한국산학기술학회논문지
Vol. 9, No. 5, pp. 1139-1144, 2008

Automatic Virtual Platform Generation for Fast SoC Verification

Jun-Mo Jung[1*]

고속 SoC 검증을 위한 자동 가상 플랫폼 생성

정준모[1*]

Abstract In this paper, we propose an automatic generation method of transaction level(TL) model from algorithmic model to verify system specification fast and effectively using virtual platform. The TL virtual platform including structural properties such as timing, synchronization and real-time is one of the effective verification frameworks. However, whenever change system specification or HW/SW mapping, we must rebuild virtual platform and additional design/verification time is required. And the manual description is very time-consuming and error-prone process. To solve these problems, we build TL library which consists of basic components of virtual platform such as CPU, memory, timer. We developed a set of design/verification tools in order to generate a virtual platform automatically. Our tools generate a virtual platform which consists of embedded real-time operating system (RTOS) and hardware components from an algorithmic modeling. And for communication between HW and SW, memory map and device drivers are generated. The effectiveness of our proposed framework has been successfully verified with a Joint Photographic Expert Group (JPEG) and H.264 algorithm. We claim that our approach enables us to generate an application specific virtual platform 100x~1000x faster than manual designs. Also, we can refine an initial platform incrementally to find a better HW/SW mapping. Furthermore, application software can be concurrently designed and optimized as well as RTOS by the generated virtual platform

Key Words : SoC, Virtual Platform,

요 약 본 논문에서는 가상 플랫폼을 이용하여 빠르고 효과적으로 시스템을 검증하기 위한 추상레벨의 자동 생성에 대하여 제안한다. 추상레벨 가상 플랫폼은 효과적인 검증 방법이긴 하지만 시스템이 변경될 때 마다 가상 플랫폼을 재생성하고 추가적인 설계/검증을 요구되며 이 작업은 매우 많은 시간을 요구한다. 이러한 문제점을 해결하기 위하여 본 논문에서는CPU, 메모리, UART 등을 기본적인 요소로 구성하여 추상레벨의 라이브러리로 생성하였다. 이 라이브러리를 이용하여 가상 플랫폼을 자동 생성하는 툴을 개발하였다. 이 툴은 임베디드 RTOS를 구성하는 가상 플랫폼을 자동 생성하며 HW/SW 간의 통신을 위한 메모리 맵과 디바이스 드라이버 등도 생성한다. 제안한 방법은 JPEG과 H.264에 성공적으로 적용하였으며 기존의 수동 작업에 비하여 매우 빠르게 가상 플랫폼을 자동 생성할 수 있었다.

1. Introduction

SoC (system-on-chip) designs have become so complicated that they demand high-level modeling for both design and verification. The state-of-the-art design methodologies [1] have been proposed to increase the productivity by raising the level of abstraction and these have been provided seamless design flows. Among them, IP reuse-based design and platform-based design methodologies [2, 3] have gained lot of attention from researchers and developers. Especially, design methodology based on virtual platform [4], which

[1]School of electronic & Information Eng. Kunsan National University(Professor)
*Corresponding Author: jung jun mo(jmjung@kunsan.ac.kr)
Received July 7, 2008 Revised October 14, 2008 Accepted October 16, 2008

captures the concept of the platform-based design approach, has been widely accepted as a crucial research issue.

However, most of the current design methodologies[1-4]based on the virtual platforms are focused only on building a new virtual platform by assembling pre-designed IP blocks. Therefore, each IP block must have been completely designed before the system integration has done. Also, a different integration of IP blocks will require redesign of software part from scratch. Furthermore, even though most commercial EDA tools [12, 13, 14] are capable of generating interfaces between hardware components and software parts automatically, they cannot automatically partition hardware/software and generate virtual platformfor a given HW/SW partition. Hence, it takes long time to manually redesign HW and SW parts for various HW/SW partitions, and it takes even longer time to verify each implementation.

