Evolution of CDC recipe: Learning through real case studies and methodology improvements

Amit Kulkarni
Intel Corporation
Amit.kulkarni@intel.com

Suhas DS
Intel Corporation
Suhas.d.s@intel.com

Deepmala Sachan
Intel Corporation
Deepmala.sachan@intel.com

***Abstract*- With the increasing complexity of SoC, and ever-increasing count of multiple clocks and resets, Clock domain crossing and Reset domain crossing verification has become an essential part of RTL design verification. Hence, performing thorough CDC and RDC checks are an integral part of any SoC verification cycle. This paper describes RTL bugs found and fixed with bottom-up hierarchical abstract model flow for a graphics chip. Moreover, making sure quality of IP abstract model is up to the mark and interface connections of integrated IPs w.r.t SoC are correct is one of the challenges in SoC CDC verification.**

This article proposes in context of a graphics chip: 1. Run time reduction in the initial phase of CDC/RDC verification 2. IPs integrated in SoC are assumed to be CDC clean, this assumption is verified at the SoC level. 3. Pessimistic clock domain grouping approach followed by grouping source clock and its divided clock in different domains, for strict CDC analysis.

This paper will take you through the journey of key CDC challenges, methodology recommendations and bug finding case studies of a complex graphics SoC.

I. Introduction

Today's System-on-chip (SoC) comprises of diverse multiple blocks offering different functionality. Usually, these blocks operate on different clocks. Each block has several clocks which do not have definite frequency and phase relationship between them. All such clocks are associated with different clock domains. Any path from one clock domain to other clock domain is called clock domain crossing (CDC).

As the demands of end users grow continuously, the designs are forced to support a wide range of peripheral components that implement various communication interfaces. Often those components are provided by a third party and are optimized for concrete clock frequencies that are different from the system's core and other parts of the design. There might be other reasons to utilize multiple independent clocks as well, i.e., to optimize circuit performance or power consumption [2].

A System-on-chip has millions of clock domain crossing paths. All such paths must be protected from metastability by inserting a synchronizer at the destination. For CDC verification at SoC clock domain crossings within the block are verified by the block owner and reverifying them at SoC is unnecessary and redundant. To avoid CDC verification of block at SoC, abstract models having CDC related boundary attributes of the block are generated after block level CDC verification is complete and are used at SoC. This approach of CDC verification is called hierarchical bottom-up CDC verification. Fig 1 shows a typical bottom-up hierarchical CDC flow.

The rest of paper is organized as follows: Section II explains challenges faced in bottom-up hierarchical CDC verification and key recommendations. Later, Section III discusses case studies for the Register Transfer Level (RTL) bugs found in SoC CDC verification. In Section IV proposed future work is explained and finally Section V concludes the paper.

