A Mixed-level Virtual Prototyping Environment for Refinement-based Design Environment

Sanggyu Park, Sangyong Yoon, and Soo-Ik Chae
School of electrical engineering and computer science, Seoul National University
San 56-1 Sillim-dong, Gwanak-gu, Seoul, Korea
{sanggyu, syyoon}@sdgroup.snu.ac.kr

Abstract

The Communication Architecture Template Tree (CATtree) is an abstraction of the specific range of communication functions and architectures, which can facilitate system function capture and communication architecture refinement. In this paper, we explain a TLM-RTL-SW mixed-level simulation environment that is useful for the functional verification of partially refined system models. We employed SystemC, GNU Gdb and a HDL simulator for the simulation of CATtree-based TLM, SW and HW, respectively. We also employed a new operating system, DEOS so that each SystemC-based TLMs can be cross-compiled to be executed as software models on the target processors. We evaluated the flexibility and simulation performance of the virtual simulation environment with an H.264 decoder design example.

1 Introduction

Recently, several works [1][2][3] have been published about the refinement-based design (RBD) methodology, where its system function model is first captured at a higher abstraction level and then a part of it is refined into an architectural model until the whole system is fully refined. A main concept in this methodology is orthogonalisation: function from architecture and computation from communication [4]. Although the orthogonalisation concept was first introduced for the platform-based design, it is also important to the refinement-based design. In the processor-based design such as MPSoC, the orthogonalisation of function and architecture is obvious because function is usually implemented in software. For the separation of computation and communication, researchers have proposed a layering concept to bridge the gap between the software computation and the hardware communication system [5].

In many previous works, the FIFO and the bus are assumed for communication although these assumptions limit the communication design space substantially. The FIFO is an abstraction of the point-to-point (P2P) communication. To find an optimal architecture of a FIFO channel, a set of architecture templates should be provided that covers a wide range of the possible design space. The bus is a primitive building block in many current system-on-chip designs. However, the bus is not a good abstraction of communication because many other communication functions cannot be captured with the bus model. Furthermore the bus model already contains too many architectural decisions that limit the design spaces to be explored.

Many design approaches that employ the hardware libraries assume that all necessary hardware components can be provided. However, it is almost impossible to provide such hardware libraries for all applications and additional hardware components should be designed and verified in a specific design. Therefore a design flow should provide an efficient design flow for hardware components, which is utilized in the refinement process.

In one of our previous works [6], we proposed a concept of Communication Architecture Template Tree (CATtree), which is a collection of a communication function, its interfaces and several architecture templates in transaction, register transfer, and software levels. We constructed a set of CATtrees that covers a wide range of communication functions and architectures. For a communication function that is captured by a specific CATtree, the designer can refine it by exploring the architecture templates in the CATtree. Moreover, the concept of the CATtree makes the computation modeling easier, especially for RTL hardware modeling because a computation can be modeled without communication-related details, which is modeled with a CATtree.

In the process of developing a CATtree-based system design environment, which will be described in another paper in the future, we exploited the CATtree library for communication refinement, a C-to-RTL synthesis tool for HW computation refinement, DEOS for SW computation refinement. DEOS is a C++-based light-weight operating system that enables SystemC TLMs to be directly executed as software models. Although its basic concept is already described in [11], the hardware dependent software (HdS) of DEOS is configured with the SW templates of CATtrees.

In the RBD, it is essential to have a mixed- level simulation environment, which can simulate partially refinement system models that contain TLMs, RTLs and SWs together. Therefore, we developed a CATtree-based mixed-level

Proceedings of the Seventeenth IEEE International Workshop on Rapid System Prototyping (RSP'06)
0-7695-2580-6/06 $20.00 © 2006 IEEE

simulation environment by using SystemC, a commercial logic simulator and a GNU Gdb.

This paper mainly focuses on the DEOS and the mixed-level simulation environment for the CATtree-based design flow. The concept of the CATtree and the design flow are introduced in Section 2. The DEOS is described in Section 3. We explain the mixed-level simulation environment and evaluated the simulation environment with an H.264 decoder design example in Sections 4 and 5, respectively which is followed by a conclusion.

2 Design Flow using the CATtree

OSCI announced a TLM library that contains FIFO and handshaking channels. These two channels are not sufficient to capture a current complex communication although they are essential. Furthermore, the OSCI TLM library does not provide any architectural information required in refining the channels. Therefore, we employed the CATtree to alleviate these problems.

