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

Yuichi Nakamura, Kouhei Hosokawa, Ichiro Kuroda
Media and Information Research Laboratories

Ko Yoshikawa
Computers Division

NEC Corporation
{yuichi@az, k-hosokawa@da, kuroda@ah, k-yoshikawa@bx}.jp.nec.com

Takeshi Yoshimura
Graduate School of Information, Production and Systems
Waseda University
t-yoshimura@waseda.jp

ABSTRACT

This paper describes a new hardware/software co-verification method for System-On–a-Chip, based on the integration of a C/C++ simulator and an inexpensive FPGA emulator. Communication between the simulator and emulator occurs via a flexible interface based on shared communication registers. This method enables easy debugging, rich portability, and high verification speed, at a low cost. We describe the application of this environment to the verification of three different complex commercial SoCs, supporting concurrent hardware and embedded software development. In these projects, our verification methodology was used to perform complete system verification at 0.2-1.1 MHz, while supporting full graphical interface functions such as "waveform" or "signal dump" viewers, and debugging functions such as "step" or "break".

Categories and Subject Descriptors

B.7.2 [**Hardware**]: INTEGRATED CIRCUTIS – *Design Aids*

General Terms

Design, Experimentation, Verification.

Keywords

Co-Verification, FPGA Emulation, C/C++ Simulator.

1. INTRODUCTION

System-On-a-Chip (SoC) refers to a system designed by integrating IP (Intellectual property) cores such as CPUs, DSPs, and various types of function. An important requirement in SoC design is that the embedded software for SoC processors should be designed in conjunction with the hardware design. Although SoC verification is difficult, because of the size and complexity of modern SoCs, accurate verification integrating hardware and software verification is indispensable since it enables full hardware and software development prior to fabrication.

The conventional approach to SoC verification is Register Transfer Level (RTL)[1][2] software simulation. This method is easy to set up since the verification designer needs only prepare the simulation models. This approach is, however, very slow. For example, an application program requiring 1 second to run on a real chip with 100MHz frequency, might take more than 10 days to simulate with the RTL simulator because the simulator can only be run at a speed of about 10 - 100 Hz. Because of this speed limitation, RTL simulation can barely be used for hardware verification, and cannot be used for embedded software verification.

To speed up system verification, various methods such as C/C++ level simulation and hardware emulation have been proposed. C/C++ [3][4] simulation runs much faster than RTL simulation, and provides an accuracy/efficiency tradeoff through various abstraction levels (Fig. 1). When the C/C++ abstraction is at the algorithm level, the simulation speed can reach about 1 MHz, but the simulation results have no cycle accuracy. On the other hand, in the case of a cycle-true simulation, the cycle accuracy is the same as in an RTL simulation, but the simulation speed is up to 1 KHz, *i.e.,* a little faster than RTL simulation.

Another method is based on a hardware emulator consisting of re-configurable devices, such as FPGAs. The speed of such emulators is typically up to 1 MHz, and they preserve cycle accuracy. In a world of nearly-MHz-speed verification with cycle accuracy, a hardware(HW) / software(SW) co-verification approach can be applied. The Ultra Sparc I verification project [5] is an example of the use of an emulator for HW/SW verification of production design. This project verified the Ultra Sparc I CPU before fabrication through emulation of the CPU for a test-bench based on the Solaris OS booting code. After this verification, both the Sparc I CPU and Solaris OS were cleaned up. Several CPU

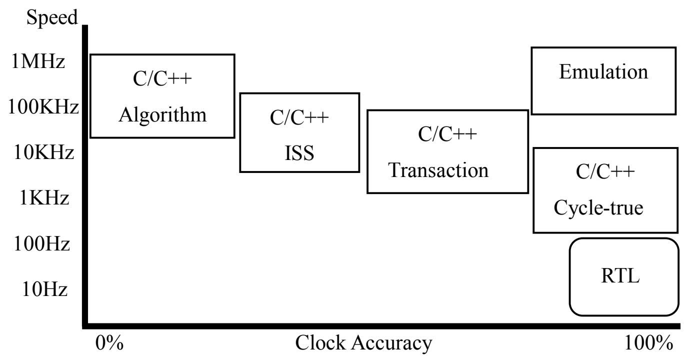


Fig. 1 Trade-off between simulation accuracy and speed

design projects have used an emulator to verify their CPU system, and archived effective results [6][7][8]. However, full chip emulation environments for large SoCs are expensive and complicated, and their debugging interface is poor compared to that of a software simulator.

