A Two-State Methodology for RTL Logic Simulation

Lionel Bening
Hewlett-Packard Company
P. O. Box 833851
Richardson, TX 75083-3851
+1-972-497-4013
bening@rsn.hp.com

ABSTRACT

This paper describes a two-state methodology for register transfer level (RTL) logic simulation in which the use of the X-state is completely eliminated inside ASIC designs. Examples are presented to show the gross pessimism and optimism that occurs with the X in RTL simulation. Random two-state initialization is offered as a way to detect and diagnose startup problems in RTL simulation. Random two-state initialization (a) is more productive than the X-state in gate-level simulation, and (b) provides better coverage of startup problems than X-state in RTL simulation. Consistent random initialization is applied (a) as a way to duplicate a startup state using a slower diagnosis-oriented simulator after a faster detection-oriented simulator reports the problem, and (b) to verify that the problem is corrected for that startup state after the design change intended to fix the problem. In addition to combining the earlier ideas of two-state simulation, and random initialization with consistent values across simulations, an original technique for treatment of tri-state Z's arriving into a two-state model is introduced.

Keywords

RTL, simulation, 2-state, X-state, pessimism, optimism, random, initialization.

1. INTRODUCTION

Whether it is VHDL or Verilog or C, designers are using register transfer level (RTL) of abstraction [3][6][9][12] as their source language for their detailed design work and design verification simulation. As elsewhere, this is true in our design laboratory, where we design large-scale high-performance systems.

In our most recent design work, the systems that we simulated were scalable ensembles of 2 to 512 CPU's in a ccNUMA (cache coherent nonuniform memory access) architecture. To support the scalability and ccNUMA features of this architecture, the systems included 7 to 512 instances of 5 ASIC types.

Four of these ASIC types designed in our laboratory provided data path and control interfaces to the CPU's, memory, I-O, and sets of CPUs. A fifth ASIC type provided the crossbar interface between the other four ASIC types. Each ASIC averaged about 500K gate-equivalents.

Our system simulation model consisted of:

- RTL models of our ASICs.

  We simulated the ASICs within our system model as standard Verilog using either the vendor compile with full accessibility to internal logic, or our own cycle-based compile with limited accessibility to internal logic.

- Bus functional models of the CPU's and main memory.

- Board level logic, consisting of:

  a) Power-on, environment, JTAG, and boot up sequence control.

  b) Board-level clock fan-out.

  Though very small compared with the ASICs in terms of the amount of logic, the board level control logic had an intimate tie-in with the start-up logic within the ASIC instances.

In this design project, we used a two-state RTL simulation design verification methodology. It began with each ASIC type in its individual simulation test bench, and continued into all of our system simulations.

Two-state in this paper refers to eliminating the X, and using only 0, 1 and Z states. Although tri-state busses had an important place in our system design and simulation, the bulk of the logic and nodes were only two-state, not tri-state. Most of this paper discusses the two-state simulation method, but it does include a discussion of our original technique for treatment of tri-state Z's arriving into a two-state model.

Our initial purpose in eliminating the X was simulation performance. We are not alone in eliminating the X. In the past two years, new vendor simulator releases [4][11] have provided the option of simulating without an X-state in order to achieve greater simulation performance.

However, we believe that using the X state in RTL simulation is a bad idea, even without the performance penalty that it causes. Section 2 of this paper describes how RTL simulation can be both excessively pessimistic and optimistic, and how attempts at overcoming these shortcomings are impractical. Section 3 introduces some aspects of start-up state design in our ASICs. Section 4 describes the application of two-state simulation in our design process. Section 5 tells about the results of our experience

with two-state simulation. Section 6 presents suggestions about further developments needed to better support two-state simulation.

2. RTL X-STATE PROBLEMS

2.1 Pessimistic

Arithmetic operations are one example of gross pessimism in X-state RTL simulation. Consider the following example.

```
reg [0:15] a,b,c;
...
begin
        b = 16'b0000000000000000;
        c = 16'b000000000000x000;
        a = b + c;
        $display(" a = %b",a);
end
```

The result for "a" in a 4-state Verilog simulator will be

"a = xxxxxxxxxxxxxxxx".