2.1 Concept of the CATtree

A CATtree captures several abstraction levels of a channel that covers a specific domain of communication functions and also contains architecture templates, which are parameterized implementations, for communication refinement. They can be a composition of channels and adapters. Two types of adaptors exist. A channel adaptor is one for matching the difference of protocols or data structures and an abstraction adaptor is one for matching two different abstraction levels.

Each architecture template in the CATtree for a channel has a TLM in SystemC, a RTL model in HDL, and a SW model in C++. To be able to simulate a mixed-level simulation model smoothly after refining some components from TLM to RTL in a system, each CATtree should provide two abstraction adaptors: one from TLM to RTL and the other from RTL to TLM. Abstraction adapters for SWs are not needed because a processor model does abstraction adaptation.

During an H.264 decoder system development, we have constructed a CATtree library as shown in Table 1 and we will add more CATtrees to the library in the future so that we can cover communication functions more effectively. Although the FIFO CATtree can be a better example because it is more fundamental and easier to be understood, the Array CATtree is explained in this paper because the FIFO CATtree was introduced once in [6]. In its diagram shown in Figure 1, which is just a part of the Array CATtree, the bus-wired array architecture contains a bus channel, two channel adapters (ArrayBusMasterAdpt and ArrayBusSlaveAdpt), and an embedded array channel. The embedded array channel can be refined into the architecture of RegArray or CachedArray. Similarly, the embedded array in the CachedArray architecture can be refined into a bus-wired array architecture.

Table 1 Currently Developed CATtrees

CATtree Name	Description
FIFO	Point-to-point ordered and synchronized data transmission
Array	Indexed data storage with one writer and one reader.
Variable	Data storage with (possibly) multiple writers and (possibly) multiple readers.
Event	Point-to-point event notification with simple and light handshaking protocol.
Handshaking (H/S)	Offset-addressable point-to-point handshaking data transmission.
Shared bus	A group of handshaking channels
Shared event	A group of event channels.
Block-read FIFO	An augmented FIFO channel which can read fixed number of data items.
Block-write FIFO	An augmented FIFO channel which can write fixed number of data items.
Multi-read FIFO	An augmented FIFO channel which can read arbitrary number of data items.
Multi-write FIFO	An augmented FIFO channel which can write arbitrary number of data items.
2D array	An augmented array channel which has two dimensional indices.
3D array	An augmented array channel which has three dimensional indices.
Packet FIFO	An augmented FIFO channel which can store packetized data stream.

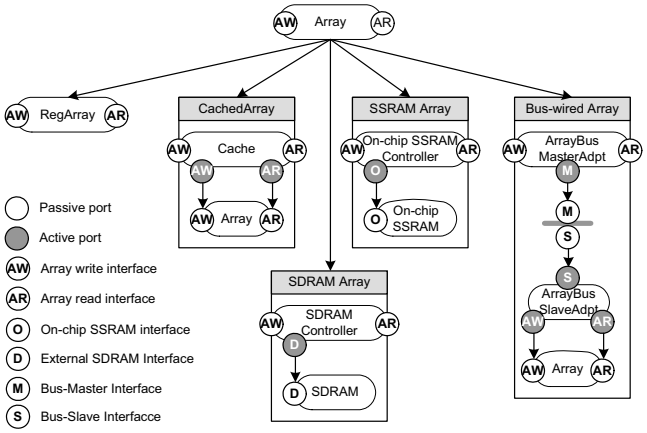

Figure 1 Array CATtree

2.2 CATtree-based Design Flow

Figure 2 shows the CATtree-based design flow in which the designer partitions computation and communica-

tion first from the informal specification in the function capture step, which is relatively easier because the designers decide only what is captured with the CATtree library as communication functions. Therefore, to develop a system TLM, the designer should describe computation TLMs by themselves and integrate them with the communication TLMs in the CATtree library.

After integrating a system TLM, the designer should execute and improve it through verification and validation. Therefore, it is essential to construct a complete set of testbenches, with which the correct-by-construction can be confirmed easily in the architecture refinement process.

