Handling Nondeterminism in Logic Simulation So That Your Waveform Can Be Trusted Again

Kai-Hui Chang, Hong-Zu Chou, Haiqian Yu, Dylan Dobbyn, and Sy-Yen Kuo

Abstract

The increasing complexity of integrated circuits pushes for more aggressive design optimizations, such as resetting only part of design registers, that can leave some registers in nondeterministic (X) states. Such Xs may invalidate the correctness of logic simulation due to X-optimism and X-pessimism, producing simulation waveforms that cannot be trusted. Although formal methods can resolve the nondeterminism problem, they are not scalable enough to handle today's multi-million gate designs. To address this problem, we developed a scalable X-analysis methodology and successfully applied it to solve three real industrial problems — one identifies missing Xs in RTL designs while the other two remove incorrect Xs to repair gate-level simulation.

Keywords: Nondeterminism, symbolic simulation, X-analysis, X-verification

1 Introduction

To design today's multi-million gate circuits, aggressive optimizations are being used. For instance, to deal with the reset signal routing problem, one can reset only part of design registers and use software sequences, called reset sequences, to initialize the rest. This approach, however, creates new challenges in design verification because uninitialized registers will exhibit nondeterministic behavior. In particular, Xs are often used to represent the nondeterminism of uninitialized registers. However, as we will show in Section 2, logic simulation cannot handle Xs correctly. As a result, what designers see in the waveform may be inaccurate, resulting in bugs escaping verification. Although formal-verification techniques such as [4] can accurately handle the Xs, they are not scalable enough to handle today's design sizes and long reset sequences. In addition, engineers may be unfamiliar with formal tools. To address these problems, we strive to develop a scalable methodology that leverages the proving power of formal methods while working seamlessly with the prevalent simulation-based verification flow. Our methodology, called eXact [3], utilizes the principle of most astonishment by pointing out only the issues that will surprise the engineers. More specifically, we identify problems in the simulation waveform that engineers are not aware of. This philosophy allows us to significantly reduce the amount of formal analysis without sacrificing analysis quality. To further improve the scalability of eXact, we intervene logic simulation with formal methods and apply novel partitioning techniques. By carefully integrating different verification methods, we make sure no bugs will be missed, although false negatives may become possible. We successfully applied eXact to resolve three industrial issues: one for finding unexpected Xs after reset and two for fixing gate-level logic simulation problems. With our methodology, simulation waveforms can be trusted again. In the rest of the paper we first provide relevant background. Next, we describe the X issues in detail and show how our methodology solves the problems.

2 X-Pessimism and X-Optimism in Logic Simulation

X-handling in logic simulation is inaccurate due to X-optimism and X-pessimism [2, 7, 8]: X-optimism incorrectly removes Xs while X-pessimism introduces unnecessary Xs. Take the code shown in Fig. 1(a) for example, if "*a*" is X, "*out*" can be either "*b*" or "*c*", which should really be X. However, due to X-optimism in logic simulation, only one branch is considered and "*out*" will be equal to "*c*". Fig. 1(b) shows another example where signal "*out*" should not be affected by "*a*" but is assigned X erroneously due to X-pessimism.

(a)	**X-optimism**	(b)	**X-pessimism**
	a = 1'bx;		*a* = 1'bx; *b* = 1'b1; *c* = 1'b1;
	if (*a*) *out* = *b*;		*out* = (*a* & b) | (~*a* & *c*);
	else *out* = *c*;		
	result: *out* = *c*;		**result**: *out* = 1'bx;

Figure 1: A simple example to show X-optimism and X-pessimism problems in logic simulation.

3 Current Solutions for Handling X-Problems

Recently, Kaiss et al. [6] proposed a SAT-based method for *Sequence Equivalence Checking* (SEC) without initial state information. Their idea is to compute reset sequences that can bring the circuit to a known state before SEC. However, their method focused on generating reset sequences instead of verifying the correctness of existing ones. In addition, scalability remains an issue due to the heavy use of formal methods. Haufe and Rogin [5] proposed a technique that utilizes automatic Register-Transfer Level (RTL) code transformation to avoid unexpected X-propagation. However, their method is based on templates and cannot handle all the RTL syntax. Another way to detect X-problems is to run gate-level simulation by assigning random values to Xs and then compare the results with RTL simulation. The major advantage of this approach is that existing verification infrastructures can be used. However, setting up gate-level simulation still needs time, and gate-level simulation is slow. In addition, once a bug is found, tracing the problem from gate-level back to the RTL can be difficult.

