A Hardware/Software Co-simulation Environment for Micro-processor Design with HDL Simulator and OS interface

Yoshiyuki Ito Yuichi Nakamura

C&C Research Laboratories
NEC Corporation
Kawasaki, Japan

Abstract— We proposed a hardware/software co-simulation environment using an RTL simulator with a software language interface. The proposed simulation environment introduces the "OS interface (OSIF)," which invokes system calls in the OS on the simulation platform to execute application software. The OSIF consists of data adaption facility and function correspondence management allowing it to cooperate with the OS of the simulation platform. We show the results of experiments with an R3000-compatible processor model. This environment verified our processor model with SPEC benchmarks that require various operating system services. For example, with a lisp interpreter program *li*, our detailed RTL description for the core part of R3000 was simulated only within 20 hours on a 109 MIPS workstation.

I. Introduction

For detailed verification of micro-processor design, execution of application software with operating system (OS) functions such as system call transactions and accesses to external devices is indispensable. Since designers desire to try several types of processor architectures, the verification environment needs to be available as quickly as possible. Thus, simultaneous simulation of hardware and software on processor design is crucial.

There is a trade-off between accuracy and simulation time in the verification environment. When the hardware is modeled in higher level description, the software is easily applied to the model and simulation time is short, but the verification may be inaccurate. On the other hand, although the simulation will be accurate when the hardware is modeled in detail, it could be too difficult to execute the application software with the model, or its execution speed is extremely slow.

In order to solve this problem, in most current design styles software simulation uses hardware models which are different from detailed hardware designs. Application software is executed with a trace-level or pipe-line-level simulator. Since such simulation environments treat an ideal hardware, processor performance might not be realistic. For development of circuit-level hardware designs, hardware description language (HDL) simulation is used. At this level of simulation, hardware/software co-verification with HDL simulation is not easy, because the software must be executed with system calls and accesses to external devices. In addition, verification speed on the HDL simulator is at most 100 patterns per second. Thus, executing application software such as SPEC benchmarks [1] requires a great amount of simulation time for emulating full operating system codes on the HDL processor model.

Several simulation methods have been proposed for efficient verification and quick software development [2, 3, 4, 5, 6]. Since most of these simulation environments are aimed at developing embedded systems. Therefore they are unsuitable for a developing general micro-processors. For example, the Virtual Processor model [7] divides a processor model into two abstractions. One is a target hardware model to verify, and the other is a trace-level software CPU model. This transformation from HDL to software CPU model makes implementing a simulator for application software execution possible. Although this approach is suitable for software development, the hardware model is too simple to detect several types of flaws such as hazards.

In this paper we propose a hardware/software co-simulation environment attained through the use of an RTL simulator with a software language interface. To execute the application software, we introduce the "OS interface (OSIF)" which invokes system calls in the OS on the simulation platform. Designers can omit those parts of HDL descriptions which are dispensable in the early stages of the target processor design. The OSIF consists of *proxy functions* for the omitted parts of the HDL descriptions. It also has *transform functions* that translate several hardware-dependent parameters such as address space from the target processor into the simulation platform, and vice versa. Whereas the Virtual Processor model abstracts hardware parts, our simulation environment substitutes OSIF functions, including the entire simulation of system calls for application software, for a large part of the hardware model. Thus, our environment achieves high-speed simulation for general purpose processor design.

The simulation environment, on which application programs are executed, consists of the following components:

1. Description of the target processor core written in RTL HDL.
2. Operating system of simulation platform through the OSIF.
3. Proxy functions that equate to abridged functions on the HDL description.

Since our simulation environment combines detailed simulation for the core part of the processor with rapid simulation using proxy software, it greatly decreases simulation time.

Experimental results obtained with an R3000-compatible processor model are presented to demonstrate the validity of this approach. This environment successfully verifies the effectiveness of our processor model in executing SPEC benchmark programs which require various operating system services. As an example, with a lisp interpreter program *li*, our detailed RTL description for the core part of R3000 was simulated in less than 20 hours on a 109 MIPS workstation. Since a detailed RTL description for the whole R3000 takes weeks to simulate, our simulation environment achieves greatly improved target processor core verification in the early design stages.

