A Top-down Hardware/Software Co-Simulation Method for Embedded Systems Based Upon a Component Logical Bus Architecture

Mitsuhiro YASUDA **Barry SHACKLEFORD** **Fumio SUZUKI**
Katsuhiko SEO
Hisao KOIZUMI

Mitsubishi Electric Corporation, Yokohama 220-81 Japan
{yasu@cd.,seo@ds.,koizumi@}**lmt.melco.co.jp**

Hewlett-Packard Laboratories, Palo Alto, CA 94304

Mitsubishi Electric Corporation, Amagasaki 661 Japan
suzuki@dpe.mdl.melco.co.jp

Abstract- We propose a top-down hardware/software co-simulation method for embedded systems and introduce a component logical bus architecture as an interface between software components and hardware components. Co-simulation using a component logical bus architecture is possible in the same environment from the stage at which the processor is not yet determined to the stage at which the processor is modeled in register transfer language. A model whose design is based on a component logical bus architecture is replaceable and reusable. By combining such replaceable models, it is possible to quickly realize seamless co-simulation. We further describe experimental results of our approach.

I. Introduction

As semiconductor technology advances into deep-submicron areas, high-performance processors (microprocessors, DSPs, etc.) have begun to appear. As a result, functions which had previously been realized exclusively in hardware circuitry have come to be implemented in software, and it is now possible to design electronic products as embedded systems, combining hardware and software.

Particularly in the development of multimedia products, it has become necessary to combine flexible software with high-performance hardware as a means of exploring new markets and responding promptly to user needs and as a means of reducing product development cycles.

Hardware/software co-design techniques are now attracting attention as one means of realizing embedded systems. There have been numerous studies [1]-[5] on:

- hardware/software partitioning,
- hardware/software co-synthesis, and
- hardware/software co-simulation.

In the first stage of the top-down design, the architecture and algorithm of the target system are evaluated. Then, the embedded system is partitioned into hardware components and software components. These components are first modeled at an abstract level and are evaluated by co-simulation. These models are then subsequently replaced by detailed models.

In order to realize seamless co-simulation in the top down design flow, it is necessary to define the interface between the hardware components and the software components and the modeling methods of hardware components and software components.

Two methods for modeling hardware, software and the interface between them--

- the channel-based component architecture [6],
- the virtual CPU approach [7]

--have been reported. In the channel-based component architecture, the interfaces between components and their operation are specified to achieve co-synthesis at the system level. In the virtual CPU approach, a CPU model and its interface to other components are defined to realize a co-simulation environment.

We propose an efficient top-down approach of co-simulation for embedded systems in which upper models are replaced successively by lower detailed models.

By introducing a component logical bus architecture as the interface between hardware components and software components, this method provides an environment for co-simulation which does not depend on processor and system bus architectures

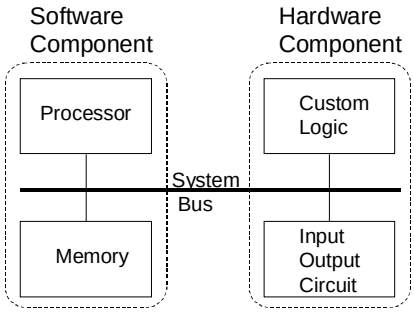

Fig.1 Embedded system hardware/software division.

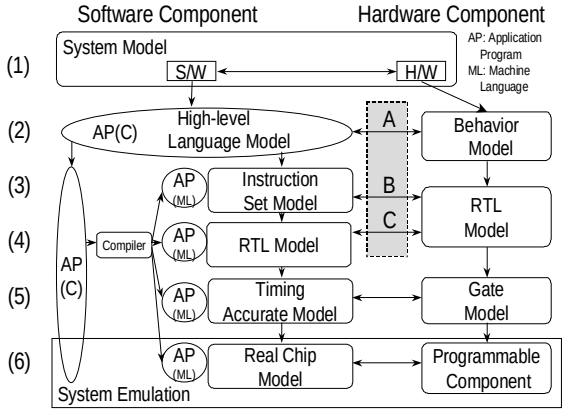

Fig.2 Verification stages in top-down design.

or on the method used for modeling hardware and software. A model whose design is based on a component logical bus architecture is replaceable and reusable. By combining such replaceable models, it is possible to quickly realize seamless co-simulation.

