Eliminate the X-Optimization during RTL Verification

Xu Huang[1, a], He Xin[2,b], LinTao Liu[3,c] and LunCai Liu[4,d]

[1]Sichuan Institute of Solid State Circuits, Chongqing, P.R. China

[2]Sichuan Institute of Solid State Circuits, Chongqing, P.R. China

[3]Sichuan Institute of Solid State Circuits, Chongqing, P.R. China

[4]Sichuan Institute of Solid State Circuits, Chongqing, P.R. China

[a]hxtt103@163.com, [b]hexin@sisc.com, [c]hgdllt@sisc.com, [d]llc@sisc.com

Keywords: X-Optimization,Verification

Abstract. Verification of complex SoC designs suffers from X-optimization issues that often conceal design bugs. The deployment of low power techniques such as power-shutdown in today's SoC designs exacerbate these X-optimism issues. To address these problems we adopted a new simulation semantic that more accurately models non-deterministic values in logic simulation. In this paper we discuss how to eliminate the X-optimism during RTL verification.

Introduction

Functional verification by simulating at the register transfer level (RTL) is a common technique used to ensure that a design meets its functional specification. However, traditional RTL simulation is unable to catch all design defects; instead, design teams typically resort to simulation of the gate-level representation to catch more bugs. Unfortunately, most designers run only a very small set of tests at the gate level because these simulations require compute servers with much higher memory requirements, and much longer runtimes.

Engineers today tend to rely more on static timing analysis and equivalence checking, and less on gate-level simulation. Frequently, only a small subset of tests is run at the gate level, and occasionally, no gate-level simulation is done at all. Also, gate-level simulation can only be done late in the design phase due to lack of availability of the design netlist. If a bug is found at this late stage, it is more costly to implement the fix than it would be during RTL validation.

There is a few of the reasons that gate-level simulation is used are: a) to check proper initialization sequences; b) to check scan operation; c) to check false and multi-cycle paths; d) to check X's in RTL simulations. The last item is quite important but is commonly overlooked. RTL verification has been the workhorse of the industry for over twenty years, yet there is a fundamental flaw in this process. Engineers that verify their design using RTL simulation and ascertain that all tests are passing may be surprised to hear that the same design will fail during gate-level simulation or in real silicon due to some-thing called X-optimism.

What is X-Optimism?

Optimism is defined as a tendency to expect a favorable or hopeful outcome. For this discussion, optimism is the condition where the simulation model yields fewer unknown values than are really possible. While optimism may generally be good, it is inadvisable in chip verification.

Verilog HDL models four different logic values: 0, 1, X, and Z. The 0 and 1 represent the Boolean logic values False and True, respectively. The Z represents an undriven (tri-state) signal, and the X represents an unknown or indeterminate value. The value X is a very useful simulation abstraction that may be intentionally designated in the model, or it may be the unintentional result of a logic operation. For example. a designer may use an X to designate the value of a variable as immaterial (a don't-care); conversely, an unintentional X value is the result of multiple drivers with different Boolean values driving the same net. A very common source of X values are uninitialized registers, that is, registers that are not initialized via a hardware reset. These registers could be part of a data path pipeline and hence not require a direct reset. Yet another (unintentional) source of X values are

error conditions such as timing violations (setup and hold) that could render a register meta-stable. The value X is a modeling abstraction because it does not exist as a distinct logic value in the actual hardware where even uninitialized registers will have either logic value 0 or 1. Next, we illustrate these concepts using use common RTL coding idioms.

Simple if Statement

```
always @*
  if (cond) c = a;
  else      c = b;
```

Figure.1. Simple IF Statement--Multiplexer

Figure 1 shows the Verilog code that models a simple multiplexer. Table 1 shows the truth table implemented by this Verilog model. The first three columns represent the inputs to the multiplexer. The fourth column shows the corresponding output value of the RTL simulation. The fifth column shows the actual hardware behavior. Inspecting the truth table, we see that when both data inputs are the same, the value of the select line is immaterial, and the output is equal to the data input value. When the data inputs are different, the output cannot be determined, hence, the fifth column shows a value of 0/1, which indicates that the real value depends on the actual value of cond. In contrast, the RTL simulation always assigns a definite logical value to the output.

cond	a	b	c (RTL)	c (HW)
X	0	0	0	0
X	0	1	1	0/1
X	1	0	0	0/1
X	1	1	1	1

Table 1. True Table for the Simple Multiplexer

The above behavior is mandated by the Verilog standard[1], which considers X-valued conditional expressions as false thereby require execution of the else clause of the if statement. Thus, when cond is X, the output c is always assigned the value of b. In other words, the if statement is evaluated "optimistically" by executing only the statements within the else clause when in reality it must consider executing both branches. This optimistic behavior is not confined to the if statement, but to all conditional controls, including the case statement.

