Formal Verification of Clock Domain Crossing using Gate-level Models of Metastable Flip-Flops

Ghaith Tarawneh
Institute of Neuroscience
Newcastle University, UK
Email: ghaith.tarawneh@ncl.ac.uk

Andrey Mokhov, Alex Yakovlev
School of Electrical and Electronic Engineering
Newcastle University, UK
Email: {andrey.mokhov, alex.yakovlev}@ncl.ac.uk

***Abstract*—Verifying clock domain boundary logic is a major challenge to the design of modern multi-clock systems. We present a novel verification approach that addresses the issue of domain crossing failures at a fundamental level. The approach relies on substituting flip-flops with model circuits and applying topological transformations to simulate the transfer of timing violations in gate-level netlists. This makes timing violations and their effects reproducible in discrete cycle-based simulation and amenable for identification and debugging similar to typical synchronous design failures. We show that this approach, when combined with formal verification, is inherently capable of reproducing many of the problematic issues at clock domain boundaries and outperforms the structural and functional heuristics used by state of the art commercial tools in several respects. It reports fewer false positives, can be applied to non-stereotypical designs, can determine failure consequences, can demonstrate failures in signal waveforms and requires no input from the designer about what design patterns are used. Case examples and verification results of several multi-clock testbench designs are presented.**

***Index Terms*—Clock domain crossing, formal verification, metastability.**

I. Introduction

The reliable transfer of data and control signals across clock domain boundaries (Clock Domain Crossing or CDC) is a well-recognized thorny issue in System-on-Chip (SoC) design [1], [2]. Signals that are generated in one clock domain and latched in another may violate the setup and hold time conditions of their destination flip-flops and drive them into metastable states. The latter may then exhibit prolonged clk-to-q transitions and violate the input timing conditions of their own destinations during subsequent cycles [3]. The consequences of such events are design-dependent and difficult to predict but can in the worst case include invalid (and potentially irrecoverable) state transitions.

A. The Challenges of CDC Verification

Despite their potential severity, clock domain boundary timing violations and their ensuing failures are entirely transparent to most digital simulation, synthesis and verification tools. This is because digital tools are built from the grounds up to handle strictly-synchronous designs in which all signals are timed according to a single clock. Flip-flops in these tools are typically abstracted as idealized storage elements that do not experience input timing violations, become metastable or exhibit prolonged output transitions. A multi-clock design may thus be synthesized correctly and pass conventional testbench and assertion-based verification but still fail catastrophically once implemented in silicon.

The inability to capture CDC failures at a digital level led to the development of specialized linting and verification tools (e.g. Conformal CDC from Cadence and Questa CDC from Mentor Graphics) that check designs against lists of pre-established structural rules and generate warnings for violations of safe design practices. Examples of these rules include (1) using synchronizer circuits to re-time control signals, (2) avoiding the use of combinational logic in crossover paths and (3) avoiding the synchronization of data signals (unless they are Gray-coded). These rules are heuristics based on theoretical understanding of clock domain crossing and metastable flip-flop behaviour. They guarantee correct behavior by preventing the design from satisfying known structural preconditions of CDC failures (e.g. lack of a synchronizer circuit). This is a safe design approach but is conservative and has two main disadvantages. First, it may report issues that the design does not suffer from in practice because it does not take into consideration design-specific properties and functional constraints. For example, quasi-stable signals from status/configuration registers that are set by software during system initialization can be safely used by other clock domains without synchronization (in violation of rule 1). Structural inspection alone cannot identify such cases and would therefore issue missing synchronizer warnings while no synchronizers are actually needed. False-positive warnings such as these can consume valuable verification time to either double-check or configure the linting tool to ignore [4], particularly in more complex scenarios in which it is not obvious to the designer whether a true CDC issue actually exists, the exact situation in which reliable issue identification is most needed.

The other main disadvantage of adhering to heuristic CDC rules is that they prohibit the exploration of design options that may be better optimized and nevertheless completely safe. Insertion of combinational logic in crossover paths, for example, can help meet performance goals but is prohibited under rule 2 because sender flip-flop transitions propagating through crossover paths may cause temporary glitches at the recipient flip-flop inputs. However, combinational logic that does not sensitize multiple crossover paths to the same destination flip-flop simultaneously does not produce glitches and is therefore an exception to the rule. Multiplexers are a common example (which most structural rule checkers attempt to recognize and ignore) but other cases of this exception may depend on functional properties and not be identifiable structurally.

