Proceedings of IEEE Asia Pacific Conference on Circuits and Systems '96
November 18 ~21, 1996, Seoul, Korea

Design Verification of Complex Microprocessors

Joonseo Yim, Changjae Park, Wooseung Yang, Hunseung Oh, Hoon Choi,
Seungjong Lee, Nara Won, In-Cheol Park and Chong-Min Kyung
Department of Electrical Engineering
KAIST,
373-1 Kusong-Dong, Yusong-Gu, 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 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 these C models in different representation levels, *i.e., Polaris, MCV(Micro-Code Verifier)* and *StreC*. An instruction behavioral level C model(*Polaris*) is verified using the slowed-down PC. In the implementation of micro-architecture, a micro-operational level model(*MCV*) and RTL model(*StreC*), both written in C, are co-simulated with consistency checking(*IPC*) between these two models. The simulation speed of C models makes it possible to test the "real-world" application programs on the C model with a software board model(*VPC*). To increase the confidence level of verifications, *Profiler* reports the verification coverage of the test program, which is fed vack to the automatic test program generator(*Pandora*). *Restartability* feature also helps significantly reduce the total simulation time. Using the proposed verification methodology, we designed and verified the K486, 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 the clock speed of several hundred MHz. 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 which saves 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. The cost of emulation hardware is very high and requires that the gate level design be already finished. Therefore, it requires large turnaround penalty to fix gate-level bugs. 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[9]. On the other hand, the general purpose Verilog simulator is much slower than the custom-tailored simulation using C language. Although hardware accelerators[10] yield significant speed-up for the gate-level design, they do not provide fundamental solution for the RTL or even for the behavior level design. In this paper we suggest a low-cost simulation method based on three *RTL C models*, *i.e.*, behavioral level, micro-operational level and RTL, and also the overall verification methodology using slowed-down PC, software system modeling, and IPC(Inter Process Communication).

The proposed verification methodology is applied to the K486, which is Intel 80486 compatible micro-

processor. This paper is organized as follows. A proposed functional verification methodology is described in section 2. Section 3 deals with the productivity issue such as test program generation, test coverage analysis and restartability feature. Finally section 4 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 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

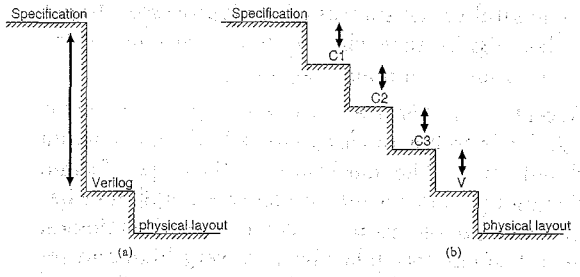

Fig. 1. (a) Traditional *vs.* (b) proposed design flow for microprocessors

In our simulation-based approaches for microprocessor design, RTL design using C language is co-simulated with a reference model, *i.e.*, a behavioral level model or micro-operation level model. The consistency is confirmed by interprocess communications(IPC) in UNIX[11]. Through the whole design procedure of K486 microprocessor, we have used clean-room and top-down approach for the design and verification as shown in Fig. 2.

B. Instruction Behavior Verification

Once the knowledge on the behavior of the instruction set is specified, the instruction set simulator, called *Polaris* is built. As Table I shows, *Polaris*

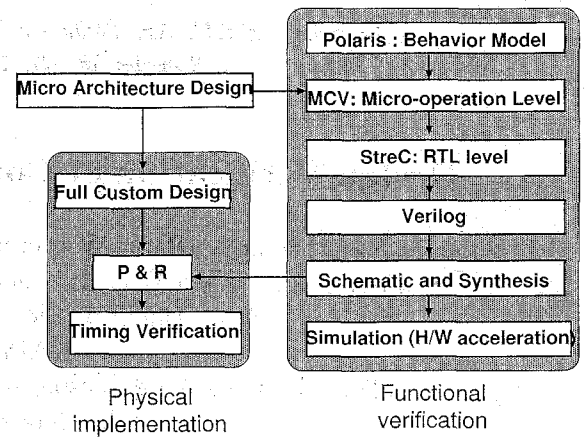

Fig. 2. K486 Design flow

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 runs at over 100 times the speed of Verilog RTL model. This execution speed of Polaris makes it possible to run the "real-world" programs consisting of several billion instructions on software model. This is impossible with commercial cycle-based simulator or gate-level HDL simulation even with hardware acceleration.

