White Paper

SYNOPSYS®
Predictable Success

Boosting Designer Productivity by Using Look-ahead Constraint Analysis Technology

August 2010

Cho Moon,
R&D Manager,
Synopsys

George Mekhtarian
Product Marketing Manager,
Synopsys

Why Constraint Analysis?

Timing constraints are a crucial specification in the modern integrated circuit (IC) design flow. Implementation and signoff tools rely on them at almost every step of the design process. The rapid increase in design size and complexity, as well as the widespread reuse of intellectual property (IP) design blocks, has led to a major increase in both the extent and the complexity of timing constraints specification. Ensuring high-quality timing constraints is paramount to efficient design implementation, especially during handoffs between teams. Incomplete, inconsistent, or conflicting constraints can cause optimization and implementation tools to run ineffectively or to never converge.

In this paper, we present a unique constraint analysis technology that checks for timing constraints problems and provides an interactive environment with context-sensitive diagnostic and fixing suggestions. Using this technology, design teams can save several weeks of engineering effort in a typical IC design project.

Design Teams and the Challenges with Timing Constraints

Today, it is not uncommon to see timing constraints files that are several hundred thousand of lines long. Over the last few years, the combination of many factors has led to the explosion of the extent and the complexity of timing constraints.

The arrival of the system-on-chip (SoC) has pushed design sizes up to the hundreds of millions of instances. Higher integration has increased the number of clocks and the interactions between them. Additional design functionality and the advent of low power design techniques have led to a significant increase in both the number of voltage domains and design scenarios that need to be analyzed and the timing constraints specifications that go along with them.

Another layer of complexity that design teams often grapple with is frequent design reuse, especially the use of third-party intellectual IP. Here, design teams face the challenge of not being the original designers of these particular design blocks and their timing constraints. Debugging problems with your own constraint specifications is difficult but manageable. Diagnosing and debugging third-party designs and their timing constraints can be challenging and a drain on precious design time.

With the globalization of IC design teams, verification of timing constraints becomes paramount during handoffs between groups, especially when they are geographically dispersed. Iterations due to bad constraint handoff between front-end and back-end design teams, for example, can be costly, especially when the design schedules are tight.

Some common issues that design teams face with timing constraints are:

- Missing constraints
- Incorrect constraints
- Conflicting constraints
- Over-constraining conditions
- Redundant constraints
- Inefficiently written constraints (causing long runtimes in tools)

Case Study: Debugging Timing Constraint Problems in Real-time

A design group was implementing a 10 million instance 40nm SoC design that had over 80 clock definitions. During the timing constraint generation phase, the timing constraints were created for the whole clock network of the design, but the timing constraint definition of one generated clock with respect to the master clock was missed. Figure 1 shows the design segment where the issue occurred. A clock was defined on the input port CLK and a divide-by-2 generated clock was defined on the FF/Q pin with its master clock being CLK.

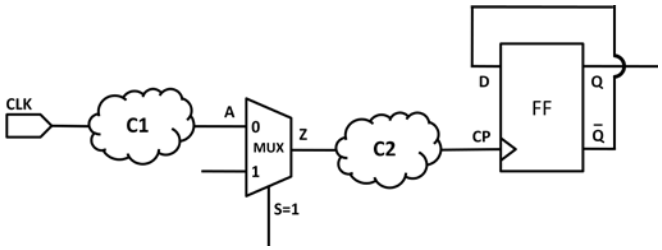

Figure 1: Generated Clock on FF/Q Has No Source Latency Path from Its Master Clock CLK

The blocks C1 and C2 represent combinational logic clusters comprising thousands of gates along the propagation path of the clock signal CLK. Due to other logic, a constant value of 1 was being propagated to the select pin of the MUX. This blocked propagation of the signal from the MUX/A pin to the MUX/Z pin. Therefore, the master clock CLK was not reaching the clock pin of the FF, leaving the generated clock's source unclocked.

Manually identifying this timing constraints problem was extremely laborious. While many conditions could block the clock signal from propagating in the way the designer expected, the designer needed to find out the root cause of this unexpected behavior before the timing constraints could be completely fixed. In this case, the direct culprit (the MUX cell) resided in a different part of the design, separated by thousands of gates from where the timing constraints problem was detected. Moreover, finding the direct culprit was only the first step in identifying the root cause. In this case, the designer then had to find out why a constant value of "1" was being propagated to the MUX/S pin when a "0" was expected. This then led to the second culprit followed by the third culprit and then many more culprits. Fixing the particular timing constraints problem completely was challenging given that there were thousands of timing constraints applied to the design, and modifying one impacted other constraints. In this case, modifying a user-defined case setting in the fanin cone of the MUX/S pin may have corrected this particular timing constraints problem, but the change in the constant value could have impacted other signals and rerouted clock signals in other parts of the design.

