Implementation of a closed loop CDC verification methodology

Andrew Cunningham, Intel, andrew.cunningham@intel.com

Ireneusz Sobanski, Intel, ireneusz.sobanski@intel.com

Abstract

Designs with multiple functional blocks and multiple clock domains require clock domain crossing (CDC) analysis in order to verify that the effects of metastability issues can be tolerated in the hardware. For a standard multiple clock domain design, CDC analysis completes having found thousands of CDC violations. Real CDC issues will be fixed in RTL but the remaining "false" violations are generally waived. As part of the design flow, the traditional CDC verification methodology is extended to automatically filter and verify the false violations generated during CDC analysis. Design rule checks are also applied to ensure CDC logic is preserved from RTL through to the final layout tapeout netlist.

The result of this verification flow permitted the design team to minimize the risk of CDC false violation being incorrectly waived. The approach significantly reduces the manual effort required to analyze real CDC violations within a design. Using the standard CDC methodology produced ~12K CDC violations which would usually require manual debug. By using the closed loop methodology, only 728 of the ~12K violations required manually inspection, reducing the number of violations required for manual debug by 94%. We will show that by analyzing the initial CDC run and using simulation and assertions, it significantly decreases the effort required to complete the CDC verification flow. By applying simulation techniques to verify the CDC violations, we reduce the risk of human factor error in the flow and take a significant step forward in terms of quality silicon itself.

1 Introduction

Clock domain crossing (CDC) occurs in a design which contains multiple clock domains. The device under test is communications chipset containing security algorithm accelerators with 26 separate functional clock domains. 19 of those clock domains are contained within the PCI-express endpoint IP which is the primary focus of this paper. In multi clock designs it is impossible to avoid metastability with the design. A well-defined CDC verification methodology will identify all signals at risk within a design and analyze whether or not those signals can tolerate the effects of metastability. The design tape-in sign-off goal for every CDC verification flow is to achieve zero RTL CDC violations prior to the generation of the final layout netlist, thus avoiding finding timing issues in silicon which are costly to fix.

Traditionally, false violations represent a high percentage of violations reported by the CDC tools in a design, i.e. they will never be selected in normal functional mode. Given the increased Time to Market (TTM) pressures which face ASIC design teams, false CDC paths need to be quickly waived in order to achieve sign off criteria. The vast number of false violations generated by a CDC tool increases the probability that a designer could easily waive a real violation. The only way to ensure that false CDC paths have been correctly waived is to verify the CDC data points through simulation and assertion techniques. This issue of waiving false violations becomes even more crucial when the design team is required to waive CDC issues in legacy "golden" IP blocks which they do not own and no longer have design support for.

This paper describes the standard SoC CDC flow adopted with closed loop flow updates required to decrease the CDC flow execution time while reducing the risk of waiving real CDC issues. The flow automates the removal of false violations from the CDC flow while also validating the assumptions made by the design team in achieving a clean RTL CDC sign off. The flow also describes how to check that synchronization logic remains consistent throughout the design process from RTL to the final tape-in netlist.

The paper is organized as follows: Section 2 provides an introduction of the effects of metastability in a design system. In Section 3 the standard CDC flow is defined. In Section 4 we discuss the issues faced when applying the flow to our design. In Section 5 we discuss the improvements we have made to enhance the CDC flow. In Section 6 we illustrate how we apply these enhancements through simulation. Section 7 contains the results from our modified flow and Section 8 the summary of our experiences.

2 Metastability

A CDC path is illustrated in Figure 1[1]. The clock source and destination domains are asynchronous. The 'bclk' register risks entering a metastable state if the input signal 'adat' violates the setup and hold timing

condition for ‘bclk’. When a register enters a metastable state it does not maintain a determinate value of logic ‘0’ or logic ‘1’. The output of the metastable circuit can lead to random behavior in the circuit. If the design cannot tolerate metastability then the risk of introducing functional errors into the system increases.

