2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

A Systematic Take on Addressing Dynamic CDC Verification Challenges

Sukriti Bisht, Sulabh Kumar Khare, Ashish Hari
sukriti_bisht@mentor.com, sulabh-kumar_khare@mentor.com, ashish_hari@mentor.com
Design Verification Technologies
Mentor, A Siemens Business

Abstract

Clock Domain Crossing (CDC) issues are the second most common reason for silicon re-spins. Most modern day designs have more than one clock, many of which are asynchronous. Signals that pass between logic clocked by different asynchronous clocks are called clock domain crossing (CDC) signals. Due to the asynchronous nature of clocks, there is a possibility of CDC signals going metastable and propagating incorrect logic downstream resulting in functional failure of the design. To mitigate these problems, synchronizers are used on CDC paths.

Each synchronizer is dependent on a set of assumptions or *protocols*, which when violated can make the transfer unreliable. To avoid such issues, it is crucial to validate each synchronizer for its reliability. Engineers have been using assertion based verification methods to verify synchronizer protocols in Formal and Simulation environments, but have fallen short of addressing the issue.

In this paper, we will present the most prominent challenges faced with the existing methodology currently being used to verify synchronizer protocols and propose a new methodology to overcome them. Following are the key challenges:

- Significant effort and time is required to setup the design before starting verification using Formal and Simulation. The setup also includes translation of design configurations and settings to both environments which requires technical expertise.
- Considerable debug effort required to review firings in Formal and Simulation.
- Missing correlation between protocol assertion checks in Formal, Simulation and associated CDC paths
- Lack of re-utilization of benefits and efforts of both Formal, Simulation for faster design closure

The proposed methodology automates the progression of setup, constraint and results from CDC to Formal to Simulation. The methodology also addresses the important issue of correlating Formal and Simulation results to CDC results by using a new technique. In this technique, we add hooks to Formal and Simulation that enable us to extract important information and associate them to the CDC results. The proposed approach has also been automated for the seamless adoption of this methodology.

The paper concludes with a demonstration of the proposed methodology on a set of real life designs. The demonstration proves that the methodology significantly reduces design and verification engineers' effort and helps in achieving faster design closure.

I. INTRODUCTION

Most designs now-a-days have multiple asynchronous clocks. Logic clocked by each asynchronous clock forms the clock domain for that clock. Signals that cross the boundaries of these clock domains are called Clock Domain

Crossing (CDC) signals. Asynchronous nature of these CDC signals may cause setup or hold time violation of flip flops resulting in metastability. To ensure that these signals are properly synchronized, synchronizers are used. Synchronizers help protect the receiver domain from sampling metastable values and prevent data loss or corruption.

However, adding a synchronizer is just part of the solution. Despite the presence of synchronizers on CDC paths, there is still a possibility of data loss or corruption due to incorrect usage or interfacing of synchronizers. Reliability of every synchronizer depends on a set of assumptions or *protocols*. If the logic associated to the synchronizer does not interact as per the synchronizer's protocol, the reliability of synchronizer might break. For example data loss can occur if the input to a two DFF synchronizer is not stable for at least two clock cycles, or if the request and acknowledge signals in a handshake synchronizer do not occur in a particular order. Figure 1 illustrates these scenarios.