The root cause analysis was a trial-and-error process that required many iterations of the locate-diagnose-fix flow before the timing constraints closure was achieved. The result was that over half a day of debugging time was spent on this one issue before any other progress could be made on the design. Had there been dozens of such issues in the design, the impact on the design schedule would have been catastrophic!

Addressing Timing Constraints Issues

Over time, design companies have developed some standard ways to deal with timing constraints issues. One way, illustrated in the previous example, is to ignore issues until they impact design implementation or signoff runs and then try to fix them case-by-case. This approach can be costly – more iterations and longer tool runtimes or worse, bad silicon due to missed timing violations.

Another approach is to use a commercial tool that performs constraints analysis and debug. While this approach has its associated investment, the cost of tool itself and the associated learning curve, integration into the design flow, and validation of results, the investment can pay major dividends if an effective commercial constraints analysis tool is selected.

Key Requirements of an Effective Constraint Analysis Tool

For a commercial constraint analysis and debug tool to be effective, it has to satisfy five key criteria:

1. Consistent constraint interpretation
2. Ease of setup and use
3. Fast runtime
4. Interactive, intuitive debug
5. Constraint comparison for the purpose of successful IP integration

As timing constraints can get very complex as we previously discussed, a constraint analysis tool must interpret exactly the same semantics of constraints as in the target implementation tools and, especially, signoff tools. Any discrepancy in the effects or precedence of constraints will lead to additional iterations, which can defeat the very purpose of using the tool in the first place. For example, when several timing exceptions like false paths and multi-cycle paths overlap, the constraint analysis tool must be able to apply the same exception precedence rules as the timing signoff tools to determine which constraints affect or do not affect the final timing results.

The constraint analysis tool has to be easy to introduce into your existing design flow. Extra training or tool setup should be minimal. The tool should be able to use existing constraint files with minimal changes. Ease-of-use is also important. A look and feel that is similar to existing design implementation and signoff tools minimizes tool ramp-up time.

Any commercial constraint analysis tool has to deliver results fast. Since this is an additional step that you are adding to your design flow, you cannot afford hours and days of extra runtime. The tool has to run fast on complete multi-million instance designs and, in particular, has to run fast during interactive debugging when multiple runs may be necessary.

Speaking of interactive debugging, the checking and diagnosis must be extremely efficient given the intense human involvement in the iterative trial-and-error cycles of debugging. Any unnecessary delay during this process will lead to increased engineering cost and delayed schedules.

The tool must allow designers to find the root causes of timing constraints problems and must aid in fixing them without introducing new problems**.** The debugging features of the tool should help you drastically reduce the search space and make more judicious fix attempts, thereby reducing the number of iterations required. The turnaround time of iterations should be very short, and changes should be reflected quickly in order to keep the debug process interactive.

Lastly, since incorporating third-party IP blocks or reusing blocks from other designs is becoming common practice in today's IC development, the ability to compare block-level versus top-level constraints is crucial. Each of these predesigned blocks usually comes with a set of constraints that you need to use to build the block. To guarantee that the block functions as expected, you must ensure that these block constraints are still valid once the block becomes part of your bigger design. Thus, the constraint analysis tool must provide block-level versus top-level constraints analysis functionality.

Introducing Galaxy™ Constraint Analyzer

Synopsys Galaxy Constraint Analyzer is a timing constraint analysis and debugging tool that was engineered and built from the ground up to satisfy the above requirements. It delivers look-ahead constraint analysis technology tuned for the Synopsys Galaxy Design Implementation Platform to enable designers to quickly assess the correctness and consistency of timing constraints. In the remainder of this paper, we will take a closer look at how Galaxy Constraint Analyzer works and highlight the following important features of the tool:

- Ease of setup and use
- Look-ahead technology
- Fast runtime, interactive user experience
- Flexible rule checking
- Powerful debug commands
- Block-level versus top-level constraint analysis

Galaxy Constraint Analyzer Usage Flow

Galaxy Constraint Analyzer can be quickly integrated into a typical design implementation flow. The tool requires a Verilog netlist, cell libraries and the design timing constraints in Tcl or SDC format as input. If you are using any of the Galaxy Design Implementation tools, the tool is extremely easy to setup and run. Using any existing PrimeTime, Design Compiler or IC Compiler run script, you can simply add a new command to analyze the design for timing constraints and run the tool. Galaxy Constraint Analyzer ignores commands in the run script intended for other tools such as reading physical constraints or reporting on timing, noise or power. Galaxy Constraint Analyzer shares a common core user interface with the Galaxy Implementation Platform tool suite, giving the new user a common, familiar look-and-feel to the tool environment.

