2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA
Sept 25-26, 2014
Hotel Park Plaza
Bangalore, India

Automated, Systematic CDC Verification Methodology Based on SDC Setup

Ashish Hari (*ashish_hari@mentor.com*), Sulabh Kumar Khare (*sulabh-kumar_khare@mentor.com*)
Design Verification Technologies, Mentor Graphics Corporation, Noida, India

***Abstract*—we present an automated approach to CDC setup using information extracted from SDC files. This approach accelerates the CDC verification process by reducing noise and making the methodology non-iterative. Another important step is to qualify the extracted CDC setup information before applying the setup to analysis, so we introduce various visual aids and show some methods that help review and qualify the extracted CDC constraints. We describe how we integrated this methodology with our CDC tool and show our results for a real ASIC design.**

Keywords— Clock domain crossing, SDC, Verification, CDC constraints extraction

I. INTRODUCTION

Clock domain crossing (CDC) failures are the second most common cause of chip failures (after functional issues); hence CDC verification is a critical sign-off requirement. Furthermore, CDC verification complexities and times-to-closure continuously increase—with large expansions of design sizes, numbers of constraints and numbers of asynchronous clock domains. Importantly, effective CDC verification depends on setting up the most accurate design constraints. Incomplete, incorrect setups hinder CDC verification results. They increase process iterations; results are "noisy" and they compromise time-to-closure.

Conventional CDC verification starts with user-defined partial clock specifications and other constraints information. This information in turn controls CDC analysis and is later verified with respect to the analysis results generated by the CDC verification tools. Based on a review of these results, the CDC analysis setup is refined iteratively.

Clocks are grouped into appropriate domains. Port constraints are finalized. Mode signals are identified and set to constants. False paths and stable signals are marked. The conventional constraint refinement process is painfully long and iterative in nature. It depends upon—and varies with—the verification engineers' knowledge and experience. It is guided by advice from the designers at various points. Each iteration involves analysis and review of partial, noisy results—for example to identify missing constraints. Adopting a constraint setup that reduces dependency on the verification engineers' design knowledge, that cuts CDC tool iterations, and that eliminates noisy constraint results is extremely beneficial.

To complicate matters, CDC analysis setup information is captured in proprietary formats designed for their CDC tools. Such non-standard specification makes it harder to keep specifications up-to-date with design changes. The Synopsys Design Constraints (SDC) format is the most common industry standard for specifying timing analysis constraints and exchanging design intent specifications among most EDA software platforms, including synthesis tools, timing analyzers, place and route systems and so on. SDC specification data evolve with the design.

Of particular interest for CDC verification, SDC format data also contain design setup information. And, this information can be extracted and used to set up for CDC analysis. However, the process has drawbacks. Extracting CDC setup information manually from SDC files is tedious and prone to errors. But it has great advantages: it greatly reduces CDC analysis setup time; it ensures that setup data are preserved in generic form during the entire design flow; and it lowers barriers to verification tool interoperability during all of the design verification phases.

An additional key challenge of using SDC specifications for CDC analysis setups relates to how information extracted from SDC correlates to CDC setup data. SDC is used for static timing analysis (STA), which is complementary to CDC analysis. Whereas STA defines paths in synchronous clock domains, CDC tools analyze paths across asynchronous clock domains. So, when a tool extracts CDC setup information from an SDC file, it must provide the extracted setup information to the user in an environment that makes it easy to review, debug and qualify.

In this paper, we present an automated approach to CDC setup information extraction from SDC files. This methodology accelerates the CDC verification process by reducing noise and making the process non-iterative. We detail our CDC information extraction engine, which helps engineers interpret the SDC information imported to the CDC setup. We also present visual aids to help them review and qualify SDC extraction and clock domain inference from SDC information. Multiple SDC commands contribute to decide the clock domain for a particular clock. Visualization helps verification engineers review and debug the process and qualify their decisions.

II. TRADITIONAL CDC VERIFICATION METHODOLOGY

Generating reliable CDC analysis results is a multiple-phase process. In the first phase, the verification engineer prepares the design setup by extracting appropriate design information, for example the design clocks specifications and the functional modes of the design. The CDC analysis tool extracts this information automatically and additionally, the engineer adjusts the extracted data manually. The next phases are iterative. The engineer runs the CDC analysis tool, reviews the results, fine-tunes the design constraints to eliminate false results and iterates the process. For the fine-tuning step, the engineer updates design constraints to group synchronous clocks in appropriate domains, specifies stable and constant signals for modal analysis, marks functional false paths and adds necessary design constraints. Reviewing results after each tool run is critical. The verification engineer must analyze the results carefully, eliminate the false violations by updating the design constraints and then re-run the tool. The iterative process of running the tool and fine-tuning design constraints continues until acceptable results are achieved.