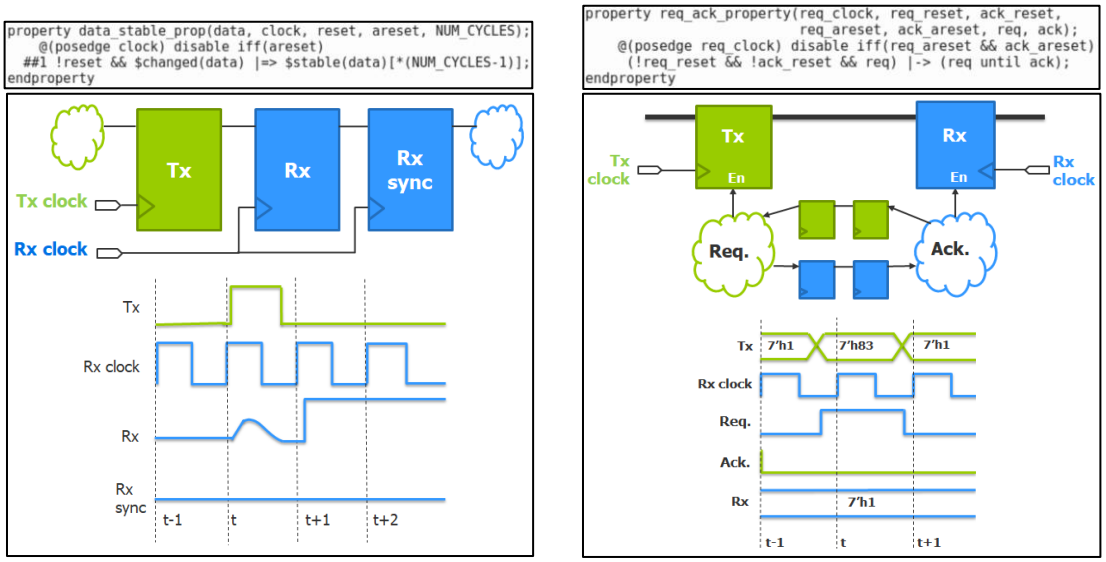

Figure 1: Data loss in two DFF and Handshake synchronizers due to protocol violation

The issues of data loss or corruption due to synchronizer protocol violation must be identified and addressed early in the design cycle, otherwise they can lead to functional failures in later stages. These functional failures can further result in increased iterations and even silicon re-spins. To ensure such issues are not missed, designers and verification engineers verify synchronizer protocols. This Dynamic CDC Protocol Verification is done by extracting assertions for synchronizer protocols and verifying them using Formal and Simulation methods.

II. Existing Dynamic CDC Protocol Verification Methodology

Figure 2 illustrates the existing approach for Dynamic CDC Protocol Verification. First step is to perform Static CDC analysis to ensure the presence of synchronizing logic on CDC paths. Post this step, assertions for synchronizer protocols are extracted based on CDC results. These assertions are then verified to ensure that the synchronizers present in the design are reliable.

To verify these protocol assertions, Simulation and/or Formal tools are used. The setup for each of these tools is different and has to be done manually, which increases the verification time and effort. Once the setup is complete, engineers perform protocol assertion verification using the Formal, Simulation tools and their results are debugged separately. While this methodology promises to identify CDC protocol bugs, it suffers from many issues such as substantial effort required for setup and debug, lack of correlation with CDC results, amongst others.

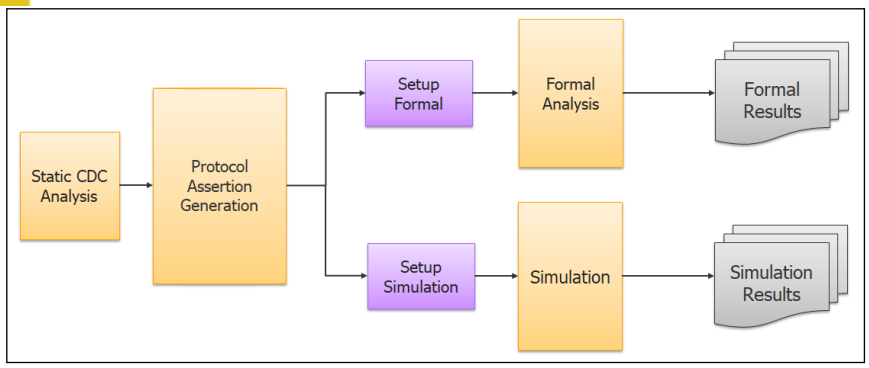

Figure 2: Existing methodology flowchart

III. CHALLENGES FACED WITH EXISTING METHODOLOGY

In the existing methodology, engineers use Simulation and Formal methods separately to generate and verify assertions for synchronizer protocols. However, the methodology entails the following common challenges:

- **Setup:** Significant effort and time in required to set up the design for Formal and Simulation runs. Design constraints and directives that were already specified in the CDC step, need to be translated and propagated to both the Formal and Simulation tools separately. For example, constant values, clock frequencies of design and so on. Issues at this step can result in unexpected design behavior in dynamic environment. Figure 4 shows an example where constant specified on input signal at the time of Static CDC was not propagated to Formal setup resulting in false two DFF protocol firing.