To verify the instruction level compatibility of Polaris, the slowed-down PC without CPU called *CMV(C Model Verifier)* is used as shown in Fig. 3. The host computer emulates the instructions through Polaris and the slowed-down PC is used as PC mother board and peripherals. The slowed-down PC and host computer are connected through the interface board which contains several FPGA chips to emulate the bus interface unit of the microprocessor with the buffer circuit which receives and sends messages from and to the host computer.

The message contains information on the bus cycle which should be performed in microprocessor through input/output pads. There is a minimum clock frequency, f_{min}, below which the stable operation of the slowed-down PC is not possible. If f_{min} is less than the frequency of the Polaris, then the interface circuit can be very simple, otherwise a complex circuit is required in order to provide a buffering and synchronization mechanism between slowed-down PC and host computer.

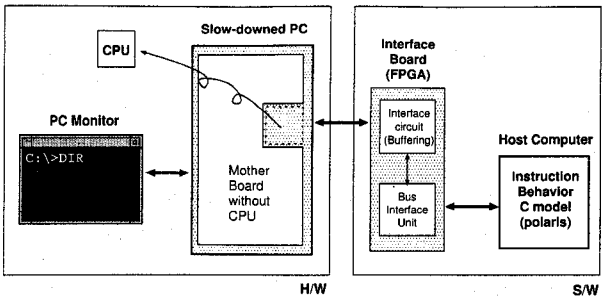

Fig. 3. H/W and S/W co-simulation environment called CMV for the verification of C model

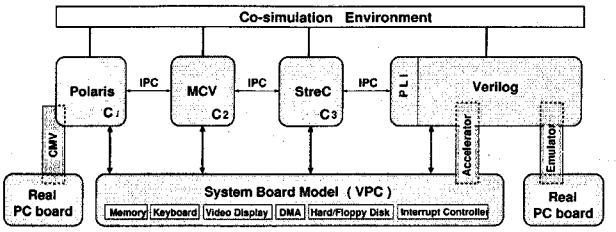

Fig. 4. Co-simulation environment for different design models(*i.e.*, Polaris, MCV, StreC and Verilog) with VPC(Virtual PC). IPC dynamically checks the consistency between models during the simulation

C. Reference Model

Once the instruction behavior of Polaris is verified using CMV, the Polaris becomes a golden-reference model for further detailed design stages. But for the micro-architecture level or RTL, the simulation speed degradation is inevitable. Then the slowed-down PC can no longer be used for slow simulation model less than 100 KHz. Therefore a software model of PC environment is needed for the micro-architecture level and RTL simulation.

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, such as MS-DOS, Window 3.1, Linux, Window95, and application programs in design model, a software model of system board called *VPC(Virtual PC)* is developed. The PC environment was modeled in UNIX workstation using X window system and consists of memory system, hard/floppy disk, interrupt controller, video display, timer and so on as shown in Fig. 4.

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)* is a C model describing the micro-operation level behavior. Neither Polaris nor MCV exactly matches the timing details as obtained via RTL model. However, the speed advantage of Polaris and MCV makes them to be used as "golden" reference model of RTL micro-architecture design.

D. 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

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

model called *StreC* accurately describes the cycle-by-cycle synchronous logic behavior as shown in Fig. 5. All the registers, combinational signals and clocks are declared as global variables. All the signals are categorized into three types : flip-flop, latch and combinational signal. All the flip-flops are updated simultaneously at the edge of clocks, P1E or P2E. The combinational logics are evaluated at the middle point of clock phase P1L and P2L, while the latch is evaluated only one of transparent period, P1L or P2L. Top module calls all the subroutines for each block 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 general-purpose HDL is liken to the assembly programming over