In this paper, we propose an automatic generation of TL modeling which effectively achieves fast system design and prototyping. We developed a set of design tools to generate a an application-specific virtual platform automatically. From an algorithmic modeling, our tools generate a virtual platform which consists of embedded RTOS (real-time operating system) including user-application and hardware components which contain user-logics. The effectiveness of our proposed framework has been successfully verified with an image compression/decompression application. We claim that our approach enables us to generate an application specific virtual platform 100x~1000x faster than manual designs. Also we can incrementally improve an initial platform to find a better HW/SW mapping. Furthermore, application software can be concurrently designed and optimized as well as RTOS by the generated virtual platform. We described briefly our idea in introduction. The related works and verification methodology are presented in section II, III. The case study and experimental results are enumerated in section IV, V.

2. Related Works

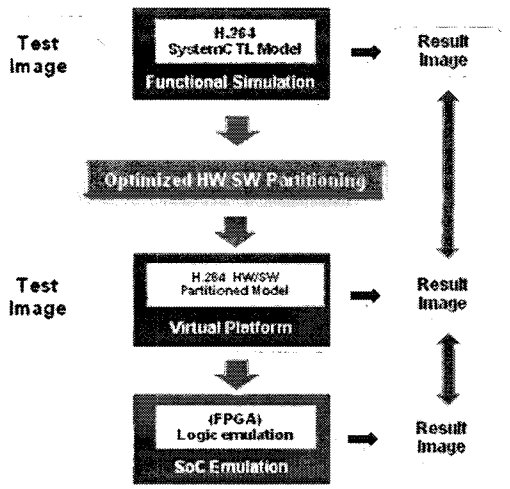
System-level design languages (SLDL) are developed like SystemC, or SpecC available for modeling and describing systems at different abstraction level. However, the languages themselves do not define any details of actualdesign flows. Recently, the transaction-level modeling (TLM) has been commonly employed using SLDLs in co-simulation frameworks to speed up the simulation. The works in [5-7] presented the general modeling issues related to the communication architecture in the TL.In [8], AMBA 2.0 is modeled as TLM using SystemC 2.0 language. However, there are no specific definitions about the level of abstraction and the semantics of transactions. Furthermore, proposals of TLM only focus on simulation and lack the transformation of models for implementation and synthesis. In [9, 10], Pasricha et al., show an approach to explore the communication design space using TLMs. However, they do not address the problem of automatic generation of the TLMs.

Recently, some commercial tools are beginning to capture designs at the TL. The ConvergenSC of CoWare [12] is one of such tools, and it explores the design space using SystemC TLM [5] and validates the optimal system model with embedded software. The CoCentric System Studio of Synopsys [13] is a co-design and co-verification tool based on a SystemC modeling. After the initial design specification is transformed into a description of high level language, each transformed module of the system is mapped into a hardware component. The Platform Express of Mentor Graphics [14]] is a platform-based SoC design and verification tool. It represents system specification as a block diagram which consists of blocks and busses. It includes high level co-verification tool (Seamless) and RT level hardware simulator (ModelSim) in order to verify the designed system. Still, the designers are not able to achieve significantly reduced system design cycles and required system performance.

This is mainly due to the non-separation of computation and communication and to non-availability of automatic generation tools of TLMs from high-level models. In contrast to the existing schematic entry tools that simply provide an interface for plugging existing database models together graphically, the contribution of this paper is to generate virtual platform, detailed TL models from algorithmic models of the system.

