Verification of a Microprocessor Using Real World Applications

You-Sung Chang, Seungjong Lee, In-Cheol Park, and Chong-Min Kyung
Dept. of EE, KAIST, Taejon, Korea
caviar@maestro.kaist.ac.kr, sjlee@duo.kaist.ac.kr, {icpark,kyung}@ee.kaist.ac.kr

Abstract

In this paper, we describe a fast and convenient verification methodology for microprocessor using large-size, real application programs as test vectors. The verification environment is based on automatic consistency checking between the golden behavioral reference model and the target HDL model, which are run in an handshaking fashion. In conjunction with the automatic comparison facility, a new HDL saver is proposed to accelerate the verification process. The proposed saver allows 'restart' from the nearest checkpoint before the point of inconsistency detection regardless of whether any modification on the source code is made or not. It is to be contrasted with conventional saver that does not allow restart when some design change, or debugging is made. We have proved the effectiveness of the environment through applying it to a real-world example, i.e., Pentium-compatible processor design process. It was shown that the HDL verification with the proposed saver can be faster and more flexible than the hardware emulation approach. In short, it was demonstrated that restartability with source code modification capability is very important in obtaining the short debugging turnaround time by eliminating a large number of redundant simulations.

1 Introduction

As the complexity of digital systems implementable as an integrated circuits increases with the ever-growing number of transistors per chip, verification has emerged as the bottleneck in the whole design and design validation procedure. Verification very often takes up more than half of the whole design time for complex microprocessors[5]. Various design verification methodologies with the relevant environmental setup for an effective verification have been proposed and/or used now[1, 2, 3, 4, 5, 6]. For the case of designing next-generation microprocessors, microcontrollers, and DSP processors which already have a large number of firmwares and application softwares running on them, the requirement on the backward compatibility with the previous products or the full compliance with all relevant softwares makes the verification process more and more difficult, error-prone and time consuming. For that purpose, one needs to use real application programs or existing operating systems as verification stimuli.

Hardware emulation, which was introduced to reduce the simulation time also contributed to the in-system verification, i.e., verification of the target design in the real system environment[2, 3, 4]. However, emulation cannot be used for the verification of all the design levels, because emulation requires the design description as a gate-level netlist, which can be obtained only at the final stage of the design process. Moreover, emulation does not provide a friendly interface for fault diagnostics and debugging, i.e., it serves mainly as a go/no-go test, and it is very difficult to observe the location of bug with emulation. Thus, in the initial phase where frequent bugs can occur, emulation is not very helpful as a verification tool. In other words, emulation is a means of confirmation, rather than the debugging facility as it asserts.

DAC 99, New Orleans, Louisiana

Using the large simulation vectors such as real application softwares often leads to excessively long verification time in the pure simulation approach. On the other hand, hardware debugging using the pure emulation approach becomes very difficult especially, when the test program is long. In this paper, we propose a verification environment including a new HDL saver, which allows 'restart' after the design change for bug fix at, or reasonably before, the point of error detection, not going all the way back to the initial point as in conventional simulation with 'save and restart' capability. The proposed HDL saver has been successfully used in our Pentium-compatible microprocessor design project using various large real-world programs as DOS and Windows as the verification vectors. We describe the construction of verification environment in section 2 and details of the proposed HDL saver in section 3. Finally, we discuss about verification strategy applied to our processor design project and the effectiveness of the verification environment.

2 Verification Based on Inter-model Consistency Check

Generally, the first stage in the design flow of the microprocessor is to build a behavioral model for the target design. In our case, the behavioral model of the microprocessor is an instruction set simulator(ISS). The ISS was written in C language and verified in cooperation with the system model for all PC peripheral components interacting with the microprocessor such as keyboard, hard disk, graphic card, etc. also written in C language. The merit of the behavioral model is that firstly, it is easy to read, diagnose and modify, and secondly, it runs fast. The HDL model is written based on the understanding of the behavioral model of the target design. The HDL model, thus produced, then needs to be verified according to the test vectors generated independently or from the simulation of the behavioral model. During the course of our Pentium-compatible processor design project, we wrote the HDL model in Verilog and utilized two simulators, Cadence Verilog-XL[7] at the early stage of the project, and Chronologic VCS Simulator[8] to increase the speed, once the design is stabilized.