In this paper, we propose a verification system that can run at 0.2 - 1.1 MHz and uses full synchronization between a C/C++ simulator and emulator. This method is especially useful for SoCs that include blocks executing software, such as CPUs and DSPs. For these SoCs, we have to co-verify and closely examine the user designed logic (UDL) and software on processors. In our verification system, the C/C++ simulator models the processors, and the FPGA emulator models the UDL. The bridge between the simulator and emulator is an array of shared registers, which either the simulator or the emulator can access and run simultaneously.

We describe conventional methods that combine a simulator and emulator in Section 2. In Section 3, we describe our shared-register architecture for communication between the C/C++ simulator and emulator. We discuss examples of application of our co-verification system to actual chip-design projects in Section 4, and conclude in Section 5.

2. CONVENTIONAL APPROACHES

To increase verification speed while maintaining clock accuracy, various hardware elements (*e.g.*, FPGAs) are required. If all blocks in an SoC are modeled using hardware emulation, the system cost, as well as the running and debugging cost, will become prohibitively expensive. Therefore, several conventional methods combine the use of a small, inexpensive emulator, and a simulator, to model and simulate SoCs.

One method is based on an abstraction approach [9]. This approach uses only a few FPGAs operating at about 1 MHz. A workstation controls the FPGA board. The FPGAs are compatible at the pin level for verification of the target SoC, but they have no clock-accurate registers for debugging or performance estimation. Since this method is similar to an accelerated C/C++ simulation, is cannot be applied for an SoC that includes pre-designed RTL components.

Another method is using a transaction-level C/C++ simulator to model the user application program [10]. In addition, an FPGA emulator models all UDL. The connection of two verification environments reduces the number of FPGAs needed and the system cost (Fig. 2). The verification speed is several hundred KHz for a 1.5 Mgate LSI. The main two features of this approach are transaction-level communication between the software modeling and hardware modeling, and use of a C/C++ simulation method to model user application programs such as an MPEG decoder. The communication method is implemented using FIFO (first in, first out) memory. It works by transferring the simulation vector from the simulator to the hardware emulator, which avoids the possibility that synchronizing the C/C++ simulator with the emulator will become a bottleneck restricting the verification speed. However, the lack of synchronization means that strict clock-level accuracy for the total SoC cannot be achieved. In addition, the user application model on a C/C++ transaction level simulator lacks the necessary system verification accuracy.

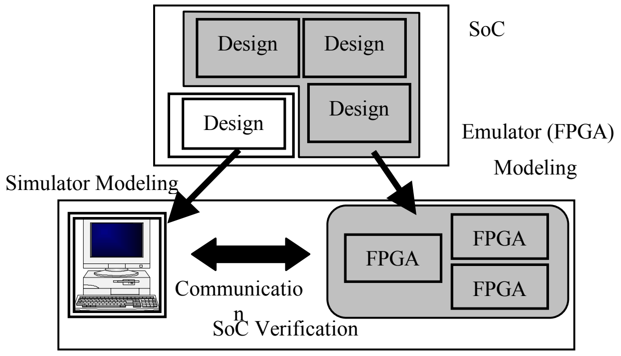


Fig. 2 Combined simulator and emulator approach

We propose a verification system that uses a C/C++ level simulator and FPGA-based emulator, as in [10]. However, a major difference of our proposed system is that it supports fully synchronized execution between the simulator and emulator. High verification accuracy is achieved since a shared register communication method is used instead of FIFO-based transaction-level communication.

3. SHARED REGISTER COMMUNICATION METHOD

3.1 Communication through Shared Registers

A key part of the proposed verification method is the connection between the simulator modeling parts and the emulator modeling parts. We use a connection bus consisting of an array of registers for this purpose. Although this register-based bus modeling is not the same as actual chip bus modeling, it is easy to set up and can be ported to various SoCs.

The connection architecture is implemented through a register array on the FPGA emulator. The set of registers is called the shared communication register (SCR). The SCR as well as the emulator design can be accessed or read/written from the software simulator. The control circuits are implemented around the PCI bus on the simulation host PC, and in the design of one FPGA in the emulator. The SCR has two roles. One is to support communication for the running verification, the other is for debugging. The SCR consists of about 2000 FFs and an associated controller. The architecture that enables remote signal-observation and register-writing through the PCI bus is shown in Fig. 3.

During the verification, the SCR performs the following operations in each SoC clock cycle.