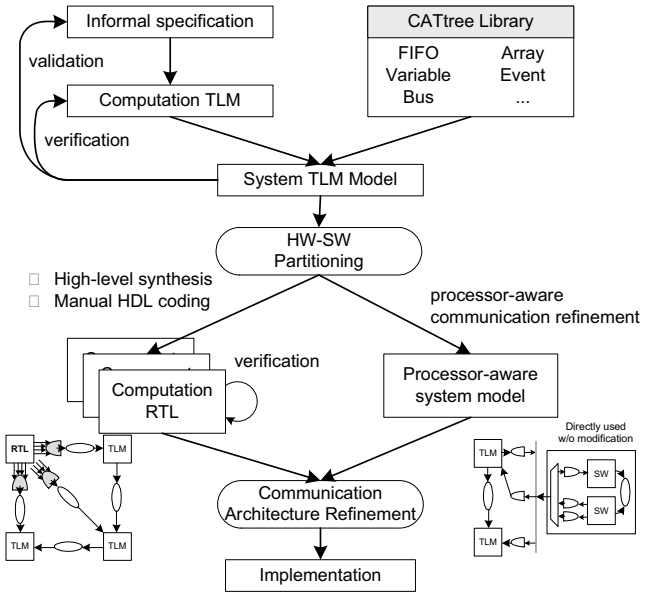

Figure 2 Simplified design flow of SoCBase-DE

In the computation refinement step, computations are partitioned into hardware and software. For a computation function mapped into hardware, we can reuse the hardware components if it is readily available. If not, however, hardware models should be newly described in RTL. Designing the communication part such as bus interfaces for a new computation component is known to be relatively complex. In the design flow, the communication parts are captured and refined only with already verified CATtrees. Thus, in the verification, the designer can focus on the computation function and its RT-level interfaces. Moreover, the system TLM can be used as a testbench for the verification of computation RTLs by inserting necessary abstraction adapters. The abstraction adapter can provide randomization techniques proposed in [10] with which the RT-level interfaces can be verified more thoroughly.

If a computation function is refined into software, its SystemC TLM is directly used as a software model with the help of a new operating system, DEOS described in the next section. Hardware and software computations can be refined separately, as shown Figure 3. Therefore, it is necessary that any intermediate models should be able to be executed in a mixed-level simulation environment, which will be described in Section 4. After finishing the software-hardware partition, the designer should refine communication and find better communication architecture by using the architecture templates of CATtrees in the design flow. We are still working on several issues on this communication refinement, which will be covered in the future.

3 DEOS and Software Refinement

DEOS, which is a C++-based light-weight operating system kernel, is specifically designed to enable CATtree-based SystemC computation models be executed on a target microprocessor. Similarly to the approach of SPACE [11], SystemC TLMs are cross-compiled and executed on the target processors. However, there are subtle differences between SPACE and DEOS. In SPACE, the authors assume the communication model to be either Un-timed Functional (UTF) or Timed Functional (TF) channels as an abstraction of the bus. In contrast, DEOS provides more extensive communication models. Moreover, DEOS can execute SW models with multiple inputs and outputs whereas SPACE supports only a UTF with an input and an output. DEOS can configure the Hardware dependent Software (HdS) flexibly with the SW templates of CATtrees.

A software computation, which is embedded in a microprocessor, requires processor-aware communication architecture. In the communication viewpoint, however, the interface of a processor is somewhat inflexible and limited. Conventional processors communicate with other processors or hardware components only through one or two bus master interfaces and one or two interrupt signals. All the channels of a SW computation that communicates with a HW component or a SW component mapped to another processor should be refined into the processor-aware architectures. For example, a FIFO channel should be refined into the bus-wired FIFO architecture [6].

As shown in Figure 3, an array channel should be refined into the bus-wired array architecture. In an example for a processor-aware architecture refinement, shown in Figure 3, there are four computations ($CMPT_A$, $CMPT_B$, $CMPT_C$, and $CMPT_D$) connected by three FIFO channels ($FIFO_A$, $FIFO_B$ and $FIFO_C$), where $CMPT_A$ and $CMPT_B$ is implemented in software and $CMPT_C$, and $CMPT_D$ remain in the transaction level. $FIFO_A$ and $FIFO_C$ are refined into the master-write bus-wired FIFO respectively, which is a composition of a bus FIFO sender, a bus FIFO receiver adapter, a H/S channel, and an event channel. $FIFO_B$ is refined into the master-read bus-wired FIFO, which is a

composition of a bus FIFO reader, a bus FIFO keeper, a H/S channel, and an event channel.