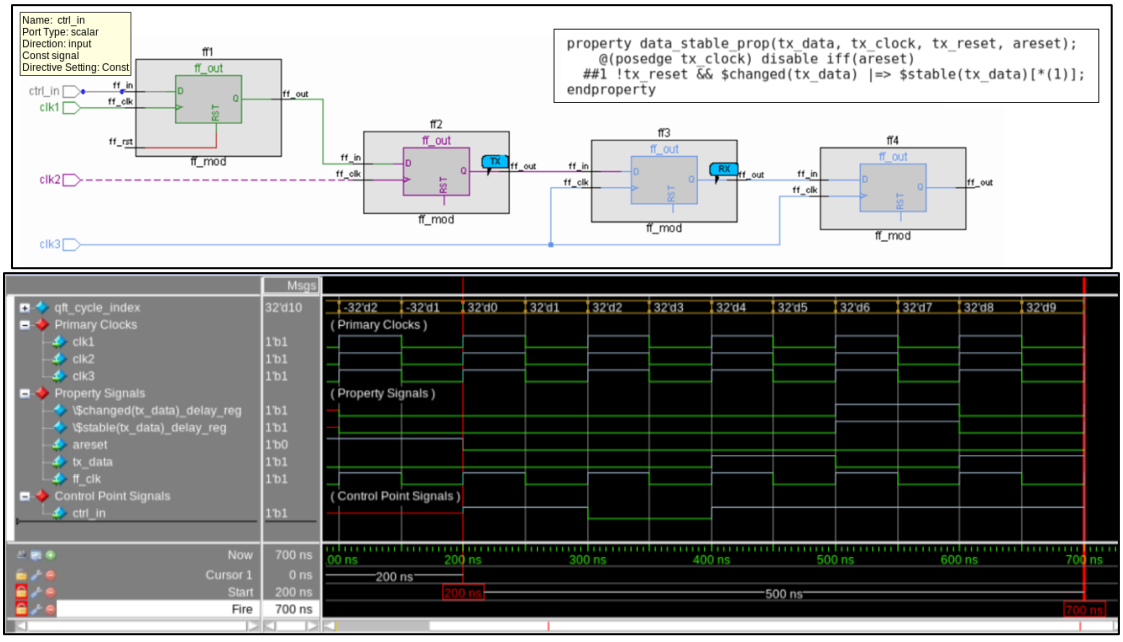

Figure 4: False two DFF firing in Formal due to constants missing in setup

- **Debug:** Analyzing and debugging Formal and Simulation results requires expert knowledge about both the environments. Issues in setup or inaccurate assertions can result in large number of false firings, which further

increase the problem. For example, incorrect clock frequency or domain specified on ports can result in unexpected behavior. Figure 4 shows an example of a false two DFF stability check firing in Formal where unspecified clock frequency in Formal setup resulted in clock incorrectly changing as per fastest clock's frequency.