B. Importance and Future Trends

Modern SoC designs contain tens of clock domains that are connected by thousands of CDC paths [5]. These numbers are expected to grow due to a number of trends. The continuous growth in integration density, for one, means that independently-developed Intellectual Property (IP) blocks that are compatible or optimized to run at particular voltage and frequency settings will occupy smaller chip areas. In consequence, a finer granularity of clock domains will be needed to take full advantage of the performance and power saving potential of these blocks. At the same time, fine-scaled Dynamic Voltage and Frequency Scaling (DVFS), Network-on-Chip (NoC) and Globally Asynchronous Locally Synchronous (GALS) architectures, all solutions to age-old problems of power conservation, data communication and clock distribution, rely on component multitude and asynchrony. The overall push towards larger numbers of clock domains resulting from the sum of these trends is met with mediocre scaling in our ability to design and verify anything but conventional clock domain interfaces. The costs of this mismatched scaling are considerable. Published accounts of verification runs on commercial SoCs (with tens to hundreds of millions of gates) report CDC warnings in the order of 100s of thousands out of which 90% are false positives due to linting tool misconfiguration and non-standard design cases [6]. Even when a design passes this laborious verification, it is still forced to tolerate the performance penalties of conservative design patterns, predominantly the low throughout resulting from synchronization latency [7].

C. Main Contributions

We present a CDC verification solution that avoids the conservatism of conventional CDC linting and offers several other advantages. Our solution involves using gate-level circuit transformations and metastable flip-flop models to reproduce timing violations and their propagation between flip-flops in discrete cycle-based simulation (e.g. using tools such as ModelSim). This makes CDC issues and their consequences observable and amenable for testing, verification and debugging by conventional digital simulation and verification tools. We propose a CDC verification flow that combines this approach with formal verification and show that, as opposed to conventional CDC verification, this flow is (1) inherently capable of identifying a wide range of CDC issues without requiring them to be explicitly specified, (2) capable of determining what failure types (e.g. data corruption, invalid state transitions) that the design may experience due to these issues, (3) capable of generating signal waveform traces showing the mechanisms behind reported failures, (4) reports fewer false positives, (5) does not require the designer to specify any details about how the interface works (e.g. what synchronization or data encoding schemes are used) and (6) capable of verifying arbitrary CDC circuits including those that do not adopt safe standards.

II. Related Work

A discussion of CDC design practices and common pitfalls is beyond the scope of this paper. We therefore survey relevant work in the domain of CDC verification while referring interested readers to the comprehensive review of CDC design strategies presented in [8]. In general, published work on using assertions and formal tools to verify multi-clock designs appears to be divided into three groups:

A. Automatic Generation of Functional CDC Assertions

Functional verification can take functional design properties into consideration when verifying CDC logic and therefore tends to report fewer false positives compared to structural linting. However, it requires some effort on part of the designer to determine what checks are needed/suitable depending on what design patterns are used (e.g. synchronization scheme). Commercial CDC verification tools and published methods can automate this process [9], [10], [11] but require the use of known design patterns which can be identified structurally. Even in the latter case the issues identified by automatically-generated assertions may still be false positives since these assertions are also heuristic rules based on safe design practices. For example, a typical functional rule in data synchronization schemes prohibits multiple signals from transitioning simultaneously. This is intended to prevent errors resulting from the arrival of transitions in different clock cycles. However, whether this will cause a failure or not depends on the design's structure and functional properties and so the rule is actually a conservative generalization.

B. Simulation of Metastability Effects

A useful method to stress-test multi-clock designs is to deliberately introduce some of the known effects of timing violations and observe their impact on the design's behavior [12]. Examples include the injection of non-deterministic one-cycle delays into synchronizer chains [10] and the latching of non-deterministic crossover bit values [13]. Such approaches simulate specific effects of timing violations but are usually limited in scope and do not capture the full range of problematic timing violation behavior. They are therefore useful for identifying certain issues but do not guarantee that a multi-clock design will function correctly.

C. Formal Models

Another approach to make up for the inability to observe timing violations using digital tools is to construct formal models that can capture timing violation effects and then use these models to prove theorems about the design [14], [15]. This is a rigorous method to ensure that multi-clock designs function as intended but requires considerable effort between constructing the model and proving the theorems.