In this paper, we present a modeling method based upon a component logical bus architecture. In section II, we describe the method used for co-simulation in the top-down design of embedded systems together with related problems. In section III, we propose a method of hardware and software modeling and co-simulation based upon a component logical bus architecture, and in section IV we describe and analyze the experimental results of a line-drawing system for graphical processing using the proposed methods. In section V we further discuss and summarize the results of this paper.

II. Problems in the Top-down Hardware/Software Co-Design

A typical embedded system (Fig. 1) consists of software components, hardware components, and an interface connecting them.

Software and hardware components exchange information via a system bus to realize the system functions for an application.

Such a process of top-down hardware/software co-design for embedded systems can be divided into the six verification stages (as shown below and in Fig. 2), corresponding to software component modeling methods.

(1) System model
In the first stage, the target application system is modeled without distinguishing between hardware and software functions.

(2) High-level language model

In the second stage, co-simulation is achieved without processor dependence. High-level language programs for applications to be executed by the software component are converted into the machine language of the host machine running the simulation and are executed directly.

Co-simulations can thus be begun before the processor architecture is determined. Moreover, in combination with behavior models of hardware components, high-speed co-simulation is possible.

(3) Instruction set model
In the third stage, co-simulation is performed by executing machine language instructions for the target processor. High-level language programs executed as software components are first converted into the target machine language, then the converted machine language is interpreted and executed by the instruction set simulator of the target processor. Hardware components are modeled at the RTL level.

(4) RTL model
In the fourth stage, co-simulation is performed on a clock-synchronized cycle base using RTL models of the target processor and hardware components. The simulation is more accurate, but more time is required for simulation execution.

(5) Timing-accurate model
In the fifth stage, timing-accurate co-simulation is executed using gates models of processors and hardware components. Simulation can be performed with accurate timing, but execution time is extremely long.

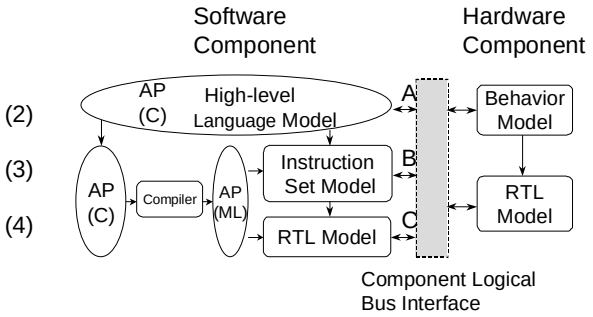

Fig.3 Seamless top-down co-simulation environment.

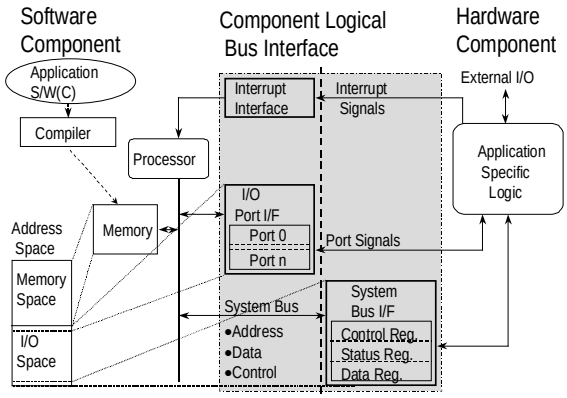

Fig.4 Configuration of the software and hardware component logical bus interface.

(6) Real-chip model

In the sixth stage, the target system is emulated by mapping the synthesizable models of processors and hardware components onto programmable logic devices and switch arrays. High-speed execution is possible in an environment approximating that of an actual system.

The following problems arise when performing top-down co-simulations of an embedded system.

Of the six verification stages, the interfaces for three--

- the high level language model (Fig. 2, part 2-A),
- the instruction set model (Fig. 2, part 3-B) ,and
- the RTL model (Fig. 2, part 4-C)

--normally depend on processor and system bus architectures, and are currently specified individually. Consequently, when conducting top-down design it may be necessary to modify the modeling at each stage, and the models created by others may not be usable without alteration. These problems may arise and impede the efficient co-design in the top-down design methodology.

This paper describes a seamless co-simulation method achieved through the introduction of a component logical bus architecture that is independent of processor and system bus architecture and is used as the interface for these three verification stages.

III. A Proposal of Seamless Co-Simulation Using a Component Logical Bus Architecture

The interfaces (A, B, C in Fig. 2) between the hardware components and the software components are currently dependent on the verification stages of the high-level language model, the instruction set model, and the RTL model.