3. Verification Methodology

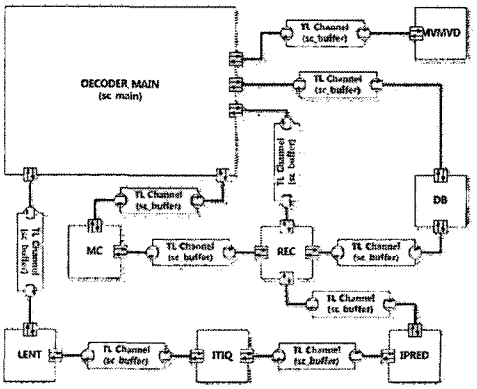
Fig. 1 shows our verification methodology conceptually. Common practice of system designs starts from abstract model verification at an algorithmic level and continues with a platform generation based on a manually determined HW/SW partition. This conventional approach of building a virtual platform is quite effective as long as we have a good partition of hardware and software. If we want to explore various options of feasible HW/SW partitions, the conventional approach takes too much time and effort to design and verify the various virtual platforms. Furthermore, if we modify algorithmic models, we cannot verify algorithm and virtual platform at a same time. In order to support various HW/SW mapping and fast virtual prototyping, we propose a top-down TL modeling which consists of two levels: algorithmic model and bus functional model.



[Figure 1] Proposed Verification Flow

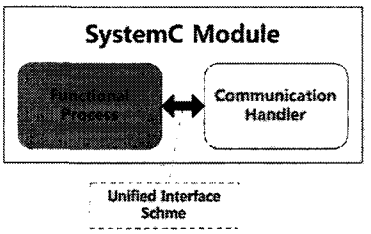
3.1 Verification Methodology

Fig. 2shows the algorithmic TL model. As is often the case with actual system design, development planning at an early stage of virtual platform creation will significantly affect the development time as well as the quality of the entire system. In our approach, an algorithmic TL model captures the main functionality and communication pattern of each individual IP and estimates the system performance and the size approximately. On the other hand, the bus functional model is employed to concretize the given algorithmic model into a valid execution model which consists of bus transaction behavior and system performance estimation.



[Figure 2] Example of Algorithmic TL model

Algorithmic TL model aims for high simulation speed and unified verification. In order to meet these criteria, we modeled IPs as KPN (Kahn Process Network) [11] models and separated the functional part from the communication part of each IP model to achieve code reusability. As shown in Fig. 3, the separated communication part can be accessed via "function calls." For various communication schemes, function calls have a unified set of programming interfaces. Therefore, all the IPs use unified interfaces, and this enables us to explore various communication models easily.



[Figure 3] IP modeling for TL communication

All IPs of the system are concurrently executed and connected by point-to-point transaction communication channels that carry sequences of data. These channels are the FIFO channels of infinite length. The concurrent processes of the system functionality read or write the data through these FIFO channels with blocked reading

and writing operations. The systems are deterministic, which implies that the execution order of the processes will not affect the results by infinite FIFO channels.

All processes can be analyzed with a flattened structure while general modules are constructed with a hierarchy. The system can be analyzed more easily if each process code is described by the legacy C/C++ code, and this method can be enabled to estimate system's costs at high level as in [6].

3.2 Bus Functional TL Model

Practically, real systems cannot have FIFOs of infinite length. Therefore, specific upper bounds for FIFOs must be specified by the designer (thus making write operations potentially blocking). In the bus functional TL model, we model communication between modules using function calls. These functional calls represent the transactions. Within TL modeling, data are exchanged between different processes by reading and writing shared data variables. From our empirical study, we learned that FIFO channel can be represented by internal memory of finite size. For scheduling of data communication by the arbiter, data is required to have information on the master number.

Table 1 shows an example code for data communication. The 'read' function of 'sc_fifo_in' channel for receiving data can be represented as 'Bus_read(type, address, data, size)' function call. Only the information of address, size, and type (i.e., burst mode or direct mode) are explicitly described but all the other details are implicitly implemented in the separate library.