During the course of the whole design procedure, we actively used the 'verified ISS' as the reference for the validity check of the HDL model. In our project, running DOS or Windows OS as the verification vectors on the HDL model requires extremely long simulation time, typically several weeks or months in the state-of-the-art workstations. Consequently, the execution traces are too large to be saved in a physical storage, and we cannot check the correctness of the simulation results of the HDL model using the reference traces obtained by the execution of the ISS using the same application programs. Therefore, we have chosen to use the on-the-fly method for correctness checking. In the environment, ISS and HDL simulator communicate with each other through the consistency check routine(CCR) based on the Inter-process Communication(IPC) as shown in Fig. 1.

The automatic-comparing facility is built using the standard IPC library and the Verilog Program Language Interface(PLI) library. It provides a function 'state_compare' as a gate of communication, through which the HDL simulator receives the running information related to synchronization as well as the state information for consistency checking from the ISS and adjusts the environment variables, if necessary, and compares its internal states with those of ISS. If the comparison discloses any discrepancy of states between two models, simulation is paused displaying the inconsistency information and allows the designer to traverse the internal

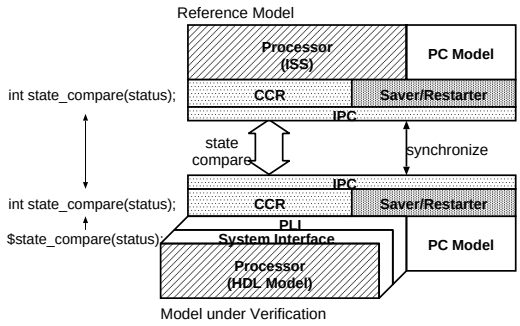

Figure 1: Simulation environment for the automatic consistency check between the behavioral reference model and the target HDL design

of design at the detection point. The information transfer and the state comparison are always made at the end of each instruction because the atomic unit of the ISS is instruction. For our HDL design, 'write-back' stage of the pipeline becomes the reference for comparison at the instruction boundary.

Comparing the whole set of states at the end of every instruction is definitely an overkill, and the selection of states being compared, therefore, needs to be made to reflect the nature of the target design. Large number of states slows down the simulation while small number yields lack of information for the correctness check. We selected all the internal programming registers of the microprocessor being designed as the state variables for comparison. The 16 internal registers were more or less enough to detect most misbehavior of the previous stages.

However, the difference in the description level of the two models may make the distance between the point of detection and the point of bug origination very large. In our Pentium-compatible processor design, the problem is the due to the instruction and data caches and BUS interface unit, which have no counterparts in the ISS. The two parts are evaluated each cycle, while ISS evaluates at the end of every instruction that can take many cycles. Three parts can disturb the order of memory operations and make long latency of the memory operations in the HDL model. Through the consistency check routine, external memory writes that appear only in ISS can be traced to assist the debugging. The trace can be managed to contain triple of access time, address, and data or other information for debugging.

Synchronization is another important issue in the implementation of the automatic-comparison facility. The running information related with the synchronization includes data from the input devices such as keyboard and mouse and interrupt signals from various device model. It is essential to generate the same situation, i.e., identical status bits and signals and to synchronize interrupt or other event timing for both simulation models which are in different description levels with respect to each other.

3 Saver with 'Modify and Restart' Capability

When real, big application programs running for hours, days and weeks, are used as test vectors, 'save and restart' is an essential feature of simulation to protect itself from any unintentional break, such as system down. Most commercial HDL simulators nowadays support the restartability but in a quite limited fashion; They just dump all running information in the simulator into a data file or into another executable file, and the hard copies of all the running information can only be 'paused and restarted'. However, if any modification is made for fixing a bug, previous files dumped before fixing the bug become useless. This eventually forces the simulation to be rewound to the very beginning for each design change for debugging.

