

Preventing Glitch Nightmares on CDC Paths: The Three Witches

Jian-Hua Yan, MediaTek Inc, Beijing, China
Ping Yeung, Mentor, a Siemens Business, Fremont, CA
Stewart Li, Mentor, a Siemens Business, Taipei, Taiwan
Sulabh-Kumar Khare, Mentor, a Siemens Business, Noida, India

Abstract - As we are investing more in automotive and safety-critical designs, there is a renewed focus on design reliability. Glitches on clock-domain-crossing (CDC) signals will undoubtedly reduce reliability and lead to potential silicon failures. An increasing number of companies are deploying CDC verification at both the RTL and the gate-level. To identify potential glitches at the gate-level, we have been using an automatic formal-based glitch detection methodology. Previously, we have been focusing on preventing and catching glitches on data multiplexing paths. After deployed gate-level CDC on several projects, we had gained more experience. We learned that it is even more critical to verify glitches on the unsynchronized and combinational CDC paths.

I. INTRODUCTION

Based on our experience, during the synthesis and power optimization process, CDC errors can be introduced. Hence, CDC verification is not only necessary at the RT-level; it is also essential at the gate-level.

At the RTL, we focus on identifying the clock domains and CDC paths by recognizing the CDC structures and schemes. At the gate-level, CDC paths with multiplexer or combinational logic are often prone to glitch defects that can be introduced during the synthesis, timing, and power optimization process. If CDC verification is only done at RTL, such glitch defects can easily be missed and lead to costly post-silicon chip failure.

This paper follows our previous inclusion in DVCon 2018, *Preventing Chip-Killing Glitches on CDC Paths with Automated Formal Analysis*[1]. After deployed gate-level CDC on several projects, we had gained experience and learned a few valuable lessons:

1. Potential glitches can be generated from different types of CDC paths. It is essential to identify them clearly so that algorithms can be created to analyze them effectively.
2. Gate-level CDC analysis takes a long time (from a few hours to a few days). If refinements have to be made, it is much more efficient to re-start from the last step instead of beginning again.
3. As the design complexity continues to grow, it is essential to support a heterogeneous hierarchical approach. At the same time, as memory consumption continues to grow, it is useful to identify parallelism in the process and enable a divide-and-conquer strategy.

In this paper, we first explain the glitch problems in various types of CDC paths. Then, we summarize the automatic formal-based glitch detection methodology [2] that we have deployed for a few years. The methodology utilizes structural CDC analysis, expression analysis, and formal methods to prune and prove real glitches in the design. In order to handle much more complex designs with even longer run time, we have fractionated the previous methodology into a much more flexible hierarchical multi-stage and multi-processing flow. We describe the stages of the flow and how to achieve parallel processing. In the end, we summarize the results from a few projects and illustrate design glitches found by the project teams.

II. THE THREE Witches OF GLITCHES

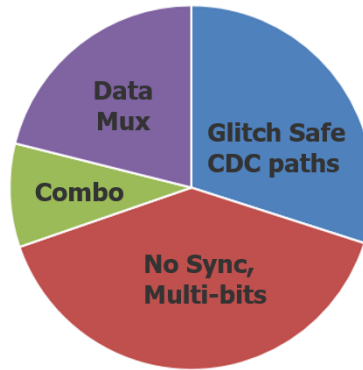


Figure 1: Distribution of CDC paths in a design project

A glitch generated in one clock domain can be captured unintentionally by a register in the receiving clock domain if the glitch passes through a CDC path. Although some unsynchronized CDC paths may be deemed safe due to stable signals such as configuration or mode registers, most CDC paths are not.

The three types of CDC paths, hence witches [3] that will cause potential glitches are:

1. Data mux synchronized paths,
2. Combinational logic before synchronizers,
3. Un-synchronized single-bit or multi-bit paths

Figure 1 summarizes the major types of CDC paths in a design. Except for the glitch safe CDC paths, static and dynamic combinational glitches can happen on the other types of CDC paths.

A. Glitch safe CDC paths

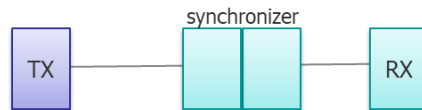


Figure 2: Combinational logic before synchronizer path

In a glitch safe CDC path, the TX register drives the synchronizer directly without any combinational logic before the synchronizer. The synchronizer can be a back-to-back two DFFs, shift register synchronizer, pulse synchronizer, latch-based synchronizer, etc.

