

Our design flow enables incremental SoC design through multiple abstraction levels for hardware and software components and communication networks. Figure 4 shows abstraction levels used in our SoC design flow.

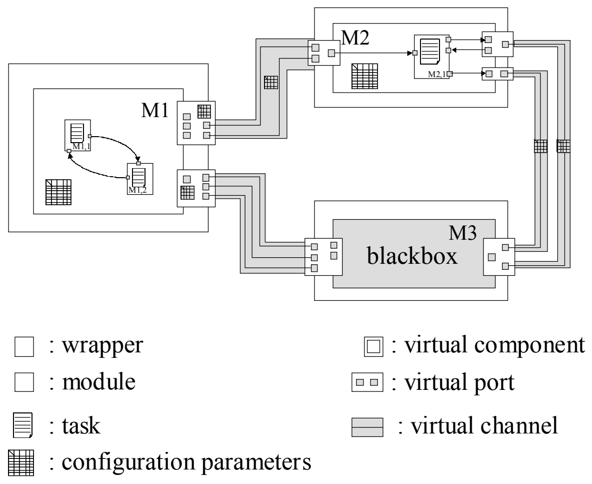
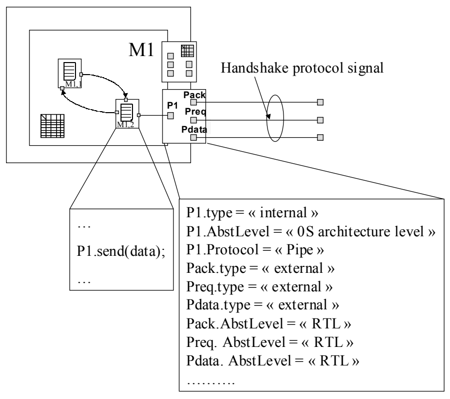


Figure 2. A virtual architecture specification.



```
…
P1.send(data);
…
```

```
P1.type = « internal »
P1.AbstLevel = « OS architecture level »
P1.Protocol = « Pipe »
Pack.type = « external »
Preq.type = « external »
Pdata.type = « external »
Pack.AbstLevel = « RTL »
Preq. AbstLevel = « RTL »
Pdata. AbstLevel = « RTL »
……….
```

Figure 3. Examples of virtual component, virtual port and virtual channels.

For the design of software wrappers, we use three abstraction levels as follows:

- at **OS architecture level,** the OS is abstract and only the system calls corresponding to the OS services are visible.
- at **driver level**, the implementation of the OS services is fixed but device drivers are still abstracted. Thus, the hardware on which software is executed (e.g. CPU, memory) can be variable. The application code is extended with OS layers implementing OS services (e.g. task scheduling management, interruption management, etc.).
- at **instruction set architecture (ISA) level**, software is described in assembly code specific to the fixed hardware (e.g. processor, memory).

For hardware design, we use two abstraction levels:

- at **behavioral level,** only the functionality is fixed but the physical implementation of this functionality is not yet fixed. Communication with the rest of system is realized via high-level communication services.
- at **register transfer level (RTL)** the physical implementation including communication protocols is fixed.

For communication networks design, we use two abstraction levels:

- at **abstract netlist level,** communication channels provide communication services (e.g. pipe, semaphore, etc.). The implementation of the services is not yet fixed.
- at **physical netlist level**, communication is modeled by physical signals (e.g. shared buses or point-to-point interconnection). At this level, all the communication details (i.e. communication protocols, interruption management, address decoding, etc.) are fixed

Note that, throughout the SoC design flow, components and communication networks can be situated at any of the above-mentioned abstraction levels.