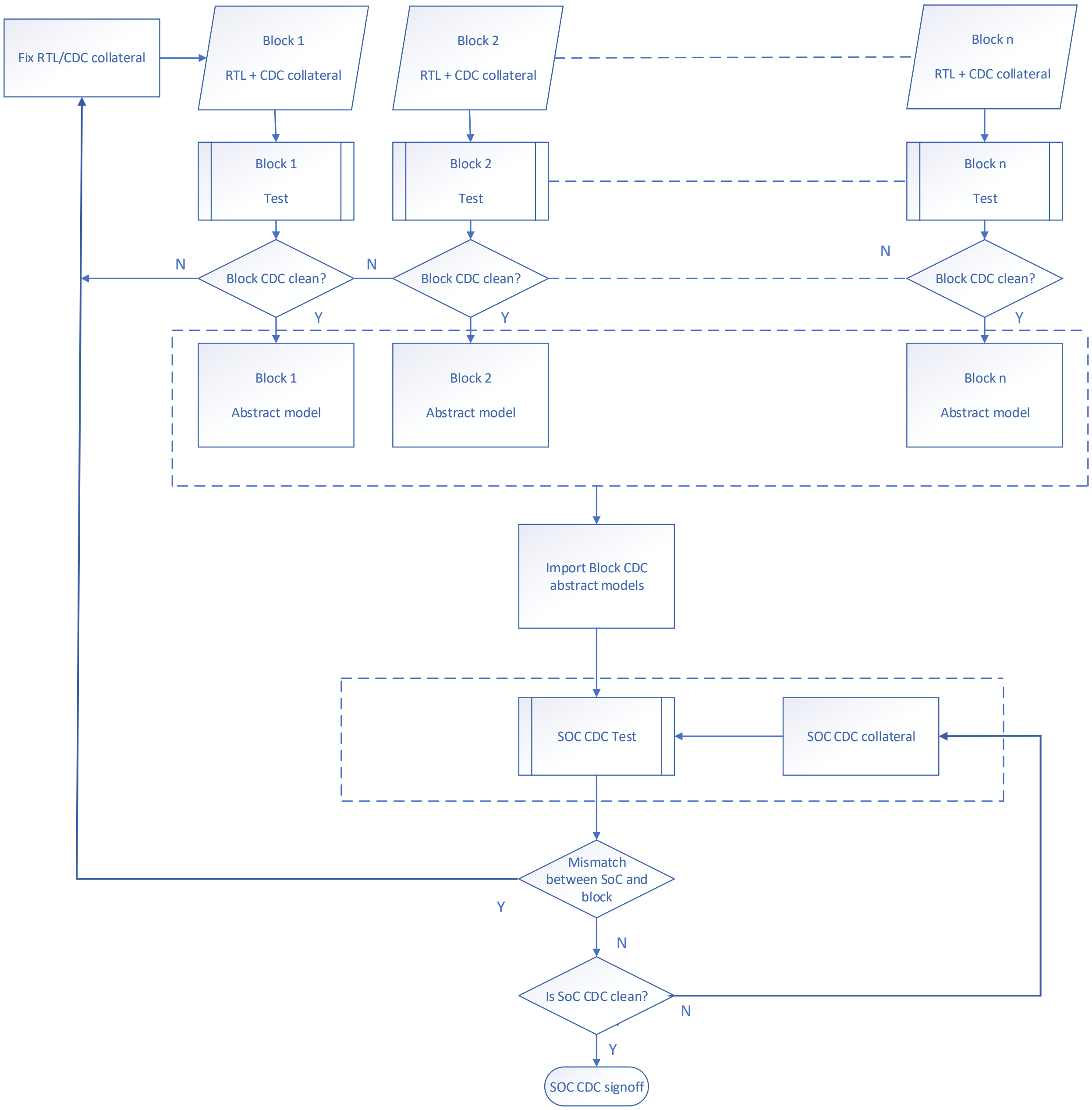


Fig. 1 Bottom-up Hierarchical CDC flow

II. Challenges and Key Recommendations with Hierarchical CDC Verification

A. *Initial phase of CDC/RDC verification*

A graphics SoC is divided into numerous partitions comprising of multiple IPs or blocks. A flat CDC SoC run takes large amount of runtime and memory resources. Moreover, there is no need to perform CDC analysis and verification on already clean CDC blocks. In hierarchical bottom-up CDC flow, as shown in Fig 2, abstract models for CDC clean blocks are loaded. Also, hierarchical CDC verification takes less amount of resources so preferred over flat SoC CDC verification.

One of the challenges in the initial phase of hierarchical bottom-up CDC or RDC verification is long run time required for CDC verification at SoC. Long run time is a result of unavailability of abstract models and incomplete or inaccurate constraints. If violation free block abstract models are not available initially then dirty abstract models can be loaded. Since abstract models have only boundary attributes, CDC tools do not perform CDC analysis inside the boundary at which abstract model is loaded. This helps in run time reduction at SoC. Any clock and reset connectivity issues can be caught upfront with this approach. Later dirty abstract models can be replaced with violation free CDC abstract models.

Not all constraints may be available at the beginning. For example, if reset ordering is not known initially then reset domain crossing (RDC) rules can be disabled, once reset ordering is available then reset domain crossing rules can be enabled. This approach will help in saving run time and memory resources.