Typical procedure for debugging using conventional simulation tools is as follows; If a fault is detected, the designer needs to trace back from the point of detection to the point of fault origination. When big simulation programs such as application programs are used, it is impossible to dump all the simulation result throughout all the simulation time because of the huge size of the dump file. Instead, the designer can choose to resimulate from a pre-set restart point which is usually set at regular time interval and generate the dump files only for the small recent time interval. As a matter of fact, if simulator does not support any kind of restartability, we have to resimulate from the start point and begin to get the signal dump from some time, t_x, sufficiently before the point of inconsistency detection(POID) such that the cause of the bug is after t_x. Once the signal dump is obtained, we find the location of the fault and start the debugging process. The debugging means modifications in the HDL source code and, therefore, it is required to resimulate from the very beginning again. This is a really wearisome routine in the simulation using long real application programs which takes several days or weeks to arrive at the bug position.

To remove the requirement of the redundant resimulation, we built a new HDL saver allowing restart at the some instruction boundary sufficiently before the POID after the HDL source code modification. Our approach is quite different from that of previous HDL savers. In our HDL saver, if a save request is invoked by the user or by a periodic routine, the saver finds the so-called 'stable point', which is defined as the time point that has a small number of designer-controllable special events. At the stable point, the saver needs to dump only the steady-state signal values(just before the event) containing all the information to restart simulation jointly with the designer-controllable special events. This is to be contrasted from the conventional restartable simulator trying to save all the information required to assure restart from a given arbitrary point.

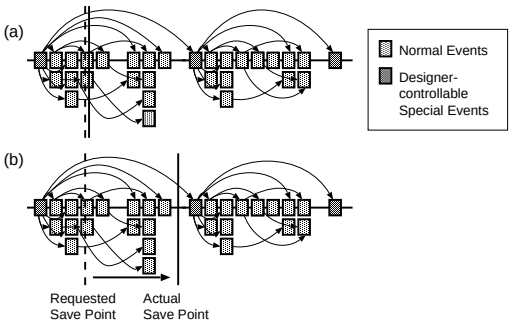

Figure 2: Two possible save strategies; (a) Conventional save at a requested point, and (b) Deferred save at a stable point

The stable point, however, is generally hard to find in the event-driven simulators. On the other hand, if the simulator is cycle-based, the stable point, which happens to be the end of clock signal, is easily found. Similarly, stable points of synchronous system simulation can be the end of each cycle. For both cases, the clock becomes the designer-controllable special event. It implies that, in most of the real cases, we have no difficulties to find the stable points.

Once the stable point is reached, all status information on internal registers and wires are read through a hierarchical search and stored in the internal database format. We used three kinds of entry format according to the type of value, i.e., scalar, vector and memory. To restart, after loading the dump of steady signals, the simulation is triggered by activating the controllable special event.

In the implementation of our processor verification environment, we chose the stable point as the point right before the end of the last clock cycle of each instruction in which every event is cleared. This is based on the fact that instruction is the atomic step in simulation and the microprocessor under design has no multi-cycle combinational path at the instruction boundaries. Architectural issues such as pipelining and super-scalar, does not affect the decision on the stable point. This holds true for the case of our Pentium-compatible microprocessor which has a 6-stage pipeline and 2-way super-scalar architecture.

By incorporating the new type of HDL saver in the automatic consistency checking interface, we can offer a complete debugging

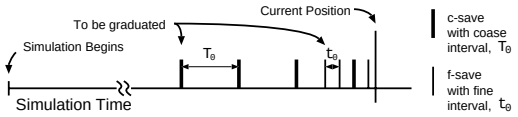

Figure 3: Generation and management of snapshots in two heterogeneous time scales, i.e., coarse interval,T_0 and fine interval t_0

environment even for large test programs. Our HDL saver records a certain number of past checkpoints in two different heterogeneous time scales, i.e., one in large and the other in small interval as shown in Fig. 3. This recording scheme works especially well for the case when a bug escapes all the recorded stable points in the fine scale but not those in the large time scale. After debugging, with the assistance of the HDL saver, we need to continue to simulate from the nearest restart position before the possible origination of the bug. In most cases, the safe restart point is that right before the bug position.

