Verification Methodology of Compatible Microprocessors

Joon-Seo Yim, Chang-Jae Park, Woo-Seung Yang, Hun-Seung Oh, Hee-Choul Lee, Hoon Choi,
Tae-Hoon Kim, Seung-Jong Lee, Nara Won, Yung-Hei Lee, In-Cheol Park and Chong-Min Kyung
Department of Electrical Engineering
KAIST
Taejon, 305-701, Korea
Tel: +82-42-866-0700
Fax: +82-42-866-0702
e-mail: kyung@dalnara.kaist.ac.kr

Abstract— As the complexity of high-performance microprocessor increases, functional verification becomes more difficult and emerges as the bottleneck of the design cycle. In this paper, we suggest a functional verification methodology, especially for the compatible microprocessor design. To guarantee the perfect compatibility with previous microprocessors, we developed three C models in different representation levels, *i.e., Polaris, MCV(Micro-Code Verifier)* and *StreC*. C models are co-simulated with consistency checking between different two models. The simulation speed of C models makes it possible to test the "real-world" application programs on the RTL design with a software board model. To increase the confidence level of verifications, *Profiler* reports the verification coverage of the test vector, which is fed back to the automatic test program generator. *Restartability* feature also helps significantly reduce the total simulation time. Using the proposed verification methodology, we designed and verified an Intel 486-compatible microprocessor successfully.

I. Introduction

The advancement of semiconductor technology has made it feasible to integrate more than ten million transistors on a single chip and to operate at faster than 500MHz clock speed. This astounding chip complexity has resulted in difficulties in the verification[1, 2, 3, 4, 5, 6, 7]. Moreover, recent microprocessors tend to maintain the instruction-level compatibility with the previous ones to save huge efforts for application software development[2]. Though compatibility can be best guaranteed by an exhaustive simulation with real application programs, the simulation time increases drastically as the design complexity increases and has been a bottleneck in a complex microprocessor design.

Therefore, it is crucial to verify the functionality of design and eliminate errors at an early stage of the design. Eradicating the functional bugs which are alive until the final gate level simulation requires excessively large amount of computing time and debugging efforts. Efficient verification methodologies become vital to the success of microprocessor design and their significance will continue to increase as we move into more complex designs.

Recently, the verification crisis of microprocessor design leads to hot research issues both in academia and industry. The hardware emulation[2], formal verification[1] and cycle-based simulation[8] have become the state-of-the-art verification methodologies. Even though the emulation is widely accepted, it requires too much cost and requires that the gate level design is already finished. Therefore, it requires large turnaround penalty to fix gate-level bugs. Formal verification method has been used successfully to verify a wide variety of moderate-sized hardware designs [9] [10] [11] [12]. The industry is beginning to look at formal verification as an alternative to the simulation for obtaining higher assurance than is currently possible. Despite the great increases in the number of organizations and projects applying formal methods, formal verification is still the case that the vast majority of potential users of formal methods fail to become actual users[13]. The hardware description language(HDL) such as VHDL and Verilog is a convenient method to describe a hardware, and a cycle-based simulation shows a clear simulation performance advantage over an event-driven simulator[14]. However, the general purpose Verilog simulator is much slower than the custom-tailored simulation using C language. Although hardware accelerators[15] yield significant speed-up for the gate-level design, they do not give any advantage for RTL or behavior level design. Most of the design time is consumed by RTL simulation rather than the description of design itself. We propose in this paper a low-cost simulation method based on *RTL C model* to speed up the RTL simulation.

To estimate how much time this methodology can save in a complex microprocessor design, one needs to understand that a design is not a linear sequence of steps and there are several iterations through design, simulation, de-

ASP-DAC '97
0-89791-851-7$5.00

bugging, code fixing, recompile, and resimulation. In the productivity issue, we will examine how much design time is consumed for each design activities, and will suggest important methodology to reduce the design verification time. Finally we will show how effective our verification methodology is over other HDL approaches.

This paper describes the functional verification methodology of K486, which is Intel 80486 compatible microprocessor which is being developed at KAIST. The paper is organized as follows. A proposed functional verification methodology is described in section II. And section III deals with the productivity issue for design verification. Finally section IV shows some verification results.