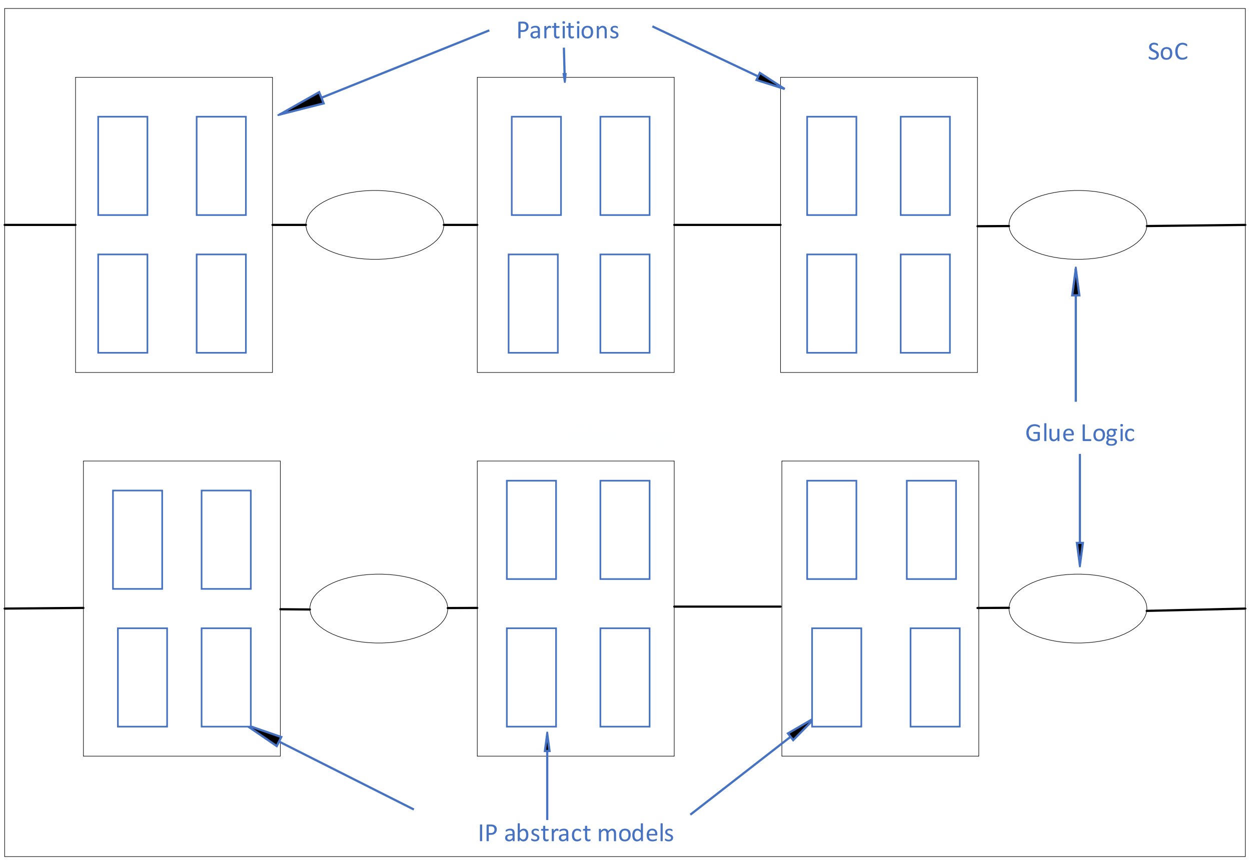


Fig. 2 Bottom-up Hierarchical SoC CDC

B. Synthesizable RTL for CDC

Synthesizable RTL is a stringent requirement to run static CDC checks. Not all RTL is synthesizable such as memories and analog modules. Behavioral model available for memories is not synthesizable. A solution to this is to use emulation models. Emulation models can be made synthesizable by proper usage of ifdef constructs in the code to exclude non synthesizable constructs and execute only synthesizable portion of the RTL code. CDC analysis is not required for analog modules. One approach to deal with analog modules is to load stubs containing I/O information of analog modules. This helps in maintaining connectivity and detects any crossings where analog modules lie in between the crossing.

C. Discrepancy in Design for Test constraints and top-level constraints

Design for test is an integral part of any chip. CDC verification is usually done on RTL where Design for Test (DFT) logic is not present. For a graphics SoC certain DFT blocks are integrated at SoC which distribute the test and functional clocks in all partitions. It is preferred to perform SoC CDC verification along with these clock distribution inserted blocks. These clock distribution blocks have similar RTL structure across multiple partitions and clock distribution logic and constraints generation is automated. However, constraints are manually specified at top level where the clock distribution blocks are inserted.

One solution is to automate the top-level constraint generation however if it is not feasible then large count of false violations are reported due to discrepancy in block level DFT constraints & top-level constraints. Both constraints should match otherwise lot of iterations and considerable amount of time is wasted to resolve this discrepancy.