II. Background to the hardware/software co-simulation with an HDL simulator

In this section, we describe the difficulties in executing application programs on the HDL simulator.

A. Executing an application program with a general HDL simulator

In the circuit design stage, a processor model is written in a hardware description language (HDL) such as Verilog or VHDL.

However, there are two main problems in simulating a new processor with an HDL simulator which does not specialize in processor modeling:

Simulation speed:

It takes several weeks merely to boot up the operating system.

Manipulation of external devices:

The time scale of an HDL simulator is about 1M times slower than the speed of the external devices.

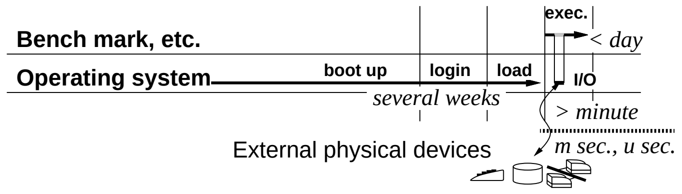

Fig. 1. Typical simulation progress

Consequently, it is not easy to verify the result of large-scale software code execution. In the last phase of development, the processor model is verified by executing the software. For this reason and because of time limitations, it is sometimes necessary to dispense with fully sufficient verification.

B. System-level behavior verification

Processor performance must be evaluated not only through the processor movements but also the system-wide behavior.

In a conventional simulation environment, system-level behavior is evaluated and verified by the HDL processor model and bus-level simulation models of external devices, because the simulator uses only an HDL description.

1. In general, bus models of the external devices are provided as existing processor families.
2. The scope of bus-level architecture is limited to that of existing external device models.
3. These external devices will be outdated by the time the processor is put on the market.

For these reasons, external devices must be developed at the same time as the processor in a conventional development sequence.

C. Process of debugging system programs with the HDL processor model

Since a new processor's system software is modified from other software, the core of the debugging process is repeated execution around the updated code, even if the processor being designed has wholly new specs. Therefore, two main problems of HDL simulation environment (cf. Section II–A) also have a very serious effect on the software debugging process. Furthermore, state control is a characteristic problem of a debugging process:

The RTL processor model is too strict and complicated to set the status safely through the debugger.

III. The hardware/software co-simulation method

In this section, we describe the design flow of a processor model through the use of a hardware/software co-simulation environment.

A. The process of gradually designing the processor description

We propose a co-simulation environment which manages partially implemented HDL processor models.

The deficient region of these HDL processor models are supplemented by software-based proxy functions (S/W-PF). Therefore, our simulation environment can handle both behavior-level HDL descriptions and detailed gate-level HDL descriptions.

Fig. 2 shows an outline of the partially implemented HDL processor model.

In the early stage of processor design, designers investigate many kinds of architectures which are partially written in HDL. In this stage, other areas of the processor model such as FPU or bus interfaces are substituted by S/W-PF (Fig. 2(a)).

As the processor design progresses, the HDL description is enlarged and increasingly detailed (Fig. 2(b)). Finally, the fixed area of the processor design is replaced with a high-speed simulation tool (Fig. 2(c)).

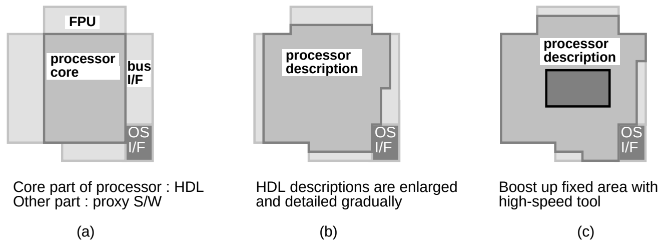
FPU
processor core
bus I/F
OS I/F
Core part of processor : HDL
Other part : proxy S/W
(a)
processor description
OS I/F
HDL descriptions are enlarged and detailed gradually
(b)
processor description
OS I/F
Boost up fixed area with high-speed tool
(c)

Fig. 2. The gradual processor description process

B. Software-based proxy functions

The functional capacity of S/W-PFs ranges from the scale of an H/W functional block to that of a software procedure which includes the execution of system calls.