II. Functional Verification

A. Design Flow

A traditional top-down design flow for microprocessors is presented in Fig. 1(a). From the design specification, design is gradually refined and moved down to the physical implementation level. An important problem in the top-down design flow is how to maintain the consistency between the consecutive design levels. As shown in Fig. 1(b), we divide the description level in more detail such as C_1, C_2 and C_3, which represent the model of microprocessor written in C for behavioral, micro-operational, and RTL description respectively.

level	model	level of description	language
C_1	Polaris	Instruction behavior	C
C_2	MCV	Micro-operation	C
C_3	StreC	RTL	C
V	V4	RTL	Verilog

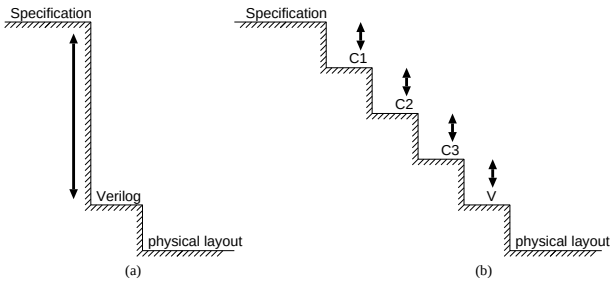

Fig. 1. (a) Traditional *vs.* (b) proposed design flow for microprocessors

In our simulation-based approaches for microprocessor design, RTL structural design using C language is co-simulated with a reference model[4, 5, 6, 7], *i.e.,* a behavioral level model or micro-operation level model. Real-world application programs are executed on these C model with a software model for the target system .

B. Reference Model

Traditionally, instruction set simulators are developed for the compiler or software development concurrently with chip design and used to determine such design parameters as instruction set, cache architecture, buffer size, bus protocols and other hardware resources. In the verification, this instruction set simulator can be used as a reference model for further detailed designs.

TABLE I
Description of CPU models in various levels

model	description	features	code lines
Polaris	Macro instruction behavior	register	9,675
MCV	Micro-operation behavior	register internal bus	18,534
StreC	Clock-Level RTL Phi 1 edge, Phi 1 level Phi 2 edge, Phi 2 level	Flip-flop,latch internal bus combinational pipeline	55,415
Verilog HDL	Clock and event-based RTL	Flip-flop,latch internal bus combinational pipeline timing	35,223

There are two types of reference models in our design. As Table I shows, *Polaris* describes the behavior of X86 instruction sets. It does not describe the detailed architecture such as pipelining, superscalar instruction pairing, parallel functional units, cache, and buffering. Representing a higher abstraction level allows us to produce a reference model that contains very few bugs and to execute over 150 times the speed of the RTL C model. This execution speed of Polaris makes it possible to run the test consisting of several billion instructions on software model. This is impossible with a commercial cycle-based simulator or gate level HDL simulation even with hardware acceleration.

For the CISC microprocessors and FPU, one macro instruction consumes multiple cycles, therefore one macro instruction is subdivided into a number of micro-operations which is executed in one clock cycle. Micro-operations are closely related to the datapath hardware or exception handling scheme. *MCV(Micro Code Verifier)* verifies a C model describing the micro-operation level behavior. Neither Polaris nor MCV exactly matches the timing details as obtained via RTL model as shown in Fig. 2. However, the speed advantage of Polaris and MCV makes them to be used as "golden" reference model of RTL micro-architecture design.

To verify all the cases which can occur in real system, such as hardware interrupts, multiple memory and I/O cycles, it is necessary to simulate *through real-world programs* rather than *by instructions*. To run the real-world programs in design model, a software model of system board[3] called *VPC(Virtual PC)* is developed. VPC contains the software models of all PC components. It in-

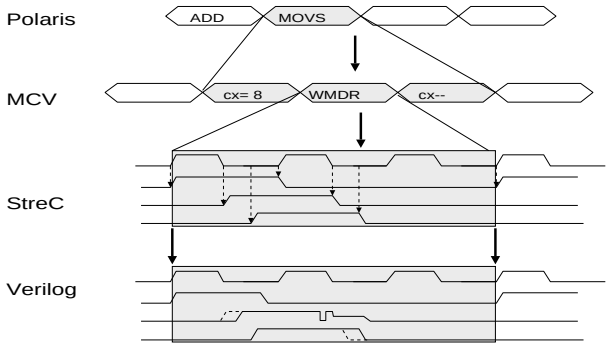

Fig. 2. Timing of each model

cludes main memory, hard/floppy disk drive, interrupt controller, timer, keyboard, and video display linked to X window system on the workstation as shown in Fig. 3.