In RTL simulation of arithmetic operations, fast simulators map these operations into host computer instructions. These fast simulators detect any x-bits in the input operands by checking an extra "flag word" for each input operand. Bits that are "1" in the "flag word" mark bit positions that are x in the input operand. So if the flag word is non-zero for either input operand, the simulator skips the addition instruction, and assigns all X's to the result. Note that the overhead added by the check for x-bits in an input operand is a single-instruction step, and therefore closely matches the performance of a single host-machine arithmetic instruction.

At the cost of reduced simulation performance, a Verilog gate-level simulation can more accurately handle this addition, resulting in "a = 000000000000X000". The gate level simulator can propagate the X more correctly because it pays the performance cost of visiting each bit in each operand, and generates a result bit-by-bit.

Another example of pessimism is the following **case** statement.

```
reg [0:1] d,e;
...
begin
   d = 2'b0x;
   case d
        2'b00 : e = 2'b01;
        2'b01 : e = 2'b11;
        2'b10 : e = 2'b10;
        2'b11 : e = 2'b00;
        default : e = 2'bXX;
   endcase
   $display(" e = %b",e);
end
```

Consider the situation where the control variable "d" is "0X." Interpreting the "X" as a possible "0" or "1," only the first two **case** branches should be reachable. So, less pessimistically, only the left bit of "e" is ambiguous, and the result should be "e = X1." However, a 4-state Verilog simulator will give "e = XX" when control variable "d" is "0X."

2.2 Optimistic

More insidious is the way that RTL simulation of **case** statements and **if-else** statements with an X-state can lead to optimistic results, and thereby hide real start-up problems in a design.

Given an XX as the start-up state for d, the following **case** statement will take the default branch. That only tests one of the four possible branches the start-up condition could actually take, if we consider the four possible two-state interpretations of the XX bits.

```
reg [0:1] d,e;
...
begin
    case d
        2'b00 : e = 2'b01;
        2'b01 : e = 2'b11;
        2'b10 : e = 2'b10;
        default : e = 2'b00;
    endcase
    $display(" e = %b",e);
end
```

2.3 Impractical

As a thought exercise, it is possible to envisage an RTL style that would intercept and process X states more accurately, moderating both the pessimism and the optimism.

Here is a **case** statement that intercepts X states and propagates their affect on the result more accurately.

```
reg [0:1] d,e;
...
begin
    case d
        2'b00 :  e = 2'b01;
        2'b0X :  e = 2'bX1;
        2'b01 :  e = 2'b11;
        2'bX1 :  e = 2'bXX;
        2'b11 :  e = 2'b00;
        2'b1X :  e = 2'bX0;
        2'b10 :  e = 2'b10;
        2'bXX :  e = 2'bXX;
    endcase
end
```

Here is an **if-else** statement similarly modified to intercept and propagate an X state.

```
if (f = = = 1'b0)
    g = 2'b00;
else
if (f = = = 1'bX)
    g = 2'b0X;
else
    g = 2'b01;
```

These reductions to absurdity illustrate how RTL usage that attempts to intercept X's everywhere is a not a good idea. Here are some reasons for not intercepting X's.

- Simulation performance. All the extra tests for X's add to the CPU processing that the simulator has to do.

- Labor content. Someone has to do the work of adding the extra statements.
- Complexification. A good feature of RTL design is that it can present a designer's intent more clearly than gate-level design. Intercepting X's detracts from the clarity.
- Completeness. There is no current method of guaranteeing that the designer's X interception and propagation is complete enough to avoid the pessimism and optimism.
- Synthesis. Doing this makes the RTL a ternary logic design, which has to be thrown out of mapping the design through to binary logic gates in synthesis.

We prohibited use of X-intercepting and X-assignments anywhere in our RTL logic design. This included the X-intercepting **default** in fully specified **case** statements as shown in [10].

```
...
case (select)
    2'b00 : mux = a;
    2'b01 : mux = b;
    2'b10 : mux = c;
    2'b11 : mux = d;
    default : mux = 'bX;
endcase
```

Our RTL design style for our ASICs required that all **case** statements be fully-specified, so assigning an X in a **default** was never needed for telling synthesis about don't-care situations.

Contemporary logic synthesis technology allows for greater optimization of generated gates for **case** statements in which certain input control variable state values are impossible. For these **case** statements, the designer doesn't care about what output states the gates generate for those control state values.

