2003년도 대한전자공학회 하계종합학술대회 논문집 제26권 제1호

In-Circuit 시스템 온 칩 검증 방법과 디버깅 환경

이재곤, 기안도*, 경종민

한국과학기술원 전기 전산학과 전기 및 전자 공학 전공
* 다이나릿 시스템

In-Circuit System-on-Chip Verification and Debugging Environment

Jae-Gon Lee, Ando Ki* and Chong-Min Kyung

Department of Electrical Engineering and Computer Science, KAIST
* Dynalith Systems Co., Ltd.
E-mail: jglee@vslab.kaist.ac.kr

Abstract

This paper presents in-circuit system-on-chip verification and debugging environment. To maximize the emulation speed, the software part is compiled natively for the host computer and the hardware part is mapped into FPGA. The two parts communicate with each other in transaction level. The operation of the hardware part and the software part is recorded independently during the emulation, and after the emulation is over, they are merged in a waveform to give user a unified view that covers both hardware and software.

I. Introduction

Advances in the semiconductor processing technology have brought more and more transistors into a single device than ever. Nowadays microprocessors, digital signal processors, memories and custom logics are integrated in a single chip to form a system-on-chip.

Verification of such a system-on-chip is quite different from that of system-on-board. Since the former embraces the entire blocks and functionality in a single chip, each functional block cannot be tested before fabrication. The component designers may have tested their designs with simulation, but that does not guarantee it will work as expected in silicon.

This kind of problem is most significant for the interface between the software and hardware. In general, the software design team and the hardware design team are separated, so there is a great possibility there to be misunderstandings between them, which might lead to re-fabrication [1, 2].

To address this problem, co-design/co-verification method is used. In one method, entire hardware model including the processor is described in HDL and simulated with the software model compiled to the target processor. With this method, the users can verify their software/hardware designs in a cycle-accurate manner. But this method wastes lots of simulation time in simulating the core processor, which already have been proved.

The second method, the software-based co-verification, removes this redundancy with Instruction Set Simulator (ISS) and Bus Functional Model (BFM). ISS models the instruction-accurate behavior of the processor, and the BFM generates cycle-accurate pin signals to access hardware models [3,4,5,6,7,8,9,10,11]. Although, this method runs faster than the previous approach, it still lacks the performance required for in-circuit verification. As a result, input stimulus needs to be provided as a test vector to verify design-under-test. But, in most cases, test vectors cannot cover all the possible situations. It means the designer tapes out his/her design with imperfect test environment.

To this problem, this paper shows how to verify system-on-chip designs with real target environment and how to observe their behavior. To gain the maximum simulation performance and to allow verification in the early design stage, the software design is compiled natively for the host computer, and the hardware design is mapped into FPGAs. We also propose a unified HW/SW debugging environment for the proposed platform.

Section II explains the verification platform and section III deals with the debugging environment for the given platform. A case study is given in section IV followed by conclusion in section V.

II. In-circuit system-on-chip verification

Fig. 1 shows the block diagram of the proposed in-circuit system-on-chip verification platform. The design is composed of two parts: software and hardware. In order to get the maximum simulation speed, software design is compiled natively for the host processor. The Application Program Interface (API) and the transactor make it possible for the software and hardware to communicate with each other. The hardware design (custom logics with IP) is synthesized and mapped into FPGA to connect the verification system to the target system.

Fig. 2 shows the proposed system-on-chip design flow.

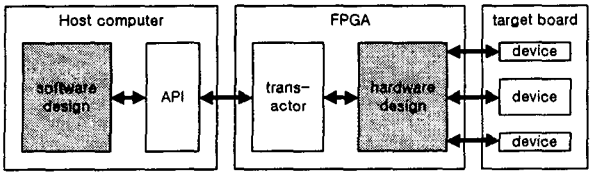


Fig. 1. Block diagram of the system-on-chip verification platform

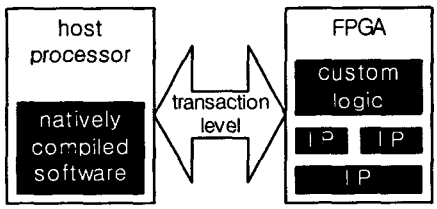
When the functionality of the software and hardware design is verified in step 1, the software design is compiled for the target processor, and the hardware design is fabricated as in step 2 of fig. 2.