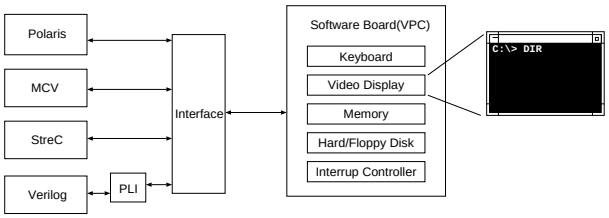

Fig. 3. VPC(Virtual PC) model interfaced with various simulators *i.e.,* Polaris, MCV, StreC and Verilog

C. StreC : RTL C model

Traditionally, RTL description is based on HDL such as Verilog. To achieve high simulation speed, we described RTL operation in C language. This model called *StreC* accurately describes the cycle-by-cycle synchronous logic behavior as shown in Fig. 4. All the registers, combinational signals and clocks are declared as global variables. All the flip-flop and latch are updated simultaneously at the edge of clocks, Phi1 and Phi2 as shown in Fig 4. The combinational logics are evaluated at the intermediate point of Phi1 and Phi2. Fig. 5 shows the RTL logic blocks stylized as C subroutines. Top module calls all the block subroutines in succession at the two clock edges and two clock levels.

As StreC is not event-driven, special care should be taken to allow signals to flow correctly between modules. Signal Flow Graph(SFG), which represents the precedence relations and temporal relations, is very useful for correcting many tricky timing problems which, although unveiled during the C-level simulation, can later be detected as hardware bugs.

To describe the synchronous circuit operation in C is not a simple job, it requires cautious efforts such as static signal ordering and asynchronous loop removal. But most of the design time is consumed by simulation rather than the description of design itself.

The speed advantage of C over the general-purpose HDL is liken to the assembly programming over the compiler-assisted high level language programming. Even though the hardware description using C is difficult than the well-formalized VHDL or Verilog in many aspects, its simulation speed can be very fast than the general-purpose commercial simulation engines. StreC was mainly used to design and debug the micro-architecture of K486. The RTL model runs program at 1400 cycle/sec as shown in Table II.

```
main()
{
    input_sim_condition(IPC,SAVE,TRACE,RESTART,PROFILE);
    if( RESTART) Load_Status();

    while( !simDone ){
        if( SAVE ) Save_Status();
        clock++;

        /* phi1 phase */
        P1E_evaluation();
        Update_flipflop();
        P1L_evaluation();

        /* phi2 phase */
        P2E_evaluation();
        Update_flipflop();
        P2L_evaluation();
        if( microcode.sequence ==Instr.End ){
            if(IPC) model_difference_check();
            InstructionCount++;
        }
    }
    report_sim_statistics(PROFILE);
}
```

Fig. 4. Top module of StreC increases the clock counter for each cycle and calls C subroutines in sequence for P1E, P1L, P2E and P2L

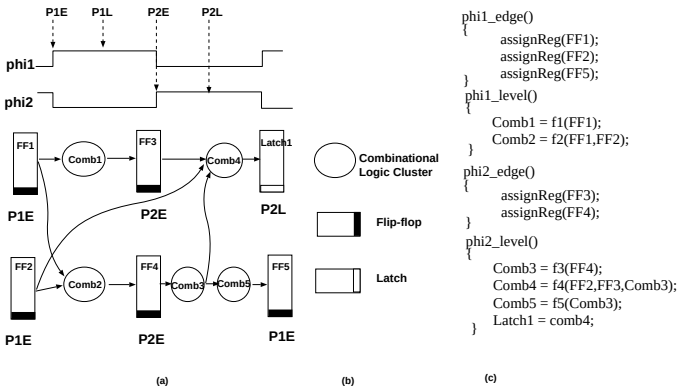

Fig. 5. (a)Signal Flow Graph showing the clock timing of flip-flops(FF's) and latches, (b) symbols for SFG and (c) the corresponding RTL C description, 2-phase clocking scheme was assumed.