There are several ways to incorporate S/W-PFs in the system. As an example, these functions can be connected through the TEXTIO feature of hardware description language. In our environment, the HDL simulator's "software language interface" is adopted, because of its speed and simplicity of installation.

C. Application programs executed by processor model

The object codes which are executed with the processor model are almost the same as the object of the final product of the processor.

If a compiler of the processor is completed, we can use it to build an object.

Our simulation environment supports the cooperative and simultaneous development of the compiler and the processor.

IV. Implementation of the simulation environment including operating system execution

We describe the implementation of an "OS interface (OSIF)" as an S/W-PF which executes the "system calls" of the UNIX operating system in their entirety.

Our environment enables trace data of the application program execution to be obtained because the OSIF solves the two major simulation problems described in Section II–A.

As seen in Fig. 3, the rough structure of our environment comprises an H/W–S/W interface (HSIF) within the processor descriptions and the OSIF within the HDL simulator. These two interfaces are described in detail in the following section.

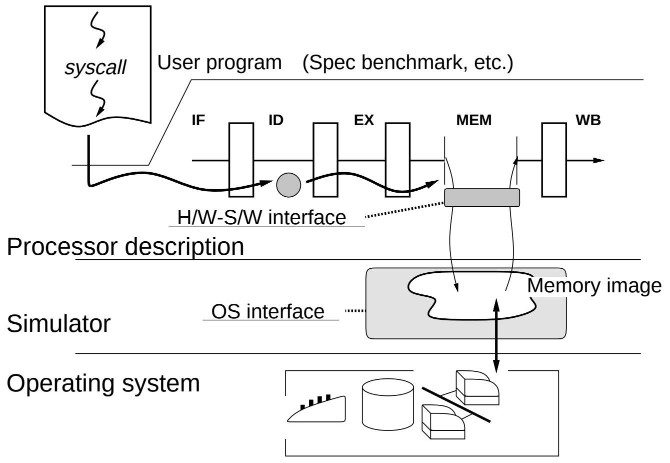

Fig. 3. Construction

A. H/W–S/W interface

The HSIF is in the HDL processor description. This function calls OSIF which executes omitted functions of the HDL processor description.

The implementation of the HSIF adopts the software language interface of HDL. These descriptions are written in the same form as in the other part of the HDL processor model. On the other hand, since the OS of the design platform accesses the target application program's memory objects, the HDL processor model operation must be stopped while the HSIF is calling the OSIF.

Therefore, the HSIF must be put in sequential statements which guarantee the order of execution by the HDL simulator.

The boundary between the HDL processor model and the unimplemented area is configurable to any position. In our environment, the boundary is made identical to the boundary of the H/W function blocks to reduce the load of the HSIF.

A.1 Data transfer

Fig. 4 shows a simulated flow of a MIPS R3000-compatible trial chip (cf. Section V–A). The HSIF is set during the MEM stage of this chip's pipeline.

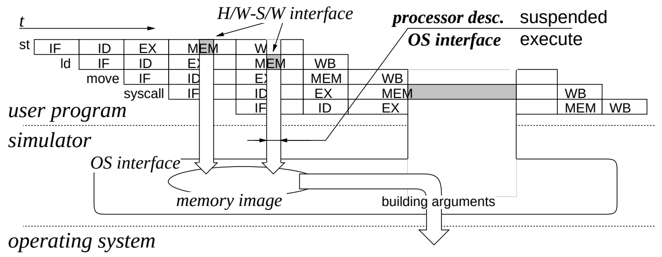

Fig. 4. Interface

To reduce the load of the HSIF, the HSIF is substituted for the MEM stage which accesses the external environment, in the same way as the original chip accesses the external devices during the MEM stage.

A.2 Instruction decoding

The simulator executes a "system call" triggered by the "syscall" instruction which is included in the target application program.