To accurately handle Xs at the RTL, our preliminary work [4] proposed an X-analysis technique based on symbolic simulation which treats Xs as symbols and produces Boolean expressions in terms of symbols. Since each symbol represents an arbitrary value, it faithfully captures real hardware behavior. Take Fig. 1(a) for example, since the X in "*a*" is treated as a symbol, both conditions can be considered and the correct Boolean value for signal "*out*" can be produced:

$$out = a \ ? \ b : c;$$

To check whether the X in *a* can propagate to output *out*, our method first duplicates the design and then performs symbolic simulation. In the duplicated version, variables with Xs are replaced with new symbols, but other symbols remain the same. Next, a miter checks whether the outputs can be different. The built Boolean expression is shown below. A SAT solver then solves the problem: if a solution can be found, then the X can propagate to the output.

$$\begin{array}{l} out = a \ ? \ b : c; \\ out' = a' \ ? \ b : c; \end{array} \quad \Rightarrow \text{solve } miter(out, out')$$

In the following sections we describe how we improve the scalability of this method to address industrial problems.

4 Issue 1: Finding Reset Nondeterminism in RTL Designs

Reset nondeterminism problems are caused by uninitialized registers in a design. Due to X-optimism, Xs may disappear in logic simulation and the nondeterminism problem can be missed by the engineer. Since our goal is to identify problems in the simulation waveform, we assume that waveforms produced by simulating the reset sequences are available. To perform X-analysis, we first replace all the Xs in uninitialized registers with symbols. We then symbolically simulate the design using the waveform as its stimulus. When Xs are encountered in the waveform, we replace them by new symbols at every input timestep. Finally, we use our formal X-analysis technique to find Xs in the design. If the formal technique can handle the whole design and the reset sequences, then the X-analysis problem is solved. However, this is impractical due to scalability issues of formal methods. Therefore, we propose three innovations to reduce the complexity of formal analysis.

4.1 Innovation 1: Utilizing the Principle of Most Astonishment

The original X-analysis method [4] reports X problems in all the registers and primary outputs. However, if registers already have Xs in logic simulation, then engineers are already aware of them, and reporting the same problems again will not be helpful. While in many fields the principle of least astonishment is emphasized, in this work we found that the principle of most astonishment is actually more useful. Therefore, we propose to check only the registers that do not have Xs in logic simulation. In our experience, this approach can point out problems that catch designers' attention right away. Runtime can also be reduced because fewer registers need to be checked.

4.2 Innovation 2: Design Partitioning

If formal X-analysis still cannot be performed with innovation 1, we apply innovation 2: design partitioning. Since it is difficult to predict the performance of formal analysis by statically analyzing the design, we propose an iterative approach based on trial-and-error and design hierarchy. To this end, one can start from the second level of hierarchy and run formal X-analysis on each block at the level. To estimate the runtime of each block, one runs a few cycles of the reset sequence and use its runtime to estimate the time required for analyzing the whole sequence. If runtime is still prohibitive for a block, one should go to the next level. This process repeats until blocks that can be efficiently analyzed are identified. We then use the waveform as the input stimulus to formally analyze each block. Finally, the logic between the top level and the partitioned blocks is analyzed by applying stimulus not only from the top module but also from the outputs of the blocks that have been partitioned away.

Since logic simulation is inaccurate when handling Xs, our use of waveforms will create stimulus that may be both over-constrained and under-constrained[1]. It may be over-constrained if there are X-problems in the block that fans out to the current one and the Xs are masked due to X-optimism, and it may be under-constrained because we treat each X as an independent symbol. An under-constrained example is shown in Fig. 2(b), and an over-constrained example is shown in Fig. 2(c).