D. Consistency Check

In traditional approaches [4, 5], simulation traces of both a reference model and RTL model are dumped. The reference trace captures how the architecturally visible states change as a result of instruction execution. The RTL trace represents the internal flip-flops, and combinational signals as well as state registers, instruction pointer, address, bus value, and flags as a result of executing the same sequence of instructions. After finishing the long simulation, the post-analysis tool compares two trace files. If some inconsistencies were detected, design error was reported. For a long simulation, the trace file size may be enormously larger than several Giga bytes. Moreover, dumping of trace file slows-down the simulation by 5 or 6 times.

As an alternative, we use a dynamic consistency check mechanism using IPC(Interprocess Communications) in UNIX[16] during the co-simulation. It neither requires extra trace files nor degrades the simulation speed.

StreC and MCV(or MCV and Polaris) run in parallel. When StreC completes one instruction execution, StreC sends its results to MCV, while MCV waiting for the results of StreC compares the received results with its own results and then tells StreC whether the results are consistent or not. Simulation stops when the differences are detected. Our experiment has shown that IPC yields a speed degradation of 10 – 20 %, depending on circuit size.

In MCV, all micro-operations are executed in a single cycle. However, because of many advanced implementation features, such as pipeline, cache, delayed handling and buffer, two models may not be identical. In StreC, the micro-operations can be delayed by more than one cycle. For example, the instruction counter, specific register values and memory map may be shifted by one cycle, but this does not mean errors in reality. An intelligence is needed for the simulation engineer to differentiate the real bugs from artifacts.

X window-based micro-architecture tool displays information such as register values, micro-operations and memory content on the screen. The designers eradicate the hardware bugs using both the micro-architecture tool and waveform displayer of RTL trace as shown in Fig. 6.

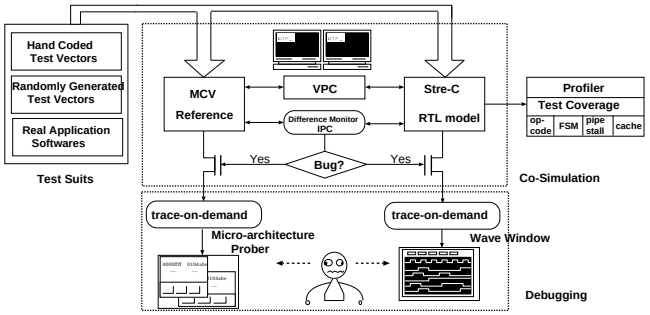

Fig. 6. Dynamic consistency checking between two C models during the co-simulation

cific block in C with a Verilog model as shown in Fig. 7-(a). C and Verilog interface is done by PLI(Programming Language Interface)[14]. In this case, the bug locating may requires large efforts. We use another method called "piggyback"[6, 7]. This is a co-simulation technique. Original C model for BUT(block under test), and Verilog model for BUT run concurrently. Verilog BUT "rode on the back" of the complete C model. The important distinction between substitution and piggybacking is that, in piggybacking, both the StreC and Verilog BUT run simultaneously. Verilog model receives a block external inputs from C model, but the outputs from Verilog model are not feed into C model. The output signals from Verilog model are compared with the output signals from C model. In addition to the external outputs from the BUT, many internal signals are also compared. Because there is a close correlation between C model and Verilog model, bugs in Verilog model can be quickly detected and isolated.

Running the RTL C simultaneously adds little overhead to the Verilog simulation because C is very fast than Verilog. This technique proved to be a very effective way of comparing the two models' consistency and obviated the need to extract and maintain large trace file.

E. Piggyback

Once the RTL C model is verified, it is one-to-one translated into the synthesizable Verilog code. During the translation process, some errors may arouse in Verilog model. There are three candidates to confirm the correct translation. The first one is to assert test vectors into block Verilog model. The only block boundary in/out signals are traced from the RTL C simulation, then the input signals are asserted and outputs are compares with trace file. The drawback of this approach is that the amount of trace is enormously large like as hundreds of mega bytes. The second one is to substitute a spe-