On this project, we felt that the extra gates saved by allowing synthesis to optimize don't-care logic were not worth:

- precluding running our gate-based ATPG[1] test vectors against the RTL ASIC models.
- the challenges it would present to fast RTL-to-gate boolean equivalence checking between the RTL and the gate [5].

3. START UP STATE DESIGN

Because some aspects of our start up state design may be unusual, some discussion of the topic is in order.

In the ASICs for our system design, we combined reset signal and ASIC instance "personalization" input ports to bring their registers to an acceptable start up state.

Some registers did not connect to either reset or input ports. They were designed to be acceptable in any state, or arrive at an acceptable state given a few clock cycles and being fed by the states of the registers that did connect to reset or personalization input ports.

Why not fan out the reset to all the registers?

[1] Automatic Test Pattern Generation, for manufacturing tests.

- Routing area. Reset fanout takes routing area, adding to the cost of physical design, or reducing the total amount of logic that can be fit on a given ASIC. Note that for full reset, reset has to go everywhere that the clock goes.
- Timing. Considering the start of reset, reset timing does not immediately seem to be critical. However, the time when reset signal goes away has to be carefully tuned so that it happens everywhere within the intended clock cycle. Otherwise, some state machines may start "moving" a clock cycle before the others that are still reset. The interaction between them will likely lead to bad outcomes.
- Design verification test. For some free-running counter registers, any start-up state should be acceptable.

4. APPLYING TWO-STATE SIMULATION

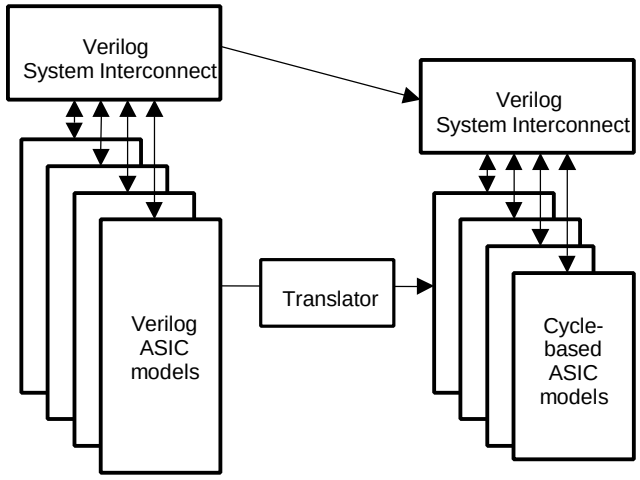
4.1 Two RTL Simulators

We used two simulation models for our ASICs in our system development RTL simulation work.

1. vendor (Verilog-XL™, VCS™) models.
2. in-house cycle-based models.

We generated both types of ASIC simulation models from the same RTL Verilog source code.

Figure 1 illustrates the transition from standard vendor ASIC models to cycle-based ASIC models in our system simulation.



a) Vendor Verilog ASIC models

b) Cycle-based ASIC models

Figure 1. System simulation models

Early in our design process, there were plenty of bugs, and simulations failed in the first few cycles. At this stage, designers used the vendor simulation models, for the debug accessibility.

As design refinement progressed, simulations began to run for a long time between bugs. Simulation performance became increasingly important.

At this point in the design process, our designers sacrificed debug accessibility for the greater performance of the cycle-based simulation of their ASICs. Our cycle-based simulation had many

of the features of cycle-based simulation described elsewhere [1][7]:

- bypassed update of internal nodes.
- limited accessibility to internal nodes.
- evaluate multi-bit buses instead of individual bits.
- two-state simulation i.e. no X state.

4.2 Zero Initialization Experience

Our progression to two-state RTL methodology began when our five ASIC designs started passing their tests using the vendor Verilog RTL simulation model. It started with registers in an X state.

We then tried running the same tests on the ASICs using our cycle-based two-state simulation that started with registers in an 0 state. All of the ASICs failed their tests.

We suspected that there was a bug in our cycle-based two-state ASIC simulation, because it was newer and not as widely used on different designs as the vendor simulator.

To verify this, we added PLI[2] calls to the ASIC RTL Verilog that initialized all registers and memory arrays to zero, and simulated them using the vendor simulator. They all failed their tests!

The problems were in the designs and not in our simulation. What the designers found was that they had tuned their RTL designs to simulate with registers initialized to X. Their **if** and **case** statements tested their control variables against two-state constants. With registers initialized to X, in the first simulation cycles **if-else** statements took the **else** branch and **case** statements took the **default** branch.