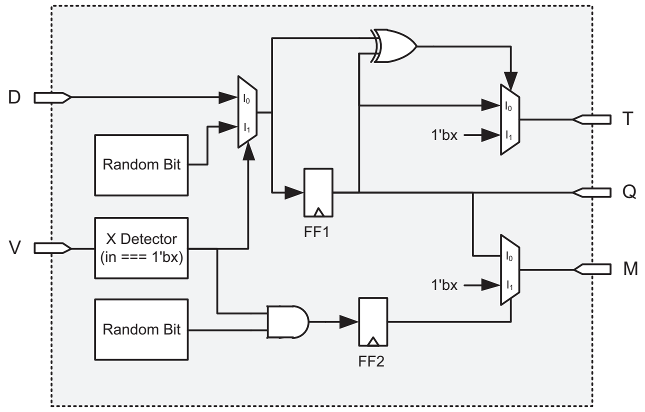
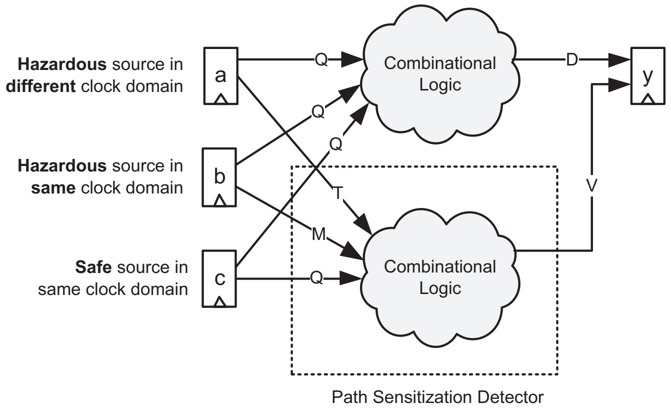


Fig. 1. Metastable flip-flop model with additional ports to communicate input (setup/hold) and output (clk-to-q) timing violation information



Hazardous source in different clock domain
Hazardous source in same clock domain
Safe source in same clock domain
a
b
c
y
Combinational Logic
Combinational Logic
Q
Q
Q
Q
T
M
D
V
Path Sensitization Detector

Fig. 2. Model inter-connectivity scheme: added ports are connected through a combinational logic duplicate to simulate the propagation of timing violations

III. Proposed Verification Approach

A. Overview

The underlying failure mechanism behind clock domain crossing issues is the violation of flip-flop setup and hold time conditions. A flip-flop whose input does not satisfy setup/hold time conditions may (1) latch an incorrect value and/or (2) exhibit a prolonged clk-to-q transition which in turn may violate the input conditions of its destination flip-flops during the following clock cycle. Our approach can be summarily described as follows. We transform the circuit such that these timing violations and their propagation between flip-flops can be modeled at the netlist level and observed using conventional discrete cycle-based simulation (e.g. using ModelSim). The transformation consists of two steps. Flip-flops are first replaced with metastable flip-flop models that have additional ports to specify when their input and output transitions violate timing conditions. Additional combinational circuit elements are then inserted to simulate the propagation of timing violations between these flip-flops.

B. Metastable Flip-flop Model

We first explain the working principle of the model and then discuss how several such models interact to simulate the transfer of timing violations. Figure 1 shows the internal schematics of the model. On top of the Data (D) and Output (Q) ports the model has a Violation (V) input, a Metastable (M) output and a Transition (T) output ports. The model contains a conventional flip-flop FF1 which acts as the data storage element and an additional flip-flop FF2 which indicates whether the model is experiencing a prolonged clk-to-q transition during the current cycle. During each cycle, the input V indicates whether the setup/hold time conditions are violated, output M indicates whether the clk-to-q timing condition is violated and T indicates whether the output Q transitioned. For reasons that will become subsequently clear the ports V, M and T use an encoding based on four-valued logic where an unknown bit (expressed in Verilog as 1'bx) indicates an active state and other values (0 or 1) indicate an inactive state. The model contains converter blocks which map the input V from {x,0/1} to {1,0} for internal use and then back to {x,0/1} at the outputs M and T.

