GLS Shift Over Through Timing Constraint Verification: A Comprehensive Framework

Amey Telang, Associate Director, Samsung Semiconductor, Bangalore, India (amey.telang@samung.com)
Shekhar Sharma, Director, Samsung Semiconductor, Bangalore, India (shekhar.sh@samung.com)
Sunil Shrirangrao Kashide, Director, Samsung Semiconductor, Bangalore, India (sunil.sk@samsung.com)
Garima Srivastava, Director, Samsung Semiconductor, Bangalore, India (s.garima@samsung.com)
Karishma Singh, Sr. Principal Product Engineer, Cadence, Noida, India (ksingh@cadence.com)
Dipankar Sharma, Principal Application Engineer, Cadence, Bangalore, India (dipankar@cadence.com)

Abstract-

Semiconductor chips today form the technological foundation of nearly every industry, from consumer electronics and automotive systems to artificial intelligence and high-performance computing. This widespread demand has driven significant advancements in semiconductor manufacturing and design methodologies, while intensifying the challenges faced by engineers developing highly integrated System-on-Chip (SoC) architectures.

Modern SoCs encompass of multiple sub-designs and blocks made-up of hundreds of complex intellectual properties (IP's), CPUs, GPUs, NPUs, and advanced processing units. This intricate architecture of SoC's requires a complex clock distribution network, with multiple asynchronous domains operating at varying frequencies and voltages. The need to deliver these complex, clock-intensive devices within aggressive schedules further amplifies the need for strategies that effectively verifies timing constraints and achieve precise timing closure before final tape-out to ensure silicon correctness and long-term reliability. Among these, Static Timing Analysis (STA) and Gate-Level Simulation (GLS) represent two of the most crucial checkpoints. Any oversight in these phases can result in complex Engineering Change Orders (ECOs), post-silicon failures, or degradation of Power, Performance, and Area (PPA) margins.

However, STA lacks an intrinsic mechanism to verify the correctness or completeness of the applied timing constraints, relying instead on user-defined inputs that may not always reflect the true design intent. On other end GLS is computationally intensive, often involves multiple iterative debug cycles that increase turnaround time and jeopardize project schedules. To address these inefficiencies, there is a growing demand for enhanced methodologies that improves constraint efficiency, uphold timing quality, and meet aggressive time-to-market goals.

The Timing Constraint Verification (TCV) methodology addresses this need by providing a comprehensive framework that ensures the accuracy, consistency, and completeness of timing constraints across the RTL-to-GDSII design flow. By fostering close collaboration among design, verification, and backend teams, TCV facilitates the analysis, propagation, and integration of high-quality SDCs—leading to improved efficiency, reduced design risk, and overall SoC reliability.

I. INTRODUCTION

Static Timing Analysis (STA) serves as a cornerstone of digital design timing sign-offs, providing a fast and exhaustive mechanism for identifying setup, hold, recovery, and removal of timing **violations without relying on dynamic simulation**. However, the **accuracy and effectiveness of STA** are fundamentally dependent on the correctness and completeness of **Synopsys Design Constraints (SDC)** format.

The **manual validation of these constraint is both impractical and highly error-prone approach**. Small inconsistencies—such as missing clock definitions, incorrect false path declarations, or unintended multi-cycle path specifications—can significantly leads to unnecessary design iterations, moreover these issues may under report critical timing failures, **allowing hidden issues to propagate into silicon.**

Further, **Functional simulation** is typically initiated in the early stages of the design cycle and focuses on validating the logical correctness of the Register Transfer Level (RTL) description. This phase is largely independent of timing analysis, as it assumes ideal conditions without accounting for path delays or cell-level timing effects.

Consequently, **Gate-Level Simulation (GLS)** becomes a critical verification step for assessing design behavior under realistic timing conditions. By incorporating **Standard Delay Format (SDF) and Synthesized Netlists, GLS enables validation of asynchronous interfaces**, setup/hold timing assumptions, and clock-domain interactions under actual delay scenarios. Despite its importance, **GLS presents significant challenges** in verification. GLS **often fails to detect incorrect or incomplete timing constraint definitions**, particularly for complex design intent scenarios such as multi-cycle paths (**MCPs**) and false paths (**FPs**). These oversights remain undetected until late-stage or post-silicon validation, leads to costly debugging cycles, schedule delays, and elevated design risks.

These limitations emphasize the growing need for **formalized Timing Constraint Verification (TCV)** methodologies, which can systematically ensure the correctness, consistency, and completeness of timing constraints throughout the **RTL-to-GDSII flow**—thereby **improving timing closure efficiency and overall SoC reliability**.