Simple Case Statement

```
case (s)
  1'b0    :   r = a;
  1'b1    :   r = b;
endcase
```

Figure.2. Simple Case Statement

Figure 2 shows the Verilog code for a simple case statement. Table 2 shows the truth table implemented by this simple case statement. The first three columns represent the inputs to the case statement. The fourth column shows the corresponding output value of the RTL simulation. The fifth column shows the actual hardware behavior. Just like the previous example, when both data inputs are the same, the value of the select expression is immaterial: the output is equal to the value of the data input value. When the data inputs are different, the output cannot be determined, hence, the fifth column shows a value of 0/1 denoting the uncertainty. In contrast, the RTL simulation does not match any case item causing no statement to be executed. As a result, the output `r` retains its previous value, which incorrectly converts the combinational block into a memory element.

s	a	b	r (RTL)	r (HW)
X	0	0	r[t-1]	0
X	0	1	r[t-1]	0/1
X	1	0	r[t-1]	0/1
X	1	1	r[t-1]	1

Table 2. True Table for the Simple Case Statement

Existing Approaches to the X-Optimism Problem

The problems associated with X-optimism have been raised before[2], and even without a systemic solution, engineers have successfully taped-out and manufactured many designs. They have accomplished this through a combination of methodologies and workarounds that alleviate the problem. Below we describe some of the most common techniques in use today.

Resetting All Registers

The X-optimism problem can be largely avoided by explicitly resetting all sequential elements in the design. After the device is reset, there are no uninitialized registers in the design. Eliminating the initial X's thus bypasses the most common cause of X-optimism issues. Likewise, if the design includes blocks that can be shutdown then all registers in the shutdown blocks can be implemented as retention registers that can be powered up to a known state. The trade-off with this scheme is that it increases the area, delay, and power consumption of the design. Rather than solve the X-optimism problem, this scheme simply circumvents the problem, and is typically not appealing for leading edge designs.

Gate-Level Simulation

The only know approach that can catch all bugs due to X-optimism is to run a complete set of functional gate-level simulations. A design description at the gate-level no longer contains conditional control structures that can cause X optimism issues. This approach requires a fully synthesized netlist and is thus available only late in the design process. Gate-level simulations tend to be very pessimistic, which make RTL equivalence very difficult. In addition, debug at the gate-level can be very time-consuming because the gate netlist depends not only on design complexity but also on the transformations and optimizations performed during synthesis. Finally, the higher memory and runtime requirements of gate-level simulations severely limit their throughput, thus, allowing only a subset of the functional tests to be completed.

Random Register Initialization

Another common approach is to perform two-state simulations and randomly initialize to 1 or 0 all registers in the design[3]. Random initialization is appealing since it emulates the initial conditions of real hardware. However, this approach suffers from several drawbacks. Initialization to a random value is less general than an X, which captures all permutations of logic values. Hence, this approach requires more simulation runs to ensure sufficient coverage: a design with n registers requires $2n$ runs in order to exhaustively cover all initialization scenarios. The increasing memory size of today's designs makes such an approach prohibitive. Often, random initialization is applied to only a small set of runs - typically one run.

VCS X-Prop Solution

Standard RTL semantics of conditional constructs in Verilog allow for indeterminate values in the control expressions to assign determinate values to the associated data path variables. This can have the consequence of failing to catch X-related bugs. A better simulation semantic model is to execute all alternatives whose execution is controlled by X values, and then merge the results in a predictable manner. This new simulation semantics eliminates X-optimism, models X values in a more realistic manner (closer to the actual hardware), reduces synthesis-simulation mismatches, and allows X-related bugs to be exposed by regular RTL simulation.

New RTL Semantics

These new RTL semantics for indeterminate values approximate the *don't-care* behavior used in logic synthesis. For every X-controlled statement, the simulator considers the effect of the control value being both 0 and 1. Conceptually, every variable assignment controlled by an X is substituted in all possible execution paths, and then all the substitutions are merged using a *merge function*. This process is illustrated using the Simple If Statement example shown in Figure 1. If `cond` has value X then values 0 and 1 both considered as shown in Table 3. When 0 is considered, the false statement of the if statement is executed, and C_0 is assigned the value of `b`. When 1 is substituted, the true statement is executed and C_1 is assigned the value of `a`. Conceptually, every executed branch creates a new temporary

variable. After all branches are considered the final value of c is computed by merging its constituent values.

Value considered	Effect	Final Value
cond == 1	C1 = a	C = merge(C1,C0)
cond == 0	C0 = b	

Table 3. True Table for the Simple Case Statement

Merge Modes

The new semantics provides two merge modes. The first mode is called T-merge, and its truth table is shown in Table 4a. T-merge, which is derived from the ternary operator (hence the T), yields X when any of its inputs are different. The merge operates bitwise so that only bits whose values actually differ will propagate an X. The T-merge semantics eliminate X-optimism, and closely emulates the actual hardware behavior.

