Verifying External Interrupts of Embedded Microprocessor in SoC with on-chip bus

Fu-Ching Yang and Jing-Kun Zhong
Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung 80424, Taiwan
Email: {fcyang, jkzhong}@esl.cse.nsysu.edu.tw

Ing-Jer Huang
Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung 80424, Taiwan
Email: ijhuang@cse.nsysu.edu.tw

***Abstract*—The microprocessor verification challenge becomes higher in the on-chip bus (OCB) than in the unit-level. Especially for the external interrupts, since they interface with other IP components, they suffer from the complicated bus protocol and IP conflict problems. This paper proposes a automatic method to verify the microprocessor external interrupt behaviors on the OCB. The verification approach is based on the Processor External Interrupt Verification Tool (PEVT) whose simulation environment is direct-connected memory. In this paper, we implement the PEVT-SoC and successfully verify two SoC platforms, one academic microprocessor and one public domain microprocessor. An interesting bug appears that is impossible to be discovered in the memory bus and not easy to be identified on the OCB. The result shows that the PEVT-SoC effectively shortens the verification time regardless of the system complexity and can be easily migrated to different platforms/microprocessors. With little human effort, even an inexperience designer can generate extensive verification cases in a systematic way.**

I. Introduction

Microprocessor external interrupt behaviors cannot be verified by instruction-based verification approach because external interrupt behaviors are caused by external interrupt signals. External interrupt behaviors are hard to verify because of two reasons. First, the external interrupts' arrival time are variables. Second, the relationship between the external interrupts and the instructions is very close; the microprocessor behaves differently when the external interrupts arrive at different instructions.

Traditional external interrupt verification approaches are usually unsystematic, impractical and not retargetable. The microprocessor core connects to the memory directly. The verification engineers design a hardware module to stimulate the microprocessor external interrupt pins while the microprocessor executes instructions. Such approach has three disadvantages. First, because the possible external interrupt behaviors are huge, the verification coverage of such human dedicated approaches is usually low. Second, modern microprocessor often equips the bus interface unit (BIU) for on-chip bus (OCB) protocol compatibility. It is impractical to ignore the BIU and verify the microprocessor core independently. Also, the microprocessor may be wrong on the OCB even though it passes the verification at core level. Third, because the microprocessor core I/O pins' functionalities depend on the implementation, verification mechanism developed at this level restricts the retargetability.

In this paper, we implement an automatic processor external interrupt verification tool for system-on-a-chip environment (PEVT-SoC) to overcome the above problems. With this tool, the only thing the user needs to do is to describe the microprocessor in the proposed exception description language (EXPDL). PEVT-SoC generates the verification cases by exploring the EXPDL automatically and systematically. The verification cases are then translated to verification hardware and software automatically. The hardware stimulates the microprocessor external interrupt behaviors and monitors the microprocessor in a SoC environment. The software creates specific microprocessor pipeline scenarios for verification. This systematic approach helps the verification engineers achieve good verification coverage within a short time. It is also practical for modern microprocessor verification.

The rest of the paper is organized as follows. Section II discusses the related work of the microprocessor verification. Section III is the overview of the PEVT-SoC. Section IV and Section V discuss the challenges and solutions of verification in the OCB environment. In Section VI, the PEVT-SoC is applied on two OCB platforms to demonstrate the usefulness, the experimental results are discussed as well. Section VII concludes this work.

II. Related work

As the system complexity increases, IP (Intellectual Property) integration in the SoC via the OCB is inevitable. However, because the OCB environment is complex and difficult for verification, the verification is divided into unit-level and system-level [2].