II. COMPARATIVE ANALYSIS OF TCV AND EXISTING METHODOLOGIES

Timing Constraint Verification addresses the limitations of existing formal verification methods **such as Fishtail and Dynamic CDC**, which suffer from restricted coverage and long debug cycles, critical for the project schedules.

Fishtail assertion-based approach flags all source–destination paths as single-cycle, demanding extensive manual review of assertion failures to determine whether they can be safely ignored or if any may impact subsequent **Gate-Level Simulation (GLS) runs**—a process that is both time-consuming and error-prone. In contrast, **TCV integrates built-in checkers that can 'inject delay' or 'X' conditions** upon detecting failures, thereby validating whether the RTL can tolerate such scenarios. Additionally, TCV supports for **Hard-Macro- Liberty files** and leverages the **Common Timing Engine (CTE)** from STA tools to interpret SDC constraints— features not available in Fishtail.

In comparison, Dynamic CDC require formal setup and get the source-destination (synchronizer or non-synchronizer) pair details for CDC path. During simulation, it injects random metastability on destination. However, Dynamic CDC **limits the selection of different metastability injection methods,** prohibiting control or modifications to latency. In addition, it **lacks the feature for X-injection and the filtration of quasi static signals.**

Previous **TCV methodologies also have certain constraints** such as:

1. Netlist based constraint verification
2. Verification enablement at both Block/ Sub-System and SoC levels
3. Mapping of RTL to Netlist based SDC constraints
4. Synthesis directive mapping for Netlist versus RTL match
5. Verification with Delay insertion for Multi Cycle Paths to replicate real time SoC behavior
6. Verification with X insertion for False Paths to simulate real time SoC behavior
7. Identification and application of quasi-static signals to reduce noise

III. IMPLEMENTATION

The **Timing Constraint Verification (TCV)** methodology introduces an automated verification flow that analyzes timing constraint and exceptions **defined in the SDC file against the RTL design representation**.

As shown in **"Figure 1: Timing Constraint Verification Flow" instead of relying on post-synthesis netlists**, TCV performs an **early-stage validation using RTL** in conjunction **with netlist-derived SDC** along with **liberty** files for mapping between RTL and Netlist based SDC paths**. The core principle of TCV is to simulate timing relationships and ensure that all timing exceptions**—multi-cycle, false, and clock-group constraints— behave as intended.

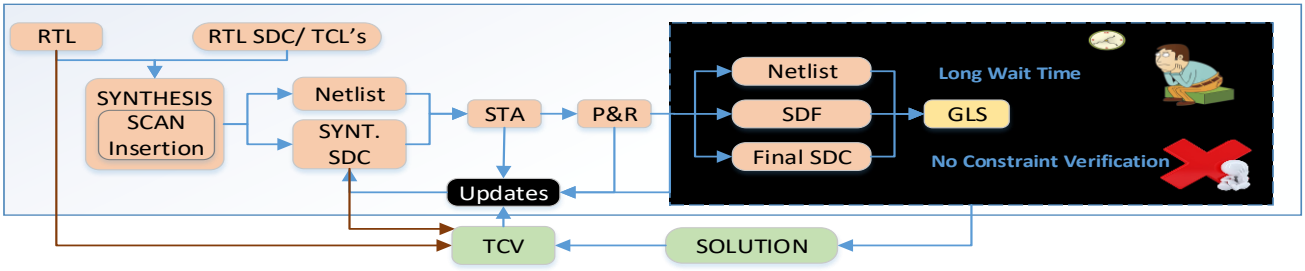

Figure 1: Timing Constraint Verification (TCV) F*low*

(The brown arrow highlights the major update in the TCV flow using Synthesized SDC). The simulator interprets the SDC file, cross-references it with the RTL hierarchy, and dynamically inserts delay or stability checks to replicate real hardware behavior. Violations trigger detailed reports, guiding designers to erroneous constraints before they manifest in silicon. By running the SDC file in simulation, **TCV identifies and reports incorrect or incomplete constraints** before they impact physical design or silicon, ensuring alignment between design intent and tool assumptions.

IV. TCV FLOW

The **TCV flow** integrates SDC-based timing exception analysis in simulation environment. The process includes:

1. **Input Preparation:** The RTL design and corresponding SDC file (typically generated post-synthesis) are loaded into the TCV simulator along with Liberty libraries for RTL and gate-level path mapping.
2. **Constraint Execution:** The simulator interprets and applies SDC directives to the RTL model.
3. **Dynamic Analysis:** During simulation, exception paths are monitored through delay injections or stability checks ensuring each constraint's functional validity aligns with real hardware expectations.
4. **Reporting:** Violations and confirmations are captured in detailed logs and HTML reports.