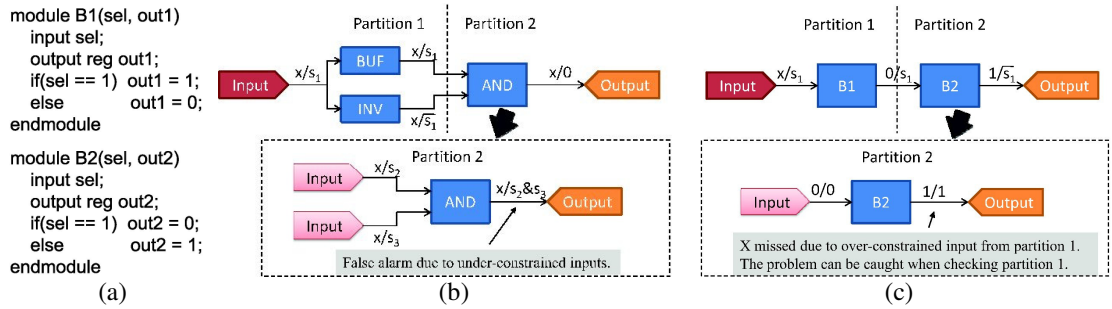

Figure 2: An example to show over-/under-constrained characteristics using stimulus from waveforms and design partitioning. In symbolic simulation, Xs in waveform are replaced by symbols (denoted as s_n). Logic and symbolic simulation results are shown using "logic/symbolic". (a) RTL code for blocks B1 and B2. (b) False alarm at AND gate's output after partitioning due to under-constrained inputs caused by replacing Xs with new symbols. (c) Missed X at B2's output after partitioning due to over-constrained inputs caused by X-optimism in B1. This X-problem will be caught when verifying partition 1.

Our use of waveforms as inputs may produce false alarms but will not miss bugs. False alarms exist because we treat each X as an independent symbol. As Fig. 2(b) shows, doing so reduces the chance to eliminate the Xs. From our case studies, however, this is not a problem because we can eliminate most of the false alarms by having the designers quickly inspect the report. Bugs may be missed because over-constrained conditions in waveforms are caused by X-optimism when simulating the block that has

[1] Over-constrained means legal inputs are not generated, while under-constrained means illegal inputs are generated.

X-problems. Therefore, as long as the problematic block is also checked, we will find unexpected Xs at its registers or outputs and discover the problem. Since in our methodology we check all the blocks, we will not miss X-problems. In Fig. 2(c), the X in B2's output may be missed because simulating B1 produces 0 instead of X at its output, making the input signal in B2 0 instead of X. Since simulating 0 produces 1 on B2's output in symbolic simulation, the X will be missed. However, the source of the problem is actually in partition 1, and the X-problem will be caught when verifying partition 1. After the engineer fixes the problem in B1, the X-problem in B2 will also be fixed.

In general, smaller blocks can reduce the runtime of each partition, but more false negatives will be produced and more partitions need to be checked. So this is a trade-off. If for some reason one does not wish to further split a block but performance issues remain, our next innovation, temporal partitioning, can be applied.

4.3 Innovation 3: Temporal Partitioning

Temporal partitioning creates checkpoints during the reset sequence and then executes formal analysis in intervals. This name is originated from the observation that formal analysis of the sequence is partitioned based on simulation time. The process of X-analysis after temporal partitioning is shown in Fig. 3. We first execute symbolic simulation to a checkpoint and then perform formal X-analysis. At the checkpoint, if X is found, designers check whether the X is acceptable. If the X is not acceptable, then a bug is found. If it is acceptable, then the non-X value in the register should not cause any problem in the future, so we can simply execute logic simulation to the current checkpoint. Next, we perform abstraction [1] by injecting new symbols for all the registers that have Xs at the current checkpoint, switch to symbolic simulation, and then simulate to the next checkpoint. X-analysis is then performed again at the next checkpoint. This process repeats until the whole reset sequence is verified.

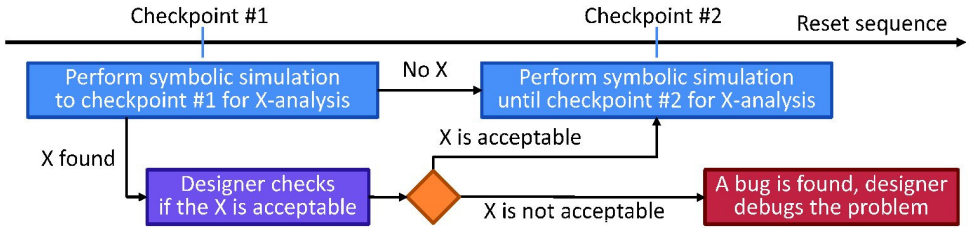

Figure 3: Verification flow after temporal partitioning.