Other solution to reduce the number of iterations is by proper usage of wildcards in constraints. Wildcards help in generalizing the constraints and constraints get applied even if there are small amount of port or pin name changes in RTL. However, care must be taken when using wildcards in constraints. If the wildcards are used strictly then the number of iterations may not reduce considerably but if they are used generously then wrong constraints might get applied on unwanted ports or paths. A trade off must be made when using wildcards.

D. Multiple iterations for abstract model generation of parametrized IPs

For a modern graphics SoC, multiple IPs from multiple vendors along with internal IPs are integrated in the SoC. Abstract model generation for an IP goes through multiple iterations due to changes in parameter values in the RTL. Every change in RTL should reflect in abstract model. If these changes happen frequently then considerable amount of time is wasted in receiving abstract models from IP owners, loading abstract models and rerunning CDC at SoC. Solution to this problem is that for parameters not affecting CDC can be treated as don’t care parameters. It is necessary to have Don’t care (DNC) abstract generation for parameterized IPs.

E. Discerning between real and false violations

The count of violations with CDC or RDC can be quite overwhelming. It is important to understand and identify real and false violations. False violations are reported due to constraint inconsistency, incorrect and incomplete constraints, unclean CDC abstract models.

Major source of false violations is due to incorrect clock grouping and incorrect reset ordering. If clocks belonging to same clock domain are grouped in different clock domains, then the number of crossings increase, giving rise to lot of incorrect missing synchronizer requirement. Similarly, if reset assertion order is incorrect then incorrect RDC violations will be flagged. Reset assertion order should be such that destination reset triggers before the source reset to avoid metastability. If that is not the case, then reset isolation signal methods should be employed to resolve RDC issues.

Also, proper control availability for selection between test clock and functional clock is required. Appropriate constraints need to be generated to ensure test clock is not propagated in functional mode. All these measures help in reduction of false violation count.

F. Severity of violations

Often some of the important violations are reported as warnings by the tool used for performing CDC static checks. It is important to promote these warnings to errors otherwise these violations can go unnoticed. It is imperative to use same severity for CDC rules for IP and SoC verification.

III. RTL BUGS FOUND IN HIERARCHICAL CDC VERIFICATION

A. Swapped connectivity during Intellectual Property (IP) integration

Although it is trivial that there are various tools that are used to check connectivity between IPs, if thorough checks are not done then connectivity issues can arise. It is a common practice to group clocks with integer multiple frequencies in a single group however a pessimistic approach was followed in clock grouping by assigning different clock domains to master clock and its divided version of clock as a possibility of phase difference between them was suspected. As shown in Fig 3, an unexpected clock domain crossing path was found between 200MHz to 100MHz clock. On further diagnosis it was found that the data paths between 200MHz and 100MHz clocks were swapped. Such connectivity issues can be fixed at an early stage.