For microprocessor, the unit-level refers to the MUV connecting to the memory directly (called simple verification environment) as Fig. 1(a) shows. The primary concern is the microprocessor core's correctness. The bus protocol is very simple. First, only a small subset of microprocessor I/O pins is used for communication: address, data, read/write control signal and data size. Second, the transmission only takes one cycle to complete. On the contrary, in system-level, the microprocessor connects with other IPs via the OCB as Fig. 1(b) shows. Because of the IP conflict and the

978-1-4244-2820-5/08/$25.00 ©2008 IEEE

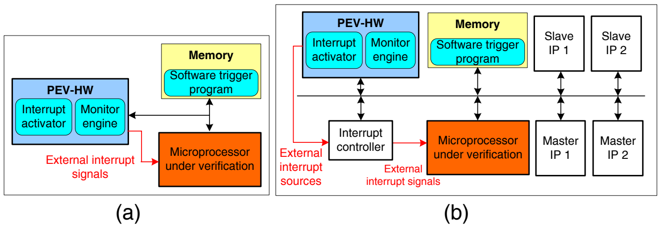

Fig. 1. PEVT-SoC verification environment. (a) Simple verification environment: connecting microprocessor and memory directly (b) On-chip bus verification environment

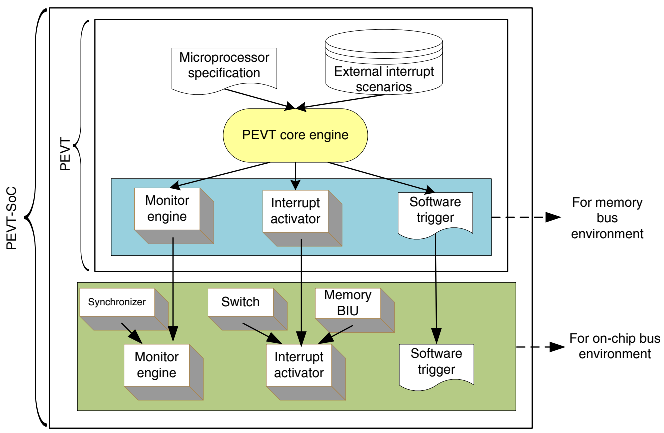

Fig. 2. PEVT-SoC framework

complex bus protocol, the microprocessor's correctness cannot be guaranteed even after passing unit-level verification [2].

As a result, there are research papers about generating transactions on the OCB to stimulate the IPs. The generation methods are either from the bus functional model [2] [8] or the software program generator [3] [4] [5]. Depending on the verification granularity and speed, the verification environment can be register-transfer level (RTL) [3] [8], system level, FPGA or a hybrid of any two of them [4] [5]. The transaction-based verification approach is good for verifying the microprocessor instructions' functional correctness. Unfortunately, the verification of microprocessor external interrupt behaviors still cannot be verified because those transactions are, in fact, instructions or data. As we mentioned in Section I, the instruction-based verification cannot exercise the external interrupt behaviors.

III. PEVT-SoC Framework Overview

The PEVT-SoC verification flow is highly automatic. It is based on the PEVT [7] which targets at the simple verification environment in Fig. 1(a). The framework of PEVT is shown in the upper part of Fig. 2. On top of the flow, the PEVT core engine reads in the microprocessor specification described in EXPDL. The EXPDL contains the microprocessor-dependent information, such as the instruction behaviors, the exception behaviors and the pipeline behaviors. On the right, the external interrupt scenarios describe the microprocessor independent interrupt behaviors for verification. There are individual interrupt behaviors, concurrent interrupt behaviors and nested-interrupt behaviors. The PEVT core engine combines the microprocessor specification information with the external interrupt scenarios to create microprocessor dependent verification cases. To apply the verification cases in the simulation environment, it generates the processor external interrupt verification hardware (PEV-HW), including the interrupt activator and the monitor engine, and the software trigger. The interrupt activator automatically stimulates the microprocessor external interrupt pins while the microprocessor executes the software trigger. The monitor engine automatically observes the microprocessor's responses and provides the verification report.

There are three advantages of this flow. First, the user only has to describe the microprocessor model in the EXPDL. The rest of the work is taken care of automatically. Second, the PEVT can generate good quality coverage cases which are hard to compile manually. Third, the simulation-based verification is automatic, including the external interrupts triggering and the results monitoring.

Although the case exploration is effective, the verification environment in PEVT is impractical. Because the MUV connects to the memory directly, there is no protocol latency and IP conflict, which is not the case in the modern SoC environment. To overcome the problems, this paper extends the verification environment of PEVT to SoC as Fig. 1(b) shows. The verification environment is complex and close to real world environment. It is effective to discover potential bugs which cannot be found in the simple verification environment.