B. Combinational logic before synchronizers

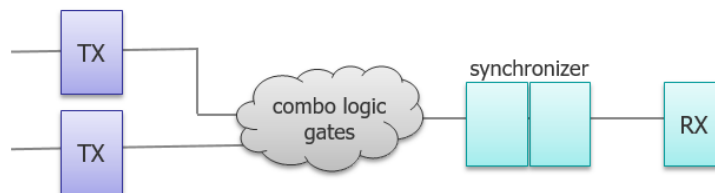


Figure 3: Combinational logic before the synchronizer path

When there is combinational logic before a good synchronizer, the CDC path is not glitch-safe anymore. Combinational logic can introduce combinational glitches before the synchronizers. And the synchronizer may sample the wrong value into the receiving domain unintentionally. Hence, this structure is a CDC violation at the RT-level. However, it can be waived if the driving signals are stable or mutually exclusive. In most cases, this was done carefully after design review, but occasionally they were waived by mistake.

C. Un-synchronized single-bit or multi-bit paths

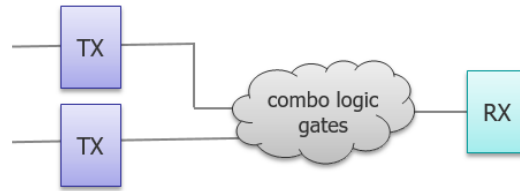


Figure 4: Un-synchronized single-bit or multi-bit path

In any unsynchronized CDC path, glitches generated by combinational logic can easily be sampled by the receiving register unintentionally. Although this structure is a violation at the RT-level, it may not be fixed. The violation can be waived if the driving signals are stable. They may remain in the design due to oversight by designers or waive by mistake. Moreover, as we have observed, they can also be introduced when project teams are performing aggressive power optimization or adding test structures and safety mechanisms into the design.

D. Data mux synchronized paths

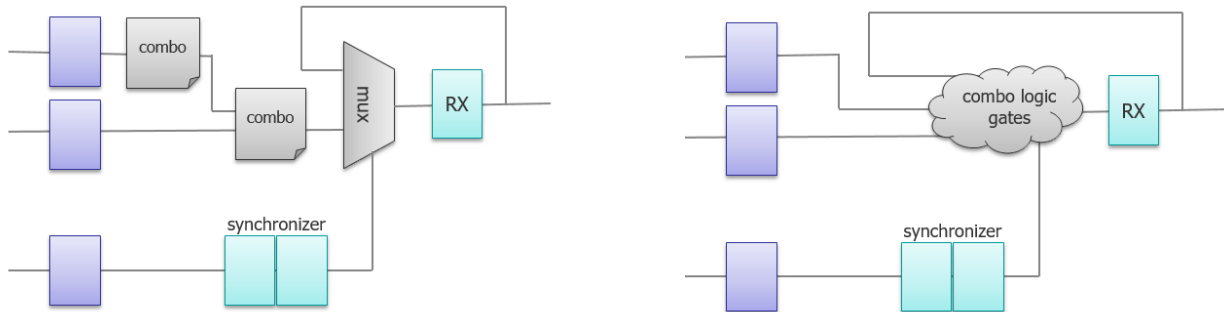


Figure 5: Data mux CDC path (before and after synthesis)

Data mux synchronized paths have to follow the protocol that the transmit data should not change when data select multiplexer is enabled. Although this is a good CDC scheme at the RT-level, the data multiplexer will be optimized during synthesis with other combinational logic in the CDC path. As a result, the protocol is violated. We have adopted a careful setup during synthesis to prevent the data select logic from moving away from the RX register, but there is no guarantee. When this happens, the data select signal is no longer controlling the sampling of the data from the transmit domain. Glitches can happen, and wrong values will likely be sampled into the receiving domain.

Previously, we have been focusing extensively on preventing and catching glitches on the data multiplexing paths[1] in the design. Two years ago, we had encountered a glitch in silicon on an unsynchronized CDC path. As a result, we learned that it is equally critical to verify glitches on the unsynchronized CDC paths, data multiplexing paths, and combinational CDC paths.