Fig. 4 explains the effectiveness of the proposed HDL saver compared with the previous ones. For the case of simulator without any kind of restartability shown as (a), for each bug, roughly twice the simulation time up to the bug position becomes the additional overhead. By introducing the conventional restartability shown as (b) in Fig. 4, the first overhead(Overhead1) disappears; however the Overhead2 still remains. Eventually, in our case, only a small portion(represented as T_{SD} in Fig. 4) remains as the overhead. The turnaround time of debugging loop for each unsuccessful bug-fix decreases dramatically with the proposed HDL saver. The effect of the our saver will be magnified as the simulation time becomes larger.

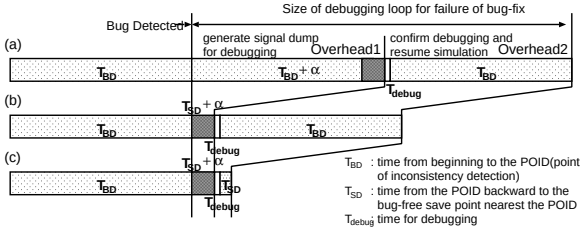

Figure 4: Comparison of the total time for simulation and debugging among (a) simulation without restartability (α is to account for time overhead for signal dump during simulation), (b) simulation with conventional `save and restart' capability, and (c) simulation with the proposed `modify and restart' capability

As the design is quite stabilized in the final stage where real application programs are used as test vectors. Therefore, minor design modifications generally do not affect the saved status. During design changes, some registers can be added or deleted. In that case, the saved information needs to be modified to reflect the changes. The modification is done by addition/initialization or deletion of corresponding saved information using a proper snapshot editing tool. Practically, there is no restriction on modification except that it degrades the confidence of the previous simulation already passed. Any risk invoked by snapshot editing can be resolved, if necessary, by the regression test. It can be used to generate special cases and to check stability. We can catch buggy situations during simulation and register the cases as test vectors for later use. The overhead due to the addition of the HDL saver feature in the simulation is negligible, because save operation is not invoked frequently.

One of the major issues for the implementation of the automatic-comparison feature is the synchronization of the save and restart point between two models written in different description levels as well as the synchronization of the status comparison point. Fig. 5 illustrates a situation in the superscalar execution. While the behavioral model advances in the step size of instruction, the step size of the super-scalar architecture varies according to whether pairing is made or not. We solved the problem by passing pairing information to ISS; in addition, we implemented a special command 'compare_skip' in the ISS as a strong means to adjust the synchronization at the restart.

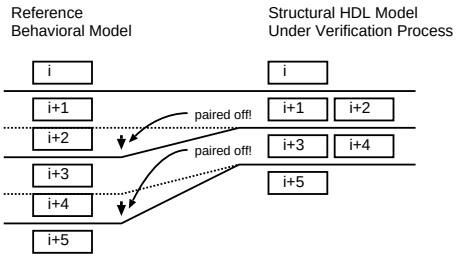

Figure 5: Comparison and possible save points are adjusted according to the decision of instruction pairing

4 Verification of the Pentium-compatible Processor Design

Fig. 6 shows the functional verification flow for the Pentium-compatible microprocessor design project[6]. Altogether, we used three kinds of verification means, microcode verifier, RTL simulation, and emulation. Microcode verifier was used to verify the microcode without architectural details in the early design stage. RTL simulation based on the HDL design was used in the wide range of verification stages. What to notice is that we applied both verification means, HDL-based simulation and hardware emulation concurrently at the final stage.

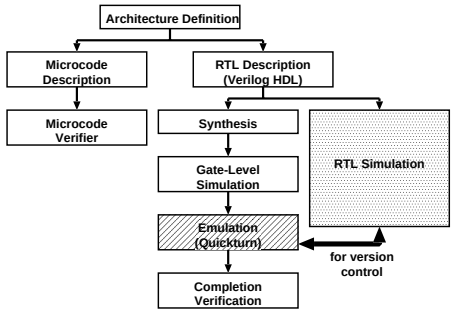

Figure 6: Functional verification for Pentium-compatible processor development

At the final stage, the strategy of using the HDL-based verification environment is as follows; for the initial phase of the final stage using real application programs, we commence the functional verification with both the verification means. When we arrive at the point where the design is sufficiently refined, we switch to emulation. Once a bug is detected for an application program, we augment the emulation with the HDL-based verification.