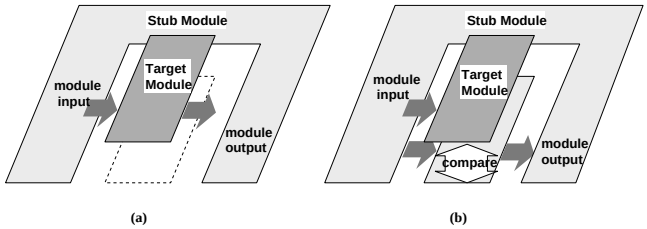

Fig. 7. Consistency Check between C model(StreC) and Verilog model (a) Substitution *vs.* (b) Piggyback

III. Productivity

A. Debugging Cost

Most of bugs found during RTL simulation result from the interaction between modules under various combinations of events. As these bugs are difficult to detect at the block level, designers hurry to integrate all the modules without complete assurance of all blocks being error-free. However, when the test vectors are applied to the fully integrated system-level design, the amount of simulation time soars, significantly degrading the design turnaround. The design complexity of complex microprocessor made it necessary to apply the "divide-and-conquer" method, *i.e.*, "module-by-module" test should precede the "post-integration" debugging. In the debugging of a module called Kunit in K486, about 86 % of bugs were found before the integration or during the integration, while the 14% was fixed during the full-chip system-level simulation after the full integration as shown in Fig. 8.

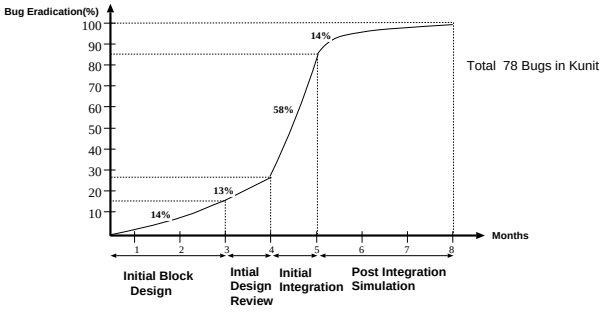

Fig. 8. Bug eradication curve for Kunit in the K486 microprocessor

During the system-level simulation, many bugs were detected at an early phase as shown in Fig. 9. Small percentage *i.e.*, 15 % of bugs remaining to the end of the design process occupies most of simulation time(50% of total debugging time). Sometimes a "careless" design modification may lead to malfunction of another block shown as a deep canyon at 17 million instructions as shown in Fig. 9. Regression test should run in company with the frontier RTL simulation in order to guarantee that proposed bug correction did not corrupt other behaviors. Built-in checkers, which are parts of RTL model, monitors certain illegal state transitions or the violation of protocols. This built-in checkers may slow down the simulation speed, but this performance degradation is compensated by bug-detecting pay-back. At an earlier verification phases, all the built-in checkers are turned on. As the design becomes stabilized, minimum checkers are alive.

B. Test Suites

Good test vectors help find design bugs quickly during the simulation. We deliberately try to stress the design

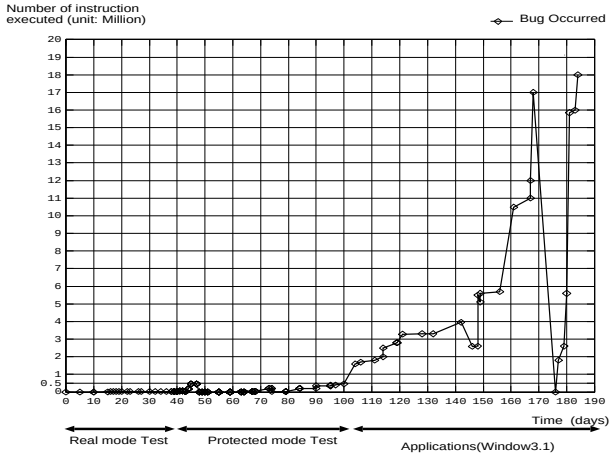

Fig. 9. StreC debugging curve to boot Windows3.1

models to their limit. In our case, there are three kinds of test suites. The first one is hand-crafted test vector, the second one is very long sequence of instructions generated in a biased random fashion. The last one is real-world application programs including operating systems.

The first hand-crafted codes are the by-product of X-86 instruction behavior discovery program that scrutinizes the real, virtual and protected model behavior of X-86 microprocessors. They are computer-generated vectors with a hand-coded template by architecture design team and test team for several years. The total number of hand-crafted test vector amounts to 500. The permutation, iteration and interleaving of existing instruction sequences into new sequences and many exceptional cases which rarely happens in real application software.