III. FORMAL-BASED GLITCH DETECTION METHODOLOGY

We have adopted the automatic formal based glitch detection methodology[2]. The approach utilizes a combination of structural CDC analysis, expression analysis, and formal methods to identify and prove real glitches in the gate-level netlist.

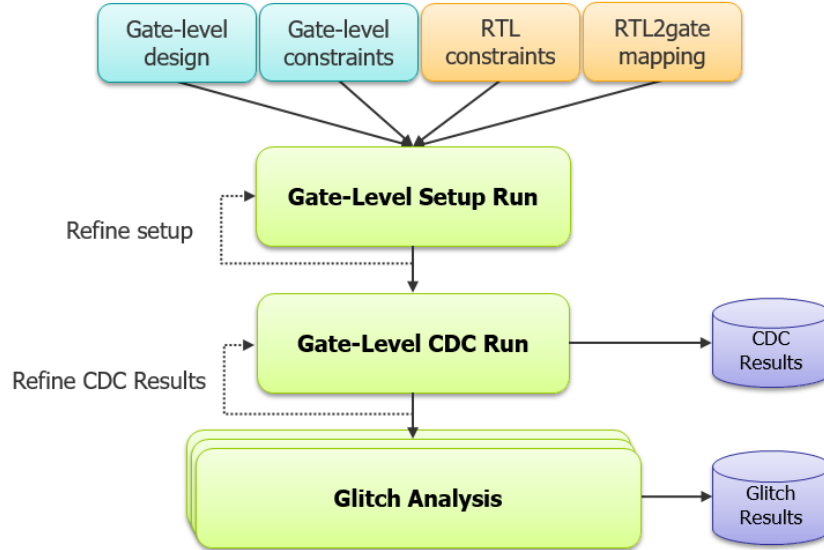


Figure 6: Multi-stage gate-level CDC and glitch analysis

The approach has various stages that are interlinked, and the entire methodology is automated. Users can refine each stage independently before moving to the next stage.

A. Gate-level setup

As in RTL CDC analysis, the user needs to specify appropriate design constraints to obtain accurate CDC results. Hence, it is important that the constraints used at the RTL can be applied as-is for the gate-level CDC run. The tool understands the naming transformation during synthesis and utilizes it to convert signal and module names from RTL format to gate-level format.

This automation saves user effort in creating and refining the setup through multiple iterations and reduces false noise as the constraints have already been validated in RTL CDC analysis. Additional RTL-to-gate-level name mapping can also be provided to refine the accuracy of the RTL constraints. In our projects, we have added explicit RTL-to-gate mappings to handle multi-dimensional arrays.

The tool can also infer constraints for logic that was not present in the RTL but got inserted during scan and design-for-test stages. We reviewed and used this constraint guidance provided by the tool to focus on the design's functional mode of operation.

B. Gate-level CDC run

The goal of gate-level CDC analysis is to identify the safe, unsafe, and waived CDC paths. The gate-level CDC paths that do not contain any combinational logic are glitch-free. Hence they are the safe paths. The unsafe CDC paths are the 3 witches: unsynchronized, combinational, and data mux synchronized CDC paths.

As some of these CDC paths have already been reviewed and waived at the RTL, the tool supports the reuse of waivers from the RTL CDC analysis. The paths that have been qualified by users as stable or false during RTL analysis need not be re-analyzed at the gate level. The RTL waivers are automatically transformed and applied to gate-level results. It ensures that the gate-level results have minimal noise, and only new issues identified at the gate-level are highlighted.

C. Gate-level glitch analysis

The final stage is to do a comprehensive expression analysis of the combinational logic tree in the CDC path to identify potential glitch candidates that can cause the glitch to propagate. This further prunes the list of glitch candidates; however, the real challenge is identifying the scenarios under which the glitch propagates. To accomplish this, we utilize formal engines to verify the glitch propagation condition and conclusively give counterexamples under which the glitch will propagate.

In addition, for the glitch prone paths, the tool performed further analysis to identify the exact location at which the signal and its complementary term will converge, hence resulting in a potential glitch during design operation. The counterexample and the convergence of logic provide the essential information to help users visualize the exact scenario under which a glitch can cause a failure.