With registers initialized to 0, wherever **if-else** and **case** statements compared control variables with zero, the statement took that branch. In effect, initializing registers to zero amounted to a different test for the ASIC than initializing to X.

Designers found that design problems brought to light by initializing registers to zero were relatively easy to track down, compared with whatever gate-level X-initialization problems they had faced in the past. This was a result of working with "real" 0 and 1 states that occur in the hardware, not the X, which only occurs in simulation.

4.3 Random Initialization Added

Because we had used PLI calls in our ASIC RTL to initialize registers to zero, it was a simple matter of programming to add a simulation run-time option to our PLI functions that loaded the ASIC registers with random values.

The PLI functions used a time-of-day or user-specified seed for generating random values. When using the time-of-day seed, the PLI functions reported that seed, so a user could re-create any problems detected with the time-of-day seed by specifying that seed in subsequent simulations.

[2] Programming Language Interface. A standard Verilog HDL feature that provides for calls with parameter passing between HDL and C functions.

A random state initialization feature is not new. Describing the DEC Alpha functional verification methods and experience in [9], mention a bug that got through to silicon due to insufficient randomization of the RTL simulation model initial state. Random state initialization was built into the *HILO* simulator [8].

A unique random state initialization feature that we added as we progressed from vendor-based ASIC simulation models to cycle-based ASIC simulation models was *consistent random initialization*[3]. The consistent result aspect is important for two reasons. Given the same seed, it allowed us to duplicate the machine state for a design problem:

1. between the cycle-based based ASIC simulation model and the vendor-based ASIC simulation model. A design problem detected by detection-oriented simulation model could be re-created and diagnosed by the other simulation model, given the same seed.
2. after a design change intended to fix the problem. This allowed the designer to verify that a design problem for a given state had been fixed after the design change. Given the same seed, all register state bits started in the same state as before the design change, except for new register bits that were added in the with the design change.

4.4 Random Initialization Verification Test

One important fact to note is that by NOT fanning out reset everywhere and NOT scanning a defined state into every register bit, random state initialization became a design verification test feature in some cases.

If we designed our ASICs in a way that all register states were defined at start up, it might take many clocks of a directed or random test sequence for the interacting state machines to arrive at states which brought out a design problem in their interaction.

By leaving registers open to random states at start up, we were able to find some state machine interaction design problems at the beginning of a simulation.

One example of this was a refresh counter. Each time we ran a simulation test, the timing of the refresh cycles interrupted the test sequence at different times. As the test passed for each run, it provided additional assurance that the interacting state machines in our design could handle the test refresh interruption correctly.

4.5 Tri-state Buses

To simulate tri-stated buses in our system design, we set up boundaries between the two-state simulation regions and the tri-state signal lines. The tri-state signal lines used 0, 1, Z and X values, while the remainder of the system signals used only 0 and 1.

The bus drivers consisted of a two-state enable signal, and the two-state signal being enabled.

The tri-state bus receivers had to deal with the situation of interfacing the potentially inactive bus signals at a Z state with two-state logic and registers. A correct logic design would not enable the bus at a Z state into any two-state region, but we

[3] Hewlett-Packard patent pending.

wanted to somehow expose the situation where a design problem caused the bus signal receiving logic to be active when the bus signal was at a Z state.

For the same reasons (simulation performance, labor content, complexification, completeness, synthesis) we didn't extend our RTL Verilog to watch for the X state in **if-else** and **case** statements, we did not extend them for Z state either.

We added a Z state trapping PLI call right at the tri-state to two-state bus receiver boundary. So, given a tri-state **pin** on one side of the boundary, and a two-state **ipin** on the other side, the

```
$TrapXZ(ipin,pin);
```

function passed through 0 and 1 values on **pin** to **ipin**, but put random 0 and 1 values on **ipin** for any bit(s) that were Z (or X) on **pin**.

The intent in the random values is to cause design problems to manifest themselves because of bad data or invalid control signals.

Including the X in the Z trapping feature was close to free in terms of labor and affect on simulation performance. In addition to dealing with the bus driver conflict problem (which would have caused the bus to resolve to X), the X trapping feature allowed us to use the same function in our system simulation model. In the system model, there were non-tri-state interfaces between commodity part models written to start at X, and our models expecting only 0 or 1.