--From software on the PC:
1) Read the signals in FPGA circuits
2) Write the registers in FPGA circuits
3) Read/Write memories in FPGA
4) Order the instructions to be designed in the FPGA
5) Generate the clock for the FPGA

--From the FPGA design:
1) Send interrupt signals to software on the PC
2) Read and Write memory space through the PC software

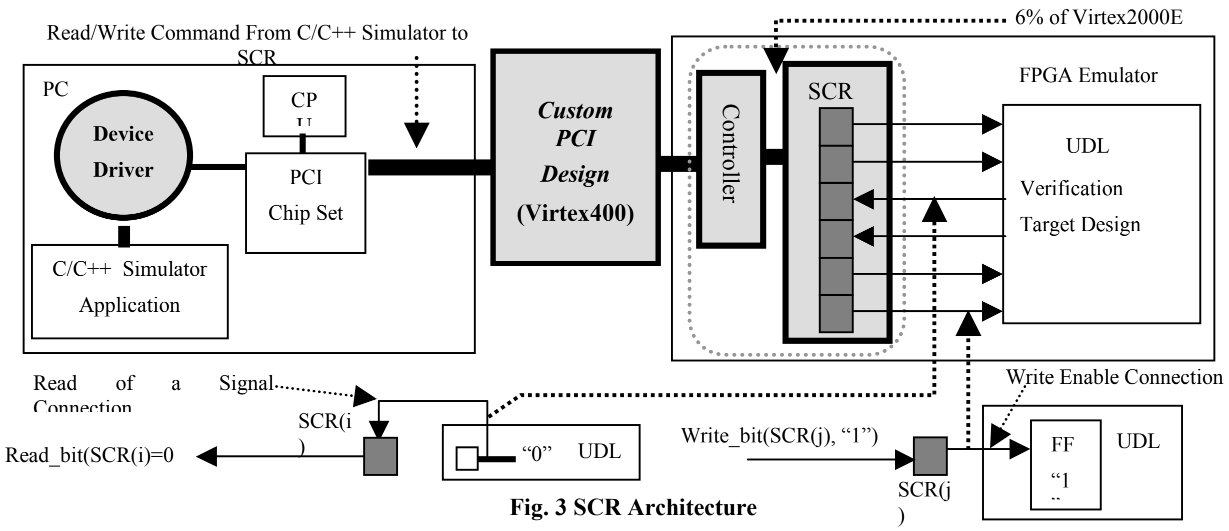


Fig. 3 SCR Architecture

3.2 SCR Architecture

The SCR architecture consists of 1) a device driver program, 2) a PCI board, and 3) interface circuits in the FPGA for emulation. Any FPGA board can be used provided it has an external interface. In the proposed verification system, we have used a simple and low cost custom FPGA board which had no devices except for a few FPGAs. Figure 3 shows the overall SCR architecture.

The device driver software communicates with the PCI board via a PCI protocol [11]. The device driver on a Windows/Linux PC has access libraries with respect to the SCR. This method partitions the SoC verification target into two components. Any software that includes a C/C++ simulator, an RTL simulator, and a simple behavior or interface description program can access the SCR of an FPGA by using an address and data. A basic operation is a 1 bit read/write; for example, *read_bit(16)* to read the data of the 16th SCR, or *write_bit(1899, "1")* to write the value "1" in the 1899th SCR. To enable easy implementation of the above functions, we prepared easy access libraries described in C/C++ language. C/C++ libraries are easy to build in the C/C++ simulator to specify the variables of a function. For example, *write_byte(010010, "0b00000111")* implies that the address starting from "010010" of the SCR is written with 1 byte data of "0b00000111". The following functions are provided as part of access libraries in C/C++. The byte and memory addresses can be assigned from among the 2048 SCRs.

1) read_bit(bit_address);
2) write_bit(bit_address, bit);
3) read_byte(byte_address,)
4) write_byte(byte_address, byte_data)
5) memory_read(memory_address)
6) memory_write(memory_address, address_pointer in C/C++ program)

Although general PCI-based commercial verification boards [12][13] have been developed by using a PLX chipset[14] based on a FIFO-based system, we developed our custom PCI chip and boards to archive simultaneously verification between the simulation and emulation through the SCR architecture.

Our custom-made PCI board was developed according to specifications based on the PCI standard [11] using a Xilinx Virtex400 [15] instead of a PLX chipset. The board can communicate with the host PC's memory and register space via the PCI chipset and PCI address space directly by our device drivers.

The address and data inputs from a software program can access the SCR of an FPGA according to the request. The interface circuit has 2048 FFs and 200 FFs that make up the SCR and the controller of the SCR. The data and address from a PC via a PCI card is decoded in these parts.