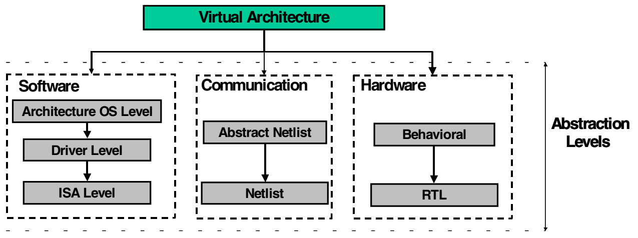


Figure 4. Abstraction levels in our SoC design flow.

4. SoC VALIDATION IN COMPONENT-BASED SoC DESIGN

The virtual architecture specification is only a specification model and it is not executable as is since the internal and external ports of the virtual ports are heterogeneous (they can have different abstraction levels and different specification languages). Thus, we need to make it executable by adapting different abstraction levels and different specification languages.

4.1 Simulation Wrapper

To resolve the problem, we use simulation wrappers for software and hardware modules as shown in Figure 5. Modules are connected with the cosimulation network via simulation wrappers. In fact, the components are separated from the cosimulation bus by simulation wrappers that implement the simulation model of wrappers specified in the initial virtual architecture.

As shown in Figure 6, the simulation wrapper has a generic architecture. A wrapper needs to adapt different abstraction levels and different languages. **Since an abstraction level and a language are, in general, independent of each other, we have two interfaces in the wrapper architecture**: simulator interface to adapt different languages and communication interface to adapt different abstraction levels.

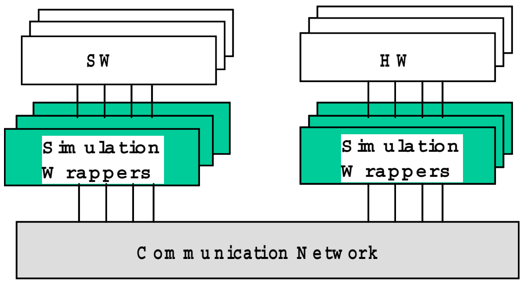


Figure 5. Simulation model for multiprocessor SoC.

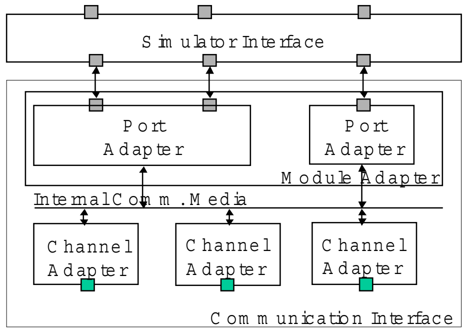


Figure 6. Generic architecture of simulation wrapper.

- **Simulator interface**

For the simulator interface, we use the methodology already presented in the literature [17] [18]: to adapt different languages, i.e. different simulators, we use simulation libraries (provided by the simulators) for communication between different simulators (e.g. CLI/PLI for VSS Vhdl simulator, S-function for Simulink/Matlab simulator) and we use IPC (Inter Process Communication), e.g. Unix shared memory or socket, for data transfer and synchronization between different simulators.

- **Communication interface**

The communication interface presents a modular internal structure presented in Figure 6. It is composed of three types of elements: module adapter (MA), channel adapter (CA), and internal communication media (ICM).

- **module adapter** (MA) provides module with required communication services. It performs also data conversion and channel resolution. As we shown in Figure 6, MA may be decomposed in several port adapters (PA), where a PA represent a grouping of related ports. For instance, to represent a bus interface for AMBA, we use an elementary interface by grouping all the ports of AMBA interface (e.g. address bus, data bus, control signals, etc.); in the case of multi-port memory, we need to use as many elementary interfaces as the number of ports.

- **internal communication media** (ICM) transfers data between module adapter and channel adapter. ICM can be implemented in any forms such as remote procedure calls (RPCs) or signals. In our simulation environment, we use RPC as the ICM.

- **channel adapter** (CA) enables the module to access the external channel. To do that, after receiving a channel access request (via MA) from the module, it uses channel communication services. To each channel, a channel adapter is assigned.