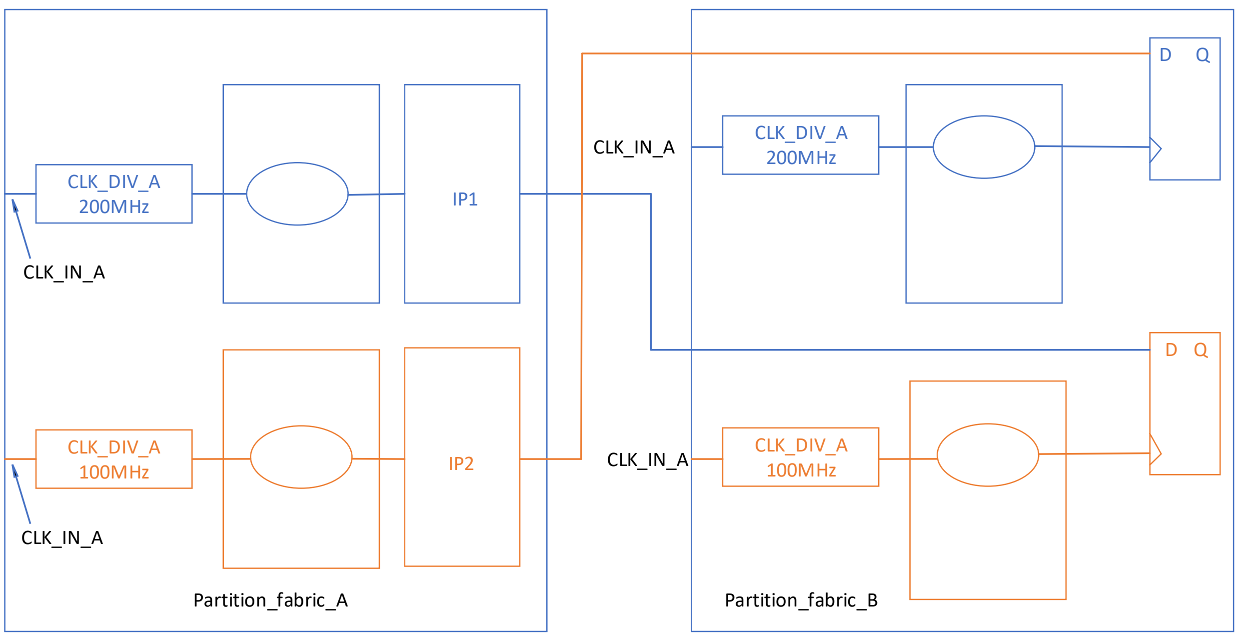


Fig. 3 Swapped Connectivity

B. Incorrect connectivity between partitions placed long apart in SoC

The graphics SoC was arranged into left and right partitions with a hard IP in between them as shown in Fig 4. These left and right partitions were placed to extreme left and extreme right of the chip respectively with a graphics hard IP in between them. No data path is expected in between such partitions. Different clock groups should be associated for the same clock if traversing between left and right partitions. Any clock domain crossings reported in between such partitions represents a potential connectivity issue. This kind of issues can be found early in the design cycle and save valuable place and route time.

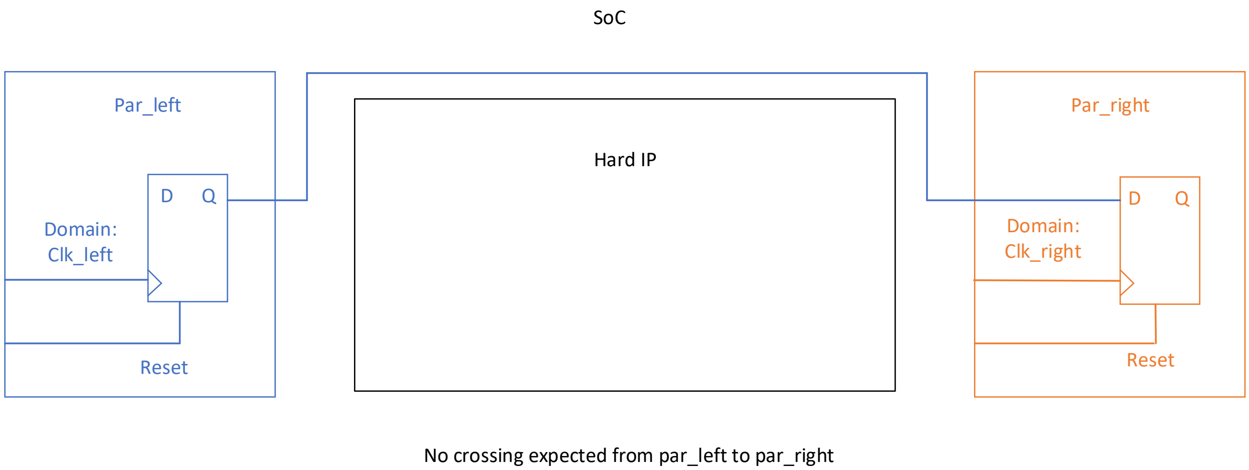


Fig. 4 Incorrect connectivity between left and right partitions

C. *Verification of assumption that IP is CDC clean at SoC*

Accurate and complete SoC CDC verification is only possible when violations free CDC clean IPs are used. Although it is trivial that third party silicon proven vendor IPs are CDC clean, new internal IPs or glue logic that interface external IP with internal logic may introduce a bug. As shown in Fig 5, a signal marked stable or quasi-static can generate wrong abstract model and mask a real violation of missing synchronizer. Since Pin_A is not a stable or quasi-static signal in the abstract model of hard IP but Pin_B is declared as stable, a mismatch is reported. Such bugs can be easily caught at SoC after these IPs are integrated at SoC.