Fig.7 shows the verification of the Pentium-compatible processor design at our HDL-based verification environment. It shows the concurrent execution of the ISS and the target design under the automatic consistency check.

5 Effectiveness of the Verification Environment

In our experience obtained during the Pentium-compatible processor design, the HDL-based simulation takes 20 days on the Ultra2 SPARC machine to boot the real OS, 'MS windows 3.1' while the emulation using Quickturn M3000 system took only 40 minutes to do the same. The speed gap is very large, about a thousand times. It is generally believed that HDL-based simulation can never be a counter-plan for emulation. However, we showed that the proposed saver could decrease the debugging turnaround time for a single bug-fix under 2 hours. Table 1 summarizes the required time for booting OS and a single bug-fix for each configuration.

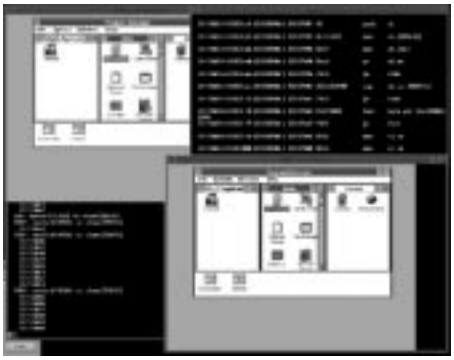

Figure 7: OS booting test of our microprocessor designed in HDL; It is running under the automatic consistency checking environment

Table 1: Comparison of required times for booting OS and a single bug-fix

Model	Method	Booting Windows	Single Bug-fix
Emulation	QTM3000	40 minutes	< 14 days
HDL Simulation	VCS + ICC	20 days	< 20 days
HDL Simulation	VCS + ICC + Proposed Saver	20 days	< 2 hours

Practically, it is to be noted that most of the time to verify a complex design is consumed on debugging. In spite of the speed gap, our experience shows that the HDL simulation with the proposed HDL saver provides a very effective solution as a verification and debugging tool for the application tests. Fig. 8 shows the verification progress of the OS booting test for the Pentium-compatible microprocessor design.

As shown in Fig. 8, HDL simulation with the proposed HDL saver which was installed 26 weeks after the start of the simulation always leads the emulation with a difference of a huge number of executed instructions. Among bugs detected by emulation, only two bugs related to the timing in the BUS interface unit were meaningful, as they were detectable only through the emulation. All other bugs were detected as well by the simulation and wiped out prior to the detection with emulation. Fig. 8 shows that an abrupt enhancement in the verification progress that was achieved by attaching the proposed saver. The HDL saver has reduced the turnaround time by one-hundredth.

It is possible that a design modification can disclose a new bug in the past that necessitates a regression test, i.e., resimulation from the beginning, after one painful pass of simulation. However, this probability is generally very low especially in a matured design that has already entered the verification phase using real application programs. To support this argument, in most of our design project, we always have passed the regression test without any additional bugs detected. The regression test could be done without any overhead in the verification schedule, because the test was executed in parallel with the main verification flow in extra machines.

We have tried the similar approach in the C-based simulation[5] to prove that the scheme is very effective in the cycle-based simulation. The C-based approach has a merit in simulation speed, however, the overhead of conversion to HDL and scheduler can not be neglected. Because the C-based approach relies on scheduling particular to the design to speed up the simulation, it may still hold bugs veiled with its own implied scheduling even though the verification is completed. C description must be translated into adequate hardware description language to be verified in timing and to be synthesized. It has the risk of bug intervention during the language conversion. In our experience of the C-based verification approach, the conversion and reverification can take over a half-year. On the other hand, rapid progress of the recent compiled code HDL simulators has significantly reduced the speed gap between the C-based simulation and HDL simulation. The difference can be less than two times in the case of our Pentium-compatible processor design project.