V. VERIFICATION

To gain confidence in timing exceptions and reduces reliance on manual debugging TCV supports:

- **Multi-Cycle Path Verification:** Confirms destination register update after the specified multi-cycle interval
- **False Path Verification:** Detects cases were destination toggles incorrectly in response to source activities
- **Clock Group Verification:** Ensures asynchronous groups are correctly identified and constrained
- **Set_Case_Analysis Support:** Validates constant propagation to enhance accuracy in analysis
- **Quasi-Static Signal Checks:** Provides a list of quasi-static signals for designer review

1. Verification of Multicycle Paths

Multicycle Paths (MCPs) represent data paths that intentionally span multiple clock cycles for signal propagation. As illustrated in **"Figure 2: Example of Multicycle Path"**, the signal transition from the launch flip-flop (FF1) to the destination flip-flop (FF2) experiences a delay equivalent to **three clock cycles** arise due to architectural design decisions, pipelined implementations, or delay insertions introduced during physical design optimizations.

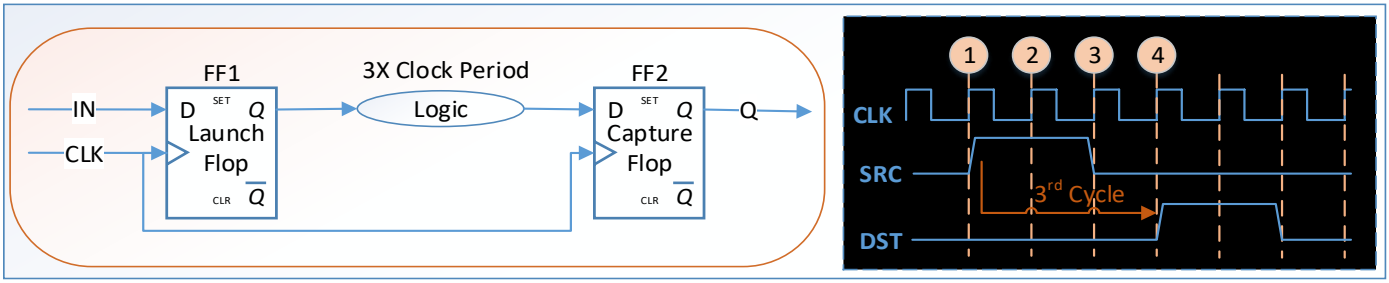

Figure 2: Example of Multicycle Path

To account for this relationship, the constraint applied is: *set_multicycle_path –setup 3 –from FF1* –to FF2**.* However, as shown in **"Figure 3: Multicycle Path Violation & Delay Insertion"** discrepancies may occur during simulation where the destination flip-flop toggles within a single clock cycle caused by incorrect assumptions, misapplied constraints, or unintentional design feedthroughs. The TCV methodology identifies the deviation and add the remaining delay (3 – 1 = 2 cycles) to validate whether the design still meets functional correctness.

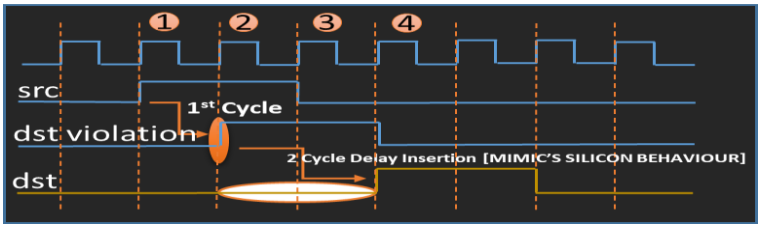

Figure 3: Multicycle Path Violation & Delay Insertion

Successful test completion indicates that the constraint matches design; Conversely any failure indicates a mismatch between the constraint and implementation demanding the review of the corresponding constraint or RTL logic.

2. Verification of False Paths

This method detects incorrect applications of the **"set_false_path"** constraint. Ideally, a false path should prevent any signal transition from the source to the destination. However, in practice, certain design regions may be marked as false paths for unused modes or features, while still allowing toggles in other configurations.