The second test program comes from the random test program generator, called *Pandora* shown in Fig. 10. It focuses on producing long sequences of legal instructions assuming that the random interaction of these instructions will exhaustively cover all the test cases and produce conditions that rarely happens. Now, we plan to develop more intelligent ATPG which generates the high quality test vector which guarantee the 100% path coverage and 100% arc coverage. Given a directed graph of the FSM(Finite State Machine)'s or micro-code, it should generate the test programs that cause the simulation to exercise every arc in the graph with minimal redundancy.

C. Test Coverage and Profiler

The "debugging-and-redesign" is an endless loop, which can be terminated only by the tape-out schedule. The test coverage[4, 5, 6, 7] probably is the single most important measure of the verification quality, while such measures as the volume of test vectors and the rate of decrease of bugs detected are all indirect measures. Random generation of test vectors for the verification of the behavior of op-code cannot guarantee that all the block interface pro-

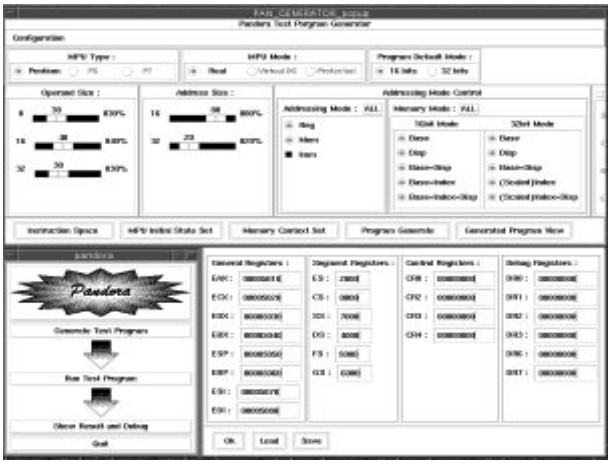

Fig. 10. ATPG(Automatic Test Program Generator) called *Pandora* generates more than 300 test programs with the biasing information of instruction and operand type

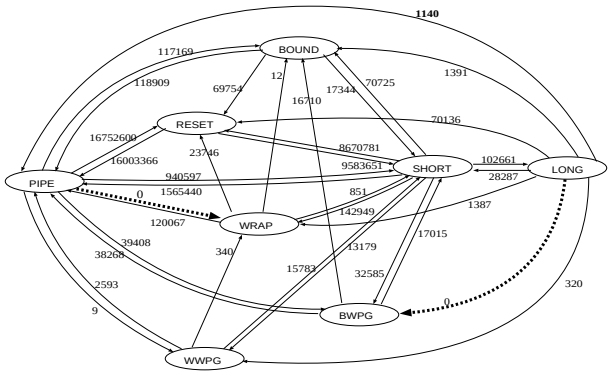

Fig. 11. Number of transitions which occurred among various states in the state transition diagram of an FSM in running DOS a pplication programs(after 20 million instructions). It is shown that some arcs denoted as 'dotted arrow' were not invoked at all. The Profiler monitors the input signals and the states, and compares them with previously states given as table format.

tocols and complex state machine traversals are covered. State-of-the-art microprocessors include complex hardware schemes such as instruction pipelining, branch prediction, superscalar multiple pipes, external bus buffering, multiprocessor cache, and many exceptional cases. Enumerating all the test combinations of various situations, signal paths, and FSM transitions is nearly impossible.

Therefore, reports on the coverage statistics are necessary to determine what percentage of events were covered and what events are to be covered. *Profiler* gives test coverage metrics such as instruction coverage, micro-operation mix, FSM transition coverage, pipeline stall event coverage, and interface protocol coverage. For example, Fig. 11 shows the FSM for a controller of segment unit in the K486, where some arcs are never activated even after the execution of 20 million instructions. These uncovered arcs might be responsible for some vicious bugs which may be captured at the final verification phase or even too late!

These coverage metrics are used subsequently to improve the quality of the test vector set, and gives the designers a feeling for the overall effectiveness of test vector set. Without meaningful test coverage metric, all simulation time is wasted by testing cases that are no longer needed to be tested, while some cases are never excited.