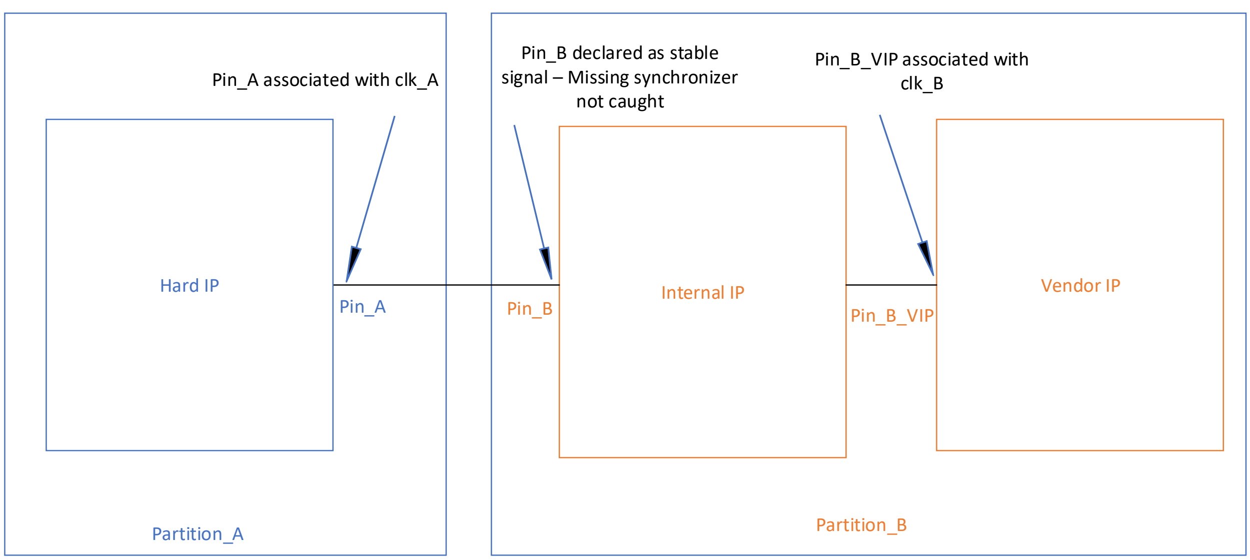
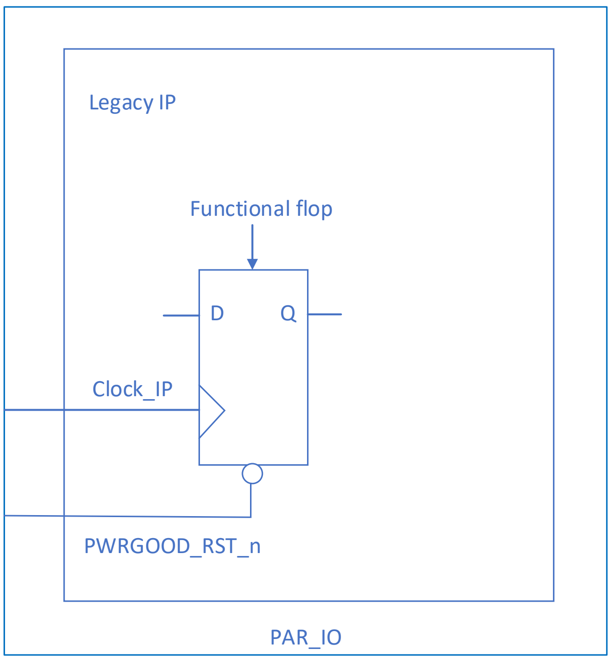