As illustrated in **"Figure 4: Example of False Path"**, if a register is fixed to '1', the source flip-flop (FF1_1) will never toggle and reach the destination (FF2). Conversely, as shown in **"Figure 5: False Path Violation"**, if this condition is not properly defined or if the false path is mistakenly retained for a mode where propagation is valid, the toggle may incorrectly appear at FF2.

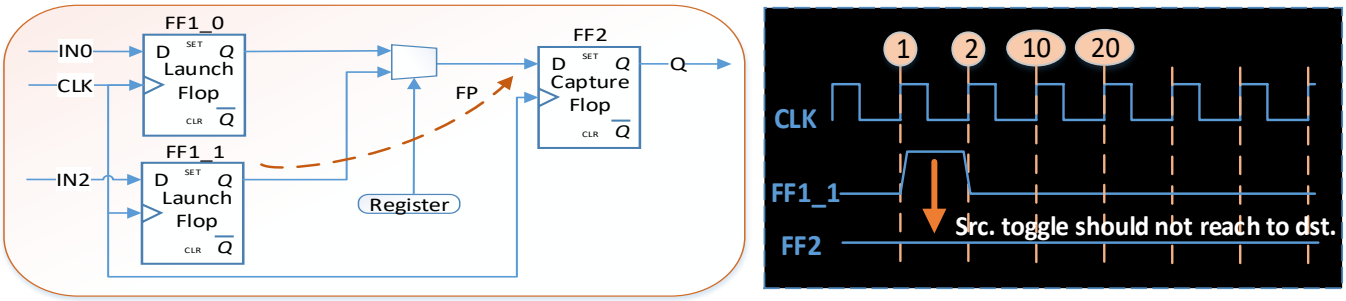

Figure 4: Example of False Path

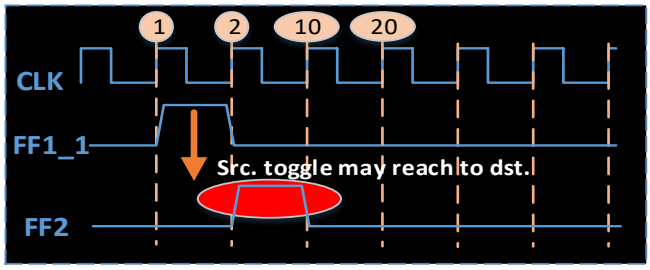

Figure 5: False Path Violation

To detect such cases, TCV utilizes an 'X'-insertion technique **("Figure 6: False Path with X Insertion"),** which introduces an unknown ('X') value along the path with a defined latency (e.g., 10 cycles). The TCV performs a stability check for a configurable duration (e.g., 100 ns). If FF1_1.Q affects FF2.D before a defined window, a violation is flagged, prompting designers to review if the constraint or design intent requires correction or waiver.

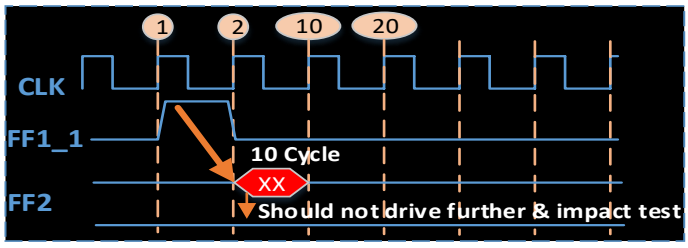

Figure 6: False Path with X Insertion

3. Verification of Clock Group Paths

Clock-group constraints (e.g., set_clock_groups -asynchronous -group {clk1} -group {clk2}) define paths between asynchronous clocks. TCV validates these groupings using the same X-propagation method applied for false-path checks, ensuring synchronous clocks are not mistakenly treated as asynchronous

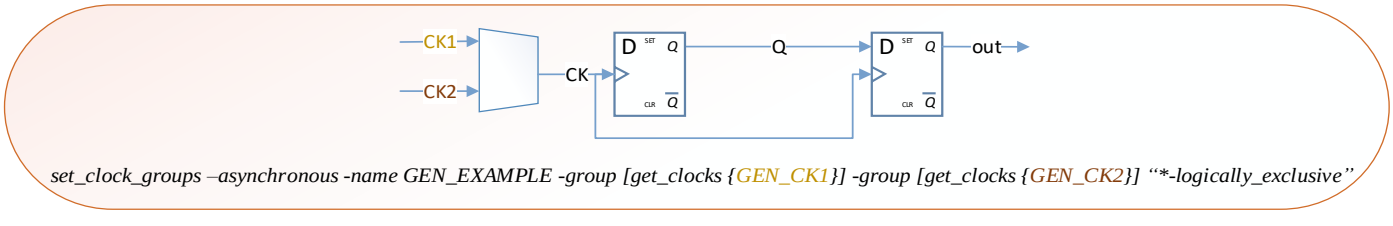

Figure 7: Clock Group Path

4. Support for set_case_analysis

The "set_case_analysis" constraint defines constant or conditional signal values to a list of pins or ports for timing analysis. TCV verifies these assignments by cross-checking defined constants against RTL behavior. **It checks whether the signal remains constant through-out simulation** if "set_case_analysis" applied on it. Additionally, quasi-static signal detection helps to minimize false positives during X-based verification and streamline debugging.