When these syscall instructions are decoded at the ID stage of the processor description, the HSIF calls OSIF during the MEM stage (Fig. 3).

In a real processor, instruction streams are flushed after the end of a system call, because the processor brings out a "return from exception" instruction.

The HSIF simulates this process by executing a "jump to the next instruction after syscall" instruction.

B. OS interface

When the OSIF is called by the HSIF, the OSIF invokes the OS system calls of the design platform as a replacement for the simulation of the OS by the HDL simulator.

The OSIF has the following functions which enable it to call the platform's OS.

- Constructing arguments
- Translation of the address space

From the implementation aspect, the OSIF is constructed with the following two structures.

- An interface description table (for common use of S/W-PF).
- Each procedure entry referred to by the interface description table, such as address translation, definition of the calling sequence, etc.

B.1 Interface description table

Our simulation environment employs an interface description table which defines "what is substituted" and "how to substitute." This table can define other types of proxies.

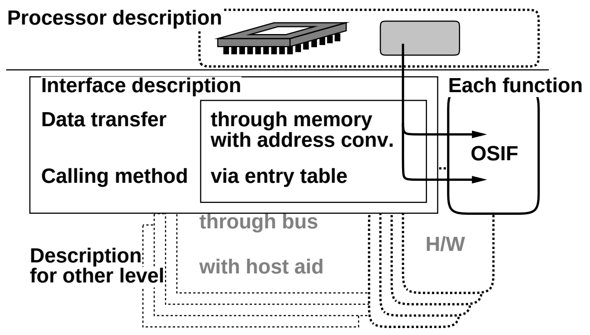

Fig. 5. Interface description table

B.2 Data conversion

Since the address space of memory objects which are operated by the simulator is assigned by the platform's OS, the address spaces of these objects are different from the address space of the processor model. Therefore, our environment makes address conversion between these two environments possible (see Fig. 6).

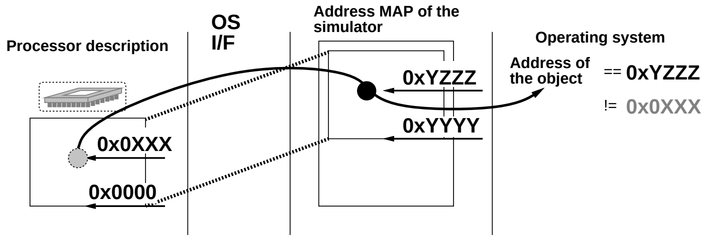

Fig. 6. Address conversion

Data conversion functions are given on the OSIF, except when using a type-convert function provided by the simulator, such as "std_logic to integer."

At this time, only two items need translating, because the data structure of the simulation platform is similar to those of the processor models. These items are:

- Address of memory object
- Descriptor of file I/O

Additional translation, such as byte endian, can be defined according to the architecture of the processor model.

C. Application programs executed by the processor model

Application programs which are executed by the processor model are made by the compiler for the new processor, except for the trigger function of the system call which uses the syscall instruction.

In our environment, the processor model simulate the execution of user level codes which include functions of the C-language standard library. Since these user-level codes are independent of a system level architecture, such as an access method of the external I/O devices, the new processor's compiler is checked and debugged simultaneously from the early stage of processor design.

The trigger function, which includes syscall instructions, is replaced with an exclusive library in order to notify the HSIF of the point of the system call.

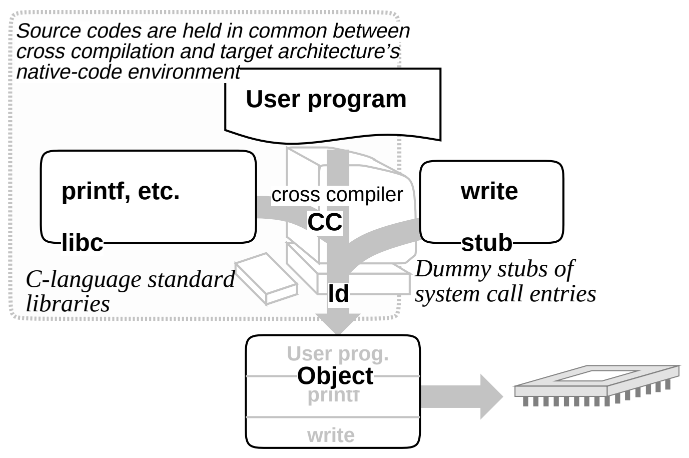
Source codes are held in common between cross compilation and target architecture's native-code environment
User program
printf, etc.
libc
cross compiler
CC
write
stub
C-language standard libraries
ld
Dummy stubs of system call entries
User prog.
Object
printf
write