The main advantage of temporal partitioning is that formal X-analysis can be performed for a shorter period of time because new symbols are used to replace the Xs in registers at each checkpoint, thus reducing the complexity of symbolic simulation. However, since such Xs are now free symbols instead of complex Boolean expressions, temporal partitioning creates under-constrained conditions for the next interval. As a result, no bugs will be missed, but there may be false alarms.

For better performance, it is recommended to consider signal activities when deciding the length of intervals. A heuristic is that shorter intervals can be used during hardware reset due to its high activities. Longer intervals can then be used for software sequences to reduce false alarms since most Xs have been removed by hardware reset. It is possible that the number of false alarms can increase to an unacceptable level if each interval becomes too short. However, we did not observe this problem in our case studies.

4.4 Case Studies

We applied the eXact methodology to a six-million gate high-speed tester design. The reset sequence was composed of two phases of hardware reset and two stages of software reset. The hardware sequences were only a couple cycles long. The first stage of software reset (called mapping sequence) was approximately 20,000 cycles long, and the second stage (called pre-pattern sequence) was roughly 500 cycles long. The total length of the sequence was approximately 40,000 cycles long because some cycles are idle. The whole sequence was prepared by the hardware team and approximately 70% of design registers were initialized. We used C++ to generate the transaction level stimulus for the sequence, and

used a SystemC interface to pass the stimulus to the Verilog testbench. Since the sequence was too long and the design was too large for brute-force formal X-analysis, partitioning was necessary. To select appropriate block sizes for partitioning, we first picked a block and ran symbolic simulation[2] for a few cycles. Next, we measured the runtime of symbolic simulation. If the runtime seems to be reasonable (at least 20 cycles per hour in our case), then the block is suitable for formal X-analysis. In this work, we found that 200K-300K is the maximum number of gates that symbolic simulation can handle efficiently, but this number may vary for different designs. Due to the tight schedule of the verification engineer, we did not verify all the blocks but only applied our methodology to five blocks that the engineer was interested in. The characteristics of the blocks are shown in Table 1.

Table 1: Characteristics of design blocks.

Block Name	Register Count		Gate Count	Lines of RTL	Description
	RTL	Gate-level			
alp_pcm	367	9247	30632	7802	Program counter memory which stores vectors used for digital sourcing.
alp_mpg	6739	109882	224986	42937	Memory pattern generator which can automatically generate test pattern for memory testing.
alp_cmem_eng	3614	13148	44291	35115	Capture memory engine which can store captured data.
alp_per_gen	331	5324	11011	5636	Period generator which can generate user period according to users requirement.
alp_lvm	15481	89848	234938	28523	Large vector memory which can be used as extra storage for vector data, capture data, etc.

4.4.1 X-Analysis Results

After we selected the blocks to verify, we chose checkpoint intervals according to simulation speed. Since the verification engineer was more interested in the correctness of the pre-pattern sequence, we performed X-analysis only on the sequence. The X-analysis results of the blocks are shown in Table 2. As our results show, most blocks could be verified without temporal partitioning. However, block alp_cmem_eng took more than one week to run without temporal partitioning and did need 129 checkpoints. Apparently, it is impractical to handle such complex X-analysis problems using brute-force methods, showing that our methodology is useful for handling realistic designs.

[2] Our experiments were performed using a commercial symbolic simulator called Insight [9] running on a Linux server farm. The machines in the server farm have Quad-Core Xeon processors with frequency ranging from 2.93 to 3.16 GHz, and they have memory between 16 GByte and 128 GByte.

Table 2: Verification results using eXact methodology.

