A C-Based RTL Design Verification Methodology for Complex Microprocessor

Joon-Seo Yim, Yoon-Ho Hwang, Chang-Jae Park, Hoon Choi, Woo-Seung Yang,
Hun-Seung Oh, In-Cheol Park and Chong-Min Kyung
Department of Electrical Engineering
KAIST, Taejon, 305-701, Korea

Abstract

As the complexity of high-performance microprocessor increases, functional verification becomes more and more difficult and RTL simulation emerges as the bottleneck of the design cycle. In this paper, we suggest C languege-based design and verification methodology to enhance the simulation speed instead of the conventional HDL-based methodologies. RTL C model(*StreC*) describes the cycle-based behaviors of synchronous circuits and is followed by model refining and optimization using LifeTime Analyzer(*LTA*) and *Cleaner*. The simulation speed of cycle-based C model makes it possible to test the RTL design with the "real-world" application programs in the order-of-magnitude faster speed than the commercial event-driven simulators. Using the proposed functional verification methodology, HK486, an intel 80486 - compatible microprocessor was successfully designed and verified.

1 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 a clock speed more than 300MHz . This astounding design complexity has resulted in the verification challenge of microprocessor both in academia and industry[1, 2, 3]. The hardware emulation[4], hardware acceleration[5], formal verification[1] and cycle-based simulation[6] have become the state-of-the-art verification methodologies. The cost of emulation hardware is very expensive and it requires that the gate level design is already finished. Therefore, it is generally used for the purpose of final verification before tape-out rather than for the early design phase. Formal verification method has been used successfully to verify a wide variety of moderate-sized hardware designs[7]. 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[8].

The hardware description language(HDL) such as VHDL and Verilog is a convenient method to describe a hardware and becomes a good bridge between RTL(Register Transfer Level) description and logic synthesis. But most of the design time is consumed by simulation rather than the description itself. A cycle-based simulation[6] and compiled-code simulator[9] shows a clear simulation performance advantage over an event-driven simulator[10]. But, the general purpose Verilog simulator is too slow to be used as a system-level simulation.

In the description of the RTL synchronous circuit, we use "C" language rather than Verilog-HDL. Therefore, there is no need to translate Verilog model into intermediate C code unlike cycle-based or compiled-code simulator. On the other hand, in the description of the hardware using C, designers should take extreme care in considering static signal scheduling and timing issues such as inappropriate usage of flip-flops or latches, asynchronous, combinational loops, and other potential timing bugs, which are not easily discovered in cycle-based simulation. To eradicate the hardware-unimplementable descriptions, we devise a "LTA(LifeTime Analyzer)", and "Cleaner" which refines and optimizes the C model based on SFG(signal flow graph) of the RTL design. This approach removes the discrepancy between C model and real hardware, and results in significant simulation speed-up.

We will show the performance of our C-based simulator compared to other HDL simulators using HK486, an intel 80486-compatible microprocessor as an example.

This paper is organized as follows. We describe our design verification methodology in section 2. A proposed RTL C-based synchronous circuit design and related problems are shown in section 3. Section 4 deals with the productivity issues in simulation. Finally, the performance result of simulator is shown in section 5.

2 Design Verification Methodology

2.1 Verification Flow

In our top-down design flow of microprocessors, design is gradually refined and realized from the specification to the physical implementation. According to the description level, C_1, C_2 and C_3 represent the model of microprocessor written in C for behavioral, micro-operational, and RTL, respectively [11].

Table 1: Description of CPU models in various levels

level	model	description	features
C_1	Polaris	macro instruction behavior	register
C_2	MCV	micro-operation	register internal bus
C_3	StreC	clock cycle-based RTL Phi1 edge, Phi1 level Phi2 edge, Phi2 level	register(FF,latch) internal bus combinational pipeline
V_1	Verilog RTL	clock and event-based RTL	register(FF,latch) internal bus combinational pipeline timing

DAC 97, Anheim, California
©1997 ACM 0-89791-920-3/97/06..$3.50

As Table 1 shows, *Polaris* describes the exact behavior of x86 instruction set without the detailed architecture such as pipelining, superscalar instruction pairing, multiple functional units and cache. Representing a higher abstraction level in Polaris allows us to produce a reference model with very few bugs and to execute at a speed more than 150 times than that of the RTL C model. The speed of Polaris makes it possible to run the test suites 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.