Fig. 7. Compilation method of the target program

V. Experimental results

Through the use of the OSIF, we were for the first time able to execute SpecInt '92 benchmark programs successfully on the HDL processor model at high speed.

A. The model case – A compatible chip for MIPS R3000

We designed an RTL description for a MIPS R3000-compatible chip with VHDL. This design is only the core part of the MPU; the design specs are shown below. The other parts of R3000's functions, such as FPU, are substituted by software functions.

Specs:

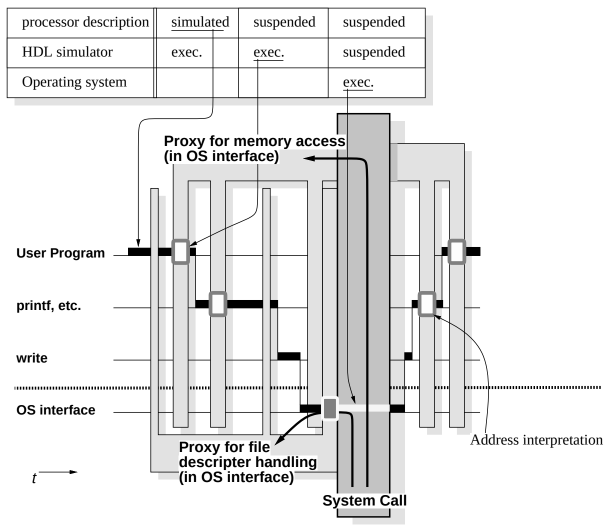

Fig. 8. Example – printf

1. MPU core is 6K lines of behavior level VHDL description.
2. The FPU and the bus interface are substituted by proxy functions.
3. Fifty-eight of R3000's 71 instructions are implemented (not including break instruction, etc.).
4. Proxy functions are 6K lines of C language description.

Using this design, we applied the OS interface to the simulation of benchmark programs in the SpecInt.

B. Verification flow

System calls used in target application programs are executed by the OS of the simulation platform through the OS interface. Other instructions including library functions are executed by a processor model which is verified by the HDL simulator.

Fig. 8 shows the execution flow of a function "printf" as a typical pattern of the library function's calling sequence.

In real systems, the printf library function analyzes its arguments and constructs an output string at the user level, and then outputs the composed string with a "write" system call.

Also, in our simulation environment, the user-level instructions of the printf are executed in the HDL processor model, therefore trace patterns of them are produced by the HDL simulator. Data objects which need translation (cf. Section IV–B.2) are interpreted one by one in the simulation and stored into the OSIF's memory pool. The write system call used in printf, executed by the simulation platform machine's OS and displays the string on the screen of the simulation environment.

Fig. 9 shows the simulation process of an "mmap" system call. The mmap system call establishes a mapping

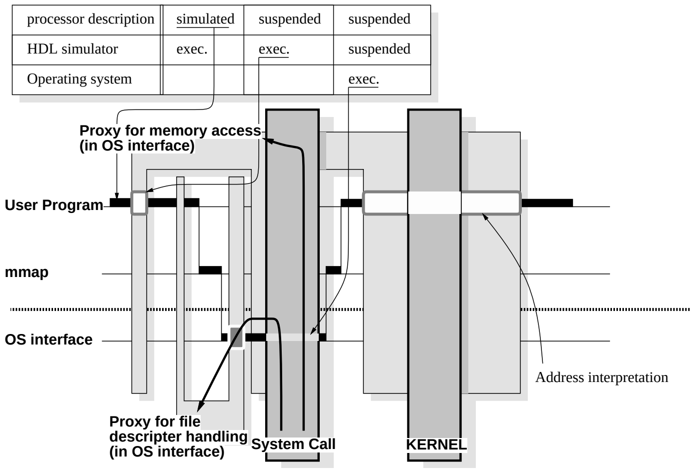

Fig. 9. Example – mmap

between a process's address space and an object, such as the contents of a regular file or shared memory pages.