Furthermore, this controller has a clock-enable system for verification target design. When this controller communicates with a PCI card, it can disable the clock of the verification target design to maintain a consistent absolute time between the C/C++ simulation and the FPGA design. Since this controller and SCR have only about 2248 FFs, they take up only 6% of the Xilinx Virtex2000E FPGA[15], which has 38000 FFs.

3.3 Access Operation to the SCR

Although the SCR has many verification applications, we describe just three basic operations showing how the SCR can be used through C/C++ simulation.

The first operation is observation of the FPGA's internal signals. The general method for observing FPGA internal signals is a JTAG-based method [16]. This JTAG-based method is very useful for observing all signals in an FPGA. However, it is difficult to observe several signals in every clock cycle since it takes about 1 sec to download data from the FPGA to the PC during one clock period. For the simulator to observe the signals during 100 cycle periods, more than 1 minute would be needed. In addition, correspondence between the RTL source and the downloaded data is very difficult. To observe signals inside the FPGA through the SCR method, the signals must be connected to the SCR. Connecting between SCR and FPGA signals is easily done by modifying the RTL source of the FPGA. If the signals are connected, the simulator can observe these signals since any signals connected to the SCR are observable. Since the number of SCRs is limited, the SCR method is useful for tracing selected signals in every time period. Connecting SCR and FPGA signals is easily done by modifying the RTL source of the FPGA. If the signals are connected, the simulator can observe these signals since any signals connected to the SCR are observable. Since the number of SCRs is limited, the SCR method is useful for tracing selected signals in every time period.

The second operation is a clock driven from software to the FPGA, which is used to run the software and FPGA synchronously. One SCR is connected in advance to the clock inputs of all FFs in the FPGA.Toggling one SCR from software allows all FFs in the FPGA to be run by the software clock; this is done by alternatively inputting one to "0" and "1". This method is very effective for debugging. If we want to issue a design breakpoint in the FPGA, we need only stop sending the clock signals (Fig. 4). When stopping the provision of a clock signal, we can observe the signals in the verification target design via the SCR. "Step" running, which is useful for debugging is also easy to archive. The C/C++ simulation program can provide a clock of about 1.5 MHz (in the maximum case) to the FPGA board and FPGA inside clock signal provided by the simulator since it takes about 270 ns to write and 260 ns to read the rising edge of the clock and the same time to write and read the falling edge. Since the falling operation of the clock signal from "1" to "0" can be done by hardware in FPGA, it is not necessary to fall the clock by simulator and, the maximum speed of clock provision is about 1.5 MHz.

The third operation is memory read/write. When we write to memory inside of a block that is mapped to the FPGA, we specify the address, and send the write data to some SCR connected to the memory address decoder and memory.

Through a combination of these three basic operations, various useful debugging applications can be developed. For example, a waveform viewer for signals internal to the FPGA and a memory system has been completed. The waveform viewer does not use the FPGA dump file, instead simultaneously reading reads FPGA signals every clock through the SCR between the C/C++ simulator clock and FPGA clock. A memory break occurs when specified memory addresses are accessed by a read or write operation. An SCR-based system can observe the specified memory and sense the read/write operation. To stop the clock provision when such an operation is sensed, the total system is subject to the "Break" condition. The details of such a debugging method are described in section 4.

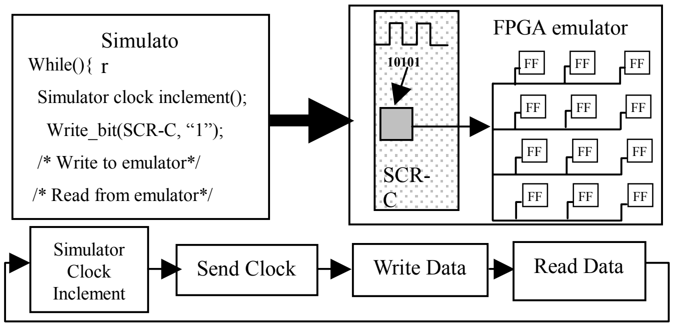


Fig. 4 Clock Provision from Simulation

4. VERIFICATION EXAMPLES

In this section, we describe how we used the proposed verification method for several commercial chip designs.

4.1 Stream Processor in Digital TV STB Chipset

This SoC is used for stream processing in a digital TV set-top box (STB) before MPEG2 decoding. The stream process consists of transforming the signals from the satellite to the MPEG2 decoding bus. The size of the whole chip is about 1.0 Mgates and the speed is 83 MHz. Its components are a RISC CPU, a host interface, a stream input block and a stream processor and DMA (Direct Memory Access). The CPU issues DMA and stream-input access instructions to the stream processor blocks.

