Shift-Left on Timing Constraints Verification: Beyond Typical Front-End Execution

Vineeth B, Deepmala Sachan, Ritesh Jain
Intel Technology India Pvt. Ltd.
Bengaluru, Karnataka, India
{vineeth.b|deepmala.sachan|ritesh1.jain}@intel.com

***Abstract*-Timing constraint verification plays a crucial role in the development of modern complex SOCs as it ensures that the timing constraints used for synthesis and timing closure are proper and accurate so that the design meets the desired performance requirements. The conventional methodology used to verify the timing constraints is a gate-level simulation (GLS). Simulations such as these require long run times, offers less coverage, and occurs too late in the SOC development cycle. A powerful alternative to GLS is the formal verification of timing constraints, which are faster and more efficient. The major drawback to this approach is that it may not be entirely possible to formally verify all the timing exceptions in the design and in such scenarios, all the formal failures must be verified using SV assertions in functional simulation. However, the sheer number of assertions generated corresponding to the formal failures can make it difficult to verify them completely in simulation. This paper presents a methodology to improve the formal verification and consequently reduce the assertions generated to converge on real timing issues as fast as possible, thereby achieving significant shift-left in the overall timing closure of the design.**

Keywords—RTL design; Timing constraints; Timing exceptions; Design constraints; Assertions; Simulation.

I. Introduction And Background

Modern System-on-Chip (SOC) designs are complex in nature which leverage the use of Intellectual Property (IP) to expedite the development process. Instead of designing every component from scratch, designers can integrate pre-existing IP blocks into their SOC design. In such designs, one of the primary goals is achieving timing closure which ensures that the design meets the specified timing requirements. The IP modules come from multiple internal as well as external vendors, along with their respective timing collateral. However, there is no guarantee that these timing constraints would hold true when the IPs are fully integrated into the final SOC as per the design requirements. Inaccurate timing constraints may result in severe issues such as data corruption, race conditions, and functional failures. Therefore, timing constraint verification is very crucial for design stability and to ensure that the design meets the desired performance requirements.

Static Timing Analysis (STA) is employed to verify that the timing requirements for all signal paths in the design are met. The presence of multiple clock domains and asynchronous interfaces increases the complexity of these timing requirements beyond a single clock period. In STA, such paths are relaxed through the application of appropriate timing exceptions. However, it is imperative to ensure that this relaxation does not compromise the logic functionality as specified in the register-transfer-level (RTL) design [1]. The conventional approach for validating the timing behavior is through gate-level simulation (GLS), where the gate-level netlist is simulated with the relevant timing information against the RTL test bench.

The GLS occurs too late in the design development cycle. It is usually conducted during the Back-End (BE) phase of design, following the completion of the Front-End (FE) RTL design. Such simulations also require long run times, often offers less coverage, and requires lot of effort to debug the failures [2]. This makes it very difficult to close all the timing violations within the scheduled project timeline and can lead to potential silicon bug escapes and costly re-spin of the silicon. This paper presents how formal verification along with SV assertion verification in simulation can be used as a faster and more efficient alternate to GLS for verifying timing constraints. The proposed methodology can be employed at FE during RTL development phase. This helps to uncover and resolve all the potential timing issues by the final RTL milestone itself, thereby providing a significant shift-left in the overall timing closure of the design. The methodology for formal verification of timing constraints is outlined in Section II, followed by the methodology for timing exception assertion verification in Section III. Section IV presents the results, upon which the paper is summarized and concluded in Section V.

II. Timing Constraint Verification Methodology

The timing constraint verification methodology is depicted in Figure 1. The primary inputs are RTL file list, Hard-IP (HIP) collateral, timing constraints, and the block level configuration file. The HIP collateral has liberty

