Exhaustive Latch Flow-through Verification with Formal Methods

Baosheng Wang, Sean Ater, Baris Piyade, Brian McMinn, and Borhan Roohipour

Advanced Micro Devices, Inc., 1 AMD Place, Sunnyvale CA 94088, USA

{FirstName.LastName}@amd.com

Antonio Celso Caldeira Junior, Bill Au, and Rajeev Ranjan

Jasper Design Automation, 707 California Street, Mountain View CA 94041, USA
{caldeira, bau, rajeev*}@jasper-da.com*

Abstract

One advantage of level-sensitive latches is their ability to enable implementation of timing-adjustment tricks such as time borrowing. However, extensive use of clock gating techniques to save power may introduce data race conditions in latch-based designs. These data race conditions (i.e., latch flow-through problems) do not exist in purely flop-based designs. To ensure data integrity and avoid yield loss due to random scan values during manufacturing test, intensive verification on those latch flow-through conditions is crucial and required.

In this paper, we present an exhaustive RTL-level verification technique to validate those latch flow-through problems with several formal methods. By utilizing the fan-in/fan-out logic cone reporting capability of equivalency checking tools plus extra homegrown scripts, all potential data racing conditions, in terms of assertions, and their related latches at the source/middle/destination locations are reported. Those assertions are fed into formal tools to ensure that clocks for corresponding latches are never on at same cycle. Notably, "data-blocking" cases, in which data feeding through the destination latch can be blocked when data racing conditions happen, will result in false negative failures. To automatically filter these false negatives, we utilize the multi-cycle-path (MCP) analysis technology from Jasper Design Automation to identify real design issues. Compared with other vector-based verification techniques, the proposed technique is proven fast and exhaustive. This has been successfully demonstrated through multiple experiments in this paper and proven to catch real silicon bugs timely at AMD.

Keywords*: Latches, Clock Gating, Latch Flow-Through, Formal, Multiple Cycle Path Analysis.*

1. Introduction

A latch [1] is a very popular storage element for high-performance designs [2-4] because of its implementation advantages on timing, power, and area over a flip-flop [5]. For example, the well-known time-borrowing or cycle-stealing technique [6] can be easily implemented in a latch-based design.

Due to ever-increasing performance requirements, clock gating emerges as the dominant approach for dynamic power reduction [7-8]. As a result, when such an effective power-reduction method is applied in a latch-based design, both timing and power goals can be easily achieved. Unfortunately, with extensive utilization of clock gating logic, we also introduce data race conditions or the latch flow-through problem.

In a latch-based design, the latch flow-through problem occurs when three or more of the logically adjacent latches are transparent in the same cycle. This is usually due to the fine clock controls for those three latches being not correlated with each other, so a random value combination can easily turn them on at the same time. We named those three latches "source," "middle," and "destination" latches. Obviously, the latch flow-through problem can cause severe data integrity issues that lead to a functional design failure. In addition, the latch flow-through problem can contribute to "overkill" scenarios during manufacturing test and thus reduce yield. This is mainly due to random scan data created for maximizing the test coverage [9].

Exhaustively validating the latch flow-through problem in a timely manner is very challenging:

1. It is difficult to identify all potential latch flow-through problems with any given stimulus because simulation-based methods can never be exhaustive.
2. Verification of all potential latch flow-through problems takes a long time because the majority of those problems occur at higher levels of a design, which is relatively larger.

The only related work publically reported to cover the latch flow-through problem is through automatic test

pattern generation (ATPG) stimulus and ATPG simulation in [9]. However, such methods -- in addition to being late in the design cycle -- fail to exercise all potential latch flow-through paths and usually take a long time to complete.

In this paper, we propose an exhaustive RTL-level verification technique to cover latch flow-through issues via several formal methods. The contributions of this paper are mainly in three aspects:

1. By coupling the logic cone trace capability of equivalence-checking tools and extra homegrown scripts, all potential latch flow-through scenarios are reported in terms of assertions at latch local clocks.
2. By utilizing several techniques to overcome the processing capacity issues, the assertions above are successfully fed into formal verification tools at higher design levels and all latch flow-through instances are exhausitively examined.
3. A special type of data-racing scenarios (see details in Section 2) cannot be easily described with assertions since it involes data paths. As a result, lots of false negatives will be reported by formal tools during prove process. In this paper, we apply the Multiple-Cycle-Path (MCP) analysis technology [10] from Jasper to filter those false negatives, which significantly increases our verification producitivy.