For the CISC microprocessors, one macro instruction is subdivided into a number of micro-operations(usually called micro-code) and consumes multiple clock cycles. *MCV(Micro Code Verifier)* is a CPU model in micro-operation level, which complements the gap between the higher level reference model and clock cycle-based RTL model.

The RTL C model, called *StreC* which is chosen instead of Verilog-HDL for its speed performance, accurately describes the cycle-by-cycle synchronous logic behavior. Basically it is similar to the cycle-based simulator. But there is no need to translate Verilog model into intermediate C code. Description of the design itself is a construction of the simulator.

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 more difficult than the well-formalized VHDL or Verilog in many aspects, its simulation speed is very fast than the general-purpose commercial simulators and compensates the difficulties of C modeling.

2.2 Inter-model Consistency Check

An important problem in the top-down design flow is to maintain the consistency between the consecutive design levels. In traditional approaches [12, 13, 14, 15], traces of both a reference model and RTL model are dumped during the simulation. After finishing the long simulation, the post-analysis tool compares two trace files. Inconsistencies in registers, flags and memory map are reported for debugging. Usually for a long simulation, trace file size may be larger than several Giga bytes. Moreover, dumping of signal trace overburdens the simulation speed by 5 or 6 times. We use a dynamic inter-model consistency check using IPC(Interprocess Communications) mechanism in UNIX during the co-simulation of StreC(RTL model) and MCV (reference model) rather than the static comparison after the simulation is finished. It neither requires large trace files nor degrades the simulation speed.

3 StreC : RTL C model

3.1 Cycle-based Synchronous Circuit Design

StreC accurately describes the cycle-by-cycle synchronous logic behavior taking care of signal precedence relation as shown in Fig. 1. The signal types are classified into three according to the timing when the signal is evaluated and updated with clocking: WIRE for combinational signals, REG for edge-triggered flip-flops, and LATCH for level-transparent latches. Combinational signals are immediately evaluated for varying inputs. Flip-flops are updated only at the clock edge and are preserved until the next clock edge, even though there are input variations. Latch has special characteristics: transparent duration and latched duration. During the transparent duration, the latch output follows the input, otherwise the output preserves the value of the input at the time of the most recent latching edge.

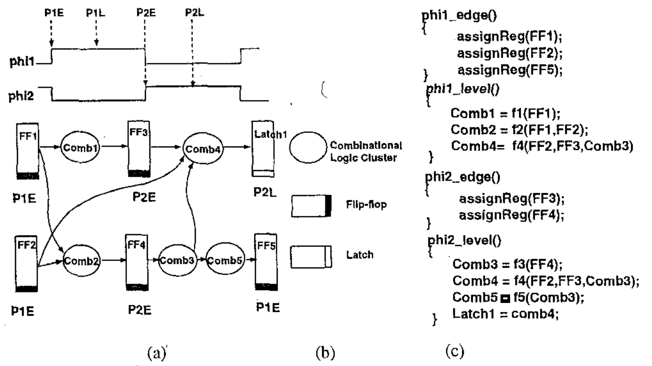


Figure 1: In 2-phase clocking scheme, phi1 is simply assumed to be the complement of phi2 in cycle-based simulation : (a)Signal Flow Graph(SFG) showing the signal relations, (b) Symbols for SFG and (c) the corresponding RTL C description.

We use a symmetric two-phase clocking(phi1 and phi2). In StreC, all the flip-flops are updated "simultaneously" at the clock edge(called P1E or P2E) without any mutual dependency using special function, **assignReg**, while the combinational logics are evaluated in the middle of clock cycle(called P1L and P2L), and the latches are evaluated during only one of clock level *i.e.*, P1L or P2L. According to our experiences, the classification of evaluation time into the edge and the level, makes the debugging simpler than the commercial cycle-based simulator which evaluates all the signals only at clock edges[6].

Fig. 2 shows a top module of StreC, **while** loop increases a `clock` counter and calls all the subroutines for each unit in succession at the two clock edges and two clock levels. All the subroutines for the units in HK486 are shown in Fig. 5. Hardware interrupts and `IPC` differences are checked at every cycle. The `SAVE`, `TRACE`, `RESTART`, and `PROFILE` feature of StreC will be explained in the next section.