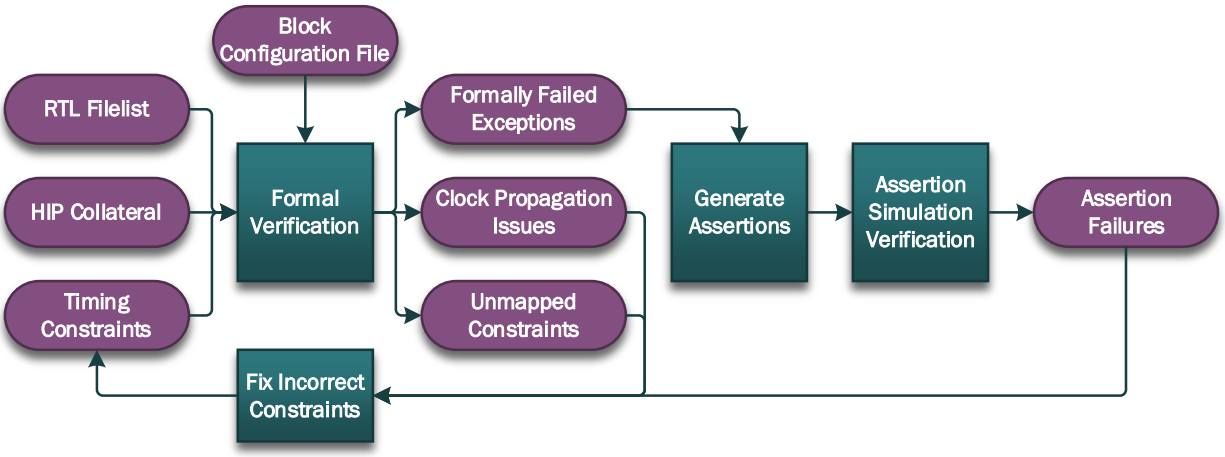

Figure 1: Timing Constraint Verification Methodology

files corresponding to each HIP module. The timing constraints are provided by the full chip timing (FCT) team in Tool Command Language (TCL) format. The block level configuration file has the flow settings as well as the TCL variables that are required to properly source the timing constraints.

A. Unmapped Constraints and Clock Warnings

During timing constraint verification, the constraints are first mapped to RTL and all the unmapped constraints are reported. These can be either due to syntax issues or hierarchy mismatches between RTL and netlist. All the syntax issues must be fixed, and the hierarchy differences should be resolved by providing a mapping file which maps corresponding hierarchies in RTL with the netlist. After the constraint mapping is done, several clock linting checks are performed to identify issues with clock propagation in the design. All the clock warnings should be reviewed, fixing critical issues in constraints, and adding appropriate waivers for the rest of the violations.

B. Formal Verification

Once all the unmapped constraints and clock propagation issues are resolved, formal verification of timing exception is done, and assertions are generated for all the exceptions that fails the formal verification. There are numerous situations where formal verification is not necessary for timing exceptions. Such scenarios are tabulated in Table 1. These exceptions contribute to noise and must be excluded from the formal verification process.

There are many scenarios where the flow may require additional inputs to formally verify a timing path. In the absence of such inputs the number of formally failing exceptions as well as the corresponding assertions generated would be large and consequently requires huge effort to validate them in simulation. The different methods to improve formal verification of timing exceptions are discussed below:

1) Specifying the list of synchronizer cells in the design: The formal verification passes for all timing paths having synchronizer cell as the end point.
2) Constraining the input ports and asynchronous reset ports impacting formal verification.
    a. Input ports must be constrained to its legal values.
    b. Asynchronous reset ports must be constrained to its non-reset value.

   The formal verification passes for all timing paths having static start point.

Table 1: Noise Elimination from Timing Exception Formal Verification

#	Scenario	Description
1	NO PATH	No valid start points/end points were found or no sensitizable combinational path found between them
2	SKIPPED (I/O Port)	Exceptions that can only be verified at higher block level where the logic that drives input ports can be inferred
3	SKIPPED (Async Clocks)	Exceptions having asynchronous launch/capture clocks
4	SKIPPED (MCP Hold<Setup)	MCPs having hold value less than setup value of the corresponding MCPs
5	SKIPPED (Duplicate)	Duplicate exceptions
6	WAIVED	Exceptions corresponding to timing don't cares that are not supported by design logic and hence cannot be formally proven

3) Importing Clock Domain Crossing (CDC) constraints: All the static signal constraints used in CDC analysis should be imported for timing constraint formal verification flow.
4) Constrain static signals based on uncovered assertions: The static start points corresponding to the uncovered assertions in simulation must be reviewed and should be constrained as such in the final sign-off formal verification run.

C. *Execution Challenges*

The different challenges faced during the execution of proposed methodology are described below:

1) Collateral Management: Due to the scale and complexity of SOC designs, timing constraint verification must be done at the partition levels. This requires the respective block configuration files as well as waiver files to be setup at each partition level. The partition level HIP collateral has liberty files corresponding to multiple corners and scenarios for each HIP module and hence, if directly used as input to the constraint verification flow, it can lead to crashes due to memory issues. To avoid such issues, the HIP collateral must be uniquified by maintaining only a single liberty file for each HIP module. All these tasks involve a lot of manual effort and hence they must be automated.
2) Cross Domain Collaboration: The successful execution of timing constraint verification requires the close collaboration between RTL, VAL, DFX and FCT teams.
3) Separation of Functional and DFX Assertions: The functional as well as DFX assertions must be verified separately in functional and DFX simulation regressions respectively and hence the same must be separated properly from the total set of generated assertions.