The model works in the following way. When V is inactive (i.e. has a value of either 0 or 1 indicating that input timing conditions are met during the current cycle) the model behaves as a conventional flip-flop and latches D (while de-asserting M and setting T based on whether the newly-latched value is different from the previous Q output). When V is active, however, the model latches a random bit (to simulate the possible incorrect latching of D) and, depending on another random bit, may assert M during the next cycle (to indicate a metastable state in which it exhibits a prolonged clk-to-q delay). When T or M are inactive, their output equals Q.

C. Circuit Transformation

To demonstrate how several such models can be connected to simulate the propagation of timing violations we refer to an example circuit described by the expression $y = f(a, b, c)$ (Figure 2). Flip-flops *a*, *b* and *c* are referred to in this context as source flip-flops while *y* is a destination flip-flop.

Flip-flop *a* belongs to a different clock domain than *y*'s and is thus "hazardous" with respect to *y* because its output transitions are not timed to *y*'s clock. Flip-flop *b* lies in the same clock domain as *y* but is nevertheless considered hazardous because it may experience prolonged clk-to-q transitions that may violate *y*'s input timing conditions (an example of this case is the first synchronizer flip-flop which is hazardous to its neighbour in the same clock domain). Finally, flip-flop *c* is in the same clock domain as *y* and its clk-to-q transitions are never delayed making it "safe" with respect to *y*. Since it is logically impossible for a source flip-flop in a different domain to be safe, these three combinations represent all the possible classifications of source flip-flops based on whether they belong to the destination clock domain and whether they can cause timing violations of a destination flip-flop's input.

Our transformation relies on the following rule for modeling the propagation of timing violations: if there exists a sensitized combinational path between a source flip-flop J and

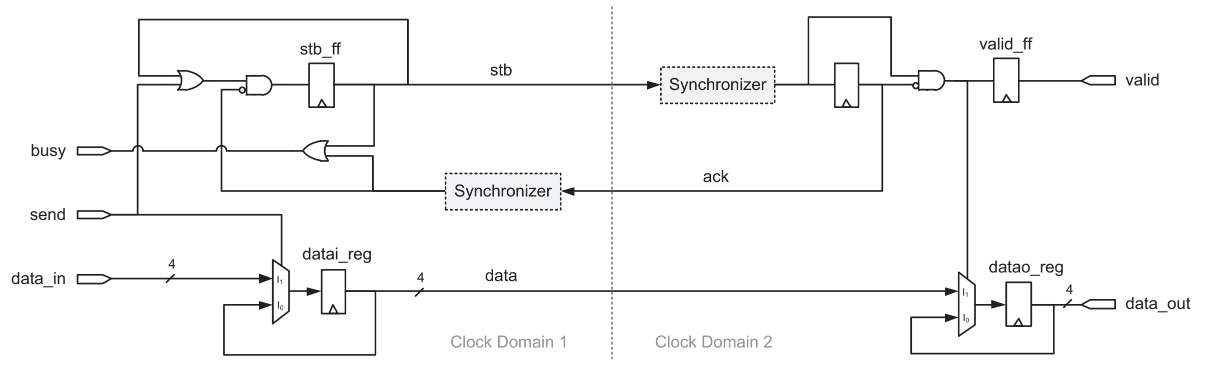


Fig. 3. Example multi-clock design: a sender and receiver using a 4-way handshake to transfer a data item across a clock domain boundary

a destination flip-flop K then a clk-to-q timing violation of J may cause a setup/hold time violation of K. To determine whether any such combinational path(s) exist in our example, we instantiate a duplicate of the combinational function of *y*, connecting its inputs to one of the ports T, M or Q of each source flip-flop depending on its classification. The basic idea behind this is that we wish to propagate an unknown bit (1'bx) from all sources which may violate *y*'s input timing conditions. If the output of the combinational function duplicate is 1'bx then we know that there exists at least one sensitized path from a hazardous source to *y*. Otherwise (if the output is 0 or 1) we know that any timing violations emanating from hazardous flops (should any have occurred) were logically masked by safe inputs. The combinational logic duplicate thus acts as a "path sensitization detector" which enables us to determine whether the timing conditions of *y*'s input are met during each cycle. In this example, the source *a* is connected to the combinational logic block through the T port because *a* belongs to a different clock domain and therefore any of its output transitions may violate *y*'s input timing conditions. The source *b* may only cause violations when it experiences a prolonged clk-to-q delay and is thus connected through its M port while source *c* can never cause violations and is connected through its Q port (whose 0 or 1 outputs always indicate an inactive state of timing violations).