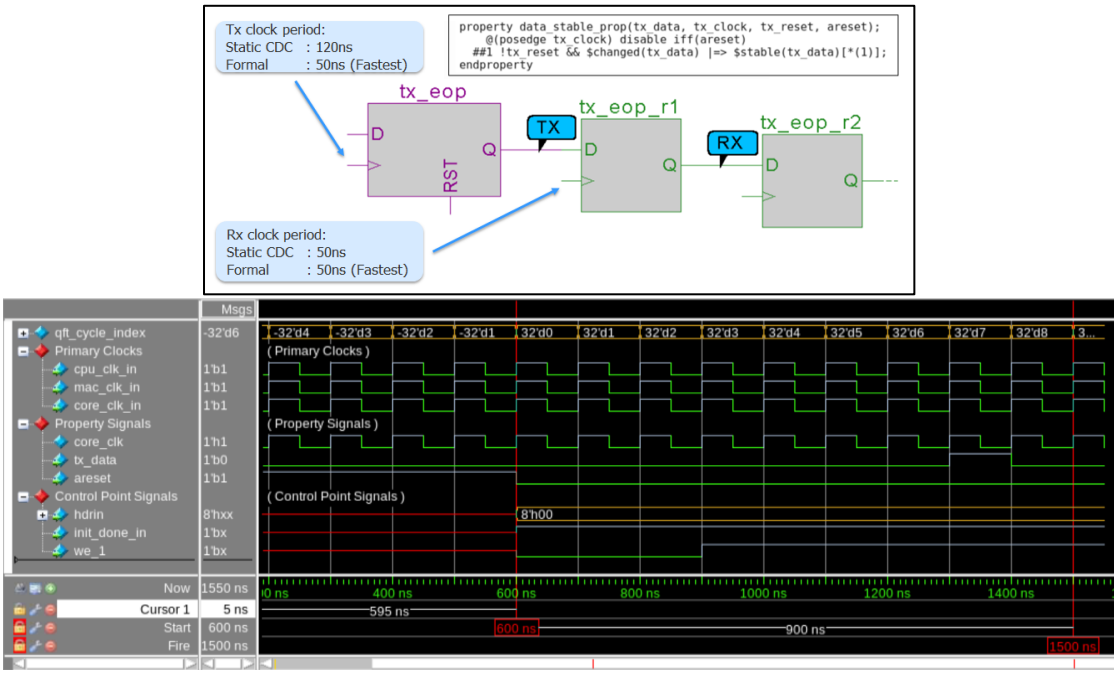

Figure 4: False 2DFF Formal firing due to unspecified clock frequency

- **Correlation to CDC:** The existing methodology suffers from lack of relation between CDC, Formal and Simulation results. The debug environments of Formal and Simulation are very different from CDC. Hence, establishing a relation between CDC, Formal and Simulation results for the purpose of coverage and review of CDC paths is cumbersome and time consuming. For example, complex synchronizers such as FIFO or handshake have multiple assertions that are treated as separate entities by both Formal and Simulation, but relate back to a single CDC path. Issues or errors during correlating results can lead to missed bugs. Figure 5 shows various handshake synchronizer protocol assertions that relate back to the same crossing. If not correlated properly, some uncovered or fired assertions may be missed resulting in data loss or corruption in the design.

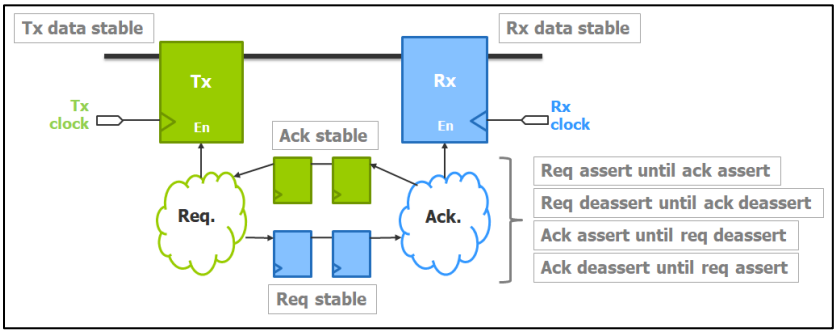

Figure 5: Multiple handshake protocol assertions

- **Lack of effort re-utilization:** Due to limitations of both Formal and Simulation methods, to save time and to ensure sufficient coverage, verification engineers may have to change from one method to another, or adopt both simultaneously. In such scenarios, the challenge faced is in terms of redundancy and lack of verification effort re-utilization. For example, if Formal step is followed by Simulation step, assertions that were already proven by Formal will have to be re-verified in Simulation.

Apart from these common challenges, both Formal and Simulation methods have their own advantages and disadvantages. While Simulation is more intuitive to understand, it runs into coverage issues. Bugs can go undetected if the quality of testbench is not good. On the other hand, Formal offers exhaustive proves but runs into infrastructural and capacity issues.

For a design of considerable size, covering all scenarios in Simulation can take infinite amount of time which may lead to missed deadlines. On the other hand, obtaining conclusive proofs for an entire design in Formal may require infinite memory due to state space explosion problem, resulting in increased cost.

IV. Proposed Methodology For Dynamic CDC Protocol Verification

In this paper, we propose a methodology that not only helps overcome the common challenges of Formal and Simulation verification, but also utilizes the advantages and efforts of both the methods. Figure 6 illustrates the proposed methodology.

With this methodology, we propose the automated propagation of setup from CDC to Formal and Simulation tools to help reduce the time and manual effort required in setting the design. This propagated setup includes design settings like clocks, resets, constants, port clock domains, amongst others that are easily available at the CDC step. This automation ensures easy and correct propagation of setup without the hurdles of technical expertise required on the engineer's part to setup each tool's environment.

The protocol assertion generation step inserts hooks into Formal and Simulation to get insights into the status of each assertion. These insights help to correlate the Formal and Simulation results back to the associated CDC paths. This helps ensure that design and verification engineers can view the dynamic verification status of each CDC path.

The methodology aims to not only save but also re-utilize designer and verification engineer's effort for faster design closure. Ensuring automated propagation of setup and barring the propagation of Formal proven assertions to Simulation helps in achieving the objective.