A. *Verification mechanism of simple verification environment*

The external interrupts can be divided into two categories: operation-independent interrupts and operation-dependent interrupts. [7] Operation-independent interrupts arrive at any cycle no matter what the microprocessor operates, such as interrupt request (IRQ) in ARM7 and external interrupt request level (IRL) in LEON2 [6]. They are generated by external components. As for operation-dependent interrupts, they only occurs when the microprocessor does specific operations, which are memory access operations mostly. For example, the data abort exception in ARM7 occurs when the microprocessor accesses an invalid memory address. Therefore, the operation-dependent external interrupt arrival time is restricted to the microprocessor memory access time. Fig. 3(a) shows the data abort exception when the load instruction accesses an invalid memory address. The data abort arrives at the 2nd execution stage since the memory access operation occurs there.

The PEVT core engine analyzes the relationship between the instructions and the external interrupts, and produces the verification cases. Each verification case generated by the PEVT core engine contains the external interrupt triggering information and the expected microprocessor reactions. The triggering information includes when to trigger (*Trigger time*) and what to trigger. This information is read in by the interrupt activator. The expected reactions include the microprocessor's interrupt response time (*Response time*), the vector address and

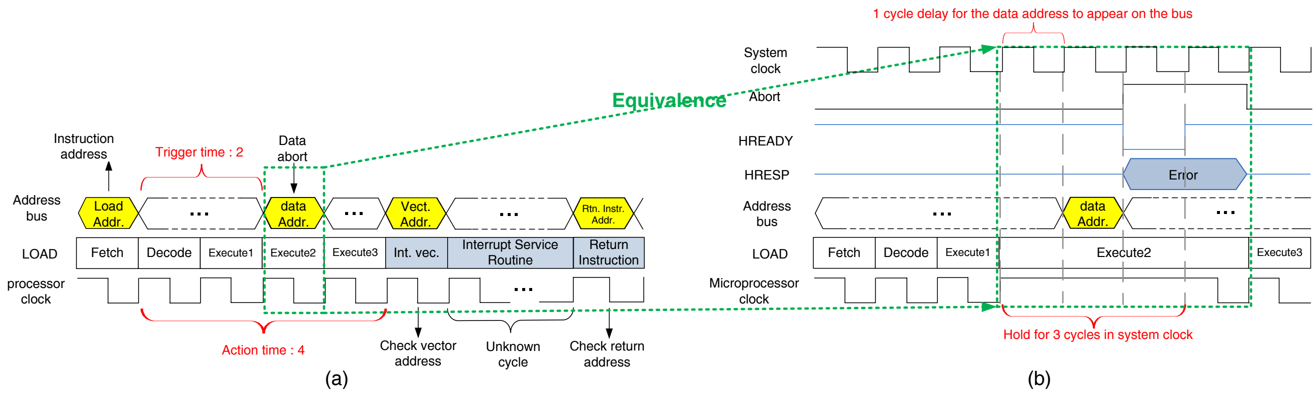

Fig. 3. PEVT triggers data abort at second execution stage of LOAD instruction and verify the MUV's response (a) In simple verification environment (b) In on-chip bus verification environment

the return address. This information is used by the monitor engine.

To trigger the external interrupts at specific time, the interrupt activator takes three steps. Using Fig. 3(a) as an example. It is the verification case that triggering data abort external interrupt at the load instruction's second execution stage. First, the interrupt activator identifies the fetch stage of the instruction the external interrupts arrive. This is achieved by comparing the MUV's instruction address with that instruction's address, which is the load instruction's address in this example. Second, it waits for *Trigger time*. As Fig. 3(a) shows, it is 2 cycles in this example. Finally, it asserts the external interrupt, which is the data abort.