4.6 Undetected Reset Problems

Even with the RTL two-state initialization methodology, a few initialization problems remained to be found at the gate-level using X initialization in our ATPG vector simulations.

One reason is that X-state gate-level initialization is pessimistic, and will show problems where there are none [2]. These are not real design problems that would show up in the hardware, but an artifact of X-state gate-level simulation.

Another reason is even though we performed RTL simulation with many initial machine states, some real machine states covered by gate-level "X" initialization had reset problems. Only a subset of all combinations of initial states were reached in our RTL simulations.

In spite of the fact that a few reset problems did get through to gate-level, designers found a majority of their reset problems using random two-state initialization in RTL simulations. Designers were happy with this, because they found most reset problems early, and these reset problems were based on real 1-and-0 machine states, not the pessimistic gate-level X-state.

5. RESULTS

5.1 Acceptance

Our designers have completely eliminated X-state from all of their RTL ASIC simulations. They use an in-house lint checker to ensure that all registers in the ASIC designs have the initialization PLI calls that initialize them to two-state values.

Unlike other HDL dialect and policy recommendations where there has sometimes been lingering disagreement, acceptance of the two-state simulation HDL style among our design team has been one hundred percent. Some designers with long-time gate-level experience simulating using the X state expressed reservations at the start. But with a couple of hours of experience debugging RTL designs with random 0 and 1 initialization, they all made the mental mind-set conversion away from using the X state.

5.2 Assertion Checkers

It is possible to envisage assertion checkers as an alternative or complementary method to two-state simulation and inserting random values. Here are some examples:

- the default in a case statement that checks for all possible two-state values could issue a diagnostic message when the case control variable has X bit values.
- bus Z-states arriving at an active input

Compared with assertions, design problems expose themselves differently when found by random values. The random values at startup and substituted for Z-state signals coming into a two-state register introduce data path parity errors and control state machine sequence malfunctions. Though a little harder to diagnose than an assertion that "talks to" the test engineer, the random values provide more complete coverage with less labor content than assertions.

As with other verification methods that may overlap in terms of what they can detect and diagnosis, assertion checkers are acceptable in a two-state methodology. However, it should be noted that we have eliminated X-detecting assertions from our RTL style.

5.3 Usage

Over our six-month peak simulation period, our design and verification engineers performed about $0.5 * 10^6$ simulations. Except when repeating a simulation to track down a bug, they ran each of their simulations with a different UNIX "time"-based seed. So in addition to their directed or random test sequence, they verified that the ASICs would start up correctly in $0.5 * 10^6$ start up states. This number of start-up states tested is far smaller than the more than $2^{200,000}$ possible start-up states for our system. However, identifying the startup design problems that we were able to find through RTL two-state simulation early in the design process was a win, compared with RTL or gate-level simulation using an X state.

For all new ASIC design work, our design and verification engineers started with two-state RTL on a conventional vendor simulator for all directed and random testing from the very beginning of their simulations. They abandoned X-state RTL simulation from the beginning of their design work, starting their first RTL simulations with all storage elements set to zero. After tests began passing with zero initialization, they tested their designs with random initialization of all storage elements.

The only RTL simulations that we ran with storage elements in an X state at start up were for manufacturing test vectors. On our most recent project, this amounted to around 500 simulations, and resulted in one design change for reset in each of our 5 ASIC types. These X-starting-state simulations were run at the *end* of the design cycle, just prior to going to silicon.

Two release cycles of our five ASIC types have made it to silicon and into our V2250 and V2500 products with no start-up state reset problems. This was unlike prior ASIC projects where X-state RTL simulation had hidden start-up state problems that made it to silicon. Zero and random two-state initialization would have caught these problems in simulation, according to the designers on those earlier ASIC projects.

Table 1 summarizes the V2250/V2500 project startup state experience, including the approximate project cost per problem.

Test phase	Startup problems	Cost/problem
RTL ASIC 2-st	~18	Hours
RTL System 2-st	2	Days
RTL/Gate ATPG 3-st	5	Days
Post-Silicon	0	(Weeks)

Table 1. V2250/V2500 project startup problem experience

The "~18" for RTL ASIC 2-state simulation in the table is approximate because it is based on designers' memories. Designers did not log every design change needed to make their ASIC pass their first tests.