In one of our projects from last year, there were close to 200,000 unsafe CDC paths at the gate-level. As each CDC path is relatively independent of each other, there is a significant benefit of performing glitch analysis concurrently in a server farm environment. Using the gate-level CDC result database from stage 2, we classify the CDC paths into different groups based on CDC schemes, priority, and design hierarchy. It enabled us to start seeing results much sooner and get all results within 24 hours.

This methodology has many advantages:

- High quality of results - A formal based approach for glitch verification generates less noisy and much higher quality results (a few glitches per thousand of paths), hence significantly reduces debug and triage effort.
- Easy to debug - With expression analysis and formal methods, the tool can identify the exact paths contributing to the glitch scenario and can pinpoint the convergence point where potential glitch can be generated.
- Ability to handle large SoCs - Hierarchical approach can be utilized to complete the CDC and glitch analysis on IP blocks first. Then, they are represented as grey boxes for analysis at the top-level.
- Quick refinement - With a multi-stage process for gate-level CDC and glitch analysis, it enables incremental refinements with much-improved turnaround time and eliminates failure within a single process.
- Multi-processing – As glitch analysis can be performed on multiple servers concurrently, it enables much more efficient usage of servers and memory.

IV. RESULTS

		Project 1	Project 2	Project 3
DMUX	Paths	14505	20015	4697
	Glitches	201	330	16
	%	1.39%	1.65%	0.34%
Combo Logic	Paths	12359	12058	5257
	Glitches	228	285	15
	%	1.84%	2.36%	0.28%
No Sync	Paths	183911	132758	43521
	Glitches	1022	1392	52
	%	0.56%	1.05%	0.12%

Table 1: CDC paths and glitch results in SOC projects

Table 1 summarizes the CDC paths and glitches identified in recent SoC projects. There are a few observations:

- Compared with combinational logic before synchronizer and data-mux synchronized paths, there are a significant amount of unsynchronized CDC paths in the design,
- Although there are a lot of CDC paths from the three witches, expression analysis and formal methods help reduce noise by identifying the real glitches only

		Project 1	Project 2	Project 3
Design	#cell	17M	16M	8.7M
	#register	844K	744K	462K
	#Hierarchical Block	71	3	21
Run time	Gate-level setup	3.45 hrs	2.15 hrs	3.56 hrs
	Gate-level CDC	3.66 hrs	2.40 hrs	3.75 hrs
	Glitch analysis	1.76 hrs	2.00 hrs	1.52 hrs

Table 2: Performance of gate-level CDC glitch analysis

Table 2 summarizes the complexity and the run time of the SoC projects. We spent a significant amount of time in the gate-level CDC setup stage with the multi-stage processing flow. After importing the setup and directives from the RT-level, we

- Refined the transformation of the RTL directives to gate-level directives
- Added additional constraints to disable scan, test, and power control logics
- Reviewed the clock trees to make sure IPs and sub-blocks are connected correctly

Then, by understanding the RTL setups and automatically convert RTL waivers to the gate level, the tool identified the waived paths and minimized noise to be reported during gate-level CDC analysis.

Finally, we performed a comprehensive glitch analysis on the three witches: unsynchronized CDC paths, data multiplexing paths, and combinational CDC paths concurrently. The multi-processing approach was set up by prioritizing the CDC paths:

- Examine the critical parts of the design where controllers are sampling the CDC signals,
- Focus on the three different types of CDC paths independently,
- Analyze the critical static glitches first, followed by dynamic glitches