The monitor engine works in the similar way as the interrupt activator does. It contains five steps. For this example, it verifies whether the MUV responses to the data abort after the third execution stage. It also verifies the vector address and return address. First, it identifies the instruction's fetch stage in the same way. Second, it waits for *Response time*, which is 4 cycles in this case. Third, it checks whether the MUV responses to the external interrupt. It is achieved by comparing the microprocessor's instruction address with the expected vector address, which is the data abort exception's vector address in this case. Fourth, it waits for the microprocessor to service the exception. Finally, it checks the return address by comparing the microprocessor's instruction address with the return address. Because the service routine processing time is difficult to be calculated beforehand, the judgment of whether the microprocessor returns is by comparing the microprocessor's instruction address with a threshold address. [7]

IV. Challenges in Verifying on OCB

Please note that the cycle numbers of the *Trigger time* and the *Response time*, are calculated beforehand. There are two reasons why they are not changed in the simple verification environment. First, there is no protocol latency between the microprocessor core and the bus. As Fig. 3(a) shows, the addresses appearing on the bus are the microprocessor's current program counter or current data address. Therefore, the microprocessor's current pipeline status can be realized by observing the current address bus. Second, because there is only one master IP in this environment, the microprocessor is not held due to IP conflict.

However, on the OCB, the first condition breaks, and therefore, the instruction address on the OCB may not reflect the current MUV's pipeline status, which we called it the latency problem. The latency problem causes that the PEV-HW fails to identify the time of the fetch stage. The OCB protocol often supports advanced transfer features that are often not implemented in the microprocessor core, such as burst transfer in AMBA. To support these features, a bus interface unit (BIU) often resides between the microprocessor and the OCB for signals translation. Because of the BIU, there may be latency from the time the microprocessor sending the address to the time the wrapper releasing the address on the bus. Fig. 3(b) is the same verification case on the OCB. Please note that the data address released at the first execution stage is delayed for one system clock cycle comparing to Fig. 3(a).

On the other hand, the microprocessor may be held on the OCB environment. As a result, the PEV-HW cannot operate with the pre-calculate *Trigger time* and *Response time*. The microprocessor could be held because of two reasons. First, the OCB usually has the handshaking protocol to solve the IP competition problem. To access memory, the microprocessor must first request to grant the bus, then it may be held for several cycles until it is granted, and then it begins transmission. Second, the slave IPs on the OCB may not response immediately. It causes that the microprocessor is held until the memory responses. As shown in Fig. 3(b), the microprocessor is held for 4 cycles in the 2nd execution stage, waits for the memory's response.

V. Solutions to the Challenges

A. Synchronization and PC Observation

One important observation is required to overcome the challenges: the internal states of the microprocessor are independent from the bus protocol. Although a memory access requires several cycles to complete, the microprocessor core still completes the memory access operation in one processor

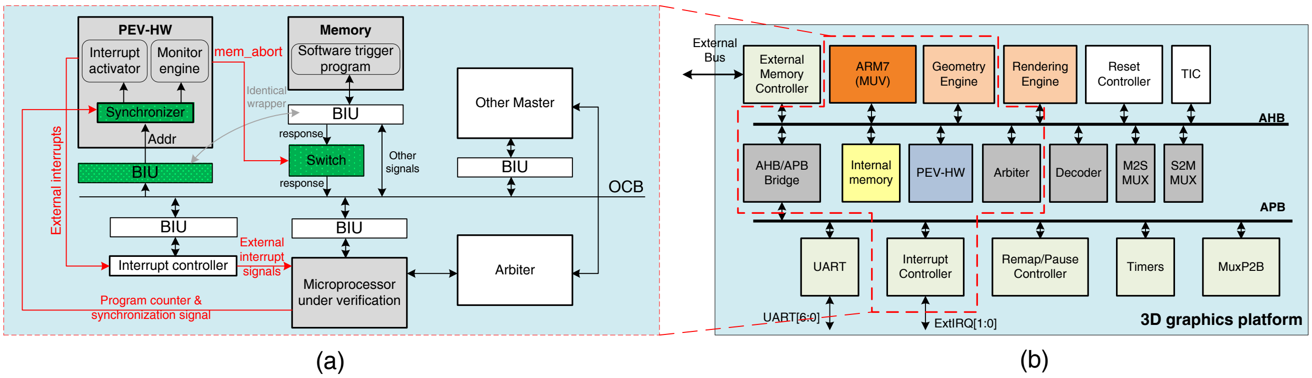

Fig. 4. (a) PEV-HW in OCB verification environment (b) 3D graphics integrated platform in EASY environment

clock cycle. As Fig. 3(b) shows, although the memory access operation takes 4 system clock cycles, it takes only one processor clock cycle. According to our survey in ARM7 and LEON2 [6], same holding mechanism exists. In ARM7, it is achieved by gating clock controlled by the BIU. In LEON2, it is achieved by disabling the enable signal of the pipeline register.