III. Debugging

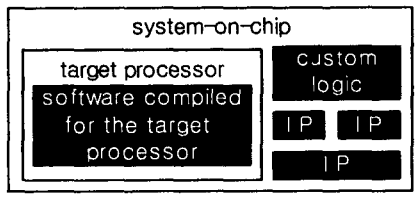
System-on-chip design is composed of two heterogeneous parts, so is the debugging environment. The hardware debugger should be able to observe the cycle-accurate behavior of the design with no overhead, while the software debugger should be able to record the value change history of variables in user's program. The former is named as Pin Signal Analyzer (PSA) and the latter is named as Software Variable Analyzer (SVA). The following subsections we will cover PSA and SVA. The last subsection deals with the synchronization problem of the results of the two analyzers to generate unified debugging information.

A. Pin Signal Analyzer

PSA samples hardware signal values with pre-defined sampling frequency and stores them in the dedicated external memory. For that purpose, PSA block is added along with the user's hardware design when it is synthesized. After the emulation is over, the stored data is translated to a waveform containing cycle-accurate behavior of the design.



(a) Step 1



(b) Step 2

Fig. 2. Design flow

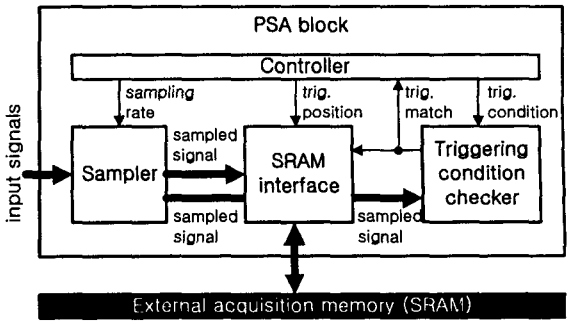


Fig. 3. PSA block diagram

The PSA block is composed of the following five sub-blocks. (fig. 3)

- **External acquisition memory** stores the sampled data.
- **Controller** contains control registers to control the rest of the PSA sub-blocks.
- **Sampler** samples the pin signals at a given sampling frequency and sends the result to the SRAM interface sub-block and the triggering condition checker.
- **SRAM interface** stores and retrieves the sampled data.
- **Triggering condition checker** checks whether triggering condition is satisfied.

The external acquisition memory is composed of six 32bit x 256k SRAMs, which can hold 256k samples of 192 signals, 128k samples of 384 signals, or 512k samples of 96 signals according to the operating modes.

Controller holds variable configurations of PSA. This allows flexible management of the PSA block. Control registers include triggering condition, triggering position, sampling rate, external triggering condition enable, etc. Recompilation is needed only when the target pin list is changed.

To overcome the limited size of the acquisition memory, triggering condition can be set to store signal values around some moment. There are thirteen kinds of triggering conditions available, analogous to logic analyzers.

The structure of the triggering condition checker is shown in fig. 4. It is composed of pattern checker blocks, a timer block, and a sequencer block. Each pattern checker block compares the input pin signal value with the pre-defined reference pattern or edge. When the input signal matches the given pattern/edge, the 'pattern match' signal is activated. (Pattern match 0, 1, 2 in fig. 4) The timer block has a counter and compares the counter value with the reference counter value. This counter can be activated by one of the pattern match signal to calcu-

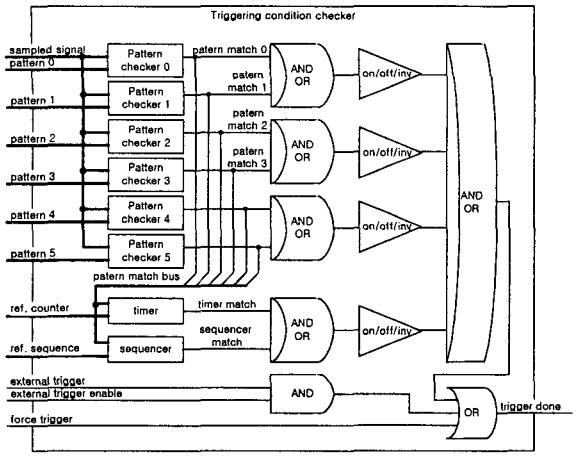


Fig. 4. Structure of the triggering condition checker

late the time elapsed after the given pattern/edge occurs. The sequencer has a state machine that records the sequence of the previous patterns/edges. When the sequence matches the pre-defined sequence, 'sequencer match' signal is activated. According to the configuration register values, one of the six pattern match signals, timer match signal, and sequencer match signal, or the combination of them is selected as a "trigger done" output signal. In addition, external trigger signal and force trigger signal can also be used to trigger the PSA logic, which is shown at the bottom of the fig. 4.