In larger designs consisting of flip-flops in different clock domains the transformation above is applied to all destination flip-flops with at least one hazardous source. A source flip-flop is hazardous to a destination if (1) it belongs to a different clock domain or (2) it is within a one flip-flop distance from a hazardous source. The second part assumes (without any loss of generality) that metastability does not propagate beyond two flip-flops on the receiving domain's side.

D. Integration with Formal Verification

After applying the transformation above a gate-level netlist will become capable of simulating the onset of timing violations across clock domain boundaries, their propagation on the receiver's side and any resulting functional failures. The non-deterministic effects of timing violations are captured by the two random bit generators inside each model flip-flop. One random bit determines the value latched by the flip-flop whose input was violated while the other determines if the flip-flop will itself become metastable during the following cycle. Since CDC failures typically involve specific sequences of non-deterministic events it is necessary to identify whether any of the possible sequences of all random bit generators can cause a deviation from the design's intended behavior. This is a task well-suited for formal verification tools which, in this context, would treat random bit sources as additional design inputs and identify any input vector sequences that violate assertions of correct behavior. We therefore propose a CDC verification flow based on our gate-level modeling of timing violations combined with formal verification. This flow consists of synthesizing the design, applying our morphological transformations to the generated netlist and then passing it to a formal tool to check for violations of formal assertions. These assertions need not describe how the CDC interface is intended to work but can be generic functional specifications.

IV. Testbench Results

We developed a software tool to apply the morphological transformation described in Section III and used it to verify a number of multi-clock designs. This section presents the results of our verification runs and is split into two parts. First, we present a case example to demonstrate how the proposed verification flow is able to identify a CDC issue and generate example waveforms demonstrating a resulting failure and its underlying mechanism. Second, we present verification results for a number of multi-clock designs to explore the flow's ability to identify common CDC issues.

A. Case Example

Figure 3 shows an example multi-clock design. The design consists of sender and receiver units that operate in different clock domains and aim to transfer data items correctly using a 4-way handshake protocol. Each unit communicates with its local neighbours in the same clock domain using a simple synchronous interface. A data item can be pushed to the sender by setting *data_in* and asserting *send* for one cycle. The sender then asserts *busy* until the transfer is completed. On the other

TABLE I
VERIFICATION RESULTS FOR THE SENDER/RECEIVER CIRCUIT

Synchronizer(s)	Assertion	Status (✓= pass)	
		Source Netlist	Processed Netlist
None	as_correct_transfer	✓	-
	as_sender_handshake	✓	-
	as_no_blocked_transfer	✓	-
Sender Only	as_correct_transfer	✓	-
	as_sender_handshake	✓	✓
	as_no_blocked_transfer	✓	-
Receiver Only	as_correct_transfer	✓	-
	as_sender_handshake	✓	-
	as_no_blocked_transfer	✓	-
Both	as_correct_transfer	✓	✓
	as_sender_handshake	✓	✓
	as_no_blocked_transfer	✓	✓

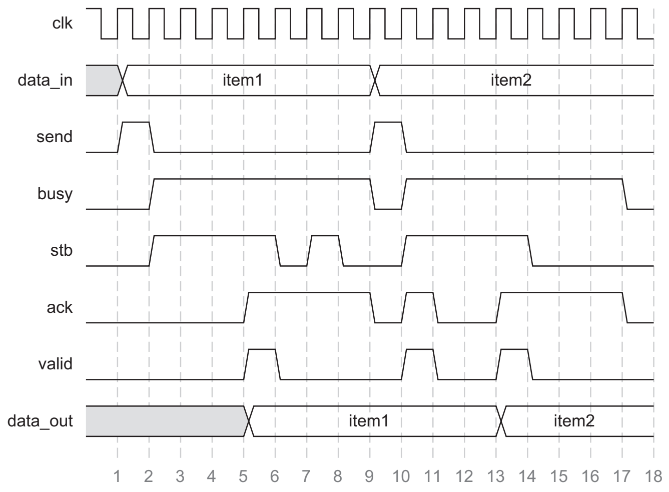


Fig. 4. An elusive failure mechanism identified by the proposed verification approach for the sender-receiver circuit (lack of sender synchronizer allows *stb_ff* to become metastable during cycles 6 and 7, initiating a bogus handshake and causing *item1* to be received twice)

end the receiver asserts *valid* for one cycle to indicate that *data_out* contains a new data item.