5. Support for Quasi Static Signal Identification

With False Path **rather more on Asynchronous Clock group** various times the nature of the signals analyzed to be a quasi- static. This creates lot of false violation when the verification starts with 'x' insertion on the FLASE PATH.

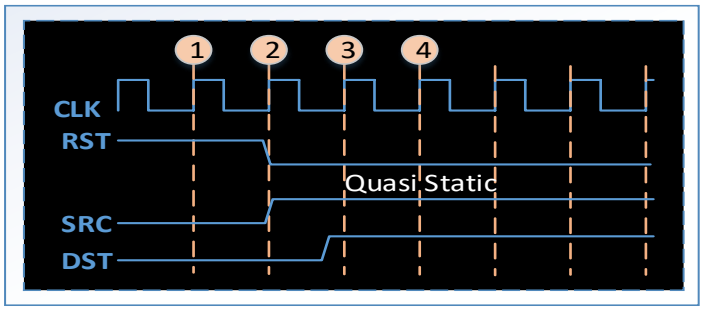

Figure 8: Quasi Static Signal

To simplify debugging, Quasi Static signal filter support has been implemented. If any signal remains static for 20 or more cycles, it'll be reported to the designer. If the signal is confirmed to be static, the Verification team can apply a waiver allowing the tool to proceed.

V. ADDITIONAL SUPPORT

TCV's integration with existing simulation flow introduces several enhancements to handle large-scale SoC environments efficiently:

1. **Mapping between SDC and RTL naming conventions**: This allows seamless hierarchy correlation.
2. **Support for synthesis directives:** Use of **translate_on/off** directives prevent false constraint v/s RTL mismatches.
3. **SYNC filter lists:** These are for RESET and DATA synchronization paths, avoiding unnecessary X-propagations.
4. **Clock filtering:** This feature filters out inactive test clocks such as SCAN or BIST.
5. **Handling of asynchronous clocks generated from the same root clock**: For differentiation between modes.
6. **Schematic generation:** This new feature will aid visual debugging of constraint violations.

These features collectively ensure robustness, scalability, and ease of adoption in SoC verification environments.

VI. VERIFICATION RESULTS

The TCV methodology has been deployed across multiple projects to evaluate its effectiveness in detecting timing-related issues already occurred silicon. The results demonstrate that TCV successfully identified problems that had previously escaped detection, including:

1. Detection of Missing Generated Clock Definitions
2. Identification of Missing MCP Constraint
3. Identification of Clock Error-1 (Wrongly marked asynchronous clocks detected between 2 synchronous clocks)
4. Identification of Clock Error-2 (Wrongly marked synchronous clocks detected between 2 asynchronous clocks)
5. Identification of incorrect exercised clocks
6. Observation of Glitch Over CDC Path
7. Verification of Double Synchronizer Behavior

These findings confirm that TCV can reliably capture critical errors earlier in the design cycle, reducing the risk of silicon escapes.