When the application program accesses the mapped region, a kernel interrupts and delivers the contents of the object to the application program, in the same way as a page fault does.

In the simulation environment, the simulation platform OS also interrupts when the target application program accesses the mapped region. However, the actual access to the memory is executed by the OSIF. The OSIF runs under the simulator's address space, and the processor model is set to safety status by the HSIF before it calls OSIF. Thus, no contradiction is generated in spite of the unexpected interruption occurring.

C. Performance

Table I presents the result of a control experiment, in which an R3000-compatible MPU's simulation time on the SUN SparcStation 10/40 is compared with the execution speed of a real system, i.e. an NEC EWS4800/220 with an R3000 MPU (30 MHz).

TABLE I
RESULTS OF THE SIMULATION

program	time		ratio
	real system (user time) sec	simulation sec	
eqntott	0.04 (0.02)	952	24,000
compress	0.09 (0.08)	12,032	130,000
espresso	0.09 (0.07)	10,660	120,000
li	0.21 (0.19)	71,518	210,000
average			120,000

VI. Conclusion

In this paper, we proposed a hardware/software co-simulation environment using an RTL simulator with a software language interface. The proposed simulation environment introduces "OS interface (OSIF)" which invokes system calls in the OS on the simulation platform to execute application software. The OSIF consists of a data adaption facility and function correspondence management to cooperate with the OS of the simulation platform.

Thus, with the OSIF, we achieve a system-level simulation environment which includes operating system functions and manipulates facilities of external devices. In addition, our simulation environment establishes detailed simulation for the core part of the processor in conjunction with rapid simulation by proxy software, and therefore a great decrease in the simulation time is achieved.

Experimental results obtained with an R3000-compatible processor model were presented. Our simulation environment successfully verifies the effectiveness of our processor model in executing SPEC benchmark programs which require various operating system services. As an example, with a lisp interpreter program *li*, our detailed RTL description for the core part of R3000 was simulated in less than 20 hours on a 109 MIPS workstation.

References

[1] *SPEC Newsletter,* SPEC, 1992.

[2] D. Becker, R.K. Singh, and S.G. Tell, "An Engineering Environment for Hardware/Software Co-Simulation," *Proceedings of the 29th DAC,* pp. 129-134, 1992.

[3] R.K. Gupta, C.N. Coelho Jr., and G. De Michel, "Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components," *Proceedings of the 29th DAC,* pp. 225-230, 1992.

[4] Y. Kra, "A Cross-Debugging Method for Hardware/Software Co-design Environments," *Proceedings of the 30th DAC,* pp. 673-677, 1993.

[5] J. Rowson, "Hardware/Software Co-simulation," *Proceedings of the 32nd DAC,* pp. 439-440, 1994.

[6] A. Ghosh, M. Bershtey, R. Casley, C. Chien, A. Jain, et al., "A Hardware-Software Co-simulator for Embedded System Design and Debugging," *Proceedings of the ASP-DAC '95,* pp. 155-164, 1995.

[7] B. Schnaider and E. Yogev, "Software Development in a Hardware Simulation Environment," *Proceedings of the 33rd DAC,* pp. 684-689, 1996.

[8] D. Patterson and J. Hennessy, *Computer Architecture — A Quantitative Approach,* Morgan Kaufmann Publishers, 1990.

[9] R. Lipsett, C. Schaefer and C. Ussery, *VHDL: Hardware Description and Design,* Kluwer Academic Publishers, 1989.

[10] G. Kane and J. Heinrich, *MIPS RISC Architecture,* Prentice Hall Inc., 1992.