IPSJ Transactions on System LSI Design Methodology Vol. 1 126–130 (Aug. 2008)

Short Papar

Embedded System Covalidation with RTOS Model and FPGA

Seiya Shibata,[†1] Shinya Honda,[†1] Yuko Hara,[†1]
Hiroyuki Tomiyama[†1] and Hiroaki Takada[†1]

This paper presents a software/hardware covalidation environment for embedded systems. Our covalidation environment consists of a simulation model of RTOS which fully supports services of ITRON, multiple hardware simulators, FPGA and a covalidation backplane. All of the simulators are executed concurrently with communication. The RTOS model can be executed on the host computer natively, therefore the software can be simulated much faster than on an instruction set simulator. FPGA can execute the hardware much faster than HDL simulators. With the RTOS model and FPGA, both application software and hardware can be validated in a short time. In the experiment, with using our covalidation environment, we perform covalidation of an MPEG4 decoder system and show the effectiveness of the covalidation environment.

1. Introduction

Software and hardware for embedded systems have been increasing their size and complexity, while the time-to-market pressure has also been increased. In order to satisfy both of the requirements, fast software/hardware covalidation is one of the key technologies.

In typical embedded real-time systems, software consists of application tasks and an RTOS, and therefore, RTOS should be incorporated in the software/hardware covalidation flow in order to verify the overall system functionality. In the past, several researchers developed simulation models of RTOSs to be used in their hardware/software cosimulation frameworks [1)–3)]. They assume that all of the system components (including software components and hardware ones) are written in a single system-level description language (SLDL) such as SystemC and SpecC. Although their approaches lead to fast cosimulation, one of their serious drawbacks is that hardware components in the SLDL are often just simulation models whose detailed functionality might be different from that of the final implementation descriptions in HDL.

In our prior work, we developed an RTOS simulation model and a multilingual cosimulation platform on which HDL simulators can be executed [4)]. While the RTOS model can be executed natively (hence fast) on a host computer, HDL simulators are inevitably slow. Such cosimulation is appropriate for hardware debugging, but inappropriate for functional verification of embedded software.

In this work, we have developed a software/hardware covalidation environment to be used for embedded software verification. In the covalidation environment, embedded software is executed directly on the host while hardware is executed on an FPGA. This work solves the two problems of the past approaches at the same time. Since final HDL designs (not simulation models) can be used for the software/hardware covalidation, unexpected inconsistency between software and hardware can be avoided. In addition, our covalidation environment brings the significant speedup compared with traditional cosimulation using HDL simulators. It should be noted that our covalidation environment presented in this paper is complementary to the past cosimulation environments. This work provides yet another covalidation solution to embedded software designers.

This paper is organized as follows. Section 2 shows our prior work on cosimulation which is the basis of this work presented in this paper. Section 3 describes the covalidation environment with an RTOS model and FPGA. A case study with an MPEG4 decoder system is presented in Section 4. Finally, Section 5 concludes this paper with a summary.

2. Prior Work

The covalidation environment presented in this paper has been built upon our prior work on cosimulation. This section describes the cosimulation environment which we developed in our prior work [4)].

2.1 Overview

In our past study, we developed a cosimulation environment and an RTOS model [4)]. The overall structure of the cosimulation environment is shown in **Fig. 1**. The cosimulation environment consists of an RTOS model, multiple hard-

†1 Graduate School of Information Science, Nagoya University

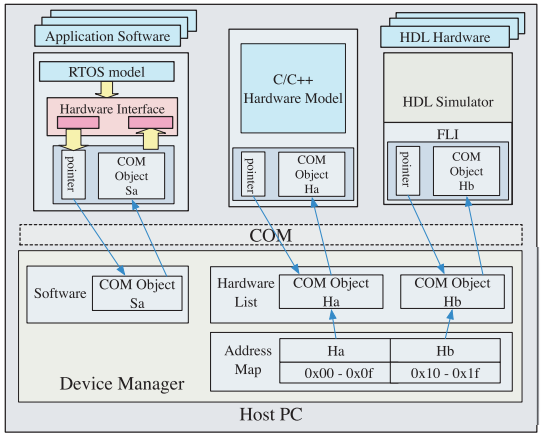

Fig. 1 Past covalidation environment overview.