Block Name	# CKPTs.	Runtime	Checked/Total Registers(%)	# Found Xs	# Investigated Xs
alp_pcm	1	7hr5m	79.0%	2	2
	58	6m	75.3%	43	
alp_mpg	1	49hr12m	90.3%	793	71
alp_cmem_eng	129	53hr16m	85.3%	848	69
alp_per_gen	1	45m	98.8%	4	4
	2	29m	93.8%	16	
	4	3m	94.1%	18	
	8	3m	93.1%	18	
alp_lvm	1	4hr49m	78.5%	1	1
	2	4hr30m	78.5%	3	

From Table 2, we can see that X-problems still exist in the design even though the design has been heavily verified. The percentage of registers with X-problems, however, is small, suggesting that our under-constrained methods did not create large amounts of false alarms. It is interesting that a relatively large number of Xs were found in alp_mpg and alp_cmem_eng after software reset. More analysis showed that most Xs were from a couple modules that were instantiated several times. After ruling out repeated ones, only 71 Xs in alp_mpg and 69 Xs in alp_cmem_eng really needed to be checked. The designers quickly pinpointed 5 registers for further inspection in in alp_mpg, and they turned out to be false alarms due to temporal partitioning. The rest of the Xs were mostly real but would not affect design correctness.

In Column 4 of Table 2 we provide the percentage of registers that need to be checked according to the method in Section 4.1. By focusing only on the registers without X in logic simulation, we can reduce the number of registers that need to be analyzed. Take alp_mpg for example, we only need to analyze 90.3% of the 6739 RTL word-level registers. If gate-level simulation were to be used for X-analysis, 109,882 bit-level registers needed to be checked, which would be even more inefficient.

To examine the effect of different number of checkpoints on X-analysis runtime and the number of false alarms, we varied the number of checkpoints for alp_per_gen, block alp_lvm and alp_pcm. As shown in Table 2, when the number of checkpoints increased, the runtime for X-analysis decreased, but the number of Xs increased. This trend is consistent with what we predicted earlier — shorter intervals will make X-analysis faster, but it could create more false alarms.

To check how well our methodology can handle long traces, we also tried running alp_per_gen for the whole reset sequence that was almost 40,000 cycles long. Without temporal partitioning, we could not finish symbolic simulation in a week. By partitioning the trace to 19,148 checkpoints, X-analysis of the whole trace finished in 43 hours and 51 minutes, and 32 Xs were reported. This result shows that the temporal partitioning technique scales well to long traces.

4.4.2 Bugs Found and Discussions

In order not to miss bugs, the designer has to check all the found Xs and decide which Xs should be analyzed. In our case study, designers found that most Xs were OK but they were not aware of the Xs in the past. In our experience, approximately 20% of the Xs were false alarms caused by under-constrained conditions, 70% of the Xs did not need to be analyzed because the designers knew right away that the Xs were not important, and only 10% of the Xs needed to be further analyzed. With our methodology, we found 3 bugs that escaped initial verification in block alp_per_gen.

The found X-problems were serious because they could break the data bus and left the chip in a nondeterministic state. This may eventually break all the functionality associated with the chip. Once the bug was found, however, fixing the problem was easy – we routed the reset signal to the problematic register.

Actually, one of the bugs was also found by a traditional method that compares gate-level and RTL simulation results. However, it took the verification team more than one man-month to set up gate-level simulation due to numerous transactors used in behavioral and RTL code. In addition, delta delay caused a

lot of race conditions, which was very difficult to solve in gate-level netlists. Once simulation mismatches between RTL and gate-level designs were found, it took the team another man-month to find the corresponding RTL code that caused the X-problem. However, it only took us three hours to set up the environment for each block, and the X-problem was found within 3 minutes. In addition to the huge reduction in bug-finding time, fixing the bug is also considerably easier because our methodology works earlier in the design cycle at the RTL. Therefore, the designer can easily identify the problematic code and fix it.

5 Issue 2: Fixing Gate-Level Logic Simulation When Xs Exist

In the previous case study we showed that too few Xs due to X-optimism at the RTL can cause problems. In this section we present two other problems, including their solutions, caused by too many Xs in gate-level simulation. The first one is called reset controllability analysis and the second one is called reconvergence path analysis.