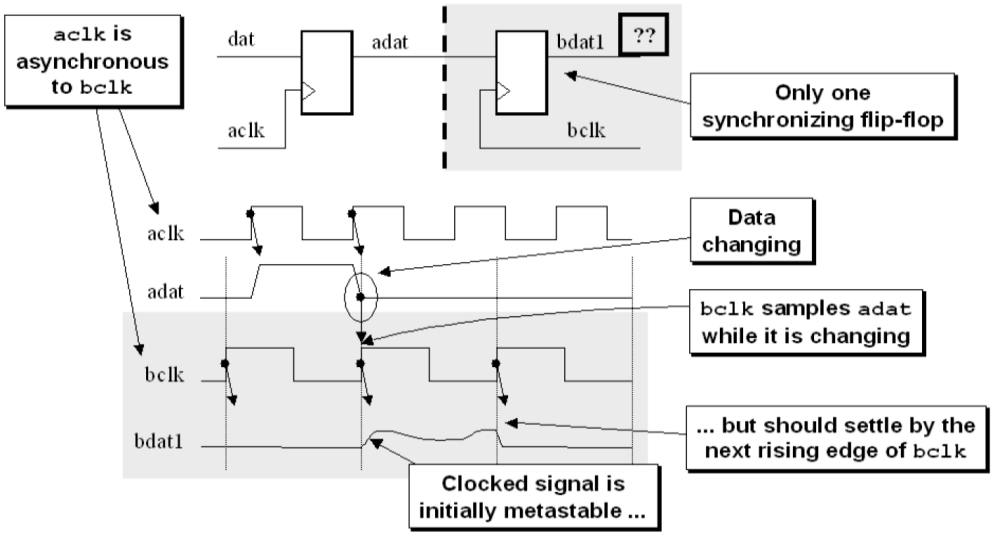

Figure 1 Metastability

Metastability can lead to functional errors in a system when the random output of receiving clock domain inadvertently causes an illegal logic condition to be propagated to the downstream logic cone. A signal in a metastable state increases the probability that different parts of downstream logic cone may sample different values for the metastable signal. The fundamentals of digital logic design require a signal to have one and only one value. Violation of this rule can lead to systematic failure within the digital system. While metastability cannot be eliminated within a multi-clock design, well established clock domain crossing techniques can be implemented to nullify the effects of metastability.

The base building block for safely transferring a signal from one clock domain to another is the synchronizer module which consists of at least two back-to-back flip flops. The first stage flop in the synchronizer samples the asynchronous input allowing one clock cycle for the output signal to stabilize. The second stage flop samples the first stage with the expectation that after one cycle the first stage flop will have stabilized and all metastability effects have been removed. The output of the second stage is assumed synchronous to the new clock domain. In some cases three or four back-to-back flops are required to remove metastability from an asynchronous input, but in general a two flop synchronizer will reduces the risk of metastability to an acceptable level.

In this section we have described the cause and effects of metastability. In the following section we will describe the methodology required to ensure the design can tolerate metastability within the system and safely transfer data across multiple clock domains.

3 Standard CDC Methodology

A complete CDC verification methodology consists of three main verification components: structural verification, protocol verification and metastability verification. For the purposes of this paper, the authors have selected an industry standard tool for CDC. The standard CDC flow is depicted in Figure 2. First the design is compiled and loaded into the CDC tool along with a control file to generate a CDC database.

To simplify processing the report generated by the structural CDC analysis, the tool presents the data in four basic types: Violations, Cautions, Evaluations and Proven. Violations do not adhere to any predefined CDC scheme, Cautions indicate the CDC path could potentially cause metastability issues and a protocol checker should be used to verify the interface logic. Evaluations and Proven types are valid signals which can be mapped to a valid CDC scheme. The violations and cautions can be broken down into unique types: Missing synchronization, reconvergence, multi-bit bus crossings and combinational logic on the input to a synchronizer cell. All violations must be carefully examined and a decision is made whether to fix or to waive. The flow is then rerun until all structural checks pass.

After the structural verification has completed, assertion checkers for the violation and caution data types can be automatically generated by the CDC tool to cover the protocol verification of the CDC signals. The assertions are bound to the simulation model and are checked via pre-silicon test case validation regressions. This requires that all further debug must be done in the validation environment. Our functional

validation environment already contains a non-synthesizable behavioral synchronizer which models the effects of metastability. This is done by randomly gating the transition of bits within the synchronizer cell and mimicking the potential cycle slip introduced by a metastability event. This enables metastability checking to be performed using the simulation environment without having to enable a license for the CDC tools Metastability PLI and reduces the cost of running the CDC flow.