ware simulators, and a cosimulation backplane named *Device Manager* (*DM*). The RTOS model supports all of the service calls which are defined by μITRON 4.0 Standard Profile [5]. ITRON is a standardized specification of RTOS for small- and mid-scale embedded systems, and is one of the most popular RTOSs in Japanese industries. The RTOS model is implemented in C, so that it is directly executable on the host computer. The cosimulation environment is very flexible in that it features plug-and-play of various simulators such as HDL simulators, the SystemC simulators, functional hardware models in C/C++, and instruction-set simulators. Each simulator is executed as an application on an MS-Windows-based host computer.

In summary, our cosimulation environment developed in the past features

- native (hence fast) execution of application software,
- complete support of a standard RTOS,
- cosimulation with various hardware simulators such as HDL simulators and C/C++ functional models.

2.2 Communication between Simulators

In the cosimulation environment, various simulators can communicate with each other using a flexible communication mechanism as follows [4].

Memory mapped I/O is assumed in our cosimulation environment, and unique address spaces are assigned to hardware simulators. DM manages a mapping table of the addresses and the hardware simulators. When the software needs to perform a read/write access to a hardware simulator, first the software sends an access request with an address to DM, and then DM selects a corresponding hardware simulator by looking up the address map and transfers the request to the hardware simulator.

The transfers of requests are implemented with a standard remote procedure call (RPC) on MS-Windows, named *COM*. COM is a mechanism for communications between MS-Windows applications. In order for simulators to communicate with each other, the RTOS model, hardware simulators and DM have so-called COM objects which realize the COM-based communication (shown in Fig. 1).

ITRON project [5] defines an API for hardware accesses. For example, application software reads from or writes to hardware devices using the following API functions.

```
x = sil_rew_mem(address);     // x=*address;
sil_wrw_mem(address, x+1);  // *address=x+1;
```

Since our cosimulator completely supports the API, application software does not have to be rewritten for cosimulation. For cosimulation, these APIs are translated to COM-based RPC calls to DM. For the final implementation, on the other hand, the APIs are translated to device driver software for the hardware.

3. Covalidation with RTOS Model and FPGA

As shown in the previous section, the hardware/software cosimulation environment which we developed in the past is very flexible. Specifically, it is useful for hardware debugging since software can serve as a fast, interactive testbench. For software debugging, however, the cosimulation environment is less efficient since HDL simulators are inevitably slow. In order to improve the execution speed, in this work, we have extended the environment to be able to connect to an FPGA.

We have developed two types of FPGA connection. One type uses RPC-based communication to make the most of communication flexibility of the environment (described in Section 2). Because of the flexibility of the RPC-based communi-

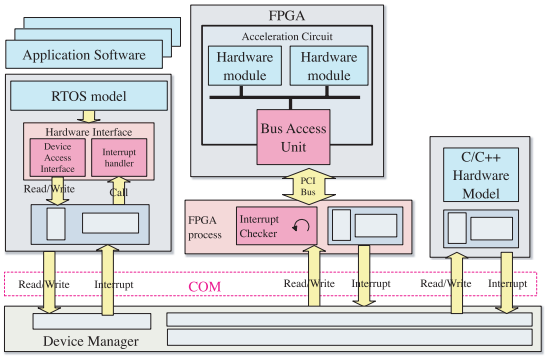

Fig. 2 Covalidation using an FPGA by RPC-based communications.

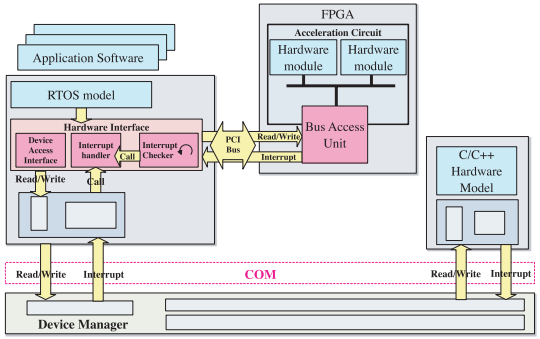

Fig. 3 Covalidation using an FPGA by direct communications.

cation, software and hardware do not have to be modified from HDL simulation to FPGA execution. Another type uses direct communication from the RTOS model to FPGA. The latter type enables faster covalidation than the former type because of less communication overhead. Application software do not have to be modified regardless of the type of FPGA connection because both communication types are implemented under the hardware interface API of the ITRON standard.