As a common interface between the hardware components and software components in these three verification stages, we propose the component logical bus architecture. By introducing the common interface, the replacement of the models between different stages and the combination of the models between hardware and software are expected to be possible with little or no modification. The purpose is to create replaceable and reusable component models, realize seamless top-down co-simulation environment as shown in Fig.3, and shorten the translation time between the verification stages.

A. Component Logical Bus Architecture

The component logical bus consists of three different
kinds of signals:

- system bus signals,
- port signals, and
- interrupt signals.

The configuration of the component logical bus which serves as the interface between software components and hardware components is shown in Fig. 4. Its implementation is dependent on the verification stages.

(1) System bus signals: The system bus consists of address, data, and control signals. The specified data is read or written according to the control signal and the address. The address space consists of the memory space and I/O space. The memory of the software component is allocated in the memory space. Ports within the software component and registers in other hardware components are allocated in the I/O space (memory mapped I/O architecture).

(2) Port signals: These are specialized signals capable of directly interfacing between software

components and hardware components. Ports within the software components are allocated in the I/O space.

(3) Interrupt signals: When software and hardware components have completed an operation, or when an error condition is detected, these signals are used to notify a software component.

Exchange of data between software components and hardware components takes place as follows:

By writing to ports and registers within hardware components, software components initiate hardware component operations and transfer data.

Hardware components use two methods to notify software components of the completion of processing. One is direct notification via an interrupt signal; the other is by setting a completion flag in the port or in the status register of the hardware component. By reading ports and registers within hardware components, software components receive data and status information from hardware components.

B. Implementations

Implementation methods for System bus signals and Port signals are shown in Fig. 5.

(1) Common Memory Method

In the high-level language model in Fig. 5 part(a), the component logical bus interface is implemented by using common memory. Consequently, I/O space is allocated as part of a common memory address space, and data is exchanged through access of this space by both components.

The software component writes data to the ports and to the hardware component registers allocated to the common memory space. When a series of operations has been completed, a startup event is passed to the hardware component.

The hardware component reads the required data from the ports and from the hardware component registers allocated to the common memory space, and then initiates the corresponding processing. When processing is completed, the processing results and status are similarly written to ports and to registers allocated to the common memory space.

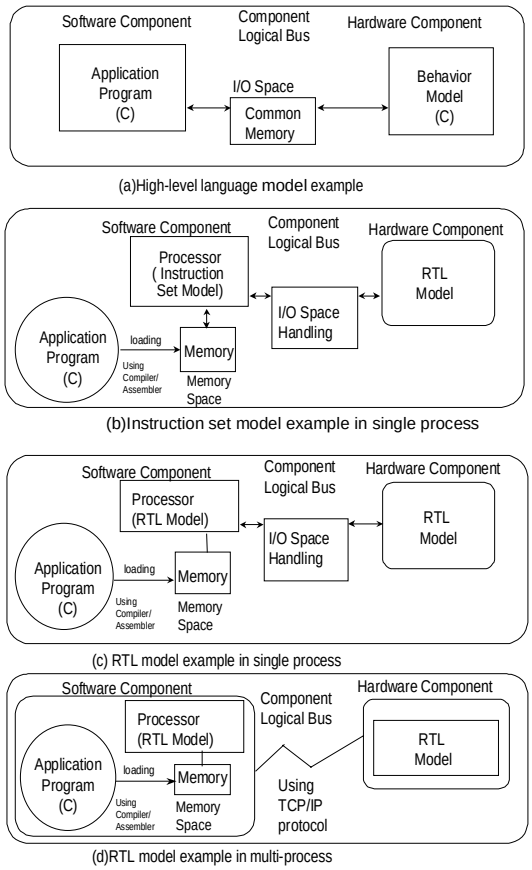

Fig.5 Implementation examples.

At fix time intervals, the software component reads the ports and hardware component registers allocated to the common memory space and so obtains the processing results and status from the hardware component.

(2)Signal Event Method in Single Process

In the instruction set model in Fig. 5 part(b) and in the RTL model in Fig.5 part(c), the component logical bus interface is implemented by signal event method in single process simulation.

When the processor accesses the I/O space in a write operation, the write data , the write address and write operation signals are sent to the hardware component. After receiving a write operation signal, the hardware component takes the data in the register specified by the write address. When the processor within the software component accesses the I/O space in a read operation, the read address and a read operation signal are sent to the hardware component.

After receiving a read operation signal, the hardware component outputs the data in the register specified by the read address.