1. **Detected Missing Generated Clock Definitions**

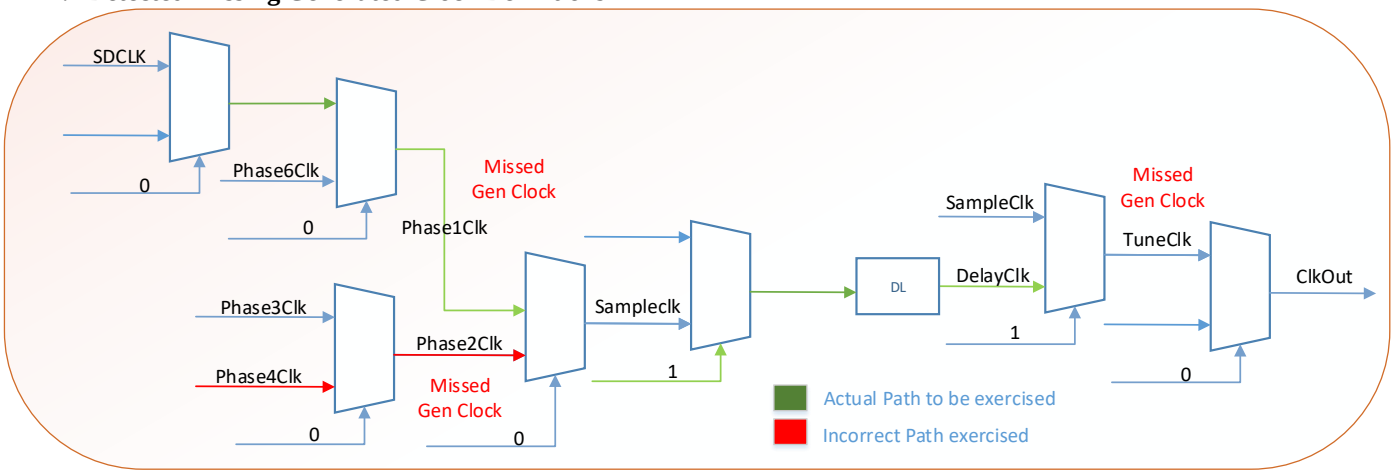

Figure 9: Clock Mux Structure for Missing Clock Definition

The issue was identified during waveform analysis, which revealed that the delay was being calculated with respect to an incorrect clock. As shown in **"Figure 9: Clock Mux Structure for Missing Clock "**, due to the complex clock-multiplexing structure, the clock definition "create_generated_clock" for **Phase1Clk** was missing in the SDC file. Similarly, the definition for **Phase2Clk** was also absent.

However, the MUX-select line remains constant at '0'. Under these conditions, the delay calculation should have referenced **Phase1Clk**. Instead, the use of **Phase2Clk for delay computation** resulted in an incorrect delay representation in the waveform.

2. **Identified Missing MCP Constraint**

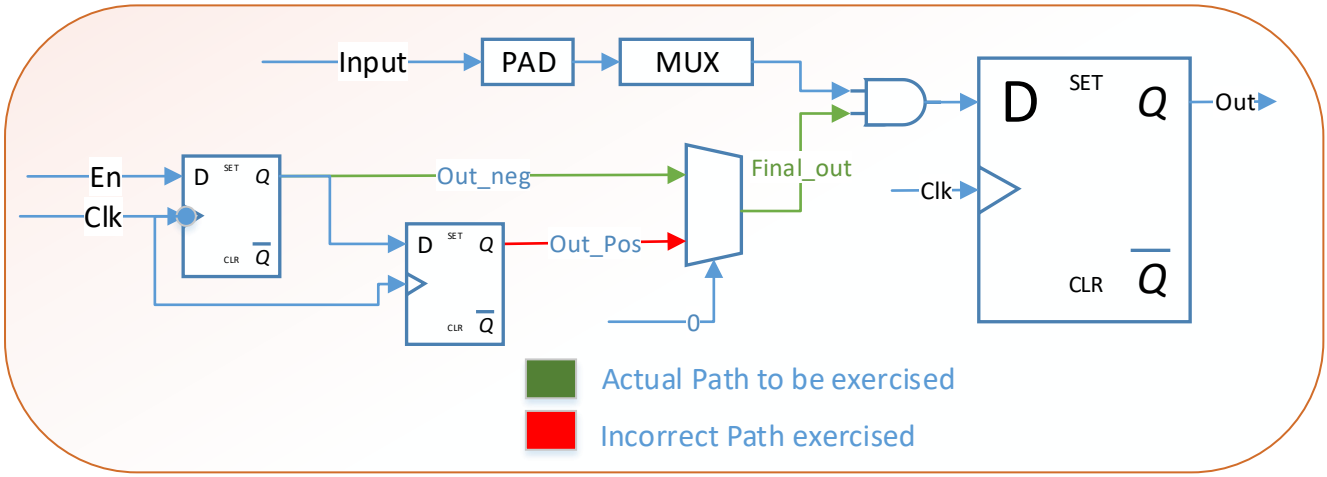

Figure 10: Absence of MCP Constraint

The issue was identified through waveform analysis, which revealed that a delay was being introduced on a path despite the absence of a defined constraint. As illustrated in **"Figure 10: Absence of MCP Constraint"**, the solid green line denotes the active path corresponding to **mux_sel = 0**, wherein the start point is **out_neg** instead of **out_pos**.