In StreC, circuit evaluation is implemented as a function call, therefore, even though there are changes for input signals, the output value remains fixed until the function call. Since the flip-flops are updated simultaneously only at the clock edge, the sequence of function calls for the flip-flops at P1E and P2E does not matter. But special care should be taken to allow the events to flow correctly between sub-functions to maintain the precedence relation of the combinational logic circuits because the StreC is not an event-driven simulator. Signal Flow Graph(SFG), which represents the precedence and temporal relations between signals in Fig. 1, is very useful to determine the sequence of the combinational and latch signals. SFG is also very useful for correcting many tricky timing problems which, although unveiled during the cycle-based simulation, can later be detected as hardware bugs.

The sequence of description for the combinational signals in StreC is similar to the levelizing mechanism in the levelized compiled-code(LCC) simulator[16], *i.e.*,

1. Assign $level(s) = 0$ to each primary input signal s
2. Assign $level(s_i)$ to each other signal s_i such that
   $\mathrm{level}(s_i) = 1 + \mathrm{MAX}\{\mathrm{level}(s_j)\}$ for all $s_j \in \mathrm{Fanin}(s_i)$

A sequence of signals is determined such that no values are referred to before they have been calculated. The sequence of signals within the same level is arbitrary because no inter-dependency exists between them. Some latches and combinational signals can be multiply scheduled because of inter-block cross referencing. This will be explained later in

```
StreC_main(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_evaluate();
            Update_FlipFlop();
            P2L_evaluation();

            if( microcode == END ){
                if( IPC) IPC_error_check();
                Instruction_count ++;
            }
        }
}
```

Figure 2: Top module of StreC increases the clock counter for each clock cycle and calls C subroutines for P1E, P1L, P2E and P2L in sequence.

```
main();
{       input(C);
        /* LTA */
        sfg_generation();
        signal_type_identification();
        lifetime_analysis();

        /* Cleaner */
        StreC_description_rule_check();
        io_timing_check();
        combinational_Latch_levelizing();
        redundant_FF_Latch_removal();
        FF_to_Latch_transformation();
        cycle_stealing_analysis();
        rescheduling();
        output(C*);
}
```

Figure 3: LTA(SFG generation) and Cleaner(refine and optimize) improves the C model

more detail. The sequence of function call is determined in a static fashion after the signal levelizing. This is different from the event-driven simulator, where dynamic scheduling overhead significantly decreases the simulation speed.

3.2 C Model Refining and Optimization

In the modeling of RTL circuit using C, there are several problems to be considered, which are static signal scheduling, combinational or asynchronous loop detection, register identification, cycle stealing, and fan-in/fan-out timing problems which, although unveiled during the cycle-based simulation, can later be detected as hardware bugs.

LifeTime Analyzer(LTA) generates the SFG from the RTL circuit descriptions and identifies the signal type(flip-flop, latch, and combinational signal) and lifetime for each signals, *i.e.*, when the fan-in signals are generated and when the signals are used as fan-out.

Using the lifetime information, *Cleaner* detects the description rule violation, such as combinational logic description at clock edges, precedence relations violation for latches or combinational signals and combinational or asynchronous loops. And *Cleaner* does the further optimization such as transformation from flip-flop to latch, cycle-stealing, and rescheduling. The pseudo code for LTA and Cleaner is shown in Fig. 3.

SFG is the key to both LTA and Cleaner. For datapath blocks, each element in datapath is represented as one vertex in SFG, while for control blocks, one vertex in SFG represents a cluster with many strongly-coupled elements. This clustered SFG also speeds up the circuit analysis and optimization process. Moreover, it provides the designers with the high-level view of circuit structure.

After constructing the refined and optimized model (shown as C^* in Fig. 3), it is one-to-one translated into synthesizable Verilog code by C2V(C-to-Verilog Compiler) and datapath schematics for the following physical implementation and timing simulation. The flow for RTL C model refining, optimization, verification and Verilog code generation is shown in Fig. 4.