When the emulation starts, the sampler begins to store the pin signal values to the acquisition memory. They are stored from the address 0 of the acquisition memory, and the address is increased with every sampled data. When the address reaches the last address of the memory, it is reset to 0 to start writing from the first entry of the memory. This continues until the triggering condition is met. When the triggering condition is met, the SRAM interface stops after predefined time according to the triggering position. For example, when middle triggering position is set, SRAM interface stops after writing 128k sample data This leaves 128k sample data prior to the triggering point and 128k sample data after that in the

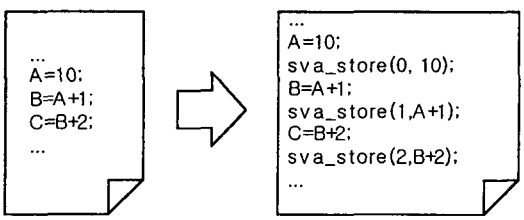


Fig. 5. Instrumentation code insertion. Sva_store(ID, new_value) gets the variable ID and the new value for the variable and stores them in the dedicated memory area with time stamp.

acquisition memory.

After the emulation is completed, the content of the acquisition memory with additional header information is translated to waveform file. The header information includes the last write address, triggering point, operating modes, sampling rate, etc. The waveform file is readily visible with popular waveform viewer such as GNU gtkwave or Veritools' Undertow VIII.

B. *Software Variable Analyzer*

As mentioned before, hardware software co-verification is one of the most important challenges in the system-on-chip design. So we need to verify the operation of the software as well as the hardware. SVA is to observe the software behavior of system-on-chip design.

SVA stores value change history of variables in the software part of the design. During emulation, the data is stored in the main memory of the host computer, and after emulation is completed, it is merged with PSA data to form a unified waveform. We'll describe the merging of the PSA and SVA data in the next subsection.

To record the value change history of variables, instrumentation code is inserted before emulation as in Fig. 5. The user's software model in C is preprocessed and the instrumentation code is inserted automatically. It is inserted where the selected variable is assigned to a new value. The insertion of the instrumentation code may slow down the simulation speed of the software part of the design. But, as the communication between the software and hardware part is done in transaction-level, overheads introduced by the instrumentation code do not matter.

The instrumentation code stores the variable ID, time stamp and the new value for the variable in the dedicated area memory in the host computer. As in the PSA, the instrumentation code continuously overwrites the previous data until triggering condition is met.

Each variable can have a triggering condition, which disables the instrumentation code to preserve the value change history at that moment. Triggering condition is checked whenever the instrumentation code is called. There are six kinds of triggering conditions for SVA.

When the triggering condition for SVA is met, SVA sends command that forces PSA to trigger. On the contrary, when PSA meets the triggering condition, PSA interrupts SVA to trigger.

C. *Synchronization between PSA and SVA*

Since the chip model is realized in part with software and the others with hardware, we need to observe their concurrent behavior in a unified time scale. In the previous two subsections, we discussed how to store the hardware and the software behavior of the chip model. In this subsection, we show how to merge the two heterogeneous data.

Each of PSA and SVA has a time stamp, i.e., synchronization counter for PSA and CPU timer value for SVA. To resolve the relation between two time stamps, each

dump data contains a header with a unit of time. In other words, the PSA header contains the sampling rate and the SVA header includes the CPU clock frequency. As we know the update frequency of each counter, we need to know the initial value for the two counters. For that purpose, we read the two counter values at some time t_0. Then, the relation between the two counter values can be correlated as follows;

$$\frac{timer_{CPU}(t) - timer_{CPU}(t_0)}{freq_{CPU}} = \frac{counter_{PSA}(t) - counter_{PSA}(t_0)}{freq_{PSA}}$$

t: arbitrary time
t_0: initial time
$timer_{CPU}(t)$: CPU timer value at time t
$timer_{CPU}(t_0)$: CPU timer value at time t_0
$freq_{CPU}$: CPU clock speed in MHz
$counter_{PSA}(t)$: PSA Synchronization counter value at time t
$counter_{PSA}(t_0)$: PSA Synchronization counter value at time t_0
$freq_{PSA}$: PSA sampling frequency

We can rearrange the formula for the $counter_{CPU}(t)$ as follows;

$$timer_{CPU}(t) = \frac{freq_{CPU}}{freq_{PSA}} \times (counter_{PSA}(t) - counter_{PSA}(t_0)) + timer_{CPU}(t_0)$$

Thus, we can calculate the synchronization counter value corresponding to a given CPU timer value. Thus, each time stamp in the SVA data is transformed to a synchronization counter value, which makes it possible to generate a unified waveform covering both hardware and software.