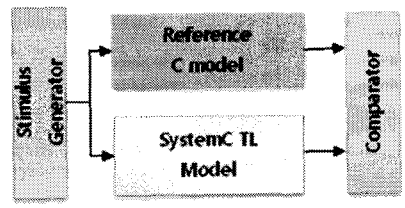
[Table 1] FIFO vs. bus

Channel	Example code
FIFO	data = FIFO_in_ch1.read()
TL Bus	slave->Bus_read(type, address, data, size)

We adopt a bus functional TL model as a data communication model to achieve fast HW/SW co-simulation. And, the bus protocol is defined as general as possible. The virtual platform consists of ISS, memory, timer, UART and user IPs. In this virtual platform, only the information of address, data, length, and type is described to achieve high-abstract bus model.

3.3 Comparison of reference C model and SystemC TL model

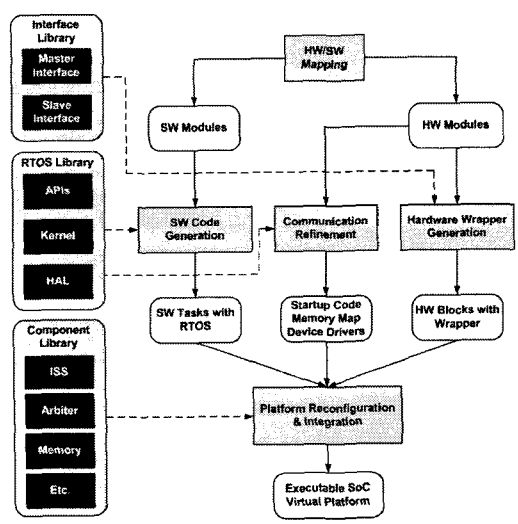
We developed stimulus generator and comparator to verify between reference C model and SystemC TL model in Fig. 4. The comparison results can certify the designated SystemC TL model from reference C model. The algorithmic TL and the bus functional TL model also can be compared by this method.



[Figure 4] Stimulus Generator and Comparator

4. Generation of bus functional TL model

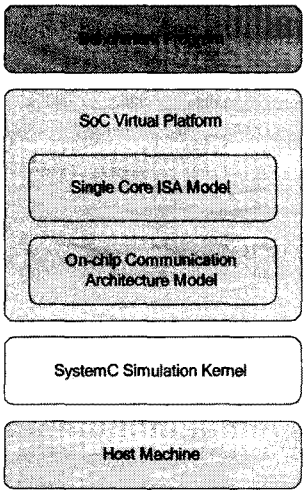
The example in fig. 5 gives us an overall framework of our virtual platform generation. The software processes of the partitioned PFG are transformed into a set of tasks for generic RTOS. Generic RTOS consists of kernel, APIs and device drivers. Hardware modules are transformed into bus functional TL model. FIFO channels of algorithmic TL model are concretized into APIs of RTOS library.



[Figure 5] Flow of virtual platform generation

Virtual platformgenerator can be divided into three parts: software code generation, hardware wrapper generation, and communication refinement. Software code generation part generates C codes in order to be simulated by an ISS. These C codes include generic RTOS. In other words, the generated codes are a set of tasks for generic RTOS. Hardware refinement part generates bus functional TL model from algorithmic TL model. The communication channels between hardware and software are refined into RTOS with APIs from FIFO channel by the communication refinement part.

Fig. 6 shows the simulator architecture. The SoC virtual platform will be run by SystemC simulation kernel on the host machine. And the benchmark program will run on the virtual platform after cross-compilation.



[Figure 6] Simulator architecture

To evaluate our methodology and the effectiveness of our approach, we specified the algorithmic TL model of JPEG and H.264 decoder using SystemC 2.1, and then applied to our tools.

We generated bus functional TL codes from algorithmic TL codes manually and automatically as in table 2. Overall model complexities are given in terms of code size using lines of code (LOC) as a metric. To quantify the actual refinement effort, the number of modified lines (Mod. LOC) is calculated as the sum of lines inserted and lines deleted whereas code coming from library modelsbecause the conventional methods and tools cannot generate any codes for user-applications. We assume that an expert can write correct 10 lines code per hours. Thus, manual refinement would require some of man-days for complex system designs. Automatic generation, on the other hand, completes in a several seconds. This means that our method and tool can reduce huge efforts of designer.