As an example of wrapper, Figure 7 shows a simulation model of the virtual architecture specification in Figure 2. Module M3 is simulated in a different simulation environment. Thus, in the simulation wrapper of module M3, we use a simulator interface as well as a communication interface. Since virtual modules of M1 and M3 have two virtual ports (see Figure 2), the MA of each of them has two PAs. The virtual module of M2 has three virtual ports. Consequently, the MA of M2 has three PAs.

4.2 Hardware Simulation with Software at Different Abstraction Levels

We apply the wrapper architecture to all the possible combinations of different abstraction levels presented in Figure 4. In other words, we have the wrapper architecture orthogonal to the abstraction levels. That is why we call the wrapper architecture **generic**.

In this paper, we will illustrate the following combinations of abstraction levels:

- RT level for the hardware – abstract netlist communication – OS Architecture level for the software
- RT level for the hardware – physical netlist communication – ISA level for the software

To allow RT level hardware simulation with software at OS Architecture level, the communication interface adapts the cycle accurate communication of the hardware part and the communication by high level communication primitives of the software part.

For the RT level hardware simulation with the software at ISA level, the communication interface acts as a classical BFM. We use corresponding simulation models: instruction set simulators (ISSs) for processors, HDL simulators for HW cores, etc. In the case of processor simulation, two types of simulation models (ISS and native OS simulation) are supported. For the native simulation we use an OS simulation model presented in [27].

4.3 Automatic Generation of Simulation Wrappers

The overview of simulation environment in our component-based design flow is illustrated in Figure 8. The simulation wrapper generator receives as input a virtual architecture specification. To generate wrappers, we use a simulation library where we have templates for ports adapters and channel adapters.

The wrapper generator analyzes the virtual architecture specification and determines the functionality of each of required wrappers, i.e. adaptation of different languages or different abstraction levels, or both and port grouping in elementary interfaces. According to the functionality, for each wrapper generation, the generator extracts and configures the appropriate elements (CA, MA/PA) from the simulation library. New CA, PA are developed by core providers and/or the designers. They can be reused for different SoCs implementations.

Figure 8 also shows simulation models in our simulation flow. Figure 8.c illustrates a simulation model in the case where the hardware (at RTL or behavioral level) is simulated with software at OS Architecture Level. Figure 8.a and Figure 8.b illustrate simulation models for systems containing software at ISA level, hardware at RT level and the communication at physical netlist level. Figure 8.a shows the ISS based simulation model for the software part and Figure 8.b shows the case where a native simulation model is used for the software part.

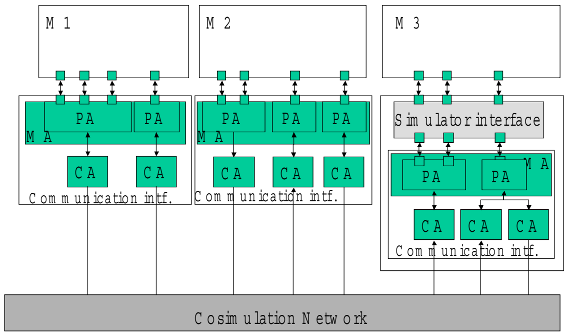


Figure 7. Example of simulation model of a virtual architecture specification

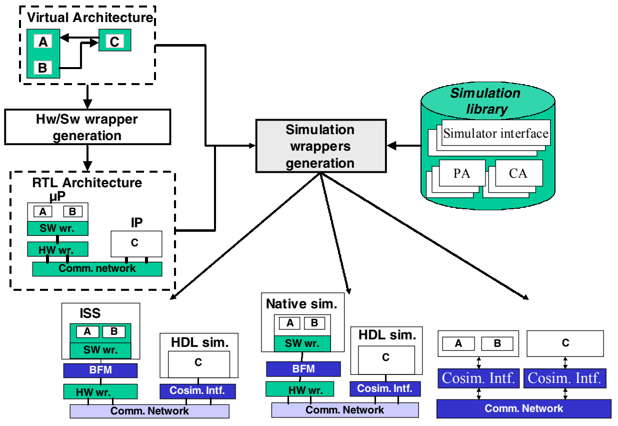


Figure 8. Automatic generation of simulation models and wrappers for multiprocessor SoC validation.