To describe the synchronous circuit in C is not a simple job, it requires cautious efforts. But most of the design time is consumed by simulation time rather than the description of design itself.

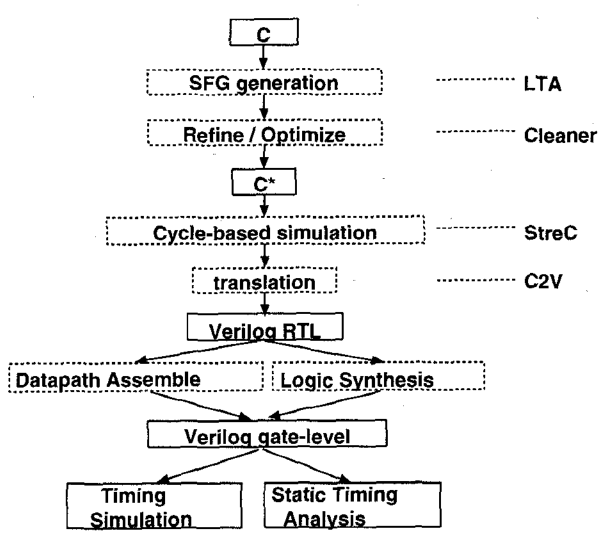


Figure 4: Design flow from RTL C model to synthesizable Verilog code

3.3 Scheduling problem in StreC

Intra-unit scheduling is fully automated with the help of SFG, while the inter-unit scheduling is very complex. Often the long sequence of function calls makes an unintentional combinational loop or asynchronous loop which contains one transparent latch, which is a main trouble maker in the cycle-based simulation.

Some units are scheduled multiple times because of inter-unit precedence relations. As shown in Fig. 5, during P1E and P2E, the sequence among C, D, F, S, K, B, X, T and G unit does not matter, while during P1L and P2L, the sequence among sub-functions is very critical to the exact function evaluation. Note that the X unit is scheduled three times at P1L, *i.e.*, `X_P1L_1`, `X_P1L_2`, `X_P1L_3`. Especially in P2L, the scheduling is more complex. Most units are multiply scheduled: Cunit – three times, while D, F, G, K, and X unit are scheduled twice.

P1E_evaluate	P1L_evaluate	P2E_evaluate	P2L_evaluate
Cunit_P1E	Cunit_P1L	Cunit_P2E	Xunit_P2L_1
Dunit_P1E	Sunit_P1L	Dunit_P2E	Kunit_P2L_1
Funit_P1E	Dunit_P1L	Funit_P2E	Gunit_P2L_1
Sunit_P1E	Funit_P1L	Sunit_P2E	Cunit_P2L_1
Kunit_P1E	Kunit_P1L	Kunit_P2E	Dunit_P2L_1
Bunit_P1E	Bunit_P1L	Bunit_P2E	Cunit_P2L_2
Xunit_P1E	Xunit_P1L_1	Xunit_P2E	Cunit_P2L_3
Tunit_P1E	Gunit_P1L	Tunit_P2E	Funit_P2L_1
Gunit_P1E	Xunit_P1L_2	Gunit_P2E	Dunit_P2L_2
	Tunit_P1L		Sunit_P2L
	Xunit_P1L_3		Funit_P2L_2
			Xunit_P2L_2
			Kunit_P2L_2
			Gunit_P2L_2
			Bunit_P2L

Figure 5: Statically-scheduled sequence of sub-function for sub units in HK486. There is no precedence relation for the function calls in P1E and P2E, while the sequences for P1L and P2L are critical to the exact function evaluation.

In Fig. 6, the sub-function `logic` is scheduled five times in P2L. At first, some signals generated in `logic`, are used at `fsm_cl`. The newly generated signals from `fsm_cl` go to `logic` again, and `logic` makes other signals to be used by `mir`, and so on. This complex schedule is due to the poor partition of the whole circuits. If some of logic blocks are migrated or replicated into other blocks, some unnecessarily complex signal handshaking need not occur. Using careful analysis of the resultant SFG, most complex signal handshaking can be localized.