With this observation, the two problems in Section IV can be solved by synchronizing the PEV-HW with the MUV and monitoring the MUV's program counter directly. First, for the latency problem, the PEV-HW can now identify the fetch stage correctly. Second, the PEV-HW can still operate according to the pre-calculate *Trigger time* and *Response time*. Fig. 4(a) shows the OCB verification environment with the enhanced PEV-HW. The additional hardware, called synchronizer, synchronized with the synchronization signal, provides the program counter to the interrupt activator and the monitor engine.

B. Reconstructing memory access patterns

However, for the interrupt activator, there is still one problem with the triggering of the operation-dependent external interrupts. As mentioned in Section III-A, the operation external dependent interrupts depend on the microprocessor operations, mostly memory operations. In the OCB environment, as Fig. 3(b) shows, the memory access abort exceptions are generated from the BIU by interpreting the memory's error responses in the bus protocol. Because the memory response time is unpredictable, the *Trigger time* cannot be calculated beforehand.

The solution is to trigger the operation-dependent external interrupts by the memory access count, instead of the cycle count. For example, in Fig. 3(b), the memory access which causes data abort exception is the first data access after the load instruction is fetched. In order to identify the memory access requests, the PEV-HW adds a BIU, which is identical to the memory BIU, as Fig. 4(a) shows. The BIU deals with the bus protocol and ends with sending the memory core the request address. Because the BIUs are identical, the interrupt activator can identify exactly when the memory requests arrive at the memory. In addition, to make sure the memory access is requested by the MUV instead of other master IPs, the PEV-HW also identifies the IP's identification of the transfers.

As we mentioned, since the operation-dependent external interrupts are generated by interpreting the memory's error response, the PEV-HW triggers them in the same way, instead of asserting the external interrupt pins directly. The motivation is to keep the verification environment as real as possible. To do this, we add the switch module which reside between the memory BIU and the OCB as shown in Fig. 4(a). It is controlled by the interrupt activator to replace the response signal: when the interrupt activator generates the abort signal, the switch module then changes the response signal from correct to error. Please note that the memory can be SRAM or SDRAM, as long as the BIUs are identical.

By carefully analyzing the OCB protocol and the microprocessor's behaviors on the OCB, only three additional hardware modules are required for the PEVT-SoC. Among them, the synchronizer and the switch module are generated automatically by the PEVT-SoC. As for the BIU, it is identical to the memory BIU. It significantly reduces the development effort of the verification environment and reduces the development time.

VI. Case Studies

A. 3D graphics SoC based on ARM's EASY platform

The PEVT-SoC is successfully applied to the microprocessor in the 3D graphic platform as Fig. 4(b) shows. This platform is established based on the AHB Example AMBA SYstem (EASY) [1] released by ARM. It is composed of basic AMBA components including the arbiter, decoder, mux and bridge. There are three master IPs in this platform. First, the academic ARM7 compatible microprocessor, which is the one under verification, controls the data flow of the 3D graphics operations. It has gone through extensive verification including MP3 and μc-OSII porting, and is verified previously by the PEVT on the simple bus. It is interesting to know what bugs might appear when connecting to the AMBA. The other two masters, geometry engine and rendering engine, play as the accelerators to speed up the 3D graphics operations.

In Fig. 4(b), the detail PEV-HW interconnection surrounded by the dash block is shown in Fig. 4(a). To keep the verification

TABLE I
BUS COMPLEXITY REFLECT ON THE SIMULATION TIME

Interrupt behavior	# of case	Sim. time (cycle) Mem. bus	OCB	Ave. time (cycle) Mem. bus	OCB
Individual interrupt	1,229	71,685	90,801	58	74
Concurrent interrupt	41,743	4,130,803	5,298,243	99	127
Nested interrupt	22	2,473	3,218	112	146
Total	42,994	4,204,967	5,392,262	98	125

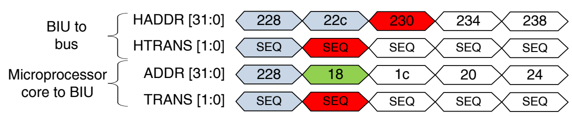

Fig. 5. Wrong TRANS signal causes the BIU sending wrong address

environment as real as possible, the PEV-HW connects the operation independent external interrupt, such as IRQ, to the interrupt controller. The correctness of the interrupt controller and its interactions with the MUV can be verified as well. Multiple masters are welcome to join the system to create the IP conflict scenario.