5. COMPONENT-BASED VALIDATION OF A VDSL APPLICATION

This section demonstrates the application of the presented method to the component-based SoC design of a VDSL application. We designed a part of a VDSL system using two ARM7 processors. The part we design as a multi-processor SoC is shown in Figure 9 (the shaded region). The VDSL core functions, analog interface, and a DSP core are implemented in a third-party block.

5.1 Virtual Architecture Specification of the VDSL Modem

Figure 10 shows a graphical representation of the virtual architecture specification of the VDSL system. Modules VM1 and VM2 correspond to the CPU1 and CPU2 in Figure 9. VM3 correspond to the rest of the VDSL designed part.

The software applications on the two CPUs (VM1 and VM2 in Figure 10) are described at OS Architecture Level using a high level language (C++). Intra-processor communication in VM1 and VM2 and inter-processor communication between VM1 and VM2 and between VM2 and VM3 are described at abstract netlist level. As explained previously, VM3 is at RTL. Since the abstraction level of VM3 (RTL) and that of channels (abstract netlist level) connected to VM3 are different, the wrapper of M3 is composed of virtual ports containing RTL internal ports (ports specific to M3) and high-level external ports (for the connection with the virtual channels connected to VM3). For instance, in Figure 10, virtual port *Pvirt* of VM3 contains an external high level port that is connected to the abstract channel (fifo) and three internal RT-level ports of which the communication protocol is called a guarded register protocol.

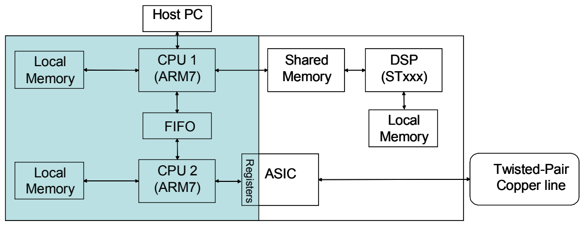


Figure 9. VDSL modem application

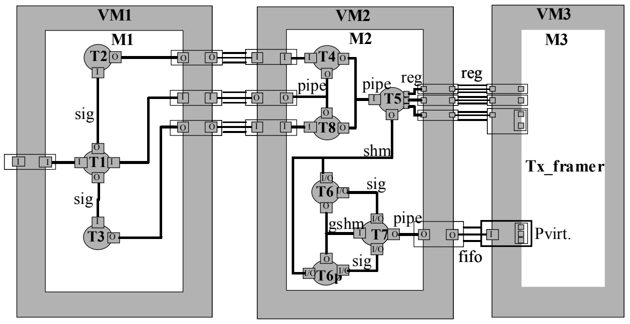


Figure 10. Virtual architecture specification for the VDSL application.

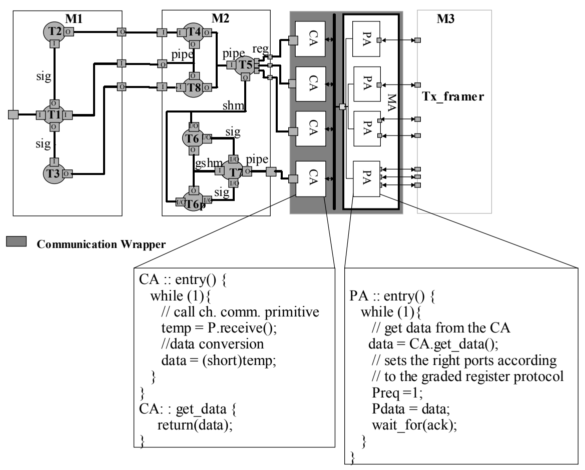
5.2 Results

To obtain the simulation model of the VDSL virtual architecture specification, a simulation wrapper has to be generated for module VM3. In Figure 11, the shaded region illustrates the simulation wrapper of module M3 generated by the simulation wrapper generator. The simulation wrapper adapts module M3 to the rest of the system. ICM is implemented as RPC (remote procedure call). PAs transfer data between the ICM and module M3, Tx_framer respecting the RTL protocols required by module M3. CAs transfer data between the ICM and the external abstract channels by calling communication services provided by the abstract communication channels.