The CPU, which is a hard-macro IP block, does not have to be verified. Rather, the verification targets are the CPU software and the behavior of the stream processor block, DMA, and overall system.

A C/C++ ISS (Instruction Set Simulator) of a processor provides high speed and a useful debugging graphical user interface (GUI). It does not have the accuracy for inside processor design and circuit, but have the perfect accuracy for the communication with outside blocks. The simulation speed is several hundred KHz. In this STB chip, the speed of ISS for a RISC CPU is about 400 KHz. The CPU developer guarantees the accuracy between a real chip and the ISS. Various debugging tools of the CPU software (such as source scroll viewer, waveform viewer, and break and step running) are implemented in the CPU ISS. However, an ISS is not suitable for developing embedded SoC software because the CPU has to communicate with other circuits in the SoC.

Since a single ISS is insufficient to verify the embedded software, we applied C/C++ simulation or RTL simulation combined with ISS and other blocks for embedded software development. In the case of full C/C++ simulation, although the speed was about 3.3 KHz, the accuracy was less than that of RTL simulation because of the high abstraction level of blocks other than the CPU. In other words, RTL simulation is more accurate, but at about 96 Hz it is too slow.

The combination of C/C++ CPU ISS and FPGA-emulation-modeled RTL blocks – DMA, stream input and stream processor, and the communication bridge of the SCR architecture – overcomes these problems and provides high verification speed and accuracy.

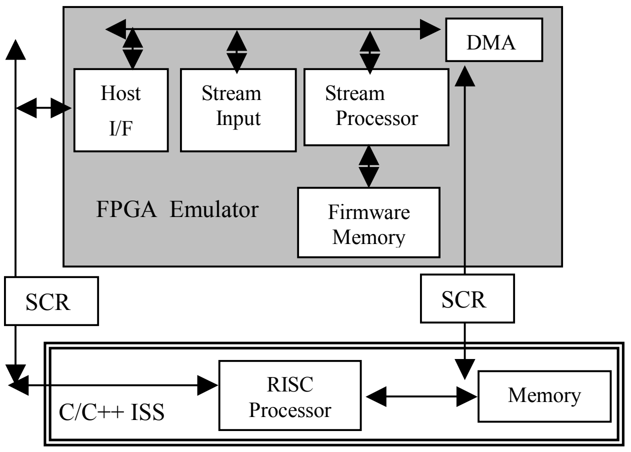


Fig 5. STB Block Diagram for Verification

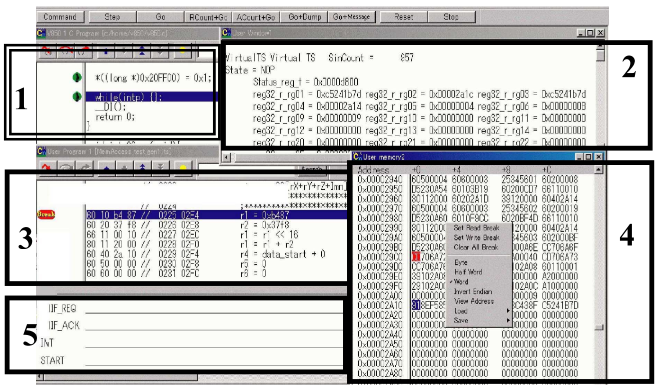
We partitioned this system into two parts (Fig. 5). One part, the CPU (0.51 Mgates) was modeled by the C/C++ ISS. The other parts (0.49 Mgates) were modeled by an emulator, which is modeled by three Virtex2000E [16] FPGAs. The firmware memory was modeled by the FPGA block memory. To model these flows, the SCR relays order sets from the CPU to other components, the signals from other components to the CPU, and the memory address/data. The emulator clock is provided from the ISS. 90 SCRs are used to bridge the two models every clock cycle.

One SCR is used for the clock provided from the ISS to the emulator, about 32 SCRs are used by order sets from the ISS, and 57 SCRs are used by debugging information in the emulation signals. When 90 SCRs communicate every clock cycle, the speed of SCR communication is about 213 KHz. Since the ISS speed is faster than the SCR communication, the bottleneck determining the total system verification speed is the 213 KHz SCR communication speed.

The debugging of this verification system is illustrated in Fig. 6.