Three SystemVerilog Assertions (SVAs) were specified to verify that this design functions as intended. Assertion 1 (*as_correct_transfer*) states that when *valid* is asserted then the value of *data_out* equals the value of *data_in* when *send* was last asserted. Assertion 2 (*as_no_blocked_transfer*) states that any assertion of *send* is followed by an assertion of *valid* in finite time. Finally, Assertion 3 (*as_sender_handshake*) states that *busy* should be asserted after *send* by 1 cycle. The design's environment was constrained by an additional property to prevent *send* from being asserted while the sender is busy.

We used Incisive Formal Verifier to verify the design's synthesized gate-level netlist with and without sender and receiver synchronizers to investigate what failures can manifest in each case. The results are presented in Table I. In summary, when verifying the source netlist all assertions receive a pass status regardless of whether synchronizers are present or not. Verifying the processed gate-level netlist, on the other hand, correctly identifies the cases when the design fails and (equally importantly) when it does not. For each failure case the formal tool generates a counter-example showing an input sequence and internal signal waveforms demonstrating how the design progresses from an initial reset state to a violation of one of the assertions. Examining these traces reveals where timing violations first occurred, their propagation between flip-flops and what abnormal design behaviour followed. Some of these scenarios are easy to foresee by inspecting the design schematics but others are less so. To showcase an example of a counter-intuitive scenario which the proposed verification flow can help uncover we present the waveform traces for a violation of *as_correct_transfer* when the sender synchronizer is absent. Although the lack of a sender synchronizer can cause handshaking problems, it is perhaps not equally clear how omitting the sender synchronizer can cause incorrect data transfers, particularly since no combinational paths exist between the sender state flip-flop *stb_ff* (the only vulnerable flip-flop in this case) and any of the data registers. The counter-example generated by the formal tool for this case (Figure 4) demonstrates an elusive mechanism behind this failure. Briefly, the assertion of *ack* following the correct transfer of a data item (cycle 5) causes the sender's *stb_ff* flip-flop to become metastable (cycle 6). During cycle 6, the combinational path *stb_ff* $\rightarrow$ *stb_ff* is sensitized and metastability propagates from *stb_ff* to itself again. In the two cycles in which *stb_ff* is metastable (cycles 6 and 7), the signal *stb* is de-asserted then asserted causing a bogus handshake to be initiated. As this unintended handshake is synchronized by the receiver, the sender enters an idle state (*busy* is de-asserted) and a new item is pushed to the sender (cycle 9). On cycle 10 the bogus handshake causes *valid* to be asserted while the value of *data_out* (still *item1*) does not equal the last pushed data item (*item2*) - a violation of the assertion *as_correct_transfer*.

B. Testbench Verification of Known CDC Issues

Our case example demonstrated that the proposed verification flow is capable of identifying missing synchronizer problems. We constructed and synthesized several other designs to test whether the proposed flow is also capable of detecting other CDC issues. Some of the verified designs use unconventional practices or exceptions to known CDC rules of thumb and were specifically included to assess the flow's capability of reporting correct behavior in these cases. The testbench verification results are presented in Table II alongside those of conventional structural checks for comparison. The results demonstrate that the proposed flow is capable of correctly identifying problematic issues in the testbench designs while reporting fewer false positives.

TABLE II
TESTBENCH VERIFICATION RESULTS

Testbench	Description	CDC Issue	Verification Result (✓= pass)	
			Structural Checks	Proposed Flow
1	Data transfer (4-phase handshaking and synchronizers)	-	✓	✓
2	Data transfer (4-phase handshaking, no synchronizers)	Data corruption	-	-
3	Data transfer (no handshaking, data synchronization, Gray coding)	-	-	✓
4	Data transfer (no handshaking, data synchronization, non-Gray coding)	Data corruption	-	-
5	Data transfer (no handshaking, no synchronization, quasi-stable data)	-	-	✓
6	Multiplexer in crossover path	-	✓	✓
7	Combinational logic in crossover path (glitch-prone)	Glitches	-	-
8	Combinational logic in crossover path (glitch-free)	-	-	✓
9	Two synchronization points	Path reconvergence	-	-
10	Two synchronization points (not activated simultaneously)	-	-	✓
	False Positives (warnings of non-present issues)		4	0
	False Negatives (present but unreported issues)		0	0

V. CONCLUSION

We presented a novel CDC verification approach based on reproducing flip-flop input/output timing violations and their propagation in gate-level netlists. This solution addresses the issue of clock domain failure unobservability in digital tools directly and offers several advantages over conventional structural and functional CDC design heuristics. Verification results from applying the approach to multi-clock testbenches demonstrated an inherent capability of identifying several CDC issues including missing synchronizers, data corruption, path reconvergence and crossover path glitches.