To be more specific, in the case of virtual port *Pvirt* explained in Section 5.1, the CA reads data from the abstract communication channel (pipe) by calling communication services of the channel (e.g. send/receive) (see Figure 11). The PA takes the data from the CA by calling an RPC (as the ICM), performs channel resolution and sets the right ports of M3 according to the guarded register protocol (see Figure 11).

In terms of combinations of different abstraction levels presented in Figure 4, the simulation wrapper in Figure 11 combines different abstraction levels: OS architecture level (SW) – abstract netlist level (channel) – RTL (HW).

Table 1 summarizes results obtained for the simulation model and wrapper generation of the VDSL application. Assuming that in manual simulation model building, the efforts of building simulation models is proportional to the code size of simulation models, we compare the code sizes of simulation models and the numbers of interconnections between modules and communication networks. Compared with the code size of virtual architecture specification, that of mixed-level simulation model is three times larger. In our simulation flow, the designer has only to write a virtual architecture specification (in this case, 150 lines of code). In our case, it took around one hour and half. Thus, compared with the case where the designer needs to build manually the mixed-level simulation model (475 lines of code), automatic generation of simulation wrapper gives a gain of factor 3 in terms of efforts in building mixed-level simulation models.



```
CA :: entry() {
   while (1){
      // call ch. comm. primitive
      temp = P.receive();
      //data conversion
      data = (short)temp;
   }
}
CA: : get_data {
    return(data);
}
```

```
PA :: entry() {
   while (1){
      // get data from the CA
     data = CA.get_data();
      // sets the right ports according
      // to the graded register protocol
      Preq =1;
      Pdata = data;
      wait_for(ack);
   }
}
```

Figure 11. Simulation model of VDSL virtual architecture

		Code size (lines)	No. of interconnections
Virtual architecture specification		150	21
Mixed-level	Generated sim. model	475	60
	Generation time	90 s	
RTL	Generated sim. model	600	187
	Generation time	10 min.	

Table 1. Results of simulation model and wrapper generation

The generation of mixed simulation models takes 90 second on a Linux PC (500MHz Pentium). This time encloses also the time needed to parse the intermediate format of our application (in our case in XML). We have spent most of the time in writing the virtual architecture specification.

The simulation runtime of mixed-level specification (4 min.) was much less then the time of building mixed-level simulation model (1h30 + 90s. in our case, around 4h30 in the case of manually building).

After the validation of the virtual architecture and the generation of RTL architecture as shown in Figure 1, we generated an RTL simulation model. Table 1 shows the code size and generation time of the RTL simulation model. Since our simulation flow generates RTL simulation models, the designer can obtain additional gain in reducing the design cycle to build RTL simulation models.

6. CONCLUSION

In this paper, we presented a validation method for multi-processor SoC component-based design. The presented method is based on automatic simulation wrapper generation starting from a virtual architecture specification, where different cores can be described at different abstraction levels and/or in different specification languages. Our case study of validating a mixed-level specification of a commercial VDSL modem application shows the effectiveness of the presented method in reducing the design cycle.