III. Assertion Verification Methodology

Once the formal verification of timing exceptions is completed, assertions are generated for all the exceptions that has failed the formal verification. These assertions capture the functional behavior of the design that must be satisfied for the exceptions they are associated. The assertions are then verified in functional simulation by running the entire verification regression suite of the design. The false path (FP) assertions check that the condition required to propagate a transition from start point to end point can never be true. The multi-cycle path (MCP) assertions check that when the start point transitions then, either in that cycle the condition required to propagate the change from the start point to the end point should not be true, or in the next cycle the end point should not transition. Thus, any assertion failure in functional simulation regression would represent real-world situations where the specified exception behavior does not hold. In this way, all the incorrect timing exceptions can be identified from the original timing constraints.

A. *Using SVAs in RTL Simulation*

The standard cells used in the design have distinct implementations in synthesis and simulation models. When assertions are generated from the synthesis model and directly used in simulation, it can lead to cross-module reference resolution errors (XMREs). This issue can be mitigated by limiting access to the internal signals of standard cells and referencing signals solely at the boundaries of standard cell wrappers during assertion generation. However, it's worth noting that signals within standard cell wrappers may still be accessed if there are sequential elements present between the signal and its input ports. In such scenarios, it becomes necessary to provide an additional mapping file that maps the synthesis hierarchy to the corresponding simulation hierarchy. This mapping ensures that assertions are generated with the appropriate simulation hierarchies and can be seamlessly incorporated into the simulation for verification.

B. *Sign-off Methodology for Uncovered Assertions*

The Figure 3 illustrates the sign-off criteria for uncovered assertions. The assertions that remain as uncovered during functional simulation regression can typically be attributed to one of the following two scenarios:

1) The exception start point has not toggled.
   In these instances, it is imperative to review all the untriggered exception start points and determine whether they are indeed static signals or not. If they are static, they should be explicitly constrained as such for formal verification, enabling the formal verification to successfully verify all the timing paths associated with these static start points so that assertions are not generated for these paths in the first place. However, if these start points are not static, appropriate test cases should be executed to ensure coverage of these start points and the verification of associated assertions.
2) The launch clock fails to reach the start point.
   In such scenarios, it is essential to conduct a thorough review of all uncovered assertions to confirm whether the launch clocks are not intended to reach the corresponding start points. Should it be determined that the launch clocks should indeed reach these start points, additional test cases must be executed to cover these assertions.

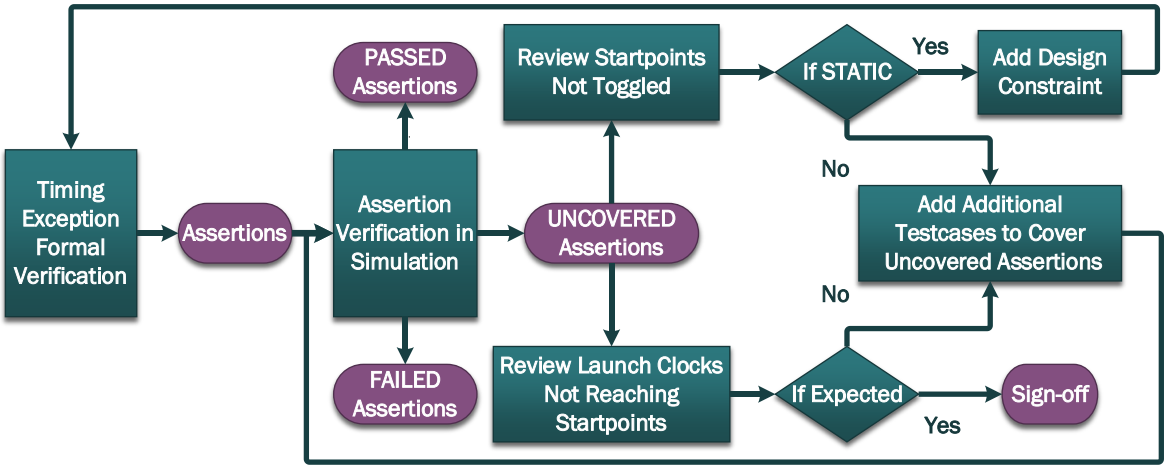

Figure 2: Uncovered Assertions Sign-off Methodology

IV. Results And Discussion

The SOC design being evaluated had nearly 2M timing constraints including more than 600 master clocks, 1100 generated clocks and 20K MCP specifications. The timing constraint verification was done at the partition level as per the proposed methodology. Prior to the formal verification of timing exceptions, all the unmapped constraints and clock propagation issues were resolved. The critical issues were fixed in the timing constraints, while appropriate waivers were applied to the remaining violations, thereby achieving 100% timing constraints mapping across all the SOC partitions. The challenges faced during execution of proposed methodology were addressed by developing and implementing appropriate TCL based automation scripts. The automation results are tabulated in Table 2.