Table I shows the verification case number of the MUV. The individual interrupt verifies the reaction of single external interrupt arriving at a instruction. The concurrent interrupt verifies the behaviors when multiple interrupt arrive at a period of time. The nested interrupt verifies the nested interrupt state transition. The cases are applied in both the simple bus environment and the OCB. Due to the bus protocol handshaking and microprocessor hold caused by IP conflict on the OCB, the simulation time on the OCB is much higher than on the memory bus. It takes 1.28x cycles on average to complete one case. The total simulation time in real time is about 5 and half hours using the Verilog-XL HDL simulator on 1.2 GHz SUN Blade 2000.

By applying the extensive verification cases, one corner case bug appears which is not found previously. When the microprocessor jumps to the vector address and requests the memory access, it sends the wrong transfer type to the BIU, even though the address is correct. The transfer type indicates the current memory request is a sequential (SEQ) address or non-sequential (NONSEQ) address of the previous address. When jumping to the vector address, it is clearly that the address is not a sequential address. However, the designer does not consider the external interrupt effect on the transfer type.

This bug is hard to be found in the simple verification environment. It is because the transfer type signal is not used in this environment. Fig. 5 shows the example. When the external interrupt is accepted, the vector address 0x18 and the wrong transfer type SEQ are released by the microprocessor core. The designer may not discover the transfer type is incorrect because the microprocessor works fine. However, the BIU processes the address with the transfer type signal. Since the transfer type indicates that it a sequential transfer, the BIU sends the incremented address 0x230. It causes that the microprocessor fetches the wrong instruction. This bug indicates that even with extensive verification on the memory bus, the microprocessor may be failed on the OCB.

Even in the SoC verification environment, this bug is not easy to be discovered, because the transfer type may not be always SEQ when jumping to the vector address. In fact, by applying the individual interrupt verification cases, the error implementation still passes 71% cases. It shows that if the verification engineer uses the unsystematic ad-hoc approach, the bug may not be discovered.

B. LEON2

The PEVT-SoC is also applied on the public domain microprocessor core, SPARC, in an integration system LEON2 [6]. LEON2 has three kinds of external interrupt: interrupt request (IRL), instruction cache error (ICO) and data cache error (DCO). IRL is composed by four pins which represent 15 interrupt sources. The experiment shows that the PEVT-SoC can be painlessly applied to different microprocessors in the SoC environment. About one day is spent to describe the EXPDL. The case generation and PEV-HW generation take a few seconds to complete.

By applying the PEVT-SoC, it generates 290 verification cases as shown in Table II. The verification case is much less comparing to ARM7 because the SPARC does not have many long multi-cycle instructions: only one multi-cycle instruction whose cycle number exceeds 5. Since the case generation is considered at every cycle of the instructions under verify [7], the case number shrinks as the cycle count of the multi-cycle instruction decreases. In addition, the nested interrupt is not supported by LEON2.

We compare the PEVT-SoC with the huge self-verify program delivered along with the LEON2 hardware. It contains 15 c files to verify the microprocessor core, cache, memory and peripherals including the interrupt controller, UART, timer. However, the verification of external interrupt is not stressed: the ICO and DCO exception never occur. As for the IRL, the program is intent on verifying the interrupt controller's function instead of the microprocessor's reaction to the IRL. The program verifies the interrupt controller's masking, pending and priority function by asserting the 15 external interrupts one at a time. Therefore, there are totally 45 IRL triggering cases regardless of what instruction is in the pipeline stage. However, in an additional case, it does verify the reaction of triggering one IRL on a multi-cycle instruction. So the total verification cases are 46.