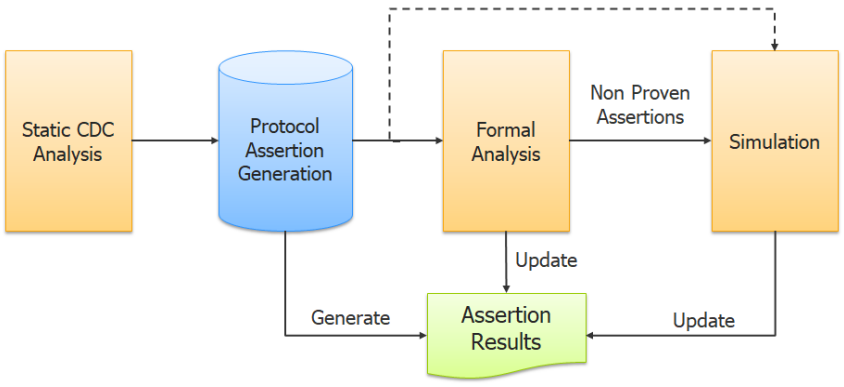

Figure 6: Proposed methodology flowchart

The proposed methodology has the following workflow:

1. **Static CDC Analysis**: Perform static CDC analysis on the design to ensure that all the relevant CDC paths are synchronized using proper synchronizers.

2. **Protocol Assertion Generation**: Generate assertions for protocols of synchronizers present in the design. Based on static CDC analysis, each synchronizer on a CDC path is analyzed and protocol assertions are generated depending upon the type of synchronizer and its connections. Simultaneously, the design settings and configurations passed to the CDC step are translated to Formal and Simulation language, generating the required setup for running both environments. This step also includes inserting hooks in both environments, which helps in correlating Formal and Simulation results back to CDC environment.

3. **Formal Analysis**: Verify the generated synchronizer protocol assertions in Formal environment using the setup generated. The automated setup reduces significant effort required to setup the design and also avoids the possibility of false firings due to incomplete or incorrect setup.

4. **Formal Debug**: Debug the assertion firings generated by Formal tool by analyzing the counterexample. Design constraints, if any, can be added to eliminate false scenarios and the design can be re-run incrementally with the added constraints.

5. **Simulation**: Run Simulation on the design using the auto-generated setup to verify the protocol assertions that are non-proven by the Formal tool. After Formal analysis, the Simulation setup is updated incrementally to ensure only non-proven assertions are promoted to Simulation environment. This helps ensure that the effort put in to get conclusive results in Formal is re-utilized resulting in faster design closure.

6. **Review CDCs**: Review the combined results of Formal and Simulation along with CDC results. The hooks inserted in Formal and Simulation are used to extract status of various protocol assertions associated to a crossing and correlate them back to CDC environment. This helps in ensuring that bugs are not missed and protocols for synchronizers on CDC paths are not violated. Figure 7 shows the view of a combined result set along with the related CDC paths, for a unit design.

```
================================================================
Section 2 : CDC Protocol Results
================================================================

CDC ID                  Protocol ID                          Formal Result       Simulation Result
-------------------------------------------------------------------------------------------------
handshake_7492          cdc_protocol.handshake_7492          Fired               Fired
bus_two_dff_1183        cdc_protocol.bus_two_dff_1183        Fired               Covered
handshake_5883          cdc_protocol.handshake_5883          Inconclusive        Uncovered
two_dff_19174           cdc_protocol.two_dff_19174           Inconclusive        Covered
bus_two_dff_4271        cdc_protocol.bus_two_dff_4271        Proven              -
fifo_2332               cdc_protocol.fifo_2332               Proven              -
two_dff_68078           cdc_protocol.two_dff_68078           Proven              -
```

Figure 7: Formal, Simulation result correlation to CDC paths

V. CASE STUDY

Both existing and proposed methodologies were compared by using them for Dynamic CDC Protocol Verification of real life designs. The comparisons were performed on a set of real life designs, ranging from 1 to 30 million gates. The setup time required and results obtained with both the methodologies were recorded and analyzed.

A. Static CDC Analysis, Protocol Assertion Generation

The designs first underwent Static CDC analysis to ensure the presence of synchronizers on all relevant CDC paths. Post Static CDC analysis, assertions for synchronizer protocols were generated and verified in Dynamic environment as per each methodology.