However, the MCP constraint is specified only from **out_pos** as the start point. Since **out_pos** toggles concurrently with **out_neg** and both signals converge at the same destination, the delay is erroneously applied with respect to the **out_pos** path rather than the actual active **out_neg** path.

3. **Identified Clock Error -1 (**Asynchronous clocks detected between 2 synchronous clocks**)**

In the simulation, **'X' insertions were observed** on certain paths that were treated as **false paths** due to their **asynchronous** clock definitions. This behavior helped identify the root cause of the improper path operation, which, upon analysis, was traced to a constraint issue.

The clocks associated with these paths **were intended to be synchronous but were incorrectly defined as**

asynchronous. Such misclassification can result in the omission of timing analysis for these paths and may lead to critical functional failures in silicon.

4. Identified Clock Error -2 (Synchronous clocks detected between 2 Asynchronous clocks**)**

In the simulation, **delay insertions** were observed on certain paths identified as MCP paths, applied under the assumption of synchronous clock behavior. Upon detailed analysis, **the issue was traced to a constraint defect wherein both the source and destination clocks were asynchronous in nature but were incorrectly treated as synchronous**, leading to the inappropriate application of the MCP constraint.

5. Identification of incorrect exercised clocks

In the simulation the 'x' insertion was observed on the path, where **the source and the destination clock both are treated as asynchronous** and eventually considered as False Path. Upon further debug the issue identified as a slipup in constraint where both source and destination clocks applied are same.

6. Verification of Glitch Prone CDC Path

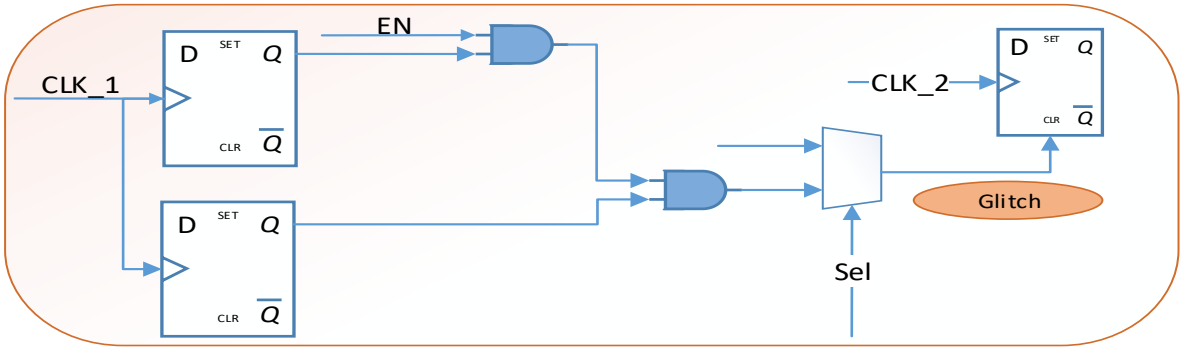

Figure 11: Identification of Glitches

As shown in **"Figure 11: Identification of Glitches"**, the source signal traverses multiple combinational logics across distinct clock domains, making it susceptible to glitches. Such paths are often waived in CDC analysis based on quasi-static or stable-control assumptions.

TCV injects glitches on these paths to verify destination resilience. If the destination fails to recover—as observed here—the path requires correction.

7. Verification of Double Synchronizer Behavior

The major function of **Double Synchronizer in any clock domain crossing design was to avoid the** metastability by reducing the probability of transferring the metastable signal from one clock domain to another clock domain. In this case the first flop may lead to go in metastable state while second flop provides an extra time for the signal to settle into a stable logic level (0 or 1) before the output is captured.

The behaviour was successfully verified in the simulation where in first flop observed to go in metastable state and second flop provides an extra time for the signal to settle down and doesn't pass down the metastable signal to output of the second flop.

VII. TARGETED ACHIEVEMENTS

Adopting TCV in place of traditional GLS provides several measurable benefits:

- **Improved Runtime:** TCV Delivers results significantly faster than GLS, enabling earlier debug and analysis
- **Higher Coverage**: TCV achieves broader verification completeness, identifying issues missed by GLS
- **Reduced Manpower**: TCV minimizes manual debug efforts and improves overall project efficiency

"Figure 12: Targeted Achievements" illustrates runtimes were outstandingly improved by around **55% reduction in days** along with coverage margin increase by around **50% increase in number of test** and debug efficiency

improvements, showing that while GLS often requires **1–2 days**, TCV achieves comparable debug within **0.3–0.5 hours**, representing an **80% reduction in debug time**.