Profiler also reports the performance statistics related to the utilization of resources such as cache access, cache hit/miss ratio, buffer, and bus traffics. Some design error leads to performance degradation without destroying the functionality. This kind of error is called a performance bug, which is difficult to detect. For example, we monitored the cache hit rate during the simulation. After certain situation, it went below 50% for a long time. We discovered later that there was a wrong description in the cache controller, but it did not cause any behavioral problem in booting operating system. Fig. 12 shows one example of performance profile.

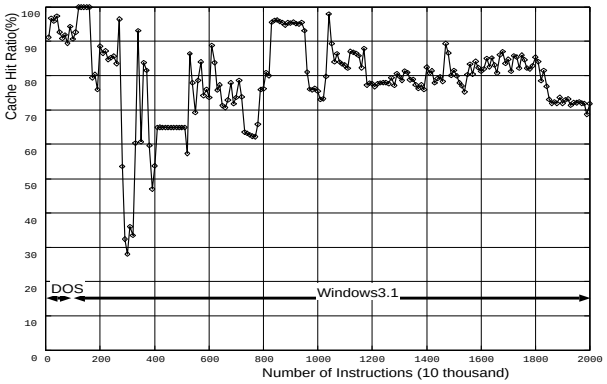

Fig. 12. Behavior of Cache hit ratio *vs.* the number of instructions executed during Windows3.1 booting

D. Restartability

Traditional simulation has an important weak point. Designers usually do not dump the signal trace in the first simulation because it is impossible to know where the error should occur beforehand, and the signal trace overburdens the simulation speed by 5–6 times. Therefore, if an error is detected, designers simulate once more from the first instruction to the bug point to dump the signal trace within the small time interval as shown in Fig. 13. After the debugging, designers modify the model and re-simulate from the first instruction. This has been a tedious but unavoidable process in the traditional simulator. In our experience, the simulation time is as much as 15 times that of the debugging itself in a traditional simulator for the microprocessor level design.

The key point is to save this redundant simulation time

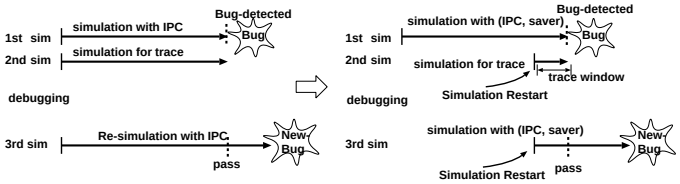

Fig. 13. Reduction of simulation time by the save-and-restart feature of StreC

by providing *restartability*. StreC saves internal states at the completion of every `K` instruction periodically. This is different from the trace dump. Only the internal states such as flip-flop signals are saved at a snapshot rather than long time trace for all signals. This makes it possible to restart simulation from arbitrary point by loading the saved snapshot. As most trivial bugs are detected and design becomes stabilized, the minor modifications of design have little effects on the system state. Restartability plays a key role to find more bugs in a shorter time by reducing the redundant simulation. Using the restartability feature, the total simulation time is minimized to 30% of the traditional simulation approach without restartability.

IV. Result

We applied the proposed functional verification methodology to the K486, which is an Intel i486^{TM}-compatible microprocessor being developed at KAIST. K486 microprocessor consists of pipelined 32-bit integer unit, 64-bit floating point unit and a 8 K-byte cache.

Most of datapath and control logic blocks are built from the cell-based 0.8μm CMOS library, while only area and time critical blocks, such as clock, cache, TLB, shifter, adder, and fast comparator are designed by full-custom layout. Total 1.25 million transistors are integrated in 1.6 x 1.6 cm^2 area at 0.8μm DLM CMOS process. A target working frequency is 60 MHz.

In our K486 project, there were limited number of designers within the limited schedule as shown in Fig. 14. One designer wrote the instruction level behavior model, one wrote the micro-operation level model, one wrote the system board model, and four designers wrote the RTL C model. But using an efficient verification methodology, total several billion cycles are simulated on the RTL C model until the tape-out. We were able to successfully boot MS-DOS and Windows3.1 on the StreC as shown in Fig. 15.