5.1 Reset Controllability Analysis

In addition to Xs due to uninitialized registers, here Xs are caused by a physical synthesis optimization that moves the reset signal of a register to its input cone, as shown in Fig. 4. One advantage of this approach is that a smaller cell can be used because the reset signal is no longer required. In addition, routing may become easier since the reset signal does not need to reach the register. However, this optimization causes problems in gate-level simulation due to X-pessimism. As Fig. 4 shows, "1 AND X" becomes X and erroneously makes the output of the NOR gate X instead of 0. This X will propagate to the rest of the design and corrupt the whole simulation. We call this problem "*reset controllability*".

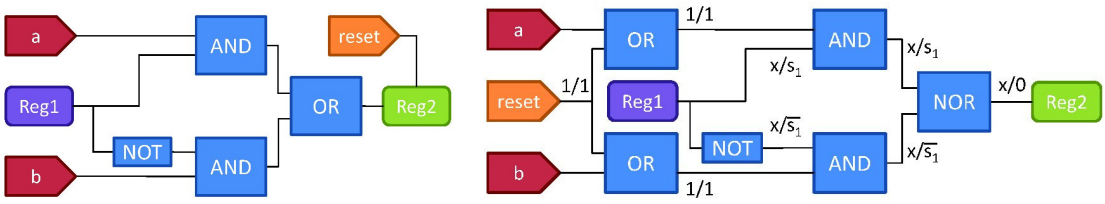

Figure 4: (a) Original netlist. (b) Netlist after optimization where the reset signal has been moved to Reg2's input cone. When reset is 1, input to Reg2 should be 0 even though Reg1 is X. However, due to X-pessimism in logic simulation, Reg2 will be X instead of 0. On the other hand, symbolic simulation uses a symbol (s_1) to represent the value of Reg1 and can produce the correct result: 0. (Simulated values for each wire are shown using logic value/symbolic value).

Our solution is that given a gate-level netlist and a reset sequence, we seek to find all registers whose values are Xs in logic simulation but should really be 0 or 1. In other words, these registers are dominated by a reset signal and should be X-free after reset. To deal with the problem, we reuse the SAT instance developed for X-analysis with one change: in the SAT instance built for X-analysis, only the Xs in registers are replaced with new symbols. To perform reset controllability analysis, we replace all the symbols injected at primary inputs that are not part of the reset sequence with new symbols as well. A miter is then built to check whether the target register can have different values. If a solution can be found, then the target register can be both 0 and 1 and is not controlled by a reset signal. Otherwise, the target register is proven to be constant.

Using this technique, we can accurately identify registers with false Xs after reset. However, to fix gate-level logic simulation, it is necessary to determine what the constant values are so that the correct values can be assigned to the registers. To solve this problem, we replace the symbols in the symbolic traces with random values and run logic simulation to evaluate the symbolic traces. The correct values of registers can then be obtained. By replacing the Xs with those correct values, we can fix gate-level simulation problems.

Note that the solution above has to build a SAT instance for each registers and can be slow. To improve the performance of reset controllability analysis, we propose a new scheme that can reduce the number of SAT calls, and it works as follows. We first execute symbolic simulation to generate the symbolic trace of the target register. Next, we perform random logic simulation on the symbolic trace several times to see if the register value is a constant. If not, then we have already found a counterexample which proves that the target register is not dominated by a reset signal. If all the random patterns produce the same register value, we use the SAT solver to check whether the register can have a different value. If no solution can be found, then the register is controlled by the reset signal and the simulated value is its constant value. Compared with the original algorithm, the number of SAT calls can be greatly reduced due to the use of random simulation. In addition, since we no longer need to duplicate the design and build miters, the SAT instances are much smaller, leading to shorter SAT solving time.

We applied our reset controllability analysis method to a multi-million gate communication chip with a reset sequence that is 74 cycles long. Our experiments were conducted on a Linux workstation with 2 GHz Quad-Core Xeon processors and 48 GByte memory. To solve scalability issues, we partitioned the design into 204 blocks. 266 registers that had Xs in logic simulation but should be X-free were identified. Gate level simulation was then fixed by replacing the Xs in the identified registers with their correct values. It is noteworthy to mention that using the scheme that builds a SAT instance for each register, runtime was 12 hours and 45 minutes. But if we perform random simulation on the symbolic traces first, runtime was reduced to 5 hours and 46 minutes. This result demonstrates the significant performance gain of our new approach for reset controllability analysis. This flow is currently in commercial production use and has solved the gate-level simulation problems for several chips.