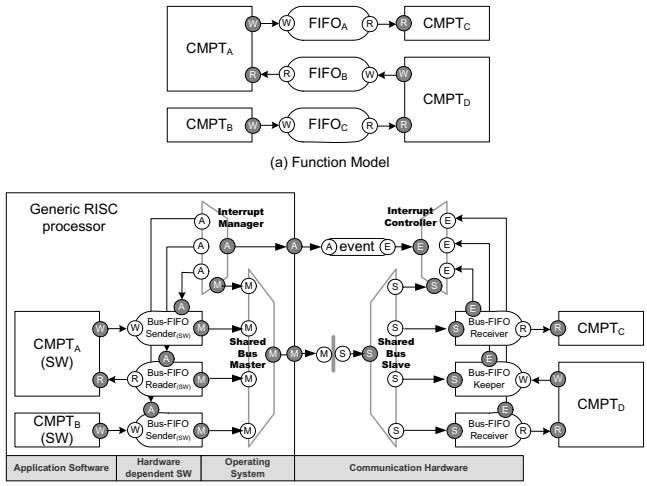

Figure 3 Example function model and its processor-aware architecture

The three event channels are grouped and mapped into a shared event CATtree. The architecture of the shared event CATtree can be mapped into a composition of an interrupt manager, an interrupt controller, an event channel, and an H/S channel. In Figure 3, the interrupt manager is a basic service of DEOS.

Four H/S channels (three from bus-wired FIFOs and one from a shared event channel) are grouped and mapped into a shared bus CATtree, A shared bus can be a composition of a shared bus master, a shared bus slave and an H/S channel. The shared bus master adapter is embedded in a processor. The shared bus can be refined into a crossbar switch or a NoC. Many researchers have emphasized the automatic synthesis of shared bus architecture [12], which can be easily adapted in the simulation environment.

The performance of SW templates is a critical issue. In fact, many researches on MPSoC design methodology targeted to its optimization, which is not a trivial problem. However, current SW templates are not yet optimized, which will be one of our future works.

4 Mixed-level Simulation Environment

The mixed-level simulation environment can execute SystemC-based TLMs, hardware models in HDL, and SW models. The SW models are running on the instruction set simulators (ISS) of a target processor. We used GNU Gdb as an ISS because of following reasons:

- Gdb contains 19 processor models including ARM, MIPS, and PowerPC etc.
- Gdb provides source-level debugging feature.
- We can modify source code to add several verification features including architecture-aware profiling.
- GDB is an open source and we can construct inexpensive design environment.

The mixed-level simulation flow is basically another hardware-software co-simulation environment that supports TLM simulation additionally. There are many works on the mixed-level simulation environment [2][11][13]. Here we describe the mixed-level simulation environment that uses the TLMs of CATtrees.

4.1 TLM-RTL Co-simulation

We assume that all HWs are described in HDL because it is a more popular language than SystemC. The abstraction adapters, which are inserted after computation and communication refinement, translate transaction function calls into RT-level signal activities, and vice versa. An abstraction adapter is a SystemC module whose RT-level interface is modeled using sc_signal channel. A RTL model in HDL can be connected to an abstraction adapter through a co-simulation linker (CSL).

The CSL can be divided into three parts: a TLM-to-IPC interface, a RTL-to-IPC interface, and a named pipe as an inter-process communication (IPC). In our approach, the TLM-to-IPC interface sends/receives the sc_signals to/from the abstraction adaptors and creates a virtual clock to synchronize the SystemC adaptors with the logic simulator. The TLM simulator should control the time to synchronize the logic simulator and the ISS. In commercial logic simulators that provide built-in SystemC simulation feature, the sc_signals in TLM can be directly connected. Even in such case, the CSL is still required to use the source-level debugger for TLM debugging.

4.2 TLM-SW Co-Simulation

When a computation function is refined into SW, its channels should be also refined into a processor-aware architecture that contains adapters either in software or in hardware. In the TLM-SW co-simulation, all SW computations and adapters are obviously executed in the target processor ISS of Gdb. Software models run in the ISS can communicate the TLMs in SystemC though a processor BFM, that is, a CSL for TLM-SW co-simulation.