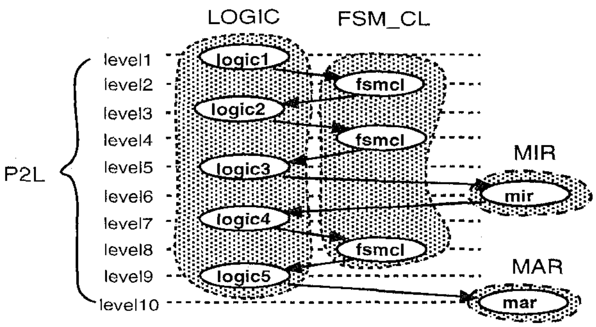


Figure 6: Multiple scheduling of `logic` block in **Cunit** during P2L

3.4 Flip-flop and Latch minimization in StreC

As an example of C model optimization, consider an example **mirco-sequencer** shown in Fig. 7 which contains five latches and three flip-flops. With a careful analysis of SFG using Cleaner, four latches and one flip-flop guarantee the same functionality as the original description as shown in Fig. 8(b). Moreover, this refined model(C^*) both improves simulation speed and clarifies the description for the designers.

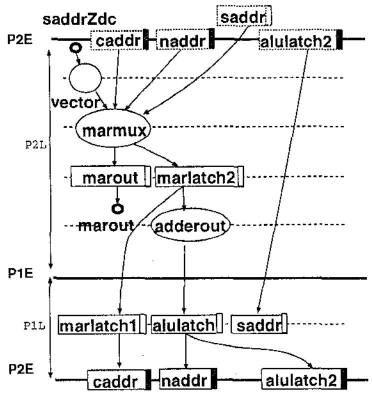
Note the signal `adderout` in Fig. 8, it utilizes a cycle-stealing profit of `marout`. By requiring a signal to arrive early, combinational logic may *borrow* time across the cycle boundaries implied by latches. This ability to borrow time provides more margin in the critical paths. Cycle-stealing analysis is an important feature of Cleaner.

3.5 Combinational Loop in StreC

Latch identification is needed to break the asynchronous loop or combinational loop. During the early stage of functional design, designers do not worry about the use of flip-flop or latch in detail. For example, in the description of `if --- else` statement, if they do not specify `else` statement explicitly, logic synthesis tools consider it as a latch rather than a *don't care* value. In such cases, Cleaner reports a warning message *'combinational loop'* and modifies the description with flip-flops.

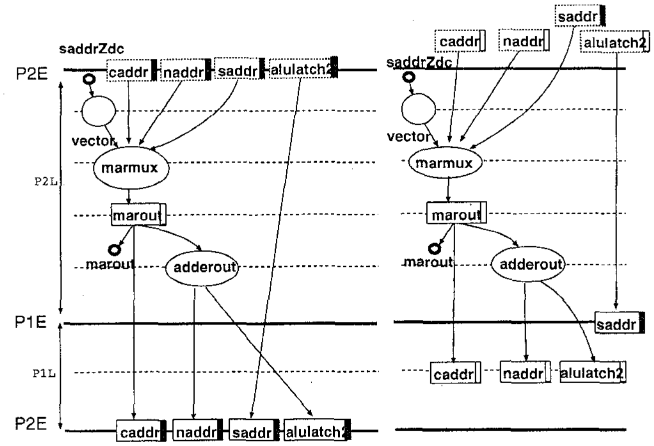
```
P1L()
{   marlatch1 = marlatch2;   /* LATCH */
    alulatch = adderout;
}   if (saddr_cp)  saddr = alulatch2;

P2E()
{   assignReg(&caddr, marlatch1);
    assignReg(&naddr, alulatch);
}   assignReg(&alulatch2, alulatch);

P2L()
{   switch (vector_sel) {
      case 0x00 : vector = 0x00;
      case 0x01 : vector = 0x40
      ....
      case 0x08 : vector = 0x80;
      case 0x10 : vector = saddrZdc;
    } default   : vector = UNKNOWN;
    switch (marmux_sel) {
    case 0x00 : marmux = 0x00000000;
    case 0x01 : marmux = JmpAddr;
    case 0x02 : marmux = saddr;
    case 0x04 : marmux = naddr;
    case 0x08 : marmux = caddr;
    case 0x10 : marmux = vector;
    default   : marmux = UNKNOWN;
    }
    marout = marmux;
    marlatch2 = marmux;   /* LATCH */
    adderout = (marlatch2 + 1) & 0xFFF;
}
```