B. Formal Verification

In the existing methodology, the protocol assertions were validated first in a Formal environment. The design settings and constraints like clocks, resets, constants, and so on were translated from Static CDC environment and re-specified in the Formal tool's language manually. This required significant effort and expertise. Once the Formal tool environment setup was complete, the design was verified formally and the assertion firings were debugged.

Similarly, in the proposed methodology, the first step was protocol assertion verification in Formal environment. The design configurations and constraints for Formal were auto-generated by using the proposed methodology. This auto-generated setup was reviewed and used to verify design formally. The setup automation saved a lot of manual effort and expertise that was earlier required to port the setup from CDC to Formal.

C. Simulation

Post Formal verification, the engineer decided to take the Simulation path. With the existing methodology, protocol assertions that were proven formally were re-verified in Simulation environment. The design, testbench were compiled and setup for Simulation. On completion of Simulation, the results were analyzed and firings were debugged.

In the proposed methodology, only the assertions that were non-proven in the Formal environment, were verified in Simulation. The auto-generated setup along with testbench, was used to run Simulation. This was followed by debugging the Simulation firings. Significant reduction in debug effort and time was observed due to running Simulation only for Formal non-proven assertions.

Tables 1 and 2 summarizes the comparison between existing and proposed methodology performed on a set of real life designs, ranging from 1 to 30 million gates. Significant reduction in Formal setup time was observed. Setup time for Simulation was also reduced. Firings in Formal also reduced for each design due to reduction in false firings caused by setup issues. Assertions passed to Simulation step also reduced due to exclusion of Formal proven assertions thereby reducing the verification effort required.

An overall reduction in effort and time was observed due to automated propagation of setup. The debug effort was also reduced due to minimization of incomplete and incorrect setup. The correlation of Formal, Simulation results to CDC paths helped in debug and easier identification of associated CDC paths.

TABLE I
RESULTS USING EXISTING METHODOLOGY

CDC Protocol Verification with Formal							CDC Protocol Verification with Simulation					
Design	Setup time	Assertions			Run Time	Formal Coverage*	Design	Setup time	Assertions		Run Time	Simulation tool Coverage**
		Total	Failed	Proven					Total	Failed		
A	1w 3d	170	79	91	24s	100%	A	10min	170	14	20min	91.7%
B	2w 2d	800	304	437	5hrs	92.6%	B	17min	800	83	35min	88.3%
C	5w 4d	8552	5673	877	7hrs	76.6%	C	30min	8552	127	1hr	79.4%

TABLE II
RESULTS USING RECOMMENDED METHODOLOGY

CDC Protocol Verification with Formal							CDC Protocol Verification with Simulation (after Formal)					
Design	Setup time	Assertions			Run Time	Formal Coverage*	Design	Setup time	Assertions		Run Time	Simulation tool Coverage**
		Total	Failed	Proven					Total	Failed		
A	2d	170	32	138	55s	100%	A	1min	170	6	20min	85.7%
B	5d	800	119	654	6hrs	96.6%	B	1min	800	34	35min	83.9%
C	1w	8552	1825	4907	10hrs	78.7%	C	5min	8552	76	1hr	70.6%

* Formal Coverage = ((Failed Assertions + Proven Assertions) / Total Assertions) * 100

** Simulation tool coverage = Coverage metric of Simulation tool = ((Failed Assertions + Covered non-failed Assertions) / Total Assertions) * 100

VI. CONCLUSION

The proposed methodology significantly reduces the time and effort required for Dynamic CDC Protocol Verification closure of designs. Not only does this approach help design and verification engineers overcome the challenges faced during Formal and Simulation verification but also enable them to reap the benefits of both the methods. The methodology is seamless to adopt and saves both time and effort to achieve faster closure.

REFERENCES

[1] Mark Litterick, “Pragmatic Simulation-Based Verification of Clock Domain Crossing Signals and Jitter using System Verilog Assertions”, Verilab

[2] William K. Lam, “Hardware Design Verification: Simulation and Formal Method-Based Approaches”, Prentice Hall, Mar 3, 2005

[3] Sulabh K. Khare, Ashish Hari, “Systematic Speedup Techniques for Functional CDC Verification Closure”, Mentor Graphics, advanced verification white paper

[4] Ping Yeung, “Five Steps to Quality CDC Verification,” Mentor Graphics, advanced verification white paper