Synchronization between the ISS and the TLM simulator is required when a SW model accesses to a TLM model. A simple solution is to synchronize the TLM and ISS at every clock cycle although it is too slow due to the IPC synchronization overhead. Optimistic ISS simulation and role-back method [14] was useful for SW-RTL co-simulation because an ISS is usually an order-of-magnitude faster than a logic simulator. However, this method cannot be applicable to the TLM-SW mixed simulation because a

TLM simulation is usually faster than an ISS simulation. A simple but effective method is to synchronize periodically after the ISS executes a pre-determined number of instructions. This method looses some accuracy for the sake of better simulation speed. In the simulation environment, the TLM-SW synchronization is performed in the following cases.

1. When the ISS accesses to TLM or RTL
2. When the ISS has executed for the pre-defined number of instructions.
3. When DEOS enters an idle state because all SW threads are waiting external events.

For the third case, we modified Gdb to handle it with a special software interrupt.

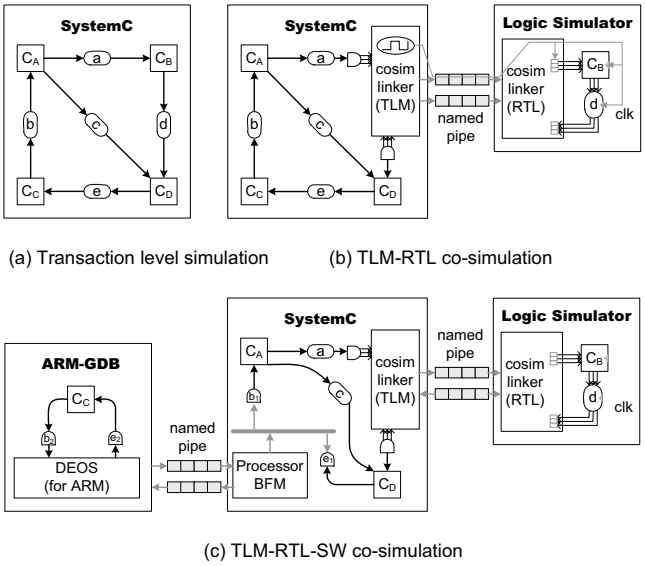

Figure 4 Mixed-abstraction simulation of SoCBase-DE

4.3 Automatic Transaction Compare (ATC)

We provide the Automatic Transaction Compare (ATC) feature that compares data written into an abstract channel with data written into a refined channel. The designer can configure a channel model to store all the written values into a file, that is, a Reference Data Stream (RDS). This RDS is a reference data to be checked by its refined channel model. The designer can control generating and comparing the RDS by configuring the template models for the purpose of verification.

5 Experiment

In this section, we also compared the speed of TLM, TLM-SW, and TLM-RTL co-simulations and RTL-SW virtual-prototype simulation for an H.264 decoder example.

5.1 An H.264 Decoder System

We designed an H.264 baseline decoder by using the design flow. We modeled twelve computation functions including a parser (PARSER), a variable length decoder (VLD), an inverse transform and inverse quantization (ITQ) unit, an inter-picture prediction (INTER) unit, an intra-picture prediction (INTRA) unit, and a de-blocking loop filter (DF). These computation functions communicate each other though 117 channels. We used all the CATtrees in Table 1 except the 2D array CATtree.

The partition of hardware and software was performed manually. We decided that only the parser was implemented in software and all the other components in hardware to achieve a higher performance. The current implementation of the H.264 decoder can decode one macroblock in 3,110 cycles on the average, including the overhead of external SDRAM accesses.

5.2 Simulation Performance Comparison

For the simulation, we used MR3_TANDBERG_B bitstream of QCIF 300 frames. The host platform was a Linux workstation with a 2.4 GHz Pentium4 CPU and a 1GB memory. Table 2 summarizes the simulation results of several configurations. We used Mentor Graphics ModelSim® as a logic simulator. The logic simulator-side CSL was implemented using the Foreign Language Interface (FLI) of ModelSim®.

Table 2 Performance of mixed-abstraction simulation

No	Configuration	Cycles	Sim. Time
1	ITU-T JM8.2 Reference C	-	4 sec
2	TLM function model	-	153 sec
3	Architecture refined TLM	-	237 sec
4	TLM with PARSER software*	68.0M	311 sec
5	TLM with VLD hardware	5.03M	546 sec
6	TLM with ITQ hardware	5.08M	474 sec
7	TLM with INTER hardware	16.0M	5838 sec
8	TLM with INTRA hardware	1.4M	339 sec
9	TLM with DF hardware	15.2M	3972 sec
10	Parser SW with all hardware	112M	34950 sec

* ISS-TLM synchronization period = 100

The transaction level function model (2) is 40 times slower than reference C implementation (1) because the SystemC-based TLM is a multi-threaded program that incurs much context switching and synchronization overhead and the TLM templates of channels contain several services for easier functional verification including the ATC technique.