(a)



(b)

Figure 7: Example **mirco-sequencer**: (a) C-model and (b) the corresponding SFG with redundant latches and flip-flops

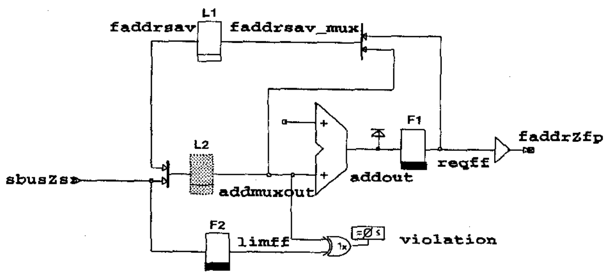


(a) (b)

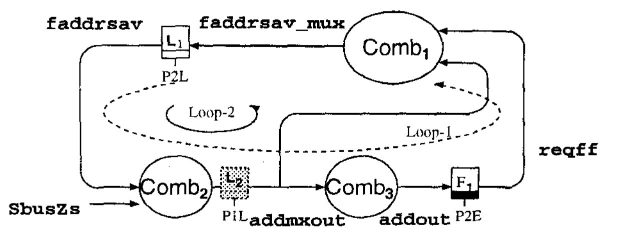
Figure 8: Example **mirco-sequencer**: Resultant SFG after (a) refining and (b) optimizing

Cleaner searches the SFG hierarchically to detect pure combinational loops or asynchronous loops with one single latch. If any inter-block global asynchronous loops are found, then the latch should be transformed into flip-flop or another latch clocked in neighboring clock phase should be inserted inside the loop. In the case of HK486 design, six complex asynchronous loops were found during the signal flow analysis.

Fig. 9 shows the datapath example with an asynchronous loop. If $delay(Comb_1 + Comb_2 + Comb_3) < T_{cycle}$, there is no problem for Loop-1 even without latch L_2. But Loop-2 constructs an asynchronous loop during phi2 clock phase and leads to the malfunction such as racing condition. Cleaner detects this asynchronous loop and proposes some remedies to guarantee the correct operations. In this example, P1L latch L_2 is inserted in Loop-2. In Fig. 9, it can be seen there is no timing problem in Loop-2 with the addition of a new P1L latch, *i.e.*, it does not affect the correct operations of other circuits.



(a) Circuit



(b) SFG

Figure 9: Example **prefetch-dpath** : (a) datapath with asynchronous loop : shadowed latch L_2 was inserted after C model refining to break the asynchronous loop and (b) the corresponding SFG : $Comb_1$, $Comb_2$ and $Comb_3$ represent two multiplexers and one adder, respectively.

4 Productivity Issues for Simulators

4.1 Save-and-Restartability of StreC

If an error is detected, designers then simulate once again from the first instruction to the bug point with the signal trace dumped. After the debugging, designers modify the source code, compile 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.

However, StreC saves the internal states at the completion of every K instructions periodically. This is quite different from the signal dump, *i.e.*, only the snapshots of flip-flop signals are saved rather than long time trace for all signals. This makes it possible to restart simulation from the arbitrary point by loading the saved snapshot. When an error is detected, we rewind the simulation time only by a little and re-simulate for the small time interval which may be enough for debugging instead of re-simulation from the very beginning to dump the signal trace. After debugging, we load the snapshot of CPU states and restart from the "safe" region which was not affected by a bug. The saved snapshot can be modified for the addition of new signals or intentionally to minimize the simulation time.

As most trivial bugs are detected and design becomes stabilized, the minor modifications of design have little effects on the CPU state. Restartability plays a key role to obtain the short turnaround time by reducing the redundant simulation. Using the *"Save-and-restartability"* feature of StreC, the total simulation time is minimized to 30% of the traditional simulation approach without restartability.

4.2 Cost of Debugging

Most of bugs found during RTL simulation result from the interaction between units under the various combinations of events. As these bugs are difficult to detect at the unit level test, designers integrate all units without the assurance that each of them is 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. In the debugging of **Kunit** in the HK486 design, about 86 % of all the 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.