This verification mechanism has several potential flaws which are overcome by the PEVT-SoC. First, the interrupt is triggered by writing the interrupt controller's register with the store instruction. It limits the trigger time to the memory stage of the store instruction. In fact, the interrupt can arrive at any cycle. Second, in [7], different instructions may have different reactions to the external interrupt. It could be dangerous to neglect the relationship. Third, the verification mechanism is

TABLE II
VERIFICATION CASES ON LEON2

Classification	Case number
Individual int.	238
Concurrent int.	52
Nested int.	Not supported
Total	290

not cycle accurate. The microprocessor could jump to the vector address one cycle earlier/later, resulting in potential errors.

The purpose of the comparison is to demonstrate that the PEVT-SoC can generate extensive cases with little human effort instead of attacking the effectiveness of the LEON2's approach. Because there is no clear definition of how to implement the interrupt control module, the external interrupt behaviors' complexity highly depends on the hardware implementation, resulting in no standard coverage measurement. The intention of the PEVT-SoC is to generate the highest coverage verification cases regardless of the hardware implementation. Because of the high automation, it can compile huge verification case number in a short time that is impossible to achieve manually.

Another advantage of the PEVT-SoC on the OCB is that the verification can be easily applied to different microprocessor as long as they are on the OCB. In this case, even if the LEON2 is written in VHDL while the generated PEV-HW is written in Verilog, it is still painless to apply the PEVT on LEON2: the verification environment establishment takes about 2 hours by a master student.

VII. CONCLUSION

We have presented the methods to enhance the PEVT in the SoC verification environment and implemented as A CAD tool - PEVT-SoC. With careful analysis of the OCB protocol and the microprocessor's behaviors on the OCB, three additional hardware modules are required. Two of them are generated automatically. One is obtained without modification.

PEVT-SoC was applied to verify an academic implementation of ARM7 microprocessor core, which had been verified previously by PEVT in the direct memory connection environment. PEVT generated 42,994 verification cases and successfully identified one bug. This bug is difficult to identify in both the simple bus environment and the SoC environment. The verification cases take about 5,392,262 cycles of RTL simulation on a SUN Blade 2000 workstation. The experiment shows that PEVT could generate highly focused verification cases which identity potential bugs with much less simulation cycles, compared with traditional regression tests which consume huge amount of simulation cycles. We also applied PEVT-SoC on the public domain LEON2 platform to prove the retargetability. With little human involvement, the PEVT-SoC can easily compile extensive verification cases at better coverage, and shorten the verification time significantly.

REFERENCES

[1] ARM. *AHB Example AMBA SYstem Technical Reference Manual*. ARM, 1999.

[2] M. Bose, M. H. Nodine, A. Chodavadia, W. R. J. Jr, L. R. Nunes, and V. Zavadsky. Modeling ip responses in testcase generation for systems-on-chip verification. In *Proceedings of the Fourth International Workshop on Microprocessor Test and Verification (MTV'03)*, pages 7–10, May 29–30, 2003.

[3] A. Cheng, A. Cheng, and C.-C. Lim. A software test program generator for verifying system-on-chips. In *Proceedings of the 10th IEEE International High-Level Design Validation and Test Workshop*, pages 79–86, Nov. 30 -Dec. 2, 2005.

[4] S.-H. Lee, J.-G. Lee, S. Kim, W. Hwangbo, and C.-M. Kyung. Soc design environment with automated configurable bus generation for rapid prototyping. In *Proceedings of 6th International Conference On ASIC (ASICON)*, pages 41–45, Oct. 24–27, 2005.

[5] Y. Nakamura, K. Hosokawa, I. Kuroda, K. Yoshikawa, and T. Yoshimura. 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. In *Proceedings of the 41th Design Automation Conference (DAC)*, pages 299–304, June 7–11, 2004.

[6] G. Research. *The LEON-2 User's Manual*, June 2002.

[7] F.-C. Yang, W.-K. Huang, and I.-J. Huang. Automatic verification of external interrupt behaviors for microprocessor design. In *Proceedings of the 44rd Design Automation Conference (DAC)*, pages 896–901, June 4–8, 2007.

[8] J. Yu, T. Li, and Q. Tan. The use of uml sequence diagram for system-on-chip system level transaction-based functional verification. In *Proceedings of the Sixth World Congress on Intelligent Control and Automation (WCICA)*, pages 6173–6177, June 21–23, 2006.