REFERENCES

[1] ITRS, http://public.itrs.net/.
[2] A. Nagari, *et al.*, A 2.7V 11.8 mW Baseband ADC with 72 dB Dynamic Range for GSM Applications, 21st annual Custom Integrated Circuits Conference, San Diego, 1999.
[3] http://www-us.semiconductors.philips.com/platforms/nexperia/
[4] Oka and Suzuoki, "Designing and Programming the Emotion Engine", IEEE Micro, vol. 19:6, Nov/Dec 1999, pp. 20-28.
[5] P. Paulin, F. Karim, and P. Bromley, "Network Processors: A Perspective on Market Requirements, Processor Architectures and Embedded S/W Tools", Proc. of DATE, Germany, 2001.
[6] A.A. Jerraya, "Application-Specific Multiprocessor Systems-on-Chip", SASIMI 2001, The Tenth Workshop on Synthesis And System Integration of MIxed Technologies, Nara, Japan, October 18-19 2001.
[7] M. Keating and P. Bricaud, "Reuse Methodology Manual", Kluwer Academic Publisher, 1999.
[8] D. Wingard, "MicroNetwork-Based Integration for SOCs", Proc. 38th Design Automation Conference 2001, Las Vegas, June 2001.
[9] Coware Inc., "N2C" , http://www.coware.com/cowareN2C.html/;
[10] Synopsys, Inc., "SystemC, Version 2.0", http://www.systemc.org/.
[11] T . W Albercht, *et al.*, "Hw/Sw CoVerification Performance Estimation & Benchmark for a 224 Embedded RISC Core Design", Proc. Design Automation Conf., June 1998.
[12] Mentor Graphics, Seamless CVE, http://www.metorg.com/semless
[13] Eaglei, http://www.synopsys.com/products/hwsw/eagle_ds.html
[14] L. Semeria et A. Ghosh, Methodology for Hardware/Software Co-verification in C/C++ , Proc. ASPDAC, Jan. 2000
[15] F. Balarin et al., "Haedware-Software Co-design of Embedded systems", Kluwer Academic Publishers, 1997.
[16] P.H. Chou, et al., « The Chinook Hardware/Software Co-Synthesis System », Proc. of International Symposium on System Synthesis, 1995.
[17] C. Valderrama *et al.*, "Automatic VHDL-C Interface Generation of Distributed Cosimulaiton : Application to Large Design Exemples ", Design Automation for Embedded Systems vol. 3, no. 2/3, Apr. 1994.
[18] P. Coste, *et al.,* "Multilanguage Design of Heterogeneous systems", Proc. Int. Workshop on Hardware-Software Codesign, May 1999.
[19] W.O. Cesario, *et al.*, "Colif: A design representation for application-specific multiprocessor SOCs", IEEE Design & Test of Computers, Vol. 18, Issue: 5, pp. 8-20, Sept.-Oct. 2001.
[20] D.J.G. Mestdagh, M.R. Isaksson, P. Odling, "Zipper VDSL: A Solution for Robust Duplex Communication over Telephone Lines", IEEE Communication Magazine, May 2000.
[21] IBM Corp., CoreConnect Bus Architecture, available at http://www-3.ibm.com/chips/products/coreconnect/
[22] ARM Ltd. AMBA Specification, available at http://www.arm.com/armtech/AMBA_Spec?OpenDocument
[23] R. Siegmund and D. Muller, "SystemC^{SV} – An Extension of SystemC for Mixed Multi-Level Communication Modeling and Interface-Based System Design", Proc. Design Automation and Test in Europe, Mars 2001.
[24] J. Smith and G. De Micheli, "Automated Composition of Hardware Components", Proc. Design Automation Conference, 1998.
[25] Motorola, Inc., available at http://e-www.motorola.com
[26] C.K. Lennard, *el al.* "Standards for System-Level Design : Practical Reality or Solution in Search of a Question ? ", Proc. of Design Automation and Test in Europe, Mars 2000.
[27] S. Yoo, G. Nicolescu, L. Gauthier, A.A. Jerraya, "Automatic Generation of Fast Timed Simulation Models for Operating Systems in SoC Design", Proc. Design Automation an Test in Europe, Mars 2001