Table II shows the simulation time needed to boot various operating systems and compares the simulation speed between C and Verilog description of K486. Enormous speed advantage of StreC over event-driven simulator comes from the cycle-based logic evaluation. In the cycle-based simulator, the sequence of logic evaluation is

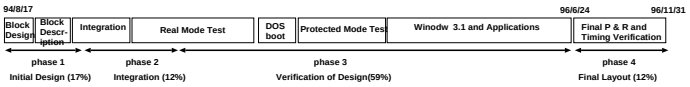

Fig. 14. K486 design milestone

Fig. 15. 20 million instructions are executed to boot Windows3.1 at StreC for 48 hours

determined completely in the static fashion during the compile time and the redundant signal transitions are not evaluated like as LCC(Levelized Compiled-Code) simulator. This gives no expensive overhead of event scheduling.

TABLE II
Comparison of simulation speed of each models for booting DOS(460,000) and Windows3.1(20,000,00 0 instructions) on Sparc20 (CPS: Cycles Per Second)

Model	execution speed(CPS)	execution time DOS	execution time Windows3.1
Polaris	210 KHz	15 secs	20 mins
MCV	50 KHz	1 mins	50 mins
StreC	1.4 KHz	2 hours	2 days
Verilog RTL	10 Hz	10 days	280 days
Verilog gate (with Zycad)	50 Hz	2 days	56 days

V. Conclusion

A functional verification methodology for the compatible microprocessor was proposed in the paper. The verification is focused on fast simulation to remove logical errors at the early design stages. This methodology was proven to be adequate for most microprocessor designs especially in CISC microprocessor such as K486. The hardware description based on C language is more efficient in terms of simulation speed over existing HDL simulator as shown in Table II. Most of the design errors can be identified through the simulation based on C. We were able to boot real-world operating systems and many application programs. The test coverage measure and restartability concept were also instrumental in minimizing the verification cost.

References

[1] A.L.Sangiovanni-Vincentelli, et.al., "Verificaiton of Electroic Systems",in *Proc. DAC, 1996*, pp.106-111

[2] Gopi Ganapathy, et.al., "Hardware Emulation for Functional Verification of K5",in *Proc. DAC*, 1996, pp.315-318

[3] Lawrence Yang, et.al., "System Design Methodology of UltraSPARC-I",in *Proc. DAC*, 1995, pp 7-12

[4] Anoosh Hosseini, et.al., "Code Generation and Analysis for the Functional Verification of Microprocessors",in *Proc. DAC*, 1996, pp.305-310 , 1996

[5] Michael Kantrowitz, et.al., "I'm Done Simulating; Now What? Verification Coverage Analysis and Correctness Checking of the DECchip 21164 Alpha microprocessor",in *Proc. DAC*, 1996, pp.325-330 , 1996

[6] Richard A. Lethin, et.al., "MDP Design Tools and Methods", in *Proc. ICCD*, 1992, pp424-435

[7] Walker Anderson, "Logical Verification of the NVAX CPU Chip Design", in *Proc. ICCD*, 1992, pp306-309

[8] "The SpeedSim/3 : Software Simulator", SpeedSim Inc., version 2.0, 1995

[9] Steven P. Miller and Mandayam Srivas. "Formal Verification of the AAMP5 microprocessor: A case study in the industrial use of formal methods," *WIF '95: Workshop on Industrial Strength Formal Specification Techniques* , pp. 2–16, Boca Raton, FL, 1995, IEEE Computer Society.

[10] Mandayam K. Srivas and Steven P. Miller. "Applying Formal Verification to a Commercial Microprocessor," *CHDL '95*, pp. , 1995.

[11] Toru Shonai and Tsuguo Shimizu. "Formal Verification of Pipelined and Superscalar Processors," *CHDL '95*, pp. 513–518, 1995.

[12] W.J. Cullyer. "Implementing Safety-Critical Systems: The VIPER Microprocessor," *VLSI Specification, Verification and Synthesis*, pp. 1–25, 1988, Kluwer Academic Publishers, Boston.

[13] J.P. Bowen and M.G. Hinchey. "Seven More Myths of Formal Methods," University of Cambridge Computer Laboratory Technical Report 357, 12pp, January 1995.

[14] "Verilog-XL Reference Manual", Cadence Design System Inc., version 1.6, 1991

[15] "ZyCAD XPlus Logic Simulation", Zycad Corporation 1994

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