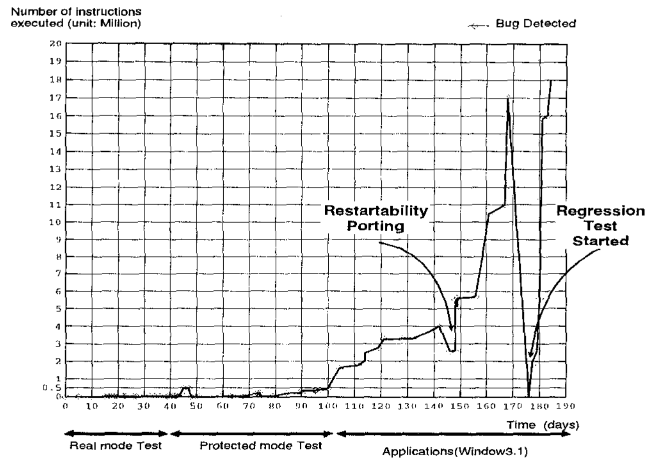


Figure 10: StreC debugging curve with system-level simulation

Fig. 10 shows the debugging curve with large test vectors after the full integration of RTL design. During the system-level simulation, many bugs were detected at an early phase as shown in Fig. 10. Small percentage, *i.e.*, 15 %, of bugs remaining to the end of the design process occupies most of the simulation time(50% of total debugging time). Therefore, it is very important that basic unit tests should be enhanced in the earlier design phase to shorten the total verification time. Sometimes a 'careless' design modification may lead to malfunction of other units shown as a deep 'canyon' at the execution of 17 million instructions as shown in Fig. 10. Regression tests should run in accompany with the frontier simulation in order to guarantee that proposed bug correction did not corrupt the otherwise-correct behavior of other units.

This "debugging-and-simulation" can become an endless loop. The criteria which guarantees the bug-free tape-out with high confidence level may be a "no-bug-detected" report by long test vectors for a specified period of time. But it is not enough. Reports on the coverage statistics of test vectors are necessary to determine what percentage of behaviors were covered and what behaviors are to be covered. The test coverage[11, 12, 13, 14, 15] probably is the most important measure of the design quality with a large volume of test suites. *"Profile"* feature of StreC gives test coverage metrics such as instruction coverage, micro-operation mix, FSM transition counts, pipeline stall count, and interface protocol coverage. These coverage metrics are used subsequently to improve the quality of test vectors and gives the designers a feeling for the overall effectiveness of test strategy. Without meaningful test coverage metric, all simulation time is wasted by testing the cases that are no longer needed to be tested, while some cases are never excited.

5 Result and Discussion

We applied the proposed design and verification methodology to HK486 microprocessor being designed at KAIST which is an intel 80486-compatible microprocessor. HK486 consists of 32-bit pipelined integer unit, 64-bit floating point unit and 8 Kbyte cache.

Most of the control logic and datapath blocks were built from standard-cell and datapath library, while area and time-critical blocks such as cache, TLB(Translation Look-ahead Buffer), shifter, adder and clock generator were designed with full-custom layout. Total 1.25 million transistors were integrated in about 1.7 x 1.7 cm^2 area using 0.8μm DLM CMOS process. A target working clock frequency is 60 MHz.

In our HK486 project, there were very limited number of designers within the limited schedule as shown in Fig. 11. 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 design verification methodology, total several billion cycles were simulated until the tape-out. MS-DOS and Windows-3.1 were successfully booted on the RTL C model as shown in Fig. 12. About five Sparc workstations are used in the design and verification of HK486.

Table 2 shows the simulation time needed to boot operating systems with the comparison of the simulation speeds between various C models and Verilog for the case of HK486. The simulation speed of proposed RTL C model is faster than the commercial event-driven Verilog RTL simulator by about 140 times. Enormous speed advantage of StreC 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. This gives no expensive overhead of event scheduling.

Even though StreC is very fast, there are still rooms for further speed up. Recently, there are approaches[1] to employ the Binary Decision Diagram(BDD) techniques to achieve the speed-up in the cycle-based simulator using the BDD's fast function evaluation feature.