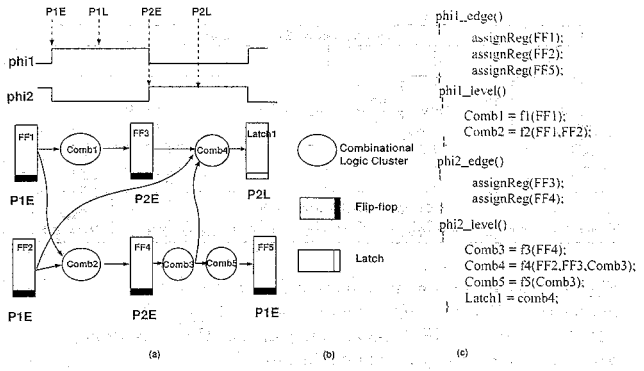

Fig. 5. (a)Signal Flow Graph showing the clock timing of flip-flops(FF's) and latches, (b) symbols for SFG and (c) the corresponding RTL C description, 2-phase clocking scheme was assumed. (In cycle-based simulation, phi1 is simply assumed to be the complement of phi2)

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.

E. Gate-Level Verilog Simulation with hardware accelerator

A hardware simulation accelerator such as Zycad's Paradigm XP series[10] is applicable for the validation of the gate-level design by executing complex test programs and application programs. In our CISC design, a Paradigm XP-2000 accelerator was incorporated with Verilog simulator though VXI software[10] in order to speed up simulation. The system configuration is shown in Fig. 6. We were able to boot MS-DOS in less than 48 hours' simulation using the accelerator for the case of HK386, an Intel 80386 compatible microprocessor.

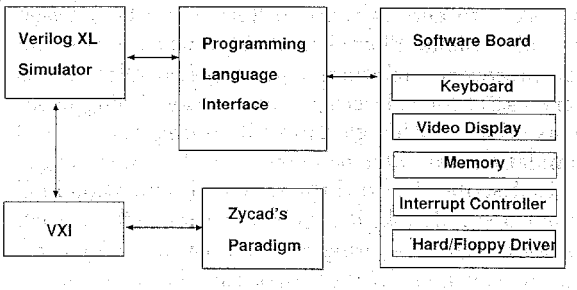

Fig. 6. Gate-level Verilog simulation with Zycad acceleration

F. Consistency Check

In traditional approaches [4, 5], simulation traces of both a reference model and RTL model are dumped. 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 slow-downs the simulation speed by 5 or 6 times. As an alternative, we use a dynamic consistency check mechanism using IPC(Interprocess Communications) in UNIX[11] 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, in StreC the micro-operations can be delayed by more than one cycle due to the pipeline stall and the external interrupt handling delay. Because of many advanced implementation features, such as pipeline, cache, delayed handling and buffer, two models may not be identical. 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 bug from artifacts.

X window-based micro-architecture probing tool displays information such as register values, microcodes and memory content on the screen as shown in Fig. 7. The designers eradicate the hardware bugs using both the micro-architecture tool and waveform displayer of RTL trace.

III. Productivity

A. Debugging Cost

During the system-level simulation, many bugs are detected at an early phase as shown in Fig. 8. Only 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). When the test programs are applied to the fully integrated system-level design, the amount of simulation time soars, significantly degrading the design turnaround. Therefore the design complexity of complex microprocessor made it necessary to apply the "divide-and-conquer"

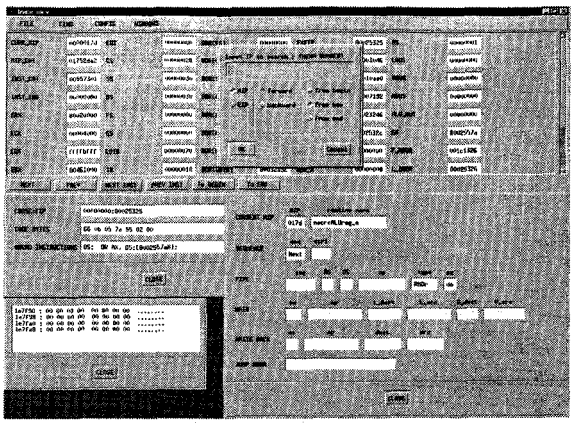

Fig. 7. Micro-architecture probing tool : Debugging information such as instruction count, register value, memory map, op-code, micro-code are shown with forward/backward trace capability.

method, *i.e.*, "module-by-module" test should precede the "post-integration" debugging. It is very important that basic block tests are enhanced in the earlier design phase to shorten the total verification 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. 8. Regression tests should run in company with the frontier RTL simulations in order to guarantee that proposed bug correction did not corrupt other behaviors.