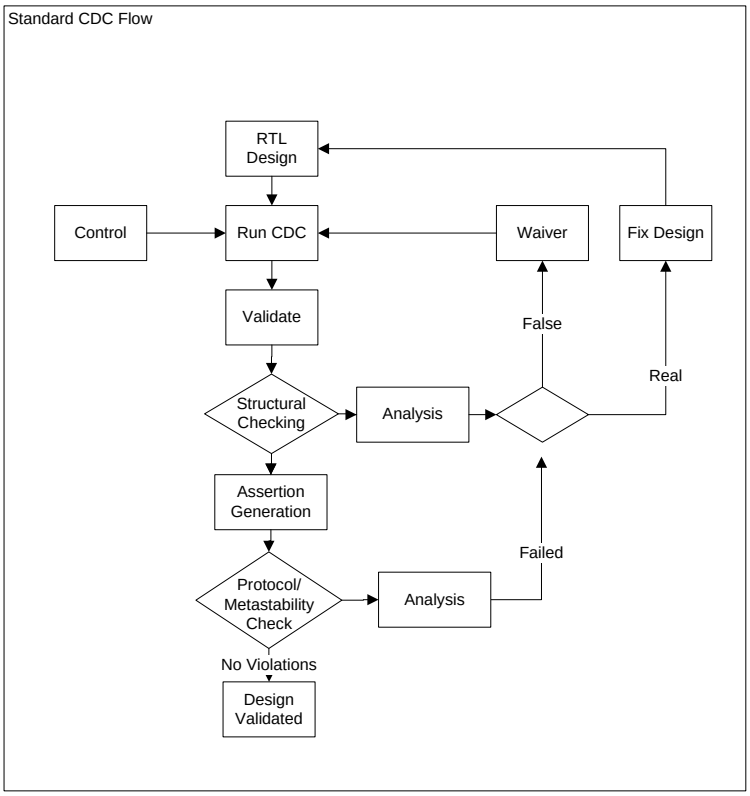

Figure 2 Standard CDC flow

In this section we have described how the standard CDC flow works, in the next section will categorize the issues found when trying to implement this flow.

4 Types of CDC Issues

During the initial structural phase of CDC analysis a significant number of violation messages can be generated which are not necessarily the real violations. Lee et al [2] describe five potential sources of such false violations and their percentage in the overall false violation number.

- Type 1 – wrong clock specification, responsible for 7 to 28% of violation, depending on the size of the project
- Type 2 – quasi-static (stable) signals, responsible for 40 to 67% of violations, depending on the size of the project
- Type 3 – dynamic async-interfaces, responsible for 16 to 20% of violations, depending on the size of the project
- Type 4 – non-standard asynchronous design, resp. for 6 to 7% of violations, depending on the size of the project
- Type 5 – CDC unfriendly design, responsible for 1 to 6% of violations, depending on the size of the project

The first three types account for approximately 90% of the overall violations. In this section we will further define each of the violation types and give an overview of how those issues were handled in our flow.

The first type, wrong clock specification, is related to the configuration of the design which is generally caused by incorrect clock port specification, incorrect clock domain specification of primary input/output, or constant values on primary input/output.

The second type is related to quasi-static (stable) signals, this represents all configuration status registers (CSR) and other (undefined) static configuration signals (e.g. Fuses). All such signals are modified just once after the reset and are generally stable during normal operation. From our experience as well as external publications [2] this kind of issue is the main source of the false violations, and may relate to as many as 67% of the all false violations.

The third type is described as dynamic asynchronous interface which use internal control logic to safely transfer data without employing synchronization mechanisms. The signals are not synchronized with a standard protocol (dmux, handshake, FIFO, etc.) and their values are only used when the source signal is guaranteed to be stable. The asynchronous interface is sampled depending on internal state of the device. It's very hard (impossible) to identify such mechanisms during static (structural) analysis, due to its dependency on the functional behavior.

The fourth type is non-standard synchronizer types which the CDC tool does not recognize. Customized synchronization modules which are guaranteed to be metastability free but do not adhere to one of the predefined synchronization schemes.

The last type is a CDC unfriendly design, which generally relates to non-synthesizable components (behavioral code, physical layers, DLLs, etc.) and sporadic usage of the gate level netlists. In the first case, all non-synthesizable components will be black boxed.

The violations discovered after CDC analysis in the design are consistent with the types defined in this section. Although the percentage of the violations varies, we can conclude that all potential false violations are covered with these 5 types. In the next section we will discuss the steps which need to be implemented to order to minimize impact of the false violation on the CDC analysis.

5 Automatic Configuration of the CDC Flow

In this section we describe the issues we faced when trying to validate the types 1, 2, 4 and 5 violations in the design and we outline the steps taken to overcome these issues.