5.2 Reconvergence Path Analysis

Reconvergence path analysis is used to repair gate-level simulation problems when the logic in real hardware can eliminate the Xs. The problem can be illustrated using the circuit shown in Fig. 5. Note that this is a simplified version and the problem we observed from the real design was far more complex. In the circuit, Reg1 is initialized to 1 while Reg2 remains X. According to the simulation result at the RTL which happened to be correct, the designer expected to see Reg2 becoming 0 two cycles later. However, due to X-pessimism, Reg1 and Reg2 both become Xs in gate-level logic simulation at the next cycle as shown in Fig. 5(a).

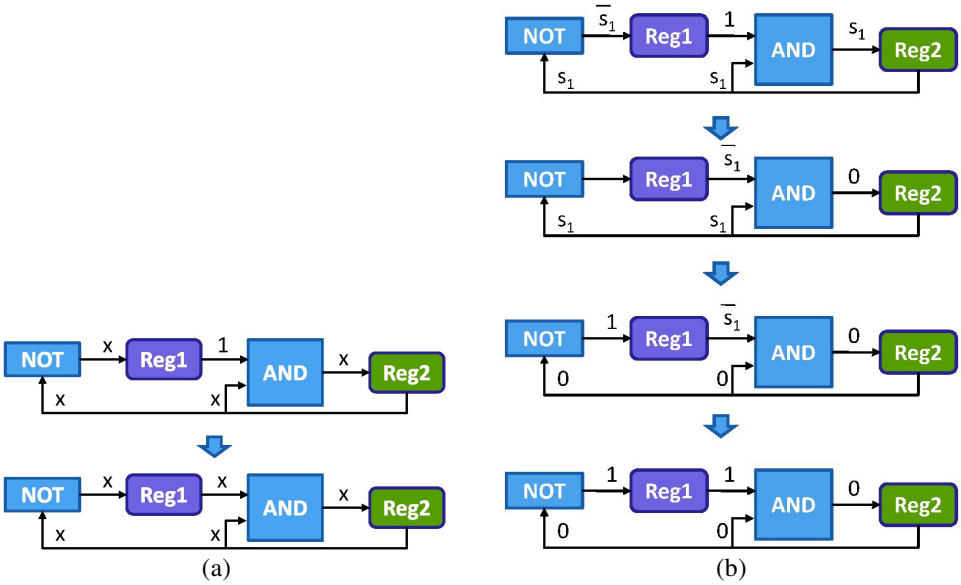

Figure 5: Example for reconvergence path analysis. Reg1 is initialized to 1 while Reg2 remains X. (a) In logic simulation both Reg1 and Reg2 become Xs at the next cycle. (b) In symbolic analysis, symbol s_1 is injected to represent the X, and all the symbols are eliminated three cycles later due to reconvergence path.

To address this problem, we reuse the controllability analysis algorithm and check the Xs to see if they are constant. Unlike reset controllability analysis that only checks the Xs at the reset cycle, here we need to check the Xs for multiple cycles after reset because Xs may be eliminated a few cycles later. As shown in Fig 5(b), our formal X-analysis can correctly prove that Reg2 becomes 0 two cycles later. To repair logic simulation, we force the constant values to the registers at the same cycle when they are proven to be X-free.

Our reconvergence path analysis has been applied to a multi-million gate communication chip from a different company. We analyzed approximately 400 blocks using 46 hours and 2206 false Xs were found. Most runtime (38 hours) was consumed by three of the blocks that were data-path and could be skipped. As a result, analyzing the blocks that the designers were interested in only required 8 hours. The X problems that we found were also found by the verification team, but it took the team a month to identify the problem and find a solution. With eXact, the problem was found in a day after the tool was set up.