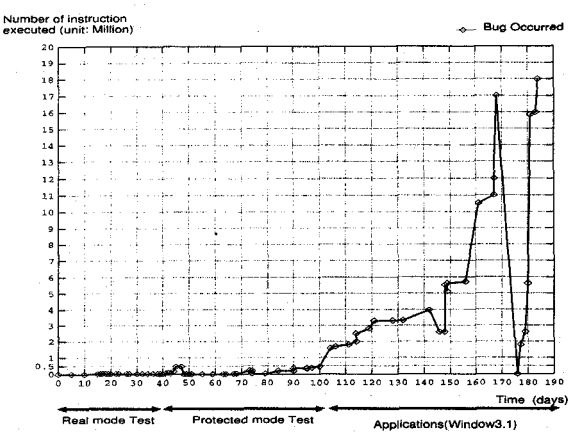

Fig. 8. Bug eradication curve to boot Windows3.1 on StreC

B. Test Suites

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

The second test program comes from the random test program generator, called *Pandora* shown in Fig. 9. 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's or micro-code, it should generate the test programs that cause the simulation to exercise every arc in the graph with minimal redundancy.

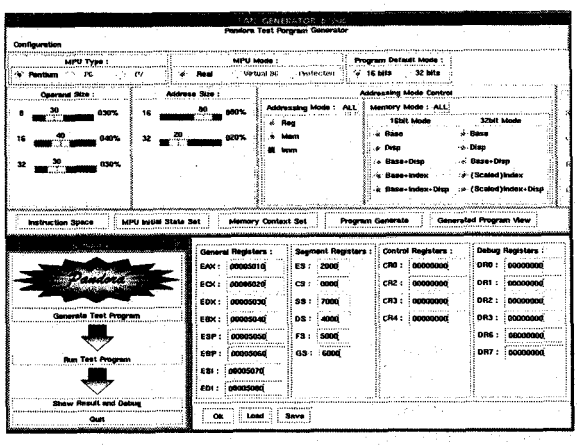

Fig. 9. ATPG(Automatic Test Program Generator) called *Pandora* generates more than 300 test programs with the biasing information of instruction and operand type

C. Test Coverage and Profiler

This "debugging-and-resimulation" forms basically an endless loop in the microprocessor design. The test coverage[4, 5, 6, 7] probably is the single most important measure of the verification quality.

Just randomly generated test vectors for the verification of the behavior of op-code cannot guarantee the coverage of all the block interface protocols and complex state machine traversals. 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 impossible.

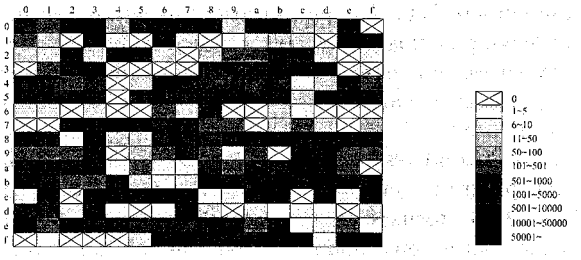

Fig. 10. Number of occurrences of each of 256 instructions in running DOS applications. It is shown that some instructions denoted by a 'cross' were not invoked at all. (after 2 million instructions)

In our simulation environment, *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. 10 shows the number of occurrences of each of 256 instructions after the execution of some "real-world" program consisting of 2 million instructions. It is shown that a significant number of instructions were never tested. These uncovered op-codes or uncovered arcs in FSM might be responsible for some vicious bugs which may be captured at the final verification phase or even too late!

The test 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 as shown in Fig. 11. Without meaningful test coverage metrics, all simulation time is wasted by testing cases that are no longer needed to be tested, while some cases are never excited.

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 occurr 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. 12.

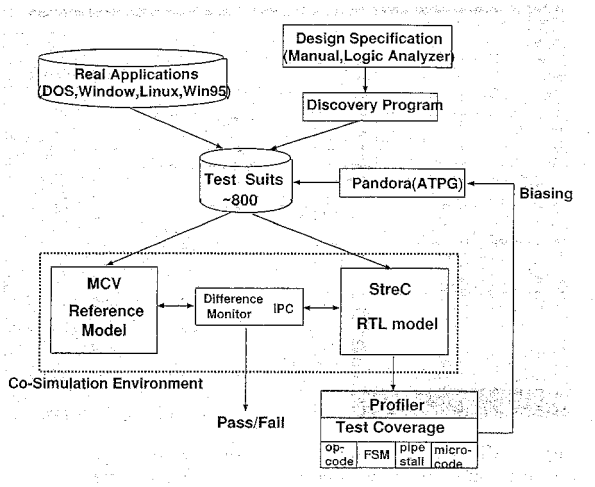

Fig. 11. Test coverage metrics are fed back to the automatic test program generator for more complete test.