Our first run of the CDC flow highlighted Type 1 violations as a significant issue. Determining all port information (clocks, clocks domains, port clock domains) if often complicated given that documentation for internal blocks was poor or in some cases non-existent. The problem is not unique to our design team and is likely to be encountered by other teams who inherit IP. The problem is especially visible in SoC designs, where we may have numerous asynchronous clock domains, with clock gating and multiplexing at different points of the design. These clocking implementations make it difficult to generate the CDC port configuration from an automated flow.

It can be overcome by applying a systematic configuration procedure using constraints from other design tools e.g. synthesis, STA, to significantly reduce the number of issues related to incorrect ports or clock definitions. Due to chip and clocking complexity it required CDC analysis to be run at a cluster or subsystem level. PrimeTime proved to be the best method found for generating cluster level configurations. Scripts were generated which extract the cluster level IO clocking information and translated them into a CDC input control file.

To address type 2 violations, a script automatically generated a stability assertion for every violation and caution detected by the structural analysis step of the flow. These assertions are then simulated in our standard validation environment. The CDC tool also supports an embedded assertion generation mechanism. In comparison to the CDC tool approach, our in-house solution gives us more flexibility in context of assertion generation, and removes the needs of using additional tool (our simulation environment relying on VCS simulator) to simulate proprietary CDC assertions (here the CDC tool would have to be run in parallel to the VCS simulation). More information about the generation as well as simulation of the assertions will be described in the next section.

For type 4 violations, the CDC tool provides the ability for the user to define custom synchronization modules and clearly define the clock behavior of the cells ports (e.g. clock domains). The standard library synchronization modules are defined as custom synchronizer cells. By defining the ports of the custom sync cell the tool does not simply waive these blocks but also automatically validates the usage of these modules within the design. It is recommended that all clock domain crossing paths are synchronized by special cells which are hardened against metastability. By limiting the tool to only using these cells, unintentional instantiations of double flip-flop RTL coded synchronizers in the design can easily be found.

By adopting this approach the tool requires that all synchronization logic within the design is built using the base library synchronizer components. Enabling the detection of FIFO, handshake and complex CDC schemes allows the tool to recognize all of our custom sync modules. Using the base library components also enables the counting of the number of synchronizer cells instantiated within the RTL design. This step was added due to an issue that arose on a previous project due to a combination of a library issue and an incorrect constraint which swapped synchronizer cells for standard registers in our tape-in netlist. By reporting the synchronizer count from the RTL and comparing it to the final PrimeTime database, it is ensured that all logic required to avoid metastability is present in the final design layout.

For type 5 violations, the case of finding a black-box module can often lead to generation of CDC violations. So it is important to monitor the logs in context of finding unwanted black boxes. It is important to identify non-synthesizable components which will be black boxed within the design. In our experience finding black box modules can generate significant numbers of violations. It is good practice to configure the CDC tool to

highlight all inferred black boxes as errors to simplify identification of these violation sources. The PrimeTime IO configuration scripts can then be modified to extract the timing information for the black box modules. When there is no IO clocking information present in the PrimeTime database we recommend treating the signal as asynchronous.

In this section we have highlighted the main causes driving the false violations in our CDC analysis and we have also identified the steps required in order to reduce the number of violations. In the following section, we will illustrate the steps that can be applied to reduce the number of false violations in the CDC flow.

6 Simulation

In Figure 3 the new modified flow for CDC analysis is defined. Most type 1 violations are removed with the application of the clocking data extracted from PrimeTime. After completing the structural CDC analysis step in the flow there is generally a large number of violations to analyze and this task maybe time intensive. In order to process these violations more efficiently we propose using functional simulation to remove a large majority of false violations, thus reducing manual debug effort. Simulation can also be used to validate waived CDC paths (if such has been defined).

Using the structural analysis reports stability assertions are generated for simulation via a script so violations related to quasi static signals – CSR registers, fuses, etc. can be automatically filtered from future CDC runs and verified in the validation environment. This type of violation can be responsible for 40 to 67% of all false violations, so early identification can significantly reduce time spent on design analysis. After completing the stability assertions task in the flow, the user can remove the false violations from the flow in one step and no longer is required to iterate between the analysis, waiver and run sections of the flow.