(3)Inter-process Communication Method in Multi Process

In the RTL model in Fig.5 part(d), the component logical bus interface is implemented by inter-process communication method in multi process simulation.

The processor and the hardware component are operated in parallel in different processes. The data communication via I/O space between the processor and the hardware component is executed by inter-process communication.

IV. Experimental Results and Evaluation

A. The Evaluation Model

In order to verify the efficacy of co-simulation in top-down design through the introduction of a component logical bus architecture, we conducted experiments which applied the proposed methodology to a line-drawing system for graphical processing.

The configuration of the line-drawing system used in evaluations appears in Fig. 6. The software component consists of a processor unit and a memory unit. The line drawing control program loaded in the memory unit is executed on the processor to control the sequence of line drawing. The hardware component is made up of a line-drawing engine which rapidly draws lines between two specified points, and a graphical display unit.

For the processor, we have used a 16-bit ASAP (application-specific adaptable processor) [8] which is generated by a high-level synthesis tool as a function of the CPU bit-width and register file size required by the application program.

The 16-bit ASAP had a cost of 5.3K gates and the hardware component cost was of 1.5K gates.

In Fig. 6, coordinates of the start point are set in the start point registers (X0,Y0) and coordinates of the end point are set in the end point registers (X1,Y1). Then, the start flag EN is set and the line drawing engine is activated. When the line engine completes the drawing of a line, the end flag DONE is set. In this case, a polling mechanism is used in place of an interrupt mechanism.

B. Results and Evaluation

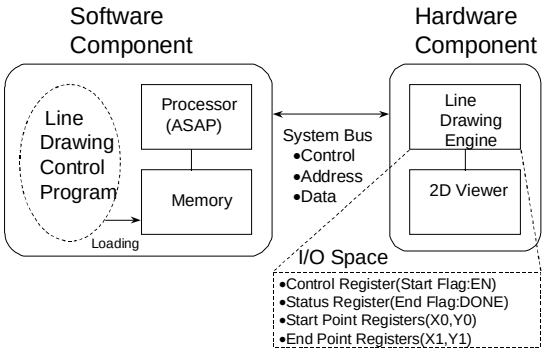

Fig.6 Configuration of a line drawing system.

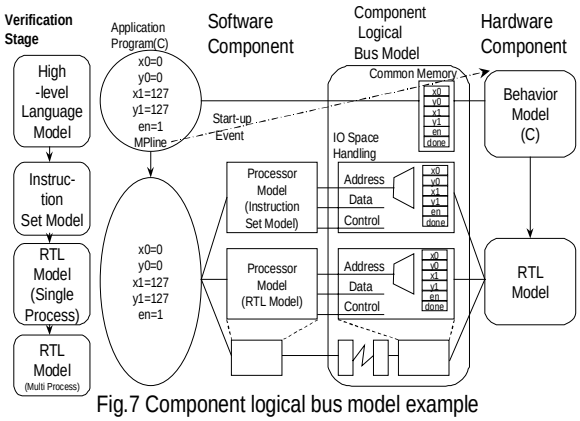

Fig.7 Component logical bus model example of a line drawing system.

In order to verify the effectiveness of our design method, we created a system model for the line-drawing system. After division into software components and hardware components, we constructed processor-independent high level language models, processor-dependent instruction set models, and conventional RTL models, and then evaluated each.

Fig.7 shows a component logical bus model example of a line drawing system. The high-level language model in the software components was replaced easily by an instruction set model with a slight modification, such as removal of the startup event. The instruction set model was replaced by the RTL model with no modification. Moreover, both the instruction set model and the RTL model of the processor could be connected to the RTL model of the hardware component with no modification. Therefore, our approach could shorten the translation time between the verification stages.

Co-simulation execution performance was measured for fore different cases:

- (a)high level language model,
- (b)instruction set model,
- (c)RTL model in single process, and
- (d)RTL model in multi-process.

The results appear in Table 1 (normalized by a value of unity for the results of the RTL model simulation in multi-process).

Co-simulations of the high-level language model were executed directly by compiling on the host machine, including the behavior model in C language of the hardware component. For the instruction set model, an ASAP instruction set simulator written in C and a VPS cycle-based simulator [10] capable of high-speed execution of the hardware component RTL model were used in the single process environment. For the RTL model in single process, VPS was used to execute simulations of the ASAP RTL model and the hardware component RTL model. For the RTL model in multi-process, an ASAP instruction set simulator and VPS were used, and VPS simulator was executed synchronously with ASAP instruction simulator using inter-process communication method based on TCP/IP protocol.