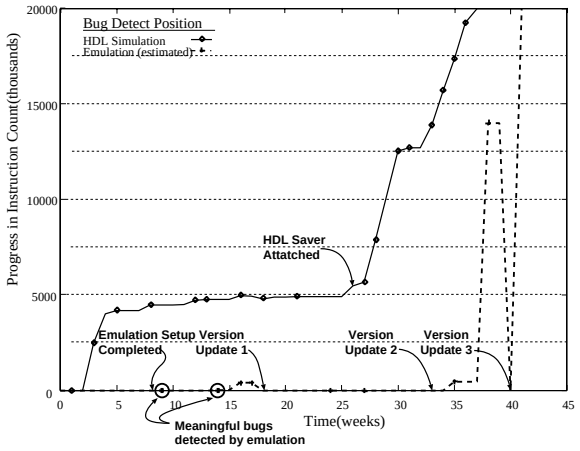

Figure 8: Debugging progress traces for HDL-based simulation and for emulation during the verification of Pentium-compatible microprocessor design using OS(DOS 6.0 and windows 3.1) booting as the verification vector

6 Conclusion

In this paper, we proposed an efficient HDL-based verification methodology that can be used for real-world application programs requiring extremely long simulation time. What we notice in the verification is that the debugging turnaround time rather than the simulation time occupies most of the verification process.

To facilitate the debugging, we developed a verification environment based on the automatic consistency checking between a golden behavioral reference model and the target HDL model. We proposed a new HDL saver which allows 'restart' after the design change for bug fix at, or reasonably before, the point of inconsistency detection, not going all the way back to the initial point as in conventional simulation with 'save and restart' capability. Therefore, the proposed HDL saver can contribute significantly to reducing the verification time by eliminating the redundant iterative simulation. In our experience, the HDL simulation environment with the proposed HDL saver enables faster verification than relying only on hardware emulation approach.

References

[1] R. C. Ho, C. H. Yang, M. A. Horowitz, and D. L. Dill. "Architecture Validation for Processors". *Proceedings of the 22th Annual International Symposium on Computer Architecture*, pp. 404–413, 1995.

[2] G. Ganapathy, R. Narayan, G. Jorden, and D. Fernandez. "Hardware Emulation for Functional Verification of K5". *Proceedings of 33th Design Automation Conference*, pp. 315–318, 1996.

[3] V. Popescu and B. McNamara. "Innovative Verification Strategy Reduces Design Cycle Time For High-End SPARC Processor". *Proceedings of 33th Design Automation Conference*, pp. 311–314, 1996.

[4] S. Mehta, S. Al-Ashari, D. Chen, D. Chen, S. Cokmez, P. Desai, R. Eltejaein, P. Fu, J. Gee, T. Granvold, A. Iyer, K. Lin, G. Maturana, D. McConn, H. Mohammed, J. Mostoufi, A. Moudgal, S. Nori, N. Parveen, G. Peterson, M. Splain, and T. Yu. "Verification of the UltraSPARC^{TM} Microprocessor". *COMPCON*, pp. 452–461, 1995.

[5] J.-S. Yim, Y.-H. Hwang, C.-J. Park, H. Choi, W.-S. Yang, H.-S. Oh, I.-C. Park, and C.-M. Kyung. "A C-Based RTL Design Verification Methodology for Complex Microprocessor". *Proceedings of 34th Design Automation Conference*, pp. 83–88, 1997.

[6] S. Lee, Y.-S. Chang, S.-I. Park, I.-C. Park, and C.-M. Kyung. "An Efficient Approach to Functional Verification of Complex Processors". *Proceedings of International Conference on Computer Systems Technology for Industrial Applications - Chip Technology*, pp. 87–92, 1998.

[7] *Verilog-XL Reference Manual Volume 1,2*. Cadence Design Systems, 1995.

[8] *VCS User's Guide*. Chronologic Simulation, 1996.

[9] *Programming Language Interface Reference Manual Volume 1,2*. Cadence Design Systems, 1992.

[10] W. R. Stevens. *Advanced Programming in the UNIX Environment*. Addison-Wesley Publishing Company, 1992.