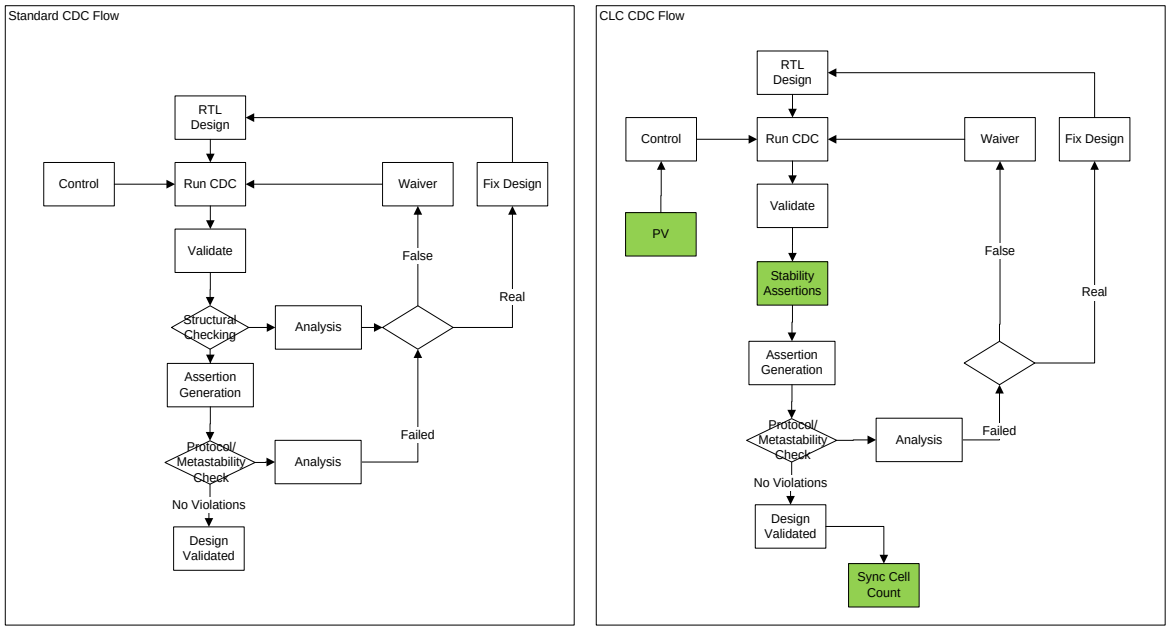

Figure 3 CLC CDC Flow

To generate the assertions we are using one of the standard CDC report file – clock crossing report, generated by the following command:

- cdc generate crossings –csv crossing.csv

The report contains the list of all clock crossing detected during structural analysis together with all basic information related to the crossing path:

- Type – Violation, Caution, Evaluation, Waived, Filtered
- Check – "Missing Synchronizer", "Multiple Bits", "Combo Logic"
- TX Signal – source register path, e.g.: *ipxppl.pciex16pl.mtd2pxppl_cfg_fifo0.entry2_synced_400to250_ff*
- RX Signal – destination register path, e.g.: *ipxppl.pciex16pl.mtd2pxppl_cfg_fifo0.sync2fifo_dout*
- TX Clock – TX register clock domain (clock group) or path to the signal (if not in the clock group)
- RX Clock – RX register clock domain (clock group) or path to the signal (if not in the clock group)
- TX Module – name of the TX module
- RX Module – name of the RX module
- Custom Sync Module – name of the custom synchronization module if applied, otherwise empty string
- TX File – TX module definition file, and line number for the output assignment (TX Signal)
- RX File – RX module definition file and line number for the input assignment (RX signal)
- ID – clock crossing path label, e.g. no_sync_96403 – uniquely identifies the clock domain crossing path.

The script analyzes the *crossing.csv* file and generates waivers for the CDC tool and assertions for simulation based on the type of crossing. Currently only three Checks (Violations) are taken into account - “Missing Synchronizer”, “Multiple Bits”, “Combo Logic”, but more may be added when needed. The assertions generated from the file are pretty simple, and they are just looking for changes on source signal after configuration phase. In order to accommodate this in the validation environment, a validation hook was added which defined our environment as either in configuration or application mode.

While in configuration mode assertions are held in reset and only activate when entering the application mode. When the design is in application mode all pseudo static (configuration) signals should be stable. Assertions which fail during simulation are assumed to be valid CDC issues and require further debug. The assertions which pass confirm the crossing as false violations and can be safely waived.