This debugging consists of five windows:

1) CPU C/C++ source scroll
2) Registers in the emulator dump
3) Assembler code of the stream processor
4) Memory in the emulator dump
5) Waveform viewer of the signals in the emulator



```
    *((long *)0x20FF00) = 0x1;
    while(intp) {};
    __DI();
    return 0;
}
```

```
VirtualTS Virtual TS   SimCount =      957
State = NOP
     Status_reg_t = 0x0000d800
     reg32_r_rg01 = 0xc5241b7d reg32_r_rg02 = 0x00002a1c reg32_r_rg03 = 0xc5241b7d
     reg32_r_rg04 = 0x00002a14 reg32_r_rg05 = 0x00000004 reg32_r_rg06 = 0x00000008
     reg32_r_rg09 = 0x00000009 reg32_r_rg10 = 0x00000000 reg32_r_rg11 = 0x00000000
     reg32_r_rg12 = 0x00000000 reg32_r_rg13 = 0x00000000 reg32_r_rg14 = 0x00000000
     reg32_r_rg20 = 0x00000000 reg32_r_rg21 = 0x00000000 reg32_r_rg22 = 0x00000000
```

```
                                      rX+rY+rZ+Imm]
                                      ************
60 10 b4 87 //  0225 02E4             r1 = 0xb487
60 20 37 f8 //  0226 02E8             r2 = 0x37f8
66 11 00 10 //  0227 02EC             r1 = r1 << 16
80 11 20 00 //  0228 02F0             r1 = r1 + r2
60 40 2a 10 //  0229 02F4             r4 = data_start + 0
60 50 00 00 //  0230 02F8             r5 = 0
60 60 00 00 //  0231 02FC             r6 = 0
```

```
Address     +0        +4        +8        +C
0x00002940  60500004  60600003  25345601  60200008
0x00002950  D5230A54  60103B19  60200CD7  66110010
0x00002960  80112000  60202A1D  39120000  60402A14
0x00002970  60500004  60600003  25345602  60200019
0x00002980  D5230A60  6010F9CC  6020BF4D  66110010
0x00002990  80112000            120000    60402A14
0x000029A0  60500004            345603    602000BF
0x000029B0  D5230A6             000A6E    CC706A6F
0x000029C0  C1706A7             A00040    CD706A73
0x000029D0  CC706A7             102A08    60110001
0x000029E0  39102A0             000000    A2000000
0x000029F0  29102A0             102A0C    A1000000
0x00002A00  0000000             000009    00000000
0x00002A10  918EF58             3C438F    C5241B7D
0x00002A20  0000000             000000    00000000
0x00002A30  0000000             000000    00000000
0x00002A40  00000000  00000000  00000000  00000000
0x00002A50  00000000  00000000  00000000  00000000
0x00002A60  00000000  00000000  00000000  00000000
0x00002A70  00000000  00000000  00000000  00000000
0x00002A80  00000000  00000000  00000000  00000000
```

```
IIF_REQ
IIF_ACK
INT
START
```

Fig. 6 GUI of debugging

Although this debugging figure is similar to a single ISS GUI, windows 2 to 5 express the emulator status. All windows from 1 to 5 are synchronized and refreshed every clock cycle, since the ISS provides the emulation clock. Because of this simultaneity, windows 2 and 4 can be specified with a breaking trigger function. When the trigger is set in window 2, where the specified register changes in value, the ISS detects the breaking timing and can break the entire system by stopping the clock provision. In windows 3 and 4, similar breaking points can be specified. This verification lasted for 3 weeks, and no hardware or software bugs were detected in the fabricated chip.

We compared the proposed combined method with the full chip emulator. The results from the verification by full-chip emulator showed that the system frequency was about 0.8 MHz, it used more than 100 FPGAs, and that the dump-based debugging operation only applied.

4.2 Application Chip for 3G Mobile Communication

This chip includes a DSP, CPU, DMA, and graphics accelerator (Fig. 7), and its size is about 3.2 Mgates. There were two verification goals: verification of the graphics accelerator, and providing the development platform for the CPU operating system (OS) before chip fabrication. A C/C++ ISS simulator modeled the DSP and CPU, and a transaction-level C/C++ simulator modeled the memory. The RTL of the DMA and that of the graphics accelerator were implemented through an emulator using one Virtex3200E and two Virtex2000E [16] FPGAs. The role of the SCR was 1) clock provision, 2) sending the orders from the DSP and CPU, 3) providing a memory read/write interface with the graphics accelerator, and 4) sending signal information from the emulator blocks. Since the memory block was in the C/C++ simulator, we could easily confirm the memory contents (i.e., the graphical data). The debugging windows and memory contents viewer are shown in Fig. 8. In the figure, "PICT" indicates the resultant image after graphics acceleration. This image is drawn at a rate of one dot per clock cycle. The other windows shown are the register dump in the emulator, source scroll in the C/C++ ISS, and the waveform in the emulator (the same as in Fig. 6).