With the former type which uses RPC-based communication, software can communicate with FPGA in the same way as when connected with an HDL simulator, because FPGA is connected to DM through *FPGA process* provided by our covalidation environment (illustrated in **Fig. 2**). The FPGA process is an MS-Windows process which intermediates the communication between DM and the FPGA. In brief, the FPGA with the FPGA process is equivalent to the HDL simulator from a view point of software. The FPGA process has a COM object to communicates with DM, and through the DM, the software performs read/write to the FPGA. When the software needs to communicate with the hardware implemented on the FPGA, first the software sends a request to DM, next DM dispatches the request to the FPGA process, and then the FPGA process actually reads from or writes to the FPGA. For the sake of flexibility, this type of connection causes a large overhead caused by RPC communication which costs over 0.1 milliseconds per communication.

With the latter type, the FPGA communicates directly with the RTOS model. The use of hardware interface API on RTOS model (described in Section 2) translates to the direct use of the device driver for the FPGA, resulting in about two order of magnitude faster communication than using COM. This type of FPGA connection is useful especially for exhaustive validation with a large amount of test patterns. The direct connection is, however, less flexible than the COM-based connection. If connection is COM-based, the same device driver can be used independent of whether the hardware is executed on an FPGA or an HDL simulator, or even the hardware is a C++ model. Thus, hardware models can be easily replaced in a plug-and-play manner. If the FPGA is connected directly with the RTOS model, however, the device driver needs to be replaced as well. However, it should be stressed that, as described earlier, software programmers do not have to care about the type of FPGA connection at the application level.

For synchronization between hardware and software, our covalidation environment supports interrupts from hardware to software. In target systems, interrupts are performed immediately by interrupt signals at any time except when CPU is locked or the interrupts are masked (hence ignored).

In our covalidation environment, interrupts are handled differently between the HDL simulator and FPGA. When the hardware modules are simulated on an HDL simulator, hardware interrupt signals are checked every clock rise in the HDL simulator, and an RPC of DM is called, so that covalidation environment

can handle hardware interrupts immediately. When the hardware modules are executed on the FPGA, however, the FPGA process or the RTOS model cannot check interrupt signals from the FPGA at every clock timing of the hardware, because it cannot synchronize with the FPGA at clock timing level. Thus, the FPGA process or the RTOS model checks interrupts from hardware modules at a certain time interval (this function is denoted as *Interrupt Checker* in Fig. 2 and in **Fig. 3**). Although this implementation causes a delay of a few milliseconds, the functionality of interrupts can be correctly performed.

4. A Design Example

This section evaluates the effectiveness of our covalidation environment through a design of an MPEG4 decoder system. Experimental environments for this covalidation are shown in **Table 1**. Note that we used an FPGA introduced in Ref. 6), which is connected to and accessed from the host computer through PCI Bus (denoted in Fig. 2).

4.1 A Case Study: An MPEG4 Decoder

Figure 4 shows the design of the MPEG4 decoder system and allocation of tasks for the simulators. The MPEG4 decoder converts input data in the MPEG4 format into the YUV format, and writes the output data to a buffer of a video graphics array (VGA) device. The decoder system consists of a processor which executes application software with an RTOS, an acceleration circuit, and the VGA device.

The MPEG4 decoder application has four tasks; VLD, dequantization, IDCT and the others (denoted in Fig. 4). In the figure, the *Others* task covers any other tasks needed for MPEG4 decoder, e.g., decoder control, motion compensation and managing input and output data. The tasks executed on a processor are described in the C language. These tasks are compiled and linked together with the RTOS model to generate a binary code which is executable on the host computer. In order to shorten execution time of the MPEG4 decoder, some tasks are implemented as an acceleration hardware circuit. We used an MPEG4 file as input data which has 49 frames. Each frame size is 192 × 192 pixels, and the total file size is approximately 120 KB. For the input data, each task of VLD, dequantization and IDCT is executed 7,766 times.