In the rest of this paper, Section 2 explains fundamentals of the latch flow-through issues. Section 3 presents the proposed formal-based latch flow-through verification method, including its flow chart and implementation details. In Section 4, several experiments demonstrate the cost-effectiveness of the proposal.

2. The Latch Flow-through Problem

In a typical synchronous design (including both latch-based and flop-based designs), logic data moves sequentially from one cycle to another. When two or more operations in a synchronous design are performed in the same cycle instead of being executed sequentially, the potential of data racing hazards exists.

Consider a series of three latches, named "source latch" L1, "middle latch" L2, and "destination latch" L3, as shown in Figure 1. Of these, L1 and L3 are scan latches and L2 is not scannable. Fine-gating signals are F1, F2, and F3. The data launched by scan latch L1 may be captured by scan latch L3 in the same cycle if the fine-gating signals assume values {F1, F2, F3} = {1, 0, 1}.

Fixing this latch flow-through problem in Figure 1 is rather obvious: do not let L2 be transparent when CLK is high. This can be implemented by F1 and F2 sharing the same driver signal, F2 and F3 sharing the same driver signal, F2 being always tied as high, or creating an assertion to ensure {F1, F2, F3} never being {1, 0, 1}.

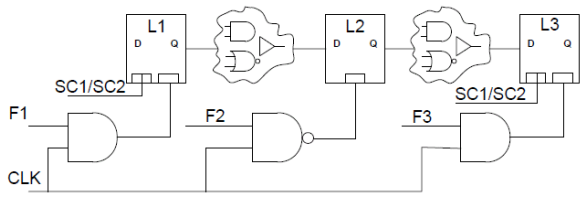

Figure 1. A Typical Latch Flow-through Case

In AMD's next-generation microprocessor core designs [11], there is a unique but valid latch flow-through scenario (named as data-blocking case internally at AMD) in which most of the assertion verification tools are not capable of filtering it out. In Figure 2, the "source latch," "middle latch", and "destination latch" have their own fine control on their clocks (i.e., the signals "FineGater1", "FineGater2", and "FineGater3", respectively). Since those fine controls are driven by different scannable flops, it is easy to create a counter example with {FineGater1, FineGater2, FineGater3} = {1, 0, 1}, which will lead to a latch flow-through condition.

However, when the "middle latch" is transparent and the signal "FineGater2" = 1'b0, the output data from the "middle latch" is blocked by this "FineGater2" signal. In other words, when all three latches are transparent in this scenario, the potential racing data from the "source latch" will not flow through the "middle latch" into the "destination latch". Instead, at this moment, the data feeding into the "destination latch" will be either constant values (as shown in Figure 2) or data from other sources.

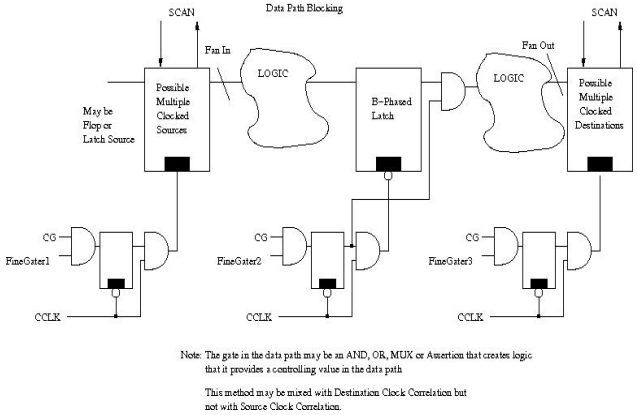

Figure 2. Data-Blocking Scenario

It is worth mentioning that the data-blocking mechanism could be an AND, OR, or MUX gate, or even equations expressed in assertions.

This is a smart implementation to fix latch flow-through problems when a corresponding clock logic change is impossible. However, this introduces a verification challenge because it is so difficult to specify for targeted verification techniques. Luckily, the MCP technology from Jasper provides a unique capability to filter those

false negatives produced by traditional assertion-based verification tools.

Lastly, there might be data racing conditions that involve two active low latches and one active high latch. However, such a design is very rare in practice. Hence, in this paper, we will focus only on the scenarios shown in Figure 1 for discussions.

3. The Proposed Verification Technique

The proposed verification technique consists of three steps:

1. Report all potential latch flow-through cases as assertions on local clock gaters.
2. Prove all assertions generated in step 1.
3. Filter failures produced in step 2 using MCP technology from Jasper.

These steps are elaborated in this section below. The overall proposed flowchart is shown in Figure 3.