Performance between single process and multi-process varies depending on the communication times. The amount of communication a time did not have much effect on the performance. In this example, the speed of co-simulation between single process and multi-process was increased by 370 times over that for the single process model. In this case communication times are about 40,000.

Performance between instruction set model and RTL model varies depending on the scale of the processor and the extent of application of hardware components. In this example, the speed of co-simulation for the instruction set model was increased by 1.3 times over that for the RTL model. Owing to the fact that RTL models were used for the hardware component in both cases and that the cycle-based simulator was used for both, there was not so great a difference in performance.

In the high-level language model, all simulations were executed directly on the host machine in the C language, and so a considerable speed increase over the RTL model in multi-process by a factor of over 11,100 was achieved.

To summarize the above results, introduction of a component logical bus architecture served to make the component models replaceable, shortened the translation time between the verification stages and expedited co-simulation at each stage of verification.

Table.1 Results for line drawing system example.

Verification Model	Modeling				Co-simulation Performance (RTL Model=1)
	S/W		Interface	H/W	
	AP	Processor			
High-level Language Model(a)	C Model		Common Memory Communication	C Model	11,100
Instruction Set Model (b)	A Slight Modified C Model ↓Compile Target Machine Inst.	Inst-ruction Set Model	Signal Events Communication in Single Process Environment	RTL Model	480
RTL Model (c)		RTL Model			370
RTL Model (d)		RTL Model	Inter-process Communication in Multi Process Env.	RTL Model	1

V. Conclusion

We have shown that by introducing a component logical bus architecture as the common interface between the software and hardware components comprising an embedded system, each component can be modeled in a manner independent of other components, and that the translation time between the verification stages can be reduced.

In particular, by adoption of a high-level language model, it was possible to execute co-simulation prior to determination of the processor architecture, and a consequent increase in execution speed by a factor of 11,100 over RTL models in multi-process was demonstrated.

The following problems must be left as themes for future research:

- Because a component logical bus architecture was introduced as the interface between hardware and software components, it may be possible to create RTL-level interface circuitry for the component physical bus. To enable efficient top-down design, further studies aiming at high-level synthesis of such elements will be needed.

- In this paper, we discussed the model based on component logical bus architecture was replaceable for each verification stage. In near future, it will be important to shift the verification stage dynamically in single verification environment. We will enhance the component logical bus architecture to apply for these needs.

- When the application program is executed on real time OS(RTOS), the verification performance is inefficient in this model. The study of co-simulation methods between embedded software on RTOS and hardware is needed.

References

[1] D.E. Thomas, J.K. Adams and H. Schmit, "A model and methodology for hardware-software codesign," IEEE Design & Test of Computers, vol. 10, no. 3, pp. 6--15, 1993.

[2] R.K. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems," IEEE Design and Test of Computers, vol. 10, no. 3, pp. 29--41, Sept. 1993.

[3] H. Yasuura, S. Nakamura, H. Tomiyama, and H. Akaboshi, "Hardware-software codesign with soft-core processor," SASIMI '95, pp. 79--84, Aug. 1995.

[4] H. Koizumi, K. Seo, F. Suzuki, Y. Ohtsuru, and H. Yasuura, "A proposal for a co-design method in control systems using combination of models," IEICE Trans. Inf. & Syst., vol. E78-D, no. 3, pp. 237--247, March 1995.

[5] J.K. Adams and D.E. Thomas, "The design of mixed hardware/software systems," in Proc. 33rd DAC, pp. 515--520, 1996.

[6] B. Lin, S. Vercauteren and H.D. Man, "Embedded architecture co-synthesis and system integration," Codes/CASHE'96, pp. 2--9, March 1996.

[7] B. Schnaider and E. Yogev, "Software development in a hardware simulation environment," in Proc. 33rd DAC, pp. 684--689, 1996.

[8] B. Shackleford, M. Yasuda, E. Okushi, H. Koizumi, H. Tomiyama, and H. Yasuura, "Satsuki: an integrated processor synthesis and compiler generation system," IEICE Trans. Inf. & Syst., vol. E79-D, no. 10, pp. 1373--1381, Oct. 1996.

[9] ZUKEN Incorporated, Tsutsuji Reference Manual, 1996.

[10] W.B. Culbertson, T. Osame, Y. Otsuru, J.B. Shackleford, and M. Tanaka, "The HP Tsutsuji logic synthesis system," Hewlett-Packard Journal, pp. 38—51, Aug. 1993.