IV. Experimental result

We adapted the system-on-chip verification methodology to MPEG-2 decoder example. The chip reads the mpeg-2 encoded image data and decodes it and generates CCIR-601 compatible video output signal. The target board contains flash memory that contains the encoded data and NTSC encoder, which gets the CCIR-601 video data from the target chip and transforms it to NTSC video signal. The hardware part contains interface logic for flash memory and NTSC encoder and the software part deals with the mpeg-2 decoding algorithm. Fig. 6 shows the block diagram of the example.

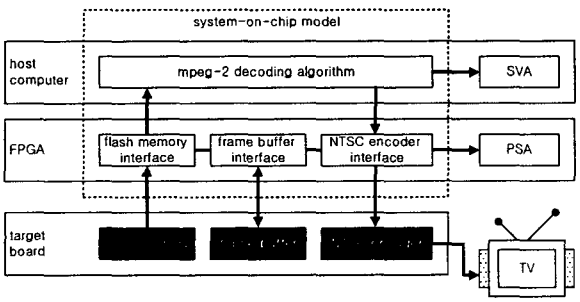


Fig. 6. Block diagram of MPEG-2 decoder example

Linux based PC with Intel Pentium-3 processor is used as a host processor, and XILINX Virtex-E 1600 is used as a FPGA. With natively compiled code and FPGA mapped interface logic, the methodology showed 15 frame/sec, which is impossible with previous co-verification tools or FPGA-based emulators.

For debugging, 111 I/O signals are observed with PSA, and nine variables are observed with SVA. Fig. 7. shows the resulting waveform. The upper signals stand for the I/O signals of the system-on-chip, and the lower signals stand for the software variables. This waveform shows when the software finishes decoding the third frame and starts to read the encoded data from the flash memory.

V. Conclusion

In this paper, in-circuit system-on-chip verification method and debugging is presented. With natively compiled code and FPGA, emulation speed can be accelerated dramatically to allow in-circuit emulation. The proposed debugging environment provides the users with run-time behavior of their hardware and software designs with target board, and makes it possible to correlate events between hardware and software.

This will free the designers from the time-consuming and error-prone jobs of test bench generation and make it a lot easier to find bugs between the hardware/software interfaces.

References

[1] P. Gupta, "Hardware-software codesign", IEEE Potentials, Vol. 20, Issue 5, pp. 31-32, 2002.
[2] W. Wolf, "Hardware/software co-design", ASIC/SOC Conference Proceedings, pp.423 –423, 1999
[3] Serge Leef, "A methodology for virtual hardware/software integration", unpublished, Mentor Graphics, http://www.mentor.com.
[4] Brian Bailey, Russ Klein, and Serge Leef, "Hardware/software co-simulation strategies for the future", unpublished, Mentor Graphics, http://www.mentor.com.
[5] Russ Klein, "Hardware/software co-simulation", unpublished, Mentor Graphics, http://www.mentor.com.
[6] T. W. Albrecht, J. Notbauer, S. Rohringer, "HW/SW coverification performance estimation & benchmark for a 24 embedded RISC core design", Design Automation Conference Proceedings, 1998.
[7] F. Petrot, D. Hommais, A. Greiner, "A simulation environment for core based embedded systems", Simulation Symposium Proceedings, pp. 86-91, 1999
[8] M. Manolescu, I. Furlan, "Software/hardware co-simulation methodology", CAS '99 Proceedings. pp. 89-92, 1999
[9] A. Hoffmann, T. Kogel, H. Meyr, "A framework for fast hardware-software co-simulation", Design Automation and Test in Europe Proceedings, pp. 760-764, 2001
[10] L. Guerra, J. Fitzner, D. Talukdar, C. Schlager, B. Tabbara, V. Zivojnovic, "Cycle and phase accurate DSP modeling and integration for HW/SW co-verification", Design Automation Conference Proceedings, pp. 964-969, 1999[11] C. Liem, F. Nacabal, C. Valderrama, P. Paulin, A. Jerraya, "System-on-a-chip cosimulation and compilation", IEEE Design & Test of Computers, Vol. 14 Issue 2, pp. 16 -25, 1997