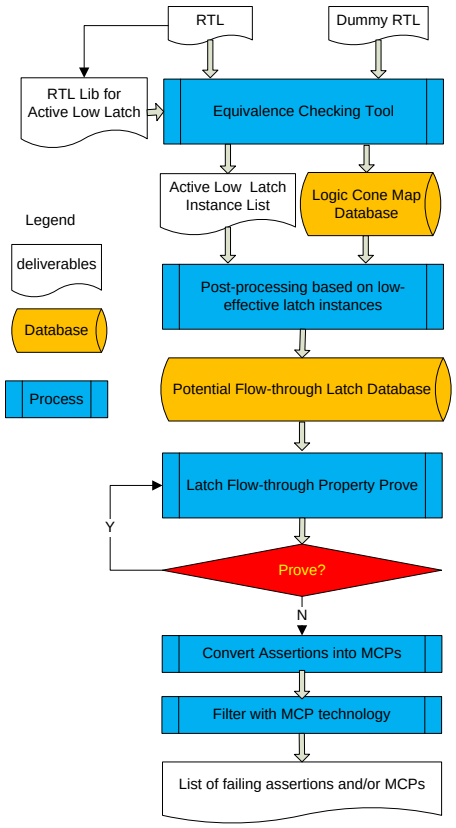

Figure 3. The Proposed Verification Technique

3.1 Identification

Unlike other stimulus-based methods that rely on "good" vectors to exercise the flow-through latch cases, the proposed technique explicitly locates all those potential verification targets. In other words, the proposal does not depend on any vectors and its identification process is exhaustive. The detailed steps of such a process are:

1. Create a dummy RTL file without contents so we can feed both the RTL under verification and this dummy RTL into equivalence-checking tools [12] to create the design database through synthesis.
2. Based on naming conventions for a active low latch library in RTL, all the instances of those active low latches are listed into a file through commands of the equivalence-checking tool. Meanwhile, the logic cone map database, which states a particular storage element and its fan-in/fan-out states, is created and stored into a text file.
3. By checking each latch instance from the active low latch instance list within the logic cone map database, both its fan-in/fan-out logic states can be looked up and reported as a latch sequence, where their respective clock gaters are formed into an OVL assertion [13]. An example of such a latch sequence is as follows:

assert_never #(.msg("ABA gaters open at the same time")) AssertABA1 (.reset_n(1'b1), .clk(~CLK), .test_expr ((foo.foo2.I_foo3.G) & ~(boo.boo2.I_boo3.G) & (poo.poo2.I_poo3.G)));

// Source latch gater = /foo/foo2/I_foo3

// Middle latch gater = /boo/boo2/I_boo3

// Destination latch gater = /poo/poo2/I_poo3

// Latch Sequences:

// /foo/foo2/I_src1 /boo/boo2/I_mid1 /poo/poo2/I_dest1

// /foo/foo2/I_src1 /boo/boo2/I_mid2 /poo/poo2/I_dest2

// /foo/foo2/I_src1 /boo/boo2/I_mid2 /poo/poo2/I_dest3

… … …

assert_never #(.msg("ABA gaters open at the same time")) AssertABA2 … …

The latch sequence file not only includes all corresponding latches but also groups those related latches together if they share the same clock gaters respectively. By doing so, the number of latch scequences to be examined can be significantly reduced, greatly accelerating the assertion analysis time.

3.2 Assertion Prove

By following traditional assertion-based verification (ABV) methodology, the assertions generated through steps described in Section 3.1 are fed into formal verification tools, e.g., the JasperGold tool from Jasper Design Automation Inc., for examining those assertions exhaustively. As is the case with typical formal-based ABV tools, the capacity issue remains the main challenge due to design size, the related state explosion as well large number of assertions associated with this flow. With

thorough analysis on latch flow-through problems, fortunately, we are able to overcome this challenge by adopting following techniques:

1. Divide the total assertions into multiple groups based on productivity requirements. For example, if empirical data reflects that in a 3 hour proof run time 1000 assertions can be verified, then the grouping mechanism can be implemented using the "ifdef" method, i.e., letting muitple formal engines work on different groups of assertions simultaneously.
2. Blackbox scannable flops which are not targets of latch flow-through instances. There are a couple of reasons why this method is effective in improving the tool processing capacity and the resulting run time efficiency:
    a. Scannable flops produces random data so that maximum manufacturing test coverage can be obtained. In other words, all scannable flops during latch flow-through verification must produce random values. This verification requirement happens to be consistent with the effect of blackboxing those flops.
    b. Purely-latching-based designs are very costly. In reality, latches are only utilized in timing-critical blocks and most of the storage elements in a typical high-performance design are still flip-flops. As a result, the assocated scanable flops interfacing with latches are with very low percentage.