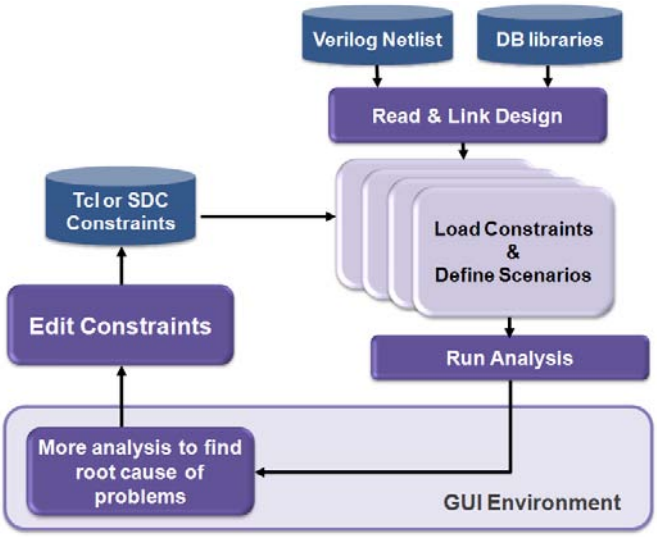

Figure 2: Galaxy Constraint Analyzer usage flow

Look-ahead Constraint Analysis

Galaxy Constraint Analyzer uses technology based on the Synopsys golden PrimeTime timing engine to propagate timing constraint information through the gate-level netlist. The tool checks for structural compliance of the timing constraints with various versions of downstream implementation and signoff tools that will

consume the timing constraints. Ensuring the correct interpretation of timing constraints during analysis gives designers a signoff-correlated view of the timing constraints early in the design process, reducing design iterations.

Fast Runtimes, Interactive User Experience

Galaxy Constraint Analyzer is designed to deliver very high performance – it can analyze multi-million instance designs in a matter of minutes. The tool uses a compact netlist representation that supports hierarchical design methodologies. Most changes during interactive debugging are performed on the timing constraints. Therefore, the Galaxy Constraint Analyzer netlist representation is tuned for fast analysis.

The compact memory footprint in Galaxy Constraint Analyzer improves the tool's capacity to handle hundred-million instance designs, it also improves the tool's runtime by reducing paging activities on the compute resource. Thus, Galaxy Constraint Analyzer can analyze multiple design modes in a single run, further improving efficiency.

Table 1 illustrates the runtime and memory performance of Galaxy Constraint Analyzer on customer designs ranging in size from 1 to 10 million instances with two of the designs having multiple modes. All results were measured on a dual, quad-core Intel Xeon 2.67 GHz machine with 16 GB memory running Red Hat Enterprise Linux version 4.0. Galaxy Constraint Analyzer was run on a single core.

Customer Design	Instances	Scenarios	Unique Clocks	Runtime* (min)	Memory (GB)
Design 1	1.0M	3	217	9.9	1.6
Design 2	1.4M	11	497	16.7	3.4
Design 3	4.4M	1	261	8.7	4.3
Design 4	6.6M	1	152	20.7	6.4
Design 5	7.6M	1	249	16.9	6.9
Design 6	10.0M	1	102	17.2	9.5

Table 1: Galaxy Constraint Analyzer runs in minutes on multi-million gate designs

As we mentioned earlier, Galaxy Constraint Analyzer can analyze modern multi-scenario/mode designs. The tool's unique internal design representation allows it to analyze multiple design modes in a single run with minimal runtime and memory penalty.

Flexible Rule Checking

Galaxy Constraint Analyzer has over 100 built-in constraint checking rules that help designers identify common constraint problems. These rules are divided into categories that span boundary conditions, exceptions, clocks, and other general constraints. Table 2 lists the rule categories in Galaxy Constraint Analyzer, followed by the rule label and type of constraint covered.