6 Conclusions

The dramatic increase in design complexity requires more aggressive optimizations that may create design nondeterminism (X) problems. Due to X-optimism and X-pessimism, such problems cannot be handled accurately by logic simulation, so designers' simulation waveforms may be incorrect. Although formal methods can handle the Xs correctly, scalability remains an issue. In this work we proposed the eXact methodology that leverages the accuracy of formal methods to augment logic simulation so that designers can trust their waveforms again. We applied this methodology to solve three industrial design problems, including finding unexpected Xs after reset and removing incorrect Xs in gate-level logic simulation. The results show that we can find and fix X-problems accurately and efficiently.

Acknowledgment: This research was supported by the National Science Council, Taiwan, under Grant NSC 97-2221-E-002-216-MY3.

References

[1] V. Bertacco, “Scalable Hardware Verification with Symbolic Simulation,” Springer, 2005.

[2] D. Brand, R. A. Bergamaschi and L. Stok, “Be Careful with Don’t Cares,” *ICCAD’95*, pp.83-86.

[3] H. Z. Chou, H. Yu, K. H. Chang, D. Dobbyn, and S. Y. Kuo, “Finding Reset Nondeterminism in RTL Designs – Scalable X-Analysis Methodology and Case Study,” *DATE’10*, pp.1494-1499.

[4] H. Z. Chou, K. H. Chang, and S. Y. Kuo, “Handling Don’t-Care Conditions in High-Level Synthesis and Application for Reducing Initialized Registers,” *DAC’09*, pp. 412-415.

[5] C. Haufe and F. Rogin, “Ad-Hoc Translations to Close Verilog Semantics Gap,” *workshop on DDECS’08*, pp. 1-6.

[6] D. Kaiss, M. Skaba, Z. Hanna, and Z. Khasidashvili, “Industrial Strength SAT-based Alignability Algorithm for Hardware Equivalence Verification,” *FMCAD’07*, pp. 20-26.

[7] R. Ranjan, Y. Antonioli, A. Hunter, and O. Petlin, “Formal verification enables safe X handling,” Dec. 2008. (available at http://www.scdsource.com/article.php? id=324)

[8] M. Turpin, “The Dangers of Living with an X,” SNUG, 2003.

[9] Avery Design Systems Inc., http://www.avery-design.com

BIOs

Kai-hui Chang is a senior member of the technical staff at Avery Design Systems, Andover, MA. His research interests include verification and logic synthesis. He has a Ph. D. in computer science and engineering from the University of Michigan, Ann Arbor. He is a member of the IEEE and the ACM

Hong-zu Chou is currently a senior engineer of SpringSoft, Inc., Hsinchu, Taiwan. He performed the work described in this article while he was with Avery Design Systems. He received the Ph.D. degree in Electrical Engineering from National Taiwan University in 2010. His research interests lie in the areas of formal verification, high-level synthesis and optimization. He is a member of the IEEE.

Haiqian Yu received her Ph.D. in Electrical and Computer Engineering from Northeastern University in 2006. She is currently a verification engineer at Teradyne Inc., North Reading, MA. Her technical interests include FPGA/ASCI verification methodology, reconfigurable computing and high level synthesis.

Dylan Dobbyn is currently a Verification Manager at Teradyne Inc., North Reading, MA, with 20 years industry experience in ASIC verification, and EDA tool development. He holds a MSECE from the University of Massachusetts, Amherst.

Sy-Yen Kuo is a Distinguished Professor at the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan. He has a Ph. D. in Computer Science from the University of Illinois at Urbana-Champaign. His current research interests include dependable systems and networks, mobile computing, and quantum computing and communications. He is a fellow of the IEEE.

Contact Information

Kai-hui Chang; 2 Atwood Lane, Andover, MA, 01810, USA; +1-978-689-7286; changkh@avery-design.com

Hong-zu Chou; No. 25, Industry East Road IV, Science-Based Industrial Park, Hsinchu 300, Taiwan, R.O.C.; +886 (3) 579-4567; hzchou@hotmail.com

Haiqian Yu; 600 Riverpark Drive, North Reading, MA 01864, USA; +1-978-370-2700; haiqian.yu@teradyne.com

Dylan Dobbyn; 600 Riverpark Drive, North Reading, MA 01864, USA; +1-978-370-2700; Dylan_Dobbyn@teradyne.com

Sy-yen Kuo; Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan; +886 (2) 3366-3577; sykuo@cc.ee.ntu.edu.tw