It is important to note that the scannable flops which are part of latch flow-through instances can not be blackboxed. This is to avoid unnecessary false negatives since formal tools needs to analyze the functionality of those in-target scannable flops during proof process.

3.3 Data Blocking Filtering

As stated in Section 2, data-blocking is a very popular mechanism to fix latch flow-through problem without requiring to modify the existing latch clocking scenario. However, if the assertions formed in Section 3.1 are utilized to express this type of clock and data structure, many assertions might fail without the violation of underlying latch flow-through scenario (essentially false negatives). This is because of its unique data blocking mechanism where data from "source/middle" latch will NOT be propagated to "destination" latch when all latches are transparent. Manual analysis of all assertion failures to eliminate false negatives will be extremely prohibitive and verification productivity will be heavily impacted.

One possible way to resolve this productivity issue is to re-write the assertions by incorporating data path components for each instance. Not only this solution is difficult to implement with much higher complexity of identification process, it will also result in explosion of assertion count. A more practical solution is to develop a post-processing mechanism to automatically filter those false negatives. This is achieved through Multi-Cycle-Path analysis, a solution available in the JasperGold product from Jasper Design Automation Inc.

An introduction to MCP can be found at: http://ece.gmu.edu/coursewebpages/ECE/ECE645/S09/resources/Multi-Cycle_Path_Tutorial.pdf. Essentially, MCP analysis checks that three end points of a path requires at least N cycles for value propagation. The typical syntax of such an MCP command exampled with latch flow-through instance is as follows:

assert –mcp –from "source" latch instance –through "middle" latch instance –to "destination" latch instance –cycle 2 –name ABA_mcp1

If the underlying formal engine in MCP technology is able to identify a counter example where the data can be propagated from "source" latch, through "middle" latch, to "destination" latch, within 2 evaluation cycles, such a failure reported from the assertion prove process in Step 3.2 could be a real data-racing incidence. This is because the formal engine could not find the tight correlation between the clock and data paths for this latch sequence. On the other hand, once such a MCP property is proven, these three latches instances successfully form a data-blocking scenario.

During MCP proof process, we encountered couple of tool capacity issues: (i) large run times due to large number of MCPs (ii) getting bounded proof results on MCP assertions instead of the full proof. We resolve the first issue by re-grouping those bit-blasted instances into a vector scenario so that the number of MCP formulations is equal to the failing assertion count, which is relatively small in practice. We resolved the second issue by starting the MCP analysis with completely uninitialized state elements. With such a general starting point, the bounded proof of depth 2 is sufficient to get the necessary confidence (since the path is of length 2).

4. Experiments

The experiments conducted in this paper are mainly for two purposes:

1. We use two block-level modules (BLMs) in [14] to prove the effectiveness of our method. Both a data blocking case and a real data-racing scenario are reported in this experiment which substantiate the capabilities of the flow.
2. We demonstrate that our flow is capable of processing large components in [14] with reasonable run time.

In the first experiment, two BLMs, BLM1 and BLM2, are selected as test cases. To identify latch-dominant design structures that are deemed as critical-area for this flow-through problem, BLM1 is a block without memory arrays

while BLM2 is a block with memory arrays. The details of the report after running our program are shown in Table 1.

Table 1. Latch Flow-through Verification Results of BLMs

BLMs	# (Latch Sequences)	# (assertions)	# (data blocking cases)	# (real latch flow-through scenarios)
BLM1	588	11	78	0
BLM2	10,730	638	530	16

From Table 1, it can be seen that more latch sequences are found in a BLM with macros. This is easily understandable because macros are the building blocks of microprocessor core design. The second observation is that our flow has successfully differentiated the real data-racing scenarios from the data-blocking cases. For example, for BLM2, we identify 16 latch scenarios that are causing a latch flow-through problem (confirmed later by designers). An example of such a violation can be shown in Figure 4:

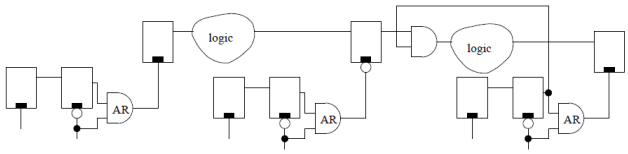

Figure 4. A Latch Flow-through Violation Found in BLM2

In this example, the data-blocking mechanism should come from the "middle latch's" fine control while it is designed with the "destination latch's" fine control.