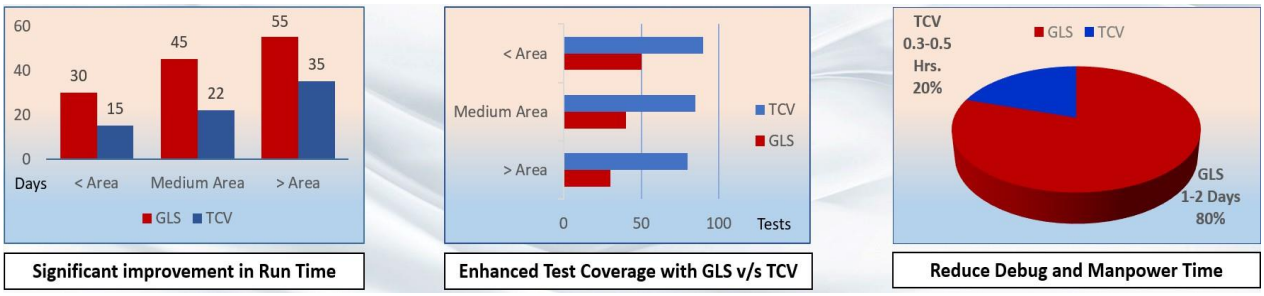

Figure 12: Targeted Achievements

- **SOC Size: < *Area:*** 28mm^2, ***Medium Area:*** 36mm^2, **> *Area:*** 46mm^2
- Test Runs **at both Block level and SoC level**
- **Tools Used for the Approach:** Cadence Xcelium (Simulator), Indago (Debugger) and Jasper (Verification Platform)
- **Constraints Count:**
  - **Multi Cycle:** Approx. 20 per block
  - **False Path:** Approx. 15 per block
  - **Async Clock Path:** Approx. 8 per block

VIII. CONCLUSION

By integrating **SDC verification at the RTL stage**, **Timing Constraint Verification (TCV)** methodology offers a structured, simulation-driven approach for validating timing exceptions in SoC designs. This approach ensures high-quality constraint delivery, reduces time-to-market, and minimizes post-silicon risk. Key benefits observed include:

- Early detection of constraint-related issues
- Enhanced coverage of STA verification
- Improved consistency between block-level and top-level constraints
- Reduction in GLS runtime and debug efforts
- Increased design reliability and silicon confidence

As the semiconductor continues to push for aggressive timelines and higher integration densities, methodologies like TCV will become essential for achieving first-time-right silicon and maintaining competitive advantage.

IX. FUTURE SCOPE

The timing constraint verification (TCV) process has significant potential for automation and improvement. Future developments in TCV will focus on several key areas:

- **Advanced Analysis Techniques:** Utilizing advance AI-based algorithms that extract and analyze simulation and data for log files, enabling more accurate and efficient TCV.
- **Automation of Timing Constraint:** The goal is to automate the analysis of failed constraints and implement effective corrections based on the type of failure, while notifying the RTL designer of any updates.
- **Integration with Industry-Standard Tools:** The TCV process can also integrates seamlessly with existing standard industrial tools like Synthesis or CDC (Clock Domain Crossing). This will enable early analysis and improve design productivity.

X. ACKNOWLEDGMENT

The authors would like to acknowledge the continuous guidance and technical inputs from Mr. Shekhar Sharma (Director, SoC Design) and Mrs. Garima Shrivastava (Director, SoC Verification). Their leadership and insight were instrumental in refining the methodology and aligning it with production workflows. Appreciation is also extended to all engineers who contributed to the implementation and validation of the proposed verification framework.

XI. ABBREVIATIONS AND ACRONYMS

TCV	Timing Constraint Verification
GLS	Gate Level Simulation
FP	False Path
MCP	Multi Cycle Path
STA	Static Timing Analysis
SDF	Standard Delay Format
SDC	Synopsys Design Constraints

XII. REFERENCES

[1] A. Khandelwal, A. Gaur and D. Mahajan, "Gate level simulations: verification flow and challenges," EDN
[2] A Faster and Efficient Timing Constraint Verification Methodology for GFX SOCs, DVCON-2023
[3] Validation of Timing Constraints on RTL, DVCON-2016
[4] static-timing-analysis-challenges
[5] Multicycle-aware At-speed Test Methodology | IEEE Conference Publication | IEEE Xplore
[6] An efficient algorithm to verify generalized false paths | IEEE Conference Publication | IEEE Xplore