When design constraints are stable and results are acceptable, the engineer advances to the next phase. Results are ready for corrective action, which is the core objective of the CDC verification process. The traditional approach has several problems:

- The constraint tuning process is iterative. It loops back through the flow multiple times (see *Fig-1*), which makes it error prone and time consuming.
- Process effectiveness and productivity depends upon the subjective expertise of the verification engineer, who determines the causes of false results and fine-tunes relevant constraints.
- Running the CDC verification tool with incorrect constraints severely impacts performance. For example, the tool spends unnecessary effort analyzing functional false-path crossings because of missing false path constraints in initial runs.
- Extracting constraint information is quite complicated. Relevant information about the design at the verification stage requires interaction with block designers—who typically are geographically isolated.
- Setting up of CDC verification is effort-intensive. Unfortunately, CDC analysis and validation are typically delayed until the design is mostly complete. In particular, CDC verification is not done regularly at the time of block development.

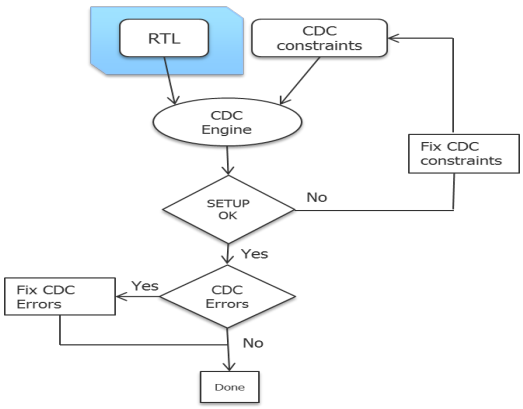

Fig-1: Conventional CDC verification method

III. CDC VERIFICATION WITH SDC SETUP

With our proposed SDC setup, the CDC verification process becomes non-iterative, as shown in *Fig-2*. The verification engineer starts with the design RTL and the SDC file used for the design. Our CDC information extraction engine reads the SDC file and generates information used to set up CDC analysis, including clock specifications, clock group specifications, port-clock relationships and constant signals. An SDC visualization graphical user interface (GUI) provides the engineer a view to quickly review the extracted clock trees and clock groups.

Multiple SDC commands can contribute to a clock group's inference. The visualization GUI shows a step-by-step command generation graph for each clock group inference. Port clock relationships and constants specification are also validated at this stage, so all CDC constraints are qualified at this phase.

Once the engineer has reviewed the CDC constraints and has validated the setup data extracted from the SDC file, they run CDC analysis using the generated CDC constraint setup.

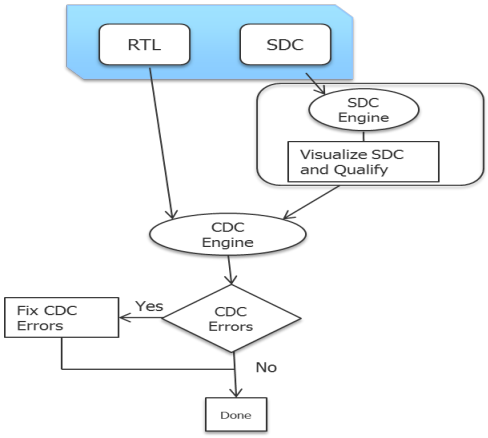

Fig-2: SDC setup-based CDC verification method

This proposed approach as many advantages:

- Constraints are accurate and reliable because they are derived from the designers' specifications.
- CDC verification, timing, synthesis and other EDA tools used in the entire design flow utilize a common setup.
- The same SDC file is used for various tools, which enhances tool interoperability.
- GUI visualization gives a complete understanding of the STA setup and how it is adapted for CDC.
- The SDC constraints extraction and visualization phase is now very fast compared to the CDC analysis run.

Validating constraints before the CDC analysis run saves lots of run time. The CDC tool focuses more on finding real bugs than on wasting time analyzing false issues.