The TLM-SW co-simulation (4) is slightly slower than the architecture refined TLM (3) because the simulation load of ISS is relatively low and the synchronization over-

head between TLM and ISS is not so high. When synchronization period was 1, 10, 100, 500 and 1000, the simulation times were 1,005 second, 420 second, 311 second, 310 second, 308 second, respectively. In general, the time unit of SW is millisecond. Thus, a timing skew of 10 or 100 cycles is negligible.

In the experiments from 5 to 9, we refined each major computation to verify its RTL description one by one. Because only one computation is in RT-level and the rest of functions are remained in TLM in each of these experiments, the total clock count is a good performance measure of that computation. Therefore, we checked the clock cycle count to improve the performance of each computation. In these configurations, the communication is still in TLM and the communication-related overhead including the latency of memory accesses is not included in the cycle counts.

In experiment 10, the system model has a completely refined architecture. Its simulation speed is very slow because all channels are refined into hardware. The simulation cycle count contains the overhead of the refined channels including the latency of external SDRAM memory accesses.

From the results of these experiments, we confirmed that the virtual prototyping environment supports refinement-base design and verification.

6 Conclusion

In this paper, we briefly introduced the concept of CATtree and a refinement-based design flow that supports mixed-level virtual prototyping and effective functional verification. We explained that user-described TLMs can be used as SW on a target processor by the help of DEOS. With DEOS and the SW templates in the CATtrees, the hardware dependent software (HdS) is configured, which is compatible with the refinement-based design methodology. For the verification of partially refined system models, we constructed a TLM-RTL-SW mixed-abstraction simulation environment using CATtree library, SystemC, a commercial logic simulator and a Gdb. With experiments, we showed that the simulation environment provides reasonable simulation performance with great flexibility.

References

[1] S. Abdi, D. Gajski, “Automatic communication refinement for system level design”, Proc. of 40th Design Automation Conference, pp. 300-305, 2003.

[2] I. Petkov, A. Jerraya, “Systematic design flow for fast hardware/software prototype generation from bus functional model for MPSoC”, Proc. 16th RSP, pp.218-224, 2005.

[3] V. Reyes, W. Kruijtzer, “CASSE: A system-level modeling and design space exploration tool for multiprocessor system-on-chip”, Proc. of the EUROMICRO systems on digital system design, 2004

[4] K. Keutzer, A. Sangiovanni-Vncentelli, “System level design: orthogonalisation of concerns and platform-based design”, Trans. on computer-aided design of integrated circuit and systems, Vol. 19, No. 12, Dec. 2000

[5] A. Jerraya, W. Wolf, “Hardware/Software interface codesign for embedded systems”, IEEE computers. Feb. 2005.

[6] S. Park, S. Chae, “Reusable component IP design using refinement-based design environment”, Proc. of 11th ASP-DAC, Jan. 2006.

[7] OSCI, “Functional specification for SystemC 2.0”, 2002

[8] A. Rose, J Femandez, “Transaction level modeling in SystemC”, Apr. 2005.

[9] S. Edwards, A. Sangiovanni-vincentelli, “Design of embedded systems: formal models, validation, and synthesis”, Proc. of the IEEE, Vol. 85, No.3, Mar. 1997.

[10] S. Park, S. Chae, “An open source based integrated framework for functional verification of system on chip”, Proc. of 16th RSP, 2005.

[11] J. Chevalier, F.Boyer, “SPACE: A hardware/software SystemC modeling platform including an RTOS”, Forun on design languages, Sep. 2003.

[12] K. Lahiri, S. Dey, “Design space exploration for optimizing on-chip communication architectures”. Trans. on computer-aided design of integrated circuits and systems, Vol. 23, No. 6, Jun. 2004.

[13] CoWare, “ConvergenSC technical overview”, http://www.coware.com/products/convergensc.php

[14] Y. Ahn, K. Choi, “An efficient simulation environment and simulation techniques for Bluetooth device design”, Design automation for embedded systems, Vol. 8, No. 2, pp. 119-138, Sep. 2003.