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 debugging.

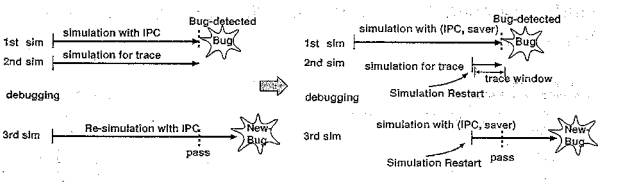

Fig. 12. Reduction of simulation time by the save-and-restart feature of StreC

The key point is to save this redundant simulation time 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 developed at KAIST. K486 microprocessor consists of 32-bit integer unit, 64-bit floating point unit and a 8 K-byte cache as shown in Fig. 13.

Most of datapath and control logic blocks are built from 0.8μm CMOS library, while area and time-critical blocks, such as clock, cache, TLB, shifter and adder 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.

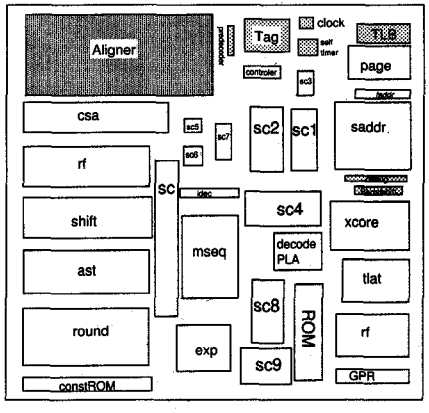

Fig. 13. Floorplan of the K486 microprocessor

In our K486 project, there were limited number of designers within the very 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 only 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 Windows-3.1 on the StreC as shown in Fig. 15.

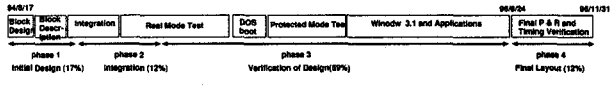

Fig. 14. K486 design milestone

Table II shows 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 and SpeedSim over event-driven simulator comes from the cycle-based logic evaluation. In the cycle-based simulator, the sequence of logic evaluation is determined completely in the static fashion during the compile time and the redundant signal transitions are not evaluated in LCC(levelized compiled-code) simulator. This gives no expensive overhead of event scheduling.

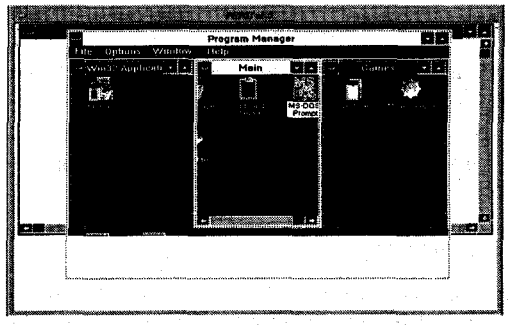

Fig. 15. Screen image showing the successful booting of Windows 3.1 using StreC, which took 48 hours running about 20 million instructions.

TABLE II
Comparison of simulation speed of each models for booting DOS(460,000) and Windows3.1(20,000,000 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
SpeedSim/3-Verilog	0.6 KHz	4 hours	4.6 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 complex microprocessor was proposed in the paper. The verification is focused on fast simulation to remove logical errors at the early design stages. The hardware description based on C language was suggested. This methodology was proven to be efficient in terms of simulation speed over existing HDL simulator for the most microprocessor designs especially in CISC microprocessor such as K486. 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 on those C models. 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

[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

[6] Richard A. Lethin, et.al., "MDP Design Tools and Methods", in *Proc. ICCD, 1992*, pp.424-435

[7] Walker Anderson, "Logical Verification of the NVAX CPU Chip Design", in *Proc. ICCD, 1992*, pp.306-309

[8] Douglas Day, "SpeedSim : The leader in Cycle-Based Simulation", 1996

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

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

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