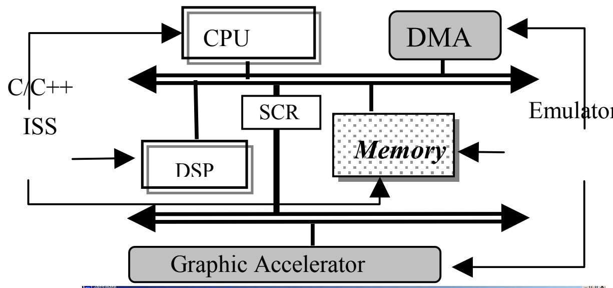


Fig. 7 Block Diagram of 3G Mobile Chip

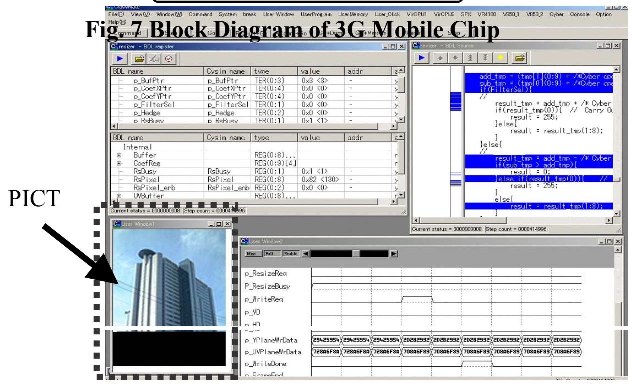


Fig. 8 GUI of C/C++ ISS

The speed from only C/C++ simulation, in which all modules are modeled by cycle true C/C++, was 2.1KHz. However, the proposed verification environment can run at about 93 KHz since about 320 SCRs communicate with the ISS and emulator every clock cycle. When the memory module is moved to the emulator, the speed rises to 199 KHz, but the image in the memory cannot be updated every clock cycle. However, in the case of memory in the emulator, we can easily view the image in memory when the system is breaking. In early verification stages, the memory block in simulation is used, and then emulation-based memory is used in the final stage. Our verification results were 1) OS development was finished before fabrication, and 2) one fatal bug in the DMA was found before fabrication. After chip fabrication, the evaluation board with an actual chip was working within one day.

4.3 ISS Alternative for Custom Processor

A typical commercial CPU is released with a fully accurate ISS provided by the CPU developer. Such an accurate ISS generally will not exist for a custom processor, though, because ISS development takes a long time and is expensive. In general, an ISS with insufficient accuracy is used for custom processor. As a result, it cases many bugs in the firmware developed by insufficient accurate ISS.

An SCR-based verification system can make it possible to provide a fully accurate ISS model to a firmware developer. The key points of accuracy are 1) the processor model describe by the RTL is mapped to the emulator, 2) a C/C++ simulator is used only in the debugging interface, and 3) the clock of the processor modeled in the emulator is provide by the C/C++ simulator (Fig. 9). After clock provision, the C/C++ simulator obtains debugging signals from the emulation design and displays these in the graphic interface windows. After obtaining the debugging data, the C/C++ simulator provides the next clock for emulation.

We developed an ISS for a custom processor (205 Kgates) by simulator for GUI and 1 Virtex2000E FPGA emulator. This custom processor has an original ISS with 270KHz, which has no cycle accuracy both inside and outside processor. The system frequency was 1.1 MHz, since the C/C++ simulator provided only clock signals and received only program counter data (16 bit) from the processor on the emulator. The source scroll window was integrated at the simulator with program counter data from the emulator and prepared firmware codes. This environment cannot only be used for logic verification, but also can be used for performance estimation (for example, cache hit rate analysis) since the processor model is based on RTL; however, all software ISSs could run at only 100-500 KHz, and so could not verify the performance estimation.

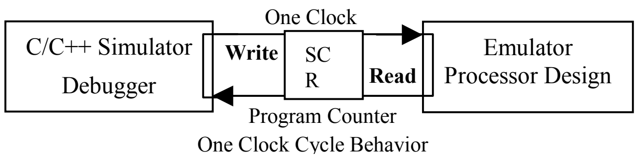


Fig. 9 Application as an alternative to a processor ISS

The result of this case study demonstrates that the proposed co-verification system can be used as an efficient ISS for any processor, avoiding the manual effort to develop an ISS, if the RTL for the processor is available.