6. NEXT STEPS

Training. Future editions of textbooks on RTL HDL usage should discuss "X" optimism and pessimism, and remove X-assignments from default branches of case statements**.**

Models. Commodity part models are still delivered that start at X values and send X values out. Just as there is a need for cycle-based commodity part models for fast simulation in system design, there is also a need for two-state RTL simulation-oriented commodity part models.

Standardization. Random initialization has proven to be a very useful method in our RTL simulations. However, our PLI call implementation is tightly linked to our design group's entire CAD tool flow. It cannot be picked up and inserted into other design groups, even within our own company, unless they carefully craft it into their scripts and programs.

The better way is for Verilog simulation vendors to add simulation run time random initialization options to their simulators. To be done right, the random initialization must be done according to a standard, so that designers can repeat their simulation tests using the same random seed. This repeatability should span different simulators, even from different vendors, as well as before and after design changes.

Ideally, this feature should be built into Verilog and VHDL simulators, so that users could get the benefits of RTL (and other) simulation random initialization in a standard manner with consistent results, without the PLI calls.

7. Conclusions

We have presented the case for eliminating the X from RTL simulation, and our experiences in using 0 and random value start-up register and memory states in ASIC and large system simulation. We discussed how we enhanced our random techniques to support (a) consistent random values across simulators, as well as before and after design changes, (b) transformation of Z (and X) inputs to random two-state values, and (c) test coverage with free-running registers. We explained that even though the random state coverage is incomplete, it brought to light many design problems early in RTL simulation. This experience completely convinced our designers that two-state and random techniques are the right way to do RTL simulation.

To the extent that others are convinced of the value of two-state RTL simulation and the random techniques discussed here, we would expect expanded support in these directions in future (a) releases of vendor simulation tools, (b) commodity part libraries, (c) standards, and (d) textbooks.

REFERENCES

[1] Ashar P. and Malik, S., "Fast functional simulation using branching programs," *Proc. IEEE ICCAD-96*, pp. 408-412, November, 1996.

[2] Breuer, M. A., "A note on three-valued logic simulation," *IEEE Trans. Computers*, vol. C-21, pp. 399-402, April, 1972.

[3] Evans, A., Silburt, A., Vrckovnik, G., Brown, T., Dufresne, M., Hall, G., Ho, T. and Liu, Y., "Functional verification of large ASICs," *Proc. Design Automation Conference*, pp. 650-655, June, 1998.

[4] Fitzpatrick, T., "Verilog modeling style guide for the Cobra cycle simulator," Cadence Design Systems, Chelmsford, MA, Rev. 2, pp. 11-12, August 17, 1998.

[5] Foster, H. "Techniques for higher performance boolean equivalence verification," *Hewlett-Packard Journal*, August, 1998, pp. 30-38.

[6] Hoehne, H. and Piloty, R., "Design verification at the register transfer language level." *IEEE Trans. Computers*, vol. C-24, pp. 861-867, September, 1975.

[7] McGeer, P. C., McMillan, K. L., Saldanha, A., Sangiovanni-Vincentelli, A. L. and Scaglia, P., "Fast discrete function evaluation using decision diagrams," *Proc. IEEE ICCAD-96*, pp. 402-407, November, 1996.

[8] System HILO - DWL Reference Manual, document 2523-0103, page 9.6, VEDA Design Automation Inc., Campbell, CA 95008, January, 1991.

[9] Taylor, S., Quinn, M., Brown, D., Dohm, N., Hildebrandt, S., Huggins, J. and Ramey, C., "Functional verification of a multiple-issue out-of-order, superscalar Alpha processor -- the DEC Alpha 21264 microprocessor," *Proc. Design Automation Conference*, pp. 638-643, June, 1998.

[10] Thomas, D. E., Moorby, P. R., *The Verilog Hardware Description Language*, Kluwar Academic Publishers, Norwell, MA 02061, pp. 136, 4th Edition, 1998.

[11] *VCS User's Guide*, Synopsys Inc., Mountain View, CA, pp. 2-19 – 2-30, December, 1998.

[12] Yim, J. S., Hwang, Y. H., Park, C. J., Choi, H., Yang, W. S., Oh, H. S., Park, I. C. and Kyunge, C. M., "A C-based RTL design verification methodology for Complex Microprocessor," *Proc. Design Automation Conference*, pp. 83-88, June, 1997.