The catastrophic and elusive nature of CDC failures has made design patterns such as synchronizers the *de facto* standard and discouraged exploring the design space lying beyond them. There are however several interesting avenues for designing systems that can function correctly (or with an acceptable error rate) despite being susceptible to metastability. A Gray-coded counter triggered by an asynchronous signal, for example, will transition to either current or next state even if one of its flip-flops becomes metastable. It is therefore possible in principle to design finite state machines whose behavior can be constrained despite the manifestation of metastable states. Such machines would be ideal controllers for applications in which infrequent data errors can be detected and corrected at higher abstraction levels (e.g. in software), tolerated or altogether ignored (e.g. when transferring lossy image pixel data). In these cases, non-conservative clock boundary design can offer zero-latency communication between truly asynchronous clocks, a feature that can amount to considerable throughput gains in modern designs based on tens of clock domains. Our approach enables the investigation of these unconventional design options that are prohibited under safe CDC design guidelines. Therefore, apart from its potency as a verification solution, it can be used as a tool for exploring heretofore uncharted regions of the multi-clock design space.

ACKNOWLEDGMENT

We thank Crescenzo D'Alessandro and Danil Sokolov for the feedback and valuable discussions. This work was supported by EPSRC grants EP/L025507/1 (project A4A) and EP/J008133/1 (project TRAMS2).

REFERENCES

[1] S. Verma and A. S. Dabare, "Understanding clock domain crossing issues," *EE Times*, 2007.

[2] V. C. Vimjam and A. Joseph, "Challenges in verification of clock domain crossings," 2013.

[3] D. Kinniment, *Synchronization and Arbitration in Digital Systems*. Wiley, 2008.

[4] S. Sarwary, "Solving the toughest problems in cdc analysis," *EE Times August*, vol. 28, 2006.

[5] P. Yeung, "Five steps to quality cdc verification," *Whitepaper, Mentor Graphics Corporation, year*, 2007.

[6] Y. Lee, N. Kim, J. Kim, and B. Min, "Millions to thousands issues through knowledge based soc cdc verification," in *SoC Design Conference (ISOCC), 2012 International*, Nov 2012, pp. 391–394.

[7] R. Ginosar, "Metastability and synchronizers: A tutorial," *IEEE Design & Test of Computers*, no. 5, pp. 23–35, 2011.

[8] C. E. Cummings, "Clock domain crossing (cdc) design & verification techniques using systemverilog," *SNUG-2008, Boston*, 2008.

[9] B. Li and C. K.-K. Kwok, "Automatic formal verification of clock domain crossing signals," in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*. IEEE Press, 2009, pp. 654–659.

[10] M. Litterick *et al.*, "Pragmatic simulation-based verification of clock domain crossing signals and jitter using systemverilog assertions," *Proceedings of DV-Con*, vol. 6, 2006.

[11] K. Kwok, B. Li, T. Ly, and R. Sabbagh, "Formal verification of clock domain crossings," Sep. 18 2012, uS Patent 8,271,918.

[12] T. Ly, K. Kwok, V. Gupta, and L. Widdoes, "Metastability effects simulation for a circuit description," Dec. 16 2014, uS Patent 8,914,761.

[13] A. Ismail and H. Saafan, "Synthesizable system verilog model for hardware metastability in formal verification," in *Electronics, Communications and Photonics Conference (SIECPC), 2013 Saudi International*. IEEE, 2013, pp. 1–6.

[14] J. Schmaltz, "A formal model of clock domain crossing and automated verification of time-triggered hardware," in *Formal Methods in Computer Aided Design, 2007. FMCAD'07*. IEEE, 2007, pp. 223–230.

[15] G. M. Brown and L. Pike, "Easy parameterized verificaton of cross clock domain protocols," in *Seventh International Workshop on Designing Correct Circuits DCC: Participants Proceedings*. Citeseer, 2006.