Boundary Conditions	CAP_xxxx	capacitance values
	DRV_xxxx	drive constraints
	EXD_xxxx	external delays
Constraints / Exception Analysis	CAS_xxxx	case analysis
	EXC_xxxx	timing exceptions
Clocks	CGR_xxxx	clock groups
	CLK_xxxx	clock properties
	CNL_xxxx	network latencies
	CSL_xxxx	clock source latencies
	CTR_xxxx	clock transitions
	PRF_xxxx	clock count
	UNC_xxxx	clock uncertainties
General	DES_xxxx	design constraints
	HIER_xxxx	hierarchical pins
	LOOP_xxxx	timing loops
	NTL_xxxx	net properties
	UNT_xxxx	library units

Table 2: Galaxy Constraint Analyzer's Built-in Rule Categories

In addition to the built-in rules, the tool also provides the designers with the capability to add user-defined rules. Using the familiar Tcl interface like other Galaxy tools, designers can specify their own rules for their design environment. For example, if a design team wants to enforce a policy of forbidding negative input/output delay values, a CAD team member or a designer can write a user-defined rule to flag any input delay or output delay constraints with negative values as errors.

In Galaxy Constraint Analyzer, the rules can be further configured into rule sets. Whether built-in or user-defined, the designer can group a set of rules together to make them applicable to a specific design stage. For example, when a design is still in the pre-layout stage, clocks should not have any propagated latency since the clock tree is still not in place. One of the rules in a "Pre-Layout" rule set, for example, can look for any clock constraints that have propagated latency and flag them as errors. Similarly, during post-layout, the designer would want to use a different rule that allows propagated clock latency but flags ideal clock latency. This flexibility allows engineers to apply a situation-specific set of rules to reduce false violations and further improve their efficiency.

Unique Debug Features

Galaxy Constraint Analyzer is unique in its extensive support for debugging constraint violations. The tool is architected to allow you to debug reported constraint-rule violations in real time as was previously described. The graphical user interface, shown in Figure 4, provides an intuitive view of the constraint-rule violations by category, including a help page that explains the violation in detail, direct access to the Tcl or SDC constraint source file, and a schematic window to visualize the violation.

Once the constraint-rule violation is identified, the tool provides debugging guidance and constraint fixing recommendations to address the issue. The fast interactive analysis engine then allows you to quickly implement the constraint fix on the design database to ensure closure.

In the situation where some constraint-rule violations are expected, as is often the case during the earlier phases of design when constraints are not refined, Galaxy Constraint Analyzer allows you to easily disable rules or specific violations directly from the GUI environment. This further enhances productivity by filtering out known and expected rule violations and allowing the designer to focus on the real constraint problems.

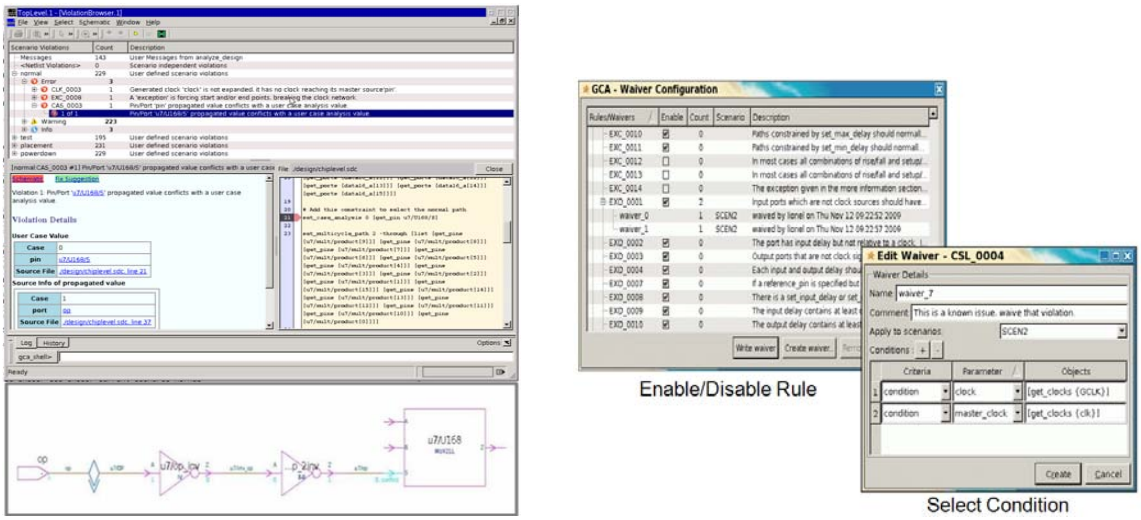

Figure 3: Galaxy Constraint Analyzer Offers a Unique GUI Debug Environment Including a Violation Waiver Feature