	0	1	X	Z
0	0	X	X	X
1	X	1	X	X
X	X	X	X	X
Z	X	X	X	X

Table 4a. T-Merge Truth Table

	0	1	X	Z
0	X	X	X	X
1	X	X	X	X
X	X	X	X	X
Z	X	X	X	X

Table 4b. X-Merge Truth Table

The second merge mode is called X-merge, and its truth table is shown in Table 4b. X-merge yields X regardless of the values of its inputs. The X-merge semantics also eliminate X-optimism, but its behavior is more pessimistic. This level of pessimism is conceptually equivalent to the projection of all X-pessimistic behaviors of any gate-level implementation of the RTL. It is useful to ensure that a gate-level implementation does not generate an X that has not been model by the RTL.

Comparison of Merge Modes and Gate-Level for the Simple If Statement

To illustrate the merge modes we consider the simple if statement shown in Figure 1. Figures 3a, 3b, and 3c show three possible gate-level implementations of the simple if statement.

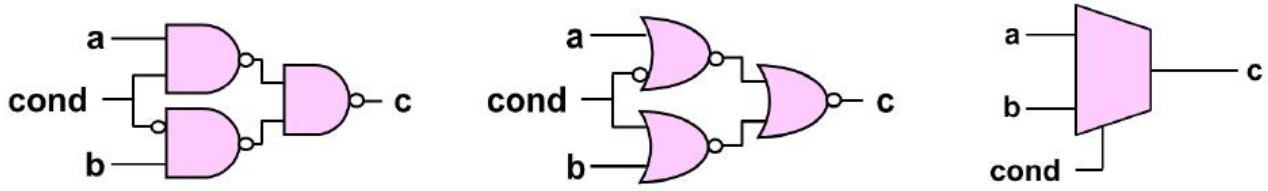

Figure.3a,3b,3c. Different Implementations of the Simple If Statement

cond	a	b	c (HW)	c (RTL)	c T-Merge	c X-Merge	c (4a)	c (4b)	c (4c)
X	0	0	0	0	0	X	0	X	0
X	0	1	0/1	1	X	X	X	X	X
X	1	0	0/1	0	X	X	X	X	X
X	1	1	1	1	1	X	X	1	1

Table 5. True Table for the Simple If Statement with T-Merge and X-Merge

Table 5 shows the truth table for the different models of the simple if statement. The first three columns represent the inputs to the multiplexer. The 4th column shows the corresponding value of the actual hardware, and the 5th column shows the output value of the standard RTL simulation. The 6th column shows the output value of the T-merge RTL simulation: it yields a determinate value when inputs a and b have the same value, and X when they differ. The 7th column shows the output value of the X-merge RTL simulation: it always yields an X value regardless of the values of a and b. It can be seen that the both the T-merge and X-merge modes remove the X-optimistic behavior exhibited by the standard RTL simulation. The 8th, 9th, and 10th columns show the output value corresponding to the gate-level simulations of each of the three implementations shown in Figure 3. It is interesting to note that the last row of the 8th column (4a) and the first row of the 9th column (4b) show an X, which is due to the reconvergent fan-out of the cond input. It is also noteworthy that the X-merge column is

the projection of the X behaviors of the three gate-level implementation, that is, it shows an X whenever any of the gate-level simulations will generate an X.

Conclusions

This paper reviews the problems that arise from the X-optimistic semantics of the standard RTL simulation model. It uses simple examples to illustrate the nuances of X semantics, and describe how these semantics lead to incorrect behaviors that often conceal defects. These hidden bugs lead to passing RTL simulations that mislead design teams, thus, creating problems that are hard to correct later in the flow.

The paper also highlights the most common approaches currently used to address the problems created by X-optimism. These include gate-level simulations, pseudo-exhaustive 2-state random simulations, and other avoidance techniques. Each of these workarounds suffers from various shortcomings and inefficiencies, hence, none offers a complete solution.

This paper proposes a new method for RTL simulation that changes the X semantics in order to eliminate the incorrect results due to X-optimism. We introduce the merge mode: a novel mechanism that allows designers to control the level of X-optimism in RTL simulations. Two merge modes, T-merge and X-merge, are discussed and contrasted. Their behavior is analyzed using simple common idioms to demonstrate how they reveal bugs that are otherwise hidden.

The new X semantics enable RTL simulation to provide hardware accuracy, but with the speed and capacity of traditional RTL simulation. The high performance and ease of use allowed the execution of a full test suite. The higher accuracy lead to increased coverage, greater confidence in the functional behavior, and overall improved designer productivity.

References

[1] IEEE Computer Society, "IEEE Standard for SystemVerilog – Unified Hardware Design, Specifi-caton, and Verification Description Language", IEEE Std 1800-2009.

[2] Harry Foster, “Semantic Inconsistency and its effect on simulation”, IEE Electronics Systems and Software, April/May 2003

[3] Lionel Bening, "A Two-State Methodology for RTL Logic Simulation," In Proceedings of 36th De-sign Automation Conference (DAC), New Orleans, June, 1999