In dealing with large circuits where the complete BDD cannot be built due to the explosion of BDD size, we can construct BDD in terms of internal nodes rather than primary inputs. When we construct SFG for RTL C model, the combinational functions are clustered into sub-circuit whose fan-ins are logically independent. The translation of StreC into BDD-based StreC is now under way to speed up the function evaluation with less memory requirement.

Table 2: Comparison of simulation speed for booting DOS(460,000) and Windows-3.1(20,000,000 instructions) (CPS: Cycles Per Second) : * is the estimated time.

Model	execution speed(CPS)	execution time		Machine
		DOS	Windows3.1	
Polaris	210 KHz	15 secs	20 mins	Sparc-20
MCV	50 KHz	1 mins	50 mins	Sparc-20
StreC	1.4 KHz	2 hours	2 days	Sparc-20
StreC	4.0 KHz	42 mins	18 hours	Ultra-Sparc
VCS RTL	40 Hz	40 hours	40 *days**	Ultra-Sparc
Verilog-XL RTL	5 Hz	14 *days**		Ultra-Sparc

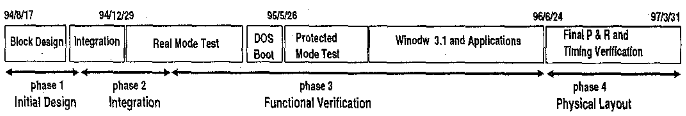


Figure 11: HK486 design milestone

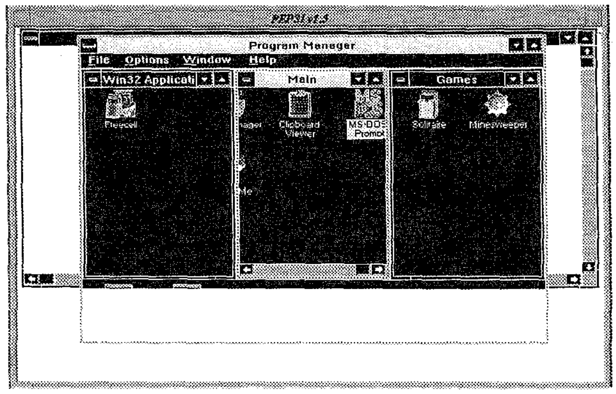


Figure 12: Screen image showing the successful booting of Windows-3.1 using StreC, which took 48 hours running about 20 million instructions on Sparc-20.

6 Conclusion

A C-based RTL design and verification methodology for complex microprocessor is described in this paper. It is focused on fast simulation to remove the logical errors at an early design stage. The cycle-based synchronous circuit description based on C is more efficient in terms of simulation time over the existing HDL simulator. This methodology was proven to be adequate for complex microprocessor such as HK486. 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., "Verification of Electronic Systems", 33rd DAC, pp.106-111 , 1996

[2] J.Monaco et al., "Functional Verification Methodoloogy for the PowerPC 604 Microprocessor", 33rd DAC, pp.319-324 , 1996

[3] V.Popescu et al., "Innovative Verification Strategy Reduces Design Cycle Time For High-End SPARC Processor", 33rd DAC, pp.311-314 , 1996

[4] G. Ganapathy et al., "Hardware Emulation for Functional Verification of K5", 33rd DAC, pp.315-318 , 1996

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

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

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

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

[9] "VCS Reference Manual", Chronologic Simulation, version 2.0, 1993

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

[11] J.S.Yim et al., "Design Verification of Complex Microprocessors", Proc. *ASP-DAC '97*, pp.173–180, 1997

[12] R.A.Lethin et al., "MDP Design Tools and Methods", ICCD, pp.424-435, 1992

[13] W.Anderson, "Logical Verification of the NVAX CPU Chip Design", ICCD, pp.306-309, 1992

[14] A.Hosseini et al., "Code Generation and Analysis for the Functional Verification of Microprocessors", 33rd DAC, pp.305-310 , 1996

[15] M.Kantrowitz et al., "I'm Done Simulating; Now What? Verification Coverage Analysis and Correctness Checking of the DECchip 21164 Alpha microprocessor", 33rd DAC, pp.325-330 , 1996

[16] L.T.Wang et al., "SSIM: A Software Levelized Compiled-Code Simulator", 24th DAC, pp.2–8, 1987