[Table 2] Automatic generation

Design	Mod. LOC	Manual	Tool
JPEG Encoder	1325	~ 6 days	< 1 sec
JPEG Decoder	1470	~ 6 days	< 1 sec
H.264 Decoder	7242	~ 30 days	< 1 sec

Table 3 compares the execution time measured on a SPARC with 1024 Mbytes of memory, running SOLARIS 8 for one macro block of JPEG image. The results show that the proposed co-simulation approach is about forty times faster than the conventional BFM for JPEG encoding/decoding of an 800 by 600 image.

[Table 3] Speedup of proposed framework

	Conventional BFM	Prop. TL Bus
one block (8 by 8)	26.4 seconds	0.63 seconds
one image (800 by 600)	35.2 hours	0.84 hours

6. Conclusion

We built TL library which consists of basic components of virtual platform such as CPU, memory, timer, UART and ETC. We developed a set of design/verification tools in order to generate a virtual platform automatically from HW/SW partitions. The effectiveness of our proposed framework has been successfully verified with an image compression/decompression application. We claim that our approach enables to generate an application specific virtual platform 100x~1000x faster than manual designs. And we can refine an initial platform incrementally to optimize HW/SW mapping. Furthermore, application software can be concurrently designed and optimized as

well as RTOS by the generated virtual platform. In future, we will expand our methodology to support various on-chip-network architectures.

7. References

[1] J. Um, et al., "A Systematic IP and Bus Subsystem Modeling for Platform-Based System Design," *DATE*, 2006.

[2] A. Sangiovanni-Vincentelli, et al., "Benefits and Challenges for Platform-Based Design," *DAC*, 2004.

[3] A. Sayinta, et al., "A Mixed Abstraction Level Co-simulation Case Study Using SystemC for System on Chip Verification," *DATE*, 2003.

[4] H. Lekatsas, et al., "Coco: A Hardware/Software Platform for Rapid Prototyping of Code Compression Technologies," *DAC*, 2003.

[5] L. Cai and D. Gajski, "Transaction Level Modeling: an Overview," *CODES*, 2003.

[6] I. Moussa, et al., "Exploring SW Performance Using SoC Transaction-Level Modeling," *DATE*, 2003

[7] A.K. Deb, et al., "System Design for DSP Applications in Transaction Level Modeling Paradigm," *DAC*, 2004.

[8] M. Caldari, et al., "Transaction-Level Models for AMBA Bus Architecture Using SystemC 2.0," *DATE*, 2003.

[9] S. Pasricha, et al. "Extending the Transaction Level Modeling Approach for Fast Communication Architecture Exploration," *DAC*, 2004.

[10] S. Pasricha, et al. "Floorplan-Aware Automated Synthesis of Bus-Based Communication Architectures," *DAC*, 2005.

[11] G. Kahn, "The Semantics of a Simple Language for Parallel Programming," in *Proc. of IFIP Congress*, 1974.

[12] ConvergenSC, Coware Inc., *http://www.coware.com*

[13] CoCentric System Studio, Synopsys Inc., *http://www.synopsys.com*

[14] Platform Express, Mentor Graphics Corp., *http://www.mentor.com*

Jun-Mo Jung [Life Member]

- Feb. 1987: Hanyang University BS in electronics engineering
- eb. 1989: Hanyang University MS in electronics engineering
- eb. 2004: Hanyang University Ph.D in electronics engineering
- eb. 1989 - Mar. 1996 : Samsung Electronics, ASIC Design Center
- ar. 2004 -Mar. 2005 : Hanyang Cyber University, Assistant Professor in computer engineering
- pr 2005 - : Kunsan National University, Associate Professor in electronics & information engineering

<Research Area>
VLSI Design, SoC Design, SoC Test & Verification, Test Scheduling