A. Formal Verification Results

The formal verification improvement techniques discussed in section II were implemented at appropriate execution milestones, leading to significant reduction in formal failures and the corresponding assertions generated. The formal verification improvement results are tabulated in Table 3. The flat run had 508999 assertions generated corresponding to the formally failed timing exceptions. Once all the formal verification improvement methods were applied, the generated assertions were reduced to 130491, thereby achieving ~75% reduction in the total assertions generated.

Among all the MCP exceptions that underwent verification, 66.17% of MCPs successfully passed formal verification, while 27.02% failed the formal verification process. The remaining MCPs were categorized as either skipped or waived. The formal verification results of MCP exceptions are tabulated in Table 4. For the MCPs that failed the formal verification, they were further classified into functional and DFX-related MCPs after a comprehensive review with the DFX team, and the corresponding assertions were also segregated accordingly for their verification in simulation.

B. Assertion Verification Results

Among the generated assertions, only 4588 corresponded to the functional (non-DFX) MCPs. These assertions were integrated into functional simulation regressions where 154 assertion failures were observed. All the assertion failures in simulation were correlated back to respective MCPs. The assertion verification results are tabulated in Table 5. The simulation failures were debugged after generating FSBD for all the failing testcases and the incorrect MCPs were fixed in the timing constraints. The fixed MCPs were incrementally verified until no more simulation failures were observed. All the assertions that remained uncovered in simulation were signed off as per the methodology discussed in the previous section.

Table 2: Automation Results

#	Execution Tasks	ETA (w/o Automation)	ETA (with Automation)
1	Block Configuration File Generation	1-2 days	15 sec
2	Uniquify HIP Collateral	1-2 hours	2 sec
3	Clock Warning Waivers Generation	2-3 days	2 min
4	Unmapped Constraint Waivers Generation	3-4 days	3 min
5	Separation of Functional and DFX Assertions	3-4 days	7 min

Table 3: SOC Formal Verification Improvement Results

#	Formal Verification	No. of Assertions	Assertion Reduction %
1	Flat Run	508999	0.00%
2	Synchronizer Cell Specification	506554	0.48%
3	Constraining Input Ports	455080	10.59%
4	Constraining Asynchronous Resets	269077	47.13%
5	Importing CDC Constraints	135435	73.39%
6	Static Signal Feedback based on Uncovered Assertions	130491	74.36%

Table 4: SOC MCP Exception Formal Verification Results

% of PASSING MCPs	% of FAILING MCPs	% of NO PATH MCPs	% of SKIPPED MCPs	% of WAIVED MCPs
66.17%	12.40%	18.18%	2.85%	0.40%

Table 5: SOC Functional (non-DFX) MCP Exception Assertion Verification Results

No. of Assertions Generated corresponding to Functional (non-DFX) MCPs	No. of PASSED Assertions	No. of FAILED Assertions	No. of UNCOVERED Assertions	No of MCPs corresponding to Assertion Failures
4588	1385	154	3049	7

V. Conclusion And Future Work

The suggested approach discussed in this paper offers a swifter and more effective method for verifying timing constraints, which can be deployed at the early stages of design development. This methodology facilitates the identification and resolution of unmapped constraints as well as clock propagation issues prior to formal verification of timing exceptions. The methodology also achieved significant noise reduction by identifying and excluding all the timing exceptions that does not require formal verification. The use of formal verification of timing exceptions along with assertion verification in functional simulation has successfully pinpointed all the incorrect timing exceptions prior to design timing closure, thus averting potential silicon bug escapes and the need for costly re-spins. This results in a significant shift-left in the overall timing closure process. Consequently, the proposed methodology enhances the precision and reliability of timing constraints, contributing to the development of a higher-quality design. This adaptability ultimately improves timing constraint verification efficiency and reduces time-to-market for a broad spectrum of semiconductor products.

As of now, assertion verification is performed for the functional (non-DFX) MCPs. Future efforts will focus on expanding the assertion verification to include DFX-related MCPs as well. The future plans also include extending the entire methodology for FP verification as well.

References

[1] P. Limmer, D. Moeller, M. Mueller and C. Roettgermann, "Validation of Timing Constraints on RTL: Reducing Risk and Effort on Gate-Level," in DVCON Europe, 2016.

[2] A. Khandelwal, A. Gaur and D. Mahajan, "Gate level simulations: verification flow and challenges," EDN, 5 March 2014. [Online]. Available: https://www.edn.com/gate-level-simulations-verification-flow-and-challenges/.