IV. SDC COMPONENTS FOR CDC

An SDC specification contains design constraints as groups of TCL commands. These commands specify timing, area and power constraints for the design. The CDC tool examines every crossing between asynchronous clock domains and validates whether or not the crossing is properly synchronized—typically by a *synchronizer*. As input constraints, the CDC tool requires the specification of the asynchronous clocks. It also must have clock domain information about the input and output ports of the design. Moreover, CDC tools must account for constant settings applied when the design operates in its various modes.

For clock constraint mapping between the SDC specification and the CDC tool, data extraction determines the relationships between different clocks and filters clocks that are asynchronous to each other. The interpretation of the SDC specification as it applies to clock domain analysis is explained in this section, including descriptions of the critical information extracted from SDC specifications.

A. *Clocks and clock domains*

SDC defines primary design clocks as well as the generated clocks with reference to primary clocks. In the SDC world, all clocks by default are synchronous except when specifically overridden. Relevant constraint commands used to extract clock information and classify identified clocks into asynchronous clock domains are as follows.

- *create_clock*
  Specfies the primary clock signals, along with their periods and other attributes. A symbolic name is specified to refer to the clocks in various other SDC constraints. In SDC, more than one clock can be specified on the same port—as is typically the case when a block can be used in multiple modes.
- *create_generated_clock*
  Specifies derived clocks. This constraint provides information about the clock division or multiplication factor, so the generated clock period can be inferred. For CDC analysis, all generated clocks are considered synchronous to their primary clocks. That is, they are considered to be in the same domain as their primary clock.
- *set_false_path*
  Typically specifies clocks that are asynchronous (since timing constraints cannot be checked across asynchronous clocks).
- *set_clock_groups*
  Specifies a set of clock groups that are mutually asynchronous, logically-exclusive or physically-exclusive. CDC identifies asynchronous clocks through this constraint and their clock groups are updated to mark them as asynchronous. Physically-exclusive clocks do not co-exist in the design so these can be ignored for CDC analysis. Logically-exclusive clocks co-exist, but their clocked logic do not interact. So if the design is appropriately constrained for mode constant settings, their clocked logic cannot interact at CDC paths. Their clock domain groupings can be ignored for CDC analysis.
- *set_multicycle_path*
  Specifies a multicycle path for timing analysis, which implies that its transmit and receive clock groups are synchronous to each other, since mulitcycle paths cannot be defined for asynchronous clocks.

Clock domain refinement happens through processing of these directives, and this helps classify the design clocks into appropriate clock domains for CDC analysis.

B. *Port clock relationships*

In the absence of information about the clock domains at primary ports, verification engineers usually take a pessimistic approach. They consider all primary input and output ports as asynchronous to design clocks. But CDC verification results are more accurate—and less time is wasted analyzing false scenarios—if this information is available. It preempts possible clock domain crossing issues when a block is integrated with other components at the chip level.

SDC specifications have port constraint information about the input and output transition delays with respect to driving and driven clocks, respectively. This information gives the CDC tool the needed information about the clock domains associated with the primary input and output data ports. The SDC constraints *set_input_delay* and *set_output_delay* are used to infer this information

C. *Design operating modes and constants*

The SDC format has constraints for constant settings (such as *set_case_analysis, set_logic_zero* and *set_logic_one*. These directives constrain the design for verification in the active functional mode. They are directly imported to the CDC setup and set the correct modal and constant constraints.

D. *Functional false paths*

False paths are paths in the design that can functionally never be sensitized. Timing analysis does not consider false paths for deciding critical paths. The same analogy applies to the CDC context and these paths are not considered for CDC analysis. When a *set_false_path* directive is not specified between clocks, but is specified between data signals, it is a functional false path for CDC analysis as well. These paths can be waived during computation of CDC results. However, the CDC tool does not ignore false paths between clocks. These are false paths because of asynchronous clocks so their directives yield useful clock domain information.

V. CDC CONSTRAINTS EXTRATION ENGINE

Our automated, integrated SDC setup enables the CDC analysis tool to start CDC verification directly, without needing any setup iterations. CDC analysis results are acceptable in the first iteration and the verification engineer can start fixing CDC problems—instead of worrying about getting the setup right. Huge SDC files and complex SDC setups are no longer a problem, as this information is automatically interpreted. Anyway, the engineer can always review the final interpretation.

Our CDC information extraction engine takes the compiled design and its SDC file as input. The engine examines the relevant SDC commands and resolves the constraints by viewing the commands together in context. Relationships between commands are interpreted. SDC format is basically used for static timing analysis, so the extraction engine uses an STA perspective when extracting information. The engine generates a report and a filtered version of the original SDC file containing only information used for the extraction. The engine also keeps track of how clock groups are formed by the algorithm heuristic.