Table I summarizes the three verification examples

Table I Summary of verification Examples

	C/C++ Simulator	FPGA Emulator	FPGA Size	System Speed	Conventional Method
STB (1MG)	-CPU -Memory -Clock -Debugging GUI	-Steam Processor -DMA -Firm Ware Memory	V2000 x3	213KHz	-Emulation 800KHz -C/C++ 3.3KHz -RTL 96Hz
Mobile (3.2MG)	-CPU,DSP -(Memory) -Clock -Debugging GUI	-DMA -Graphic -(Memory)	V3200 x1 V2000 x2	-93KHz (Simulator Memory) -199KHz (Emulator Memory)	-C/C++ Simulator 2.1KHz
CPU (205KG)	-Clock -Debugging GUI	-Processor Design	V2000 x1	1.1MHz	-Inaccurate Custom ISS 270KHz

5. Conclusions

In this paper, we have described an SoC verification method that combines a C/C++ simulator and FPGA emulation. The key idea is the use of shared communication registers (SCRs) at the boundary between the software and the FPGA. We show that the SCR method incorporates simultaneous connection between simulator and emulator. Our experimental results from design examples showed the effectiveness of the proposed method. In our experiments, the proposed verification methodology was used to complete system verification for 0.2-3.2Mgate class SoC, at 0.2–1.1 MHz, using C/C++ simulation, emulation, and SCRs. Furthermore, the proposed verification environment provides a rich debugger interface, displaying items such as the wave form or register dump, through synchronized running of the emulation and simulation via the SCRs.

The proposed methodology is thus a fast and useful means of verification for a wide range of SoCs.

6. ACKNOWLEGEMENTS

We are grateful to Dr. Anand Raghunathan and Mr. Hidefumi Kurokawa for helpful suggestions.

7. REFERENCES

[1] R. Lipsett, et al, "VHDL: Hardware Description and Design", Kluwer Academic Publishers, 1989.
[2] D. E. Thomas and P. Moorby, "The Verilog Hardware Description Language", Kluwer Academic Publishers, 1991.
[3] D. D. Gajski, et al, "SPEC C: Specification Language and Methodology", Kluwer Academic Publishers, 2000.
[4] System C, "http://www.systemc.org"
[5] J. Gateley, M. Blatt, D. Chen, S. Cooke, P. Desai, M. Doreswamy, M. Elgood, G. Feierbach, T. Goldsbury, and D. Greenley, " UltarSparc-I Emulation", Proceedings of ACM/IEEE Design Automation Conference (DAC)95, pp.13-18, 1995.
[6] F. Casaubieilh, A. McIsaac, M. Benjamin, M. Bartley, F. Pogodalla, F. Rocheteau, M. Belhadj, J. Eggleton, G. Mas, G. Barrett, and C. Berthet, "Functional verification methodology of Chameleon processor", Proceedings of ACM/IEEE Design Automation Conference (DAC)96, pp.421-426, 1996.
[7] G. Ganapathy, R. Narayan, G. Jorden, D. Fernandez, M. Wang, and J. Nishimura, "Hardware emulation for functional verification of K5", Proceedings of ACM/IEEE Design Automation Conference (DAC)96, pp.315-318, 1996.
[8] J. Monaco, D. Holloway, and R. Raina, "Functional verification methodology for the PowerPC 604 microprocessor", Proceedings of ACM/IEEE Design Automation Conference (DAC)96, pp.319-324, 1996.
[9] N. Kim, H. Choi, S. Lee, S. Lee, I-C. Park, C-M. Kyung, "Virtual chip: making functional models work on real target systems", Proceedings of ACM/IEEE Design Automation Conference (DAC 98), pp.170-173, 1998.
[10] M. Kudlugi, S. Hassoun, C. Selvidge, D. Pryor, "A Transaction-Based Unified Simulation/Emulation Architecture for Functional Verification", Proceedings of ACM/IEEE Design Automation Conference (DAC 01), pp. 623-628, 2001
[11] "PCI Local Bus Specification, Revision 2.1" PCISig, 1995
[12] Zebu, "http://www.eve-team.com"
[13] Alpha data "http://www.alpha-data.com/"
[14] PLX, "http://www.plxtech.com"
[15] Xilinx, "http://www.xilinx.com"
[16] Xilinx, Application Note, APP151 "http://www.xilinx.com/bvdocs/appnotes/xapp151.pdf"