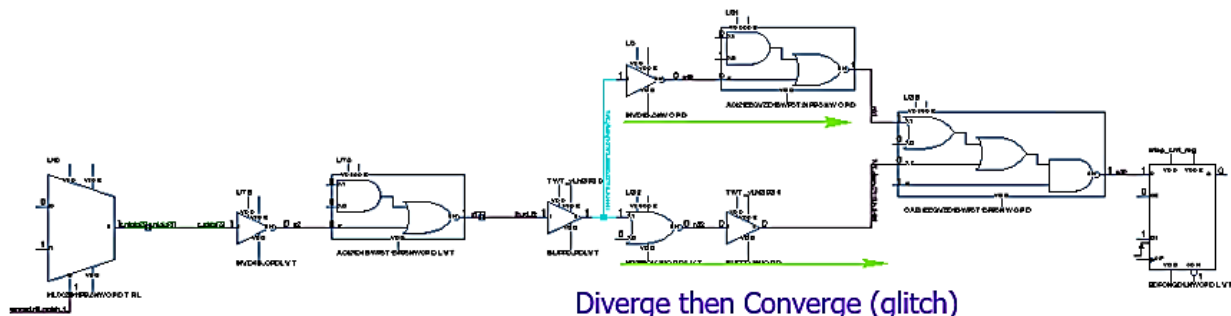


Figure 7: Potential glitch in unsynchronized CDC paths

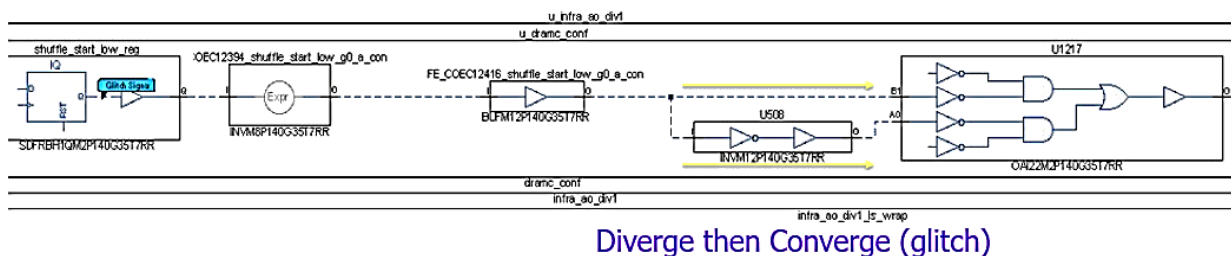


Figure 8: Potential glitch in unsynchronized CDC paths

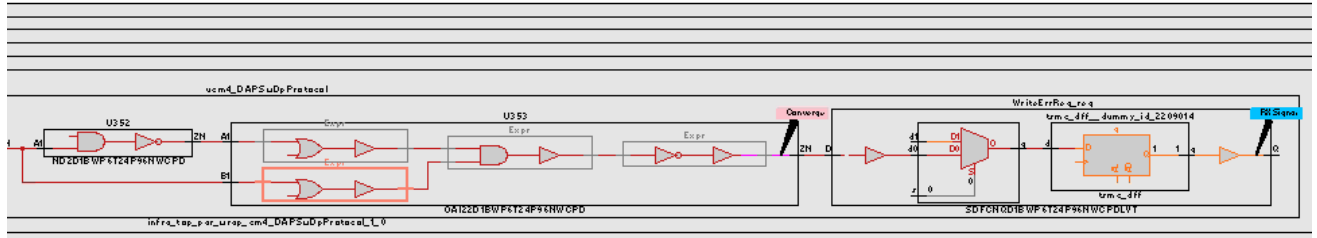


Figure 9: Potential glitch in Data mux synchronized paths

Figures 7 and 8 show two of the potential glitches in the unsynchronized CDC paths. Figure 7 is a real glitch situation that had been fixed with multiple ECOs. Figure 8 is an acceptable situation. Although the signal (in green) diverge and re-converge later, the two AND gates inside the cell will not be enabled at the same time. The two green paths are mutually exclusive functionally.

Figures 9 shows potential glitches in a data mux synchronized path.

V. SUMMARY

With the hierarchical multi-stage and multi-processing, we can perform gate-level CDC and glitch analysis much efficiently than before. By transforming the RTL directives and waivers to the gate-level, we reused the hard works that have been performed at the RTL. Though critical, gate-level glitch analysis has not been adopted widely because of the noisy results. Expression analysis and formal methods validate the CDC paths. They identify the exact paths contributing to the glitch scenario and pinpoint the potential glitch in the design.

REFERENCES

- [1] Jackie Hsiung, Ashish Hari, Sulabh-Kumar Khare, "Preventing Chip-Killing Glitches on CDC Paths with Automated Formal Analysis", Mediatek, DVCon 2018.
- [2] Anwesha Choudhury, Ashish Hari, "Accelerating CDC Verification Closure on Gate-Level Designs", DVCon 2017
- [3] The Three Witches are characters in [William Shakespeare's](#) play [Macbeth](#) (c. 1603–1607).