GUI visualization of the SDC file involves understanding the clock specifications, clock groups specifications, constants, port-clock associations, and input/output delays as explained next.

A. Clock tree visualization

Fig-3 shows and example of an SDC clock view generated by the extraction engine from an SDC file. Analyzing the group and determining inconsistency is easy. The extraction is fast and independent of the CDC analysis tool. The figure also shows an example of SDC clock tree visualization. Clocks are colored according to their groups. The clock tree shows that *div2_clk* should be in *Group_pll1_clk*. However it is a different color, indicating it is in a different asynchronous group. The verification engineer can qualify the grouping at this point since this might be an issue in the original SDC file because it creates possible unintended groups for CDC.

SdcClocks

Name	Object	Group	File	Line
pll1_clk (7)	pll1_clk	Group_pll1_clk	...mp/cdc_sdc/test.sdc	1
div1_term_clk	lrcomp.term_lrc.o	Group_pll1_clk	...mp/cdc_sdc/test.sdc	5
div2_clk (1)	iptop.digital.uslogic.clkdiv2.o	Group_div2_clk	...mp/cdc_sdc/test.sdc	6
gen_div2_clk	iptop.digital.genlogic.clkdiv2.o	Group_pll1_clk	...mp/cdc_sdc/test.sdc	18
div16_clk	clk_div0.clk_div16_reg_2_0.o	Group_pll1_clk	...mp/cdc_sdc/test.sdc	7
gen_div16_clk	clk_div0.clk_div16_reg_3_0.o	Group_gen_div16_clk	...mp/cdc_sdc/test.sdc	9
gen_div8_clk	clk_div0.clk_div8_reg_2_0.o	Group_pll1_clk	...mp/cdc_sdc/test.sdc	10
div2_ref_clk	adcclk.buf_gtech.gen_mux2.o i_pll1_refclkdiv....	Group_pll1_clk	...mp/cdc_sdc/test.sdc	12
div1_clk	lrcomp.term_lrc.clkdiv.o	Group_pll1_clk	...mp/cdc_sdc/test.sdc	14
pcie_tx_clk_src	u_PCIE_PHY.tx_clk_src	Group_pcie_tx_clk_src	...mp/cdc_sdc/test.sdc	2
mng_clk	u_core_port_0.hcsr_jtag_tck.Q	Group_mng_clk	...mp/cdc_sdc/test.sdc	3
tclk	top.clk_gen.tck0	Group_mng_clk	...mp/cdc_sdc/test.sdc	16

Fig-3: Clock domain inference snapshot (Visualizer)

B. Clock group inference debug

Viewing the information in an intuitive manner using a graph helps the verification engineer quickly understand the constraints and the CDC verification setup. *Fig-4* shows an interactive graphical view generated by the SDC extraction engine. It shows the formation of groups, step-by-step from the SDC commands.

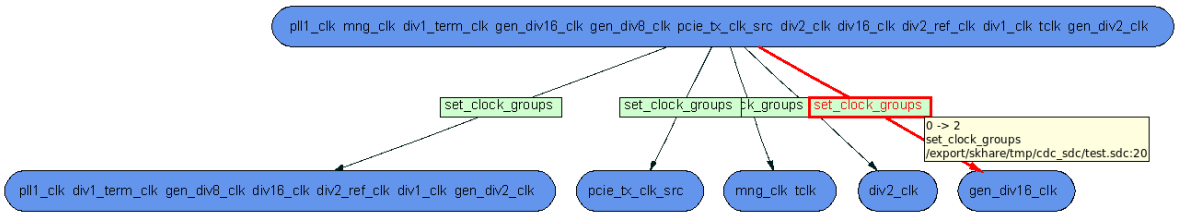

Fig-4: Clock domain inference trace from the GUI visualization utility

C. *Port clock relationship visualization*

Port clock relationships are inferred from SDC for accurate CDC verification. A port can be associated with a single clock group or multiple clock groups. Visualizing the port clock relationships in tabular format gives a view of the inferred clock groups. *Fig-5* shows an example of the port domains as inferred by the SDC extraction engine.

```
============================================================
Port                Direction           Clock Domain
------------------------------------------------------------
LAN_PWR_GOOD          input             { Group_pll1_clk }
PE_CLKp               input             { Group_pll1_clk }
DEVICE_OFF_N          input             { Group_pll1_clk }
PE_CLKn               input             { Group_pll1_clk }
VDDO_2                output            { Group_pll1_clk }
PHY_HSDACP            output            { Group_pll1_clk }
VDDO_1                output            { Group_pll1_clk }
NUM_SI                output            { Group_mng_clk }
VDDO25_3              output            { Group_pll1_clk }
------------------------------------------------------------
```