The assertion is defined in the following way:

- Name: assert property (
  @(posedge TX_clock)
  !$stable(TX_signal) |-> !config_phase);

The Name is the ID of the clock crossing path so it can easily be used to identify it and bind to the violation in the CDC report. The transmit signal and clock are taken from the crossing.csv file and config_phase enable signal is driven directly by the operation mode: configuration or application. The user can also detect simulation holes by checking the activity on the assertion precondition. A lack of activity on the assertion precondition can be inferred as a functional code coverage hole in the validation environment.

The remaining violations may match types 3 or 4, but they require closer attention and some level of manual analysis. To make the validation more robust and immune from simple mistakes, it is proposed to avoid simple waiving of these violations. From experience these types of violations have always been accompanied with assertion in the RTL. If the user can justify the lack of synchronization with any property of the design, then that assertion should be extracted from the RTL and used to waive the issue. This would work better if it can be done automatically but our current implementation requires manual identification and addition of these assertions from the RTL.

In this section we have described the modifications we have made to the CDC flow, in the next section we will describe the results of applying these modifications.

7 Results

The results from applying our modified flow which are described in Figure 3 and Section 5 are captured in Tables 1 and 2. We are able to clearly demonstrate that our flow resulted in 94% reduction in the number of CDC violations which required manual debug effort. The largest gain was achieved by applying the data gathered from the physical verification (PV) or PrimeTime database to the configuration signals, where we were able to reduce the number of violation for about 64%. After our first simulation run, when pseudo static (configuration) signals were identified, we were able to reduce the violation number for further 30%. The overall task time is reduced from 26 man days to 2 man days. By using automatically generated assertions in our validation environment we reduced the risk of human prone error in waiving CDC violations and enable a significant step forward towards a quality silicon release.

Table 1 CLC CDC Results

Description	#CDC Crossing to analyse	# Violations to debug as percentage of Total Violations	Task Time in Man Days (1 min/violation)	
All clock domain crossings	45664	n/a	-	Initial flow setup based on standard flow with all clock domain crossing bits (without reconvergence). The data is for the PCIe endpoint cluster in communications chipset product and is purely for design references.
Violations (Structural 1)	12274	100%	26	Number of violations after first run of the CDC without PV input. All clocks, clock domains as well as known stable signals have been defined. It can be seen as first clean run of the CDC – our overall number of violation.
Violations (Structural 2)	4456	36%	9	Number of violations after second structural analysis when all ports and black boxes has been configured. PV input and stability assertion generation enabled. The violations reduced by 63.7% (type 1 & 5) of the previous stage
Debugging needed	728	5.9%	2	Number of clock domain crossing, which had to be manually debugged The violations reduced by 30% (type 2) of the previous stage

The synchronizer count was extracted by counting the number of synchronizer library cells in the CDC report. A similar count was also performed on the final full chip PV database. The results are summarized in table 2.

Table 2 Synchronizer Cell Count

CDC	sync01nnx10		17
	sync02nnx10		962
	sync02nnx20		1720
	sync02nnx10		46
	Total		2745
PrimeTime	Total		2745
		Delta	0
		Result	Pass

At first the synchronizer cell count did not match up as the CDC database contained more sync cells than the PV database. After close inspection the CDC compile settings did not match the synthesis compile settings and once this was fixed the synchronizer count matched up. So we advise designers to take care when applying the correct compilation settings for CDC.

8 Summary

This paper presents an improvement to the standard CDC methodology which can be used to accelerate the time taken to complete the CDC flow and to reduce the risk of false violation escapes.

The advantages of adopting these flow updates include:

- Leveraging design information from different design tools in order to accelerate CDC configuration setup
- Initial false violations are automatically filtered and verified
- Designer only has to worry about real violations
- CDC functional validation holes can be highlighted through simulation
- Overall CDC flow execution time is reduced
- CDC synchronization logic is tracked through the design process
- Removes requirement to run proprietary CDC tool in parallel to standard RTL simulator to verify CDC assertions

This work builds on the proven CDC design methodology deployed widely within Intel.

There is still a need to fix how the flow automatically deals with RTL synchronization based assertions. Future work will look at how to extract reconvergence violations and automatically analyze.

In any case, we believe the lessons learned by the experiences described in this paper should be considered for application in future multi-clock domain designs within the SoC design community.

Acknowledgments

The authors would like to acknowledge the Intel Shannon, Ireland design team for their valued contribution to this paper.

Bibliography/References

[1] Cummings, Cliff. E, “Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog” Sunburst Design

[2] Youngchan Lee, et al, “Millions to thousands issues through knowledge based SoC CDC Verification” Samsung