Fig. 5 Verification of assumption that IP is CDC clean

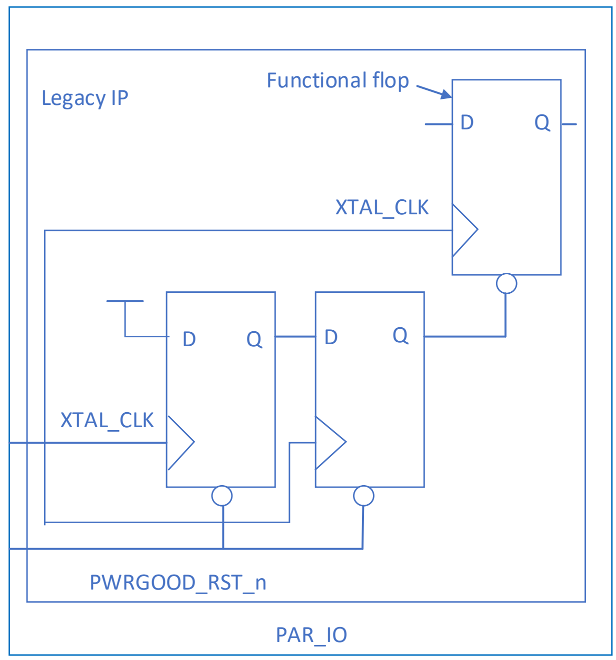
D. *Legacy IP reuse*

Silicon proven legacy IPs cannot be blindly reused. Minor modifications in legacy IPs should be thoroughly verified. Some resets such as power good resets are triggered during the initial operation of chip once power is up and remain stable for entire operation of the chip. Functional clocks are triggered after these resets are stable. In this case, clock of the functional flop i.e. Clock_IP (shown in Fig 6a), gets triggered after reset is de-asserted. Since Clock_IP gets triggered after reset de-assertion initially there was no need for de-assertion synchronized reset synchronizer, however, Clock_IP was replaced with crystal clock which triggers before the reset. So, a need arose to add reset synchronizer in the legacy IP as shown in Fig 6b. It is quite easy to miss such important synchronizer requirement so legacy IPs should be thoroughly verified if there are any clock changes.



Legacy IP before changes in clock

Fig. 6a Original Legacy IP



Legacy IP after changes in clock and reset synchronizer added

Fig. 6b Modified Legacy IP

E. *Replacement of double/triple flop synchronizer with technology specific synchronizer*

Third party IPs use hand coded or generic technology double or triple flop synchronizer. When such IPs are used in the SoC they should be replaced with technology specific synchronizer. This gives multiple benefits. First, and most important, it ensures easy recognition of synchronizers in STA and eases downstream checking. Standard-cell synchronizers also ensure the multi-stage flip-flops are placed together so they do not affect metastability.[1]

IV. FUTURE WORK

A. *Functional CDC dumped SVA to be plugged in simulation environment*

Functional CDC stage generates assumptions for the input constraints. We plan to use tool generated assumptions in the testbench/simulation environment to make sure they are not violated.

B. *Prioritize certain set of violations*

Current state of art static CDC tools dump all the violations without giving user a priority for fixing violations. If the violation count is large with multiple different categories, then it is essential to understand which category of violations need to be fixed first. Tool should have the ability to prioritize certain violations which require immediate user attention such as unsynchronized crossings over case analysis mismatch or virtual clock mismatch so that important issues are fixed quickly.

V Conclusion

This paper describes in detail about typical challenges in hierarchical bottom-up CDC verification and the methods we used to overcome them has helped us to reduce the CDC verification time. The techniques we adopted have helped in unearthing multiple connectivity issues and corner cases which are usually overlooked because of the trivial procedure of CDC verification.

We have proven the viability of our approach by providing real case scenarios that helped us to find bugs at an early stage. The case studies described in this paper has helped us uncover missing CDC and reset synchronizers which comprises of 90% of the total CDC bugs found at SoC.

This approach of verifying IP assumptions at SoC and pessimistic approach of clock domain grouping in hierarchical CDC flow has helped us to achieve pre-silicon CDC sign-off criteria prior to tape-out.

References

[1] S. Chaturvedi "Static analysis of asynchronous clock domain crossings", Proceeding DATE '12 Proceedings of the Conference on Design, Automation and Test in Europe, 2012, pp. 1122-1125.

[2] V. C. Vimjam, A. Joseph "Challenges in Verification of Clock Domain Crossings", Real Intent Inc., Design Automation Conference (DAC'47), 2010.

[3] Cadence, "Clock Domain Crossing," White paper. http://w2.cadence.com/whitepapers/cdc_wp.pdf

[4] C. E. Cummings, "Clock Domain Crossing (CDC) Design and Verification Techniques using SystemVerilog," SNUG 2008

[5] S. Sarwary, S. Verma, "Critical clock-domain crossing bugs", Atrenta Inc., 2013.

[6] SpyGlass CDC, Synopsys, May 2017. [Online]. Available: http://www.solvnetplus.synopsys.com