Table 2 shows elapsed times for covalidation for two system architectures. The second column of Table 2 shows covalidation time of the system where the IDCT task is implemented as an acceleration circuit. The third column of Table 2 shows that of the system where both dequantization and IDCT tasks are implemented as an acceleration circuit. The acceleration circuit is written in HDL at the register-transfer level, and simulated on an HDL simulator or an FPGA. The VGA simulator is written in C++ and simulated in native execution.

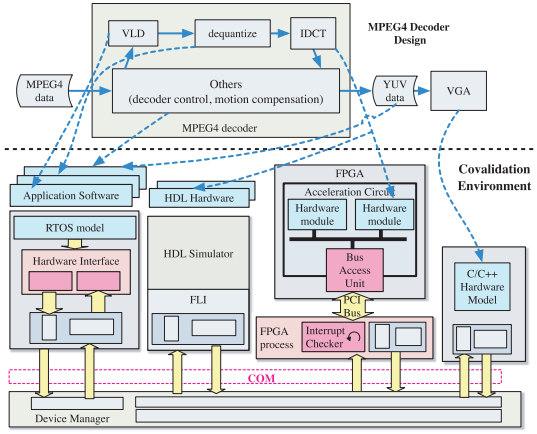

Fig. 4 The MPEG4 decoder system and covalidation environment.

Table 1 Experimental environments.

Host CPU	Intel Core 2 Duo on 2.66 GHz
Host main memory	2 GB
Host OS	MS-Windows XP Professional
HDL simulator	ModelSim SE 6.1c
FPGA [6]	Spartan3 on 15 MHz

Table 2 MPEG4 covalidation time.

Hardwares	IDCT	dequant+IDCT
with HDL simulator [4]	5.36×10^2 sec	7.53×10^2 sec
with FPGA (COM)	1.48×10^2 sec	1.49×10^2 sec
with FPGA (Direct)	9.97 sec	6.25 sec

A comparison of the second and fourth rows of Table 2 describes that our covalidation environment can perform software/hardware covalidation two orders of magnitude faster than the previous work [4]. Moreover, the second row shows that covalidation time is significantly affected by hardware complexity with the HDL simulator. On the other hand, hardware complexity has less effect on covalidation time when the hardware is emulated on the FPGA. These results shows effectiveness of our covalidation environment with RTOS model and FPGA.

5. Conclusion

This paper presented a software/hardware covalidation environment for embedded systems, which supports an RTOS model and an FPGA. The heart of our covalidation environment is the use of an RTOS model and an FPGA. The use of an RTOS model enables accurate system specification, efficient validation, and smooth implementation. In addition, the use of an FPGA enables performing covalidation in a short time. We also showed a case study in order to demonstrate the effectiveness of our covalidation environment.

Acknowledgments This work is in part supported by NEC Corporation and KAKENHI 19700040.

References

1) Gerstlauer, A., Yu, H. and Gajski, D.D.: RTOS Modeling for System Level Design, *DATE* (2003).
2) Hassan, M.A., et al.: RTK-Spec TRON: A Simulation Model of an ITRON Based RTOS Kernel in SystemC, *DATE* (2005).
3) Posadas, H., Villar, E. and Blasco, F.: Real-Time Operating System modeling in SystemC for HW/SW co-simulation, *DCIS* (2005).
4) Honda, S., et al.: RTOS-Centric Cosimulator for Embedded System Design, *IEICE Trans. Fundamentals*, Vol.E87, No.A (12), pp.3030–3035 (2004).
5) ITRON. http://ertl.jp/ITRON/home-e.html
6) Nakamura, Y., et al.: A fast hardware/software co-verification method for system-on-a-chip by using a C/C++ simulator and FPGA emulator with shared register communication, *DAC* (2004).

(Received December 25, 2007)
(Accepted February 22, 2008)
(Released August 27, 2008)

(Recommended by Associate Editor: *Tsuyoshi Isshiki*)