In the second experiment, we run the proposed program on all components in [14] to ensure that a daily latch flow-through verification regression at the component level is possible. Table 2 lists the flow run results for 3 typical components:

Table 2. Latch Flow-through Verification Results of 3 Components

Component	C0	C1	C2
Computing Memory	10G	26G	26G
# Latch Sequences	456k	2.04m	1.893m
# assertions	201	1324	18014
# scannable flops	15k	107k	89k
% scannable flops blackboxed	98.9	95.8	92.1
Assertion generation time (hours)	0.7	3.0	5.7
Assertion+MCP prove time (hours)	3.0	3.3	3.7
Total run time (hours)	3.7	6.3	9.4

From Table 2, compared with the latch flow-through verification at BLM level, the latch sequences reported at the component increase dramatically. This clearly shows latch flow-through verification at a high design level is essential because there are more latches across BLMs in this microprocessor core.

Run time on large components of our proposal is very reasonable, e.g., below 10 hours in total, which fits well with the daily-regression system. Blackboxing the non-target scannable flops and grouping large number of assertions account for the major productivity improvements.

5. Conclusion

Latch flow-through problems become more and more serious for high-performance latch-based designs, especially when their dynamic power budget on clock gaters is limited. Extensive validation of such design violations is unavoidable. With reasonable runtime, the proposed verification technique is effective and exhaustive in identifying and/or catching such design failures, even at an early design stage because it can operate on RTL description. Experiments performed in this paper demonstrate that the proposed flow is capable of identifying those design violations and can exhaustively verify such problems in a reasonable run time on large designs.

6. Acknowledgements

We thank anonymous reviewers for their insightful comments and suggestions on this work.

7. References

[1] A. Chandrakasan, W. Bowhill, F. Fox, *Design of High-Performance Microprocessor Circuits*. New York, NY: Wiley-IEEE Press, 2001.

[2] M. Butler, "Bulldozer – a new approach to multi-thread compute performance," *The IEEE 22nd HotChips conference – a symposium on high performance chips*, Session 7.2, August 22-24, 2010.

[3] T. Wood, "Test and debug features of the AMD-K7™ microprocessor," in *Proceedings of IEEE International Test Conference (ITC)*, 1999, pp. 130-136.

[4] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, R. Kumar, "An Integrated Quad-Core Opteron Processor," *The IEEE International Solid-State Circuits Conference (ISSCC)*, 2007, pp. 102-103.

[5] Electronics Bus, Latch Vs Flip-Flop Registers in Digital Design, http://electronicsbus.com/latch-vs-flip-flop

[6] S. Lee, S. Paik, Y. Shin, "Retiming and time borrowing: Optimizing high-performance pulsed-latch-based circuits," *The IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, pp. 375-380, 2009.

[7] D.R. Sulaiman, "Using clock gating technique for energy reduction in portable computers," *The International Conference on Computer and Communication Engineering*, pp. 839-842, 2008.

[8] M. Kunes, et al., “Reducing Power Consumption of an Embedded DSP Platform through the Clock-Gating Technique,” *The International Conference on Field Programmable Logic and Applications (FPL),* pp. 336-339, 2010.

[9] M. Yilmaz, et al., “The scan-DFT features of AMD’s next-generation microprocessor core,” in *Proceedings of IEEE International Test Conference (ITC)*, 2010, pp. 2.1.1-2.1.10.

[10] Jasper Design Automation, JasperGold Verificaton System and JasperCore Command Reference Manual, Version 7.3, August 2011

[11] S. Arekapudi, E. Busta, C. Dietz, T. Fischer, M. Golden, S. Hilker, A. Horiuchi, K. Hurd, D. Johnson, H. McIntyre, S. Naffziger, J. Vinh, J. White, K. Wilcox, “Design Solutions for the Bulldozer 32nm SOI 2-Core Processor Module in an 8-Core CPU,” to appear in *IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 20-24, 2011.

[12] Cadence. Encounter Conformal Equivalence Checking User Guide, Product Version 8.1, June 2009.

[13] E. Clarke, Jr., O. Grumberg, D. Peled, *Model Checking*. MIT Press, 1999.

[14] B. Roohipour, et al.,, “Exhaustive Equivalence Checking on AMD’s Next-generation Microprocessor Core,” to appear in Design & Verification Conference & Exhibition (DVCON), March 2011