Fig-5: Port domains visualization

D. *Design modes and constants identification*

Listing all constants inferred from SDC commands helps to set up the CDC tool and also it gives a good idea of the various operating modes of the design. The verification engineer often can identify from this information the correct mode needed to run the CDC analysis tool. *Fig-6* shows a sample snapshot of the constants identified by the SDC extraction engine.

```
============================================================
Constant                                        Value
------------------------------------------------------------
u_core_port_0.u_ior_tap_m_jtag.u2101.Z              0
u_core_port_0.u_ior_tap_m_jtag.tap_tmp_scantri_t.Q  0
------------------------------------------------------------
```

Fig-6: Constants identification

VI. CASE STUDY

To illustrate the benefits of our methodology, we integrated the SDC extraction engine and GUI visualization utility into our CDC verification tool. Next we highlight its benefits through a case study on a customer's ASIC design. The RTL design has 55 asynchronous clock domains. The SDC setup for the SoC consisted of 46 SDC files with a total of over 120,000 lines.

Conventional CDC verification methodology on this design required three iterations. Even after that, there was much "noise" in the results—as all mode constant settings could not be identified in the setup. The CDC tool run time during the first two iterations was bad, as the CDC tool was also analyzing false paths due to incomplete constraints. Results in initial runs were very noisy; the process required debugging effort to filter the noise; and the identified missing constraints had to be applied in next run. Acceptable results were achieved at the end of the third iteration.

We ran the same design again with the proposed SDC-based CDC setup. Automatic information extraction from the SDC files through our CDC setup inference engine grouped 179 specified SDC clocks into 55 asynchronous clock domains. The correct functional modes, false paths and port constraints were also inferred. This strategy created the CDC setup automatically—in less than a minute. We used the SDC inference visualization utility to understand and qualify the setup. The initial CDC analysis tool run returned acceptable results without the need for any iteration. The customer could start fixing CDC errors immediately. *Table-1* shows these results.

Conventional CDC Verification Methodology						
Run	**Async Clock Domains**	**Port Domain Information**	**Stable/False-Path Information**	**Mode constant Settings**	**CDC Errors**	**Review Action**
1	403	Not Available	Not Available	none	54432	1. Group synchronous inferred clocks (Designer inputs required) 2. Specify mode settings 3. Port domain inferred information (tool help)
2	55	Specified	Not Available	10 const signals	55696	1. Missing case analysis settings identified 2. Identify stable signals and false-paths
3	55	Specified	Specified	27 const signals	9296	Review results, waive false paths. Fix CDC Errors

Proposed SDC Setup based CDC Verification Methodology						
Run	**Async Clock Domains**	**Port Domain Information**	**Stable/False-Path Information**	**Mode constant Settings**	**CDC results**	**Review Action**
1	55	Inferred 3882 Port domains	Inferred	Inferred 232 const signals	9245	Qualify SDC Setup. Fix CDC Errors

Table-1: Conventional vs. proposed CDC methodology

VII. CONCLUSION

From the case study, the benefits of our proposed approach are obvious. The SDC-based setup eliminates days spent creating, refining and qualifying the CDC setup. The methodology is generic and non-iterative. Noise in CDC results is eliminated. Since the results are generated automatically, our process is recommended for qualifying the initial, inferred setup. Such qualification is aided by the SDC inference visualization utility. Verification engineers can start fixing CDC results immediately. The proposed method accelerates CDC verification closure time and it results in valuable savings of time, effort and costs.

REFERENCES

[1] Clifford E. Cummings, “Clock Domain Crossing (CDC) Design & Verification Techniques Using System Verilog”, SNUG-2008

[2] Vishnu C Vimjam and Al Joseph, “Challenges in Verification of Clock Domain Crossings”, DAC knowledge center Article

[3] Ping Yeung, “*Five Steps to Quality CDC Verification”*, Mentor Graphics, Advanced Verification White Paper

[4] “Using the Synopsys Design Constraints”, Application Note v1.9, 2010