Some of the other important features of Galaxy Constraint Analyzer that make it an effective debugger are:

- Clock propagation: The tool traces the propagation of clock sense, which is very important with generated clocks. The tool also displays generated clock source latency and traversal.
- Datapath propagation: The tool identifies constraints that are blocking data propagation in a region of the design.
- Detailed case analysis including constant case value propagation: The tool identifies sources of a case value on a pin and shows forward propagation of constants from a pin.
- Clock network identification: The tool accurately identifies clock networks including a clock interactions report.
- Analysis coverage across modes: The tool finds timing checks not covered in any mode.
- Unclocked register pins: The tool identifies causes for unclocked pins in the design.
- Timing exception reporting: The tool identifies redundant and dominant exceptions in the design and helps create a set of concise, yet complete constraints.

Block-level versus Top-level Constraint Analysis and Debug

Earlier, we explained how design teams often face the challenge of frequent design reuse, in particular the use of third-party IP blocks. The engineers who perform the top-level design integrations are usually not the experts in these particular design blocks or their timing constraints. Therefore, debugging potential issues can be time consuming.

Design teams use various strategies to create block-level constraints from top-level constraints. In a bottom-up design flow, block-level constraints are promoted from the block-level to the top-level. In a top-down flow, block-level constraints are generated from the top-level via budgeting. During the design iteration process, new constraints may be manually inserted into both the top-level and block-level designs. Galaxy Constraint Analyzer allows you to check whether the constraints are consistent after such operations. Each block comes with a set of constraints under which it was synthesized and placed. These constraints include clock definitions, timing exceptions, and boundary conditions. Galaxy Constraint Analyzer loads the constraints for both the block-level instance and the top-level design and performs a comparison between the two sets of constraints. As shown in Figure 5, the tool's block-to-top GUI and powerful debugging features then allow you to zoom-in on any differences. A side-by-side comparison of the top and block constraint files in the GUI help highlight the differences. The tool generates a set of block-to-top rule violations that can be debugged just like the general design rules.

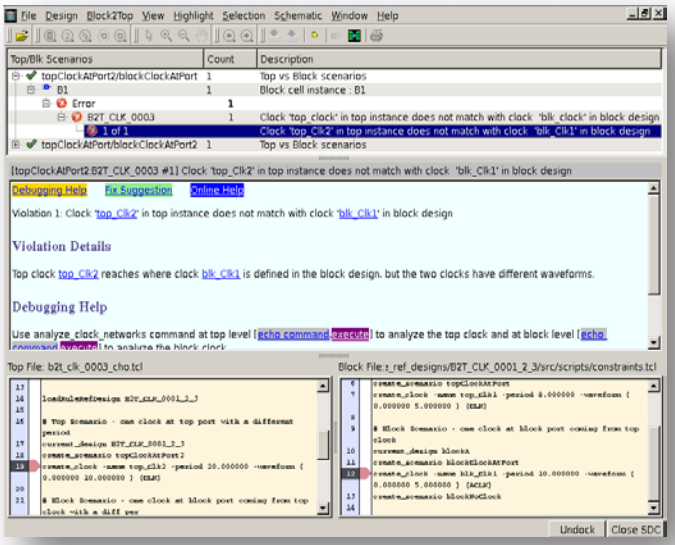

Figure 4: Galaxy Constraint Analyzer Block-level versus Top-level Constraint Analysis GUI

User Experiences

In this section, we share a few real life experiences of design teams who have adopted Galaxy Constraint Analyzer in their design flow and the benefits they realized.

Galaxy Constraint Analyzer Reduces Constraint Scrubbing Time

One set of designers, a digital technology development group, was able to reduce the constraint scrubbing time by more than 50% with Galaxy Constraint Analyzer. They cited the following features of the tool as the key enablers in improving their productivity:

- Fast runtime and small memory usage: They reported 6X faster runtime and 3.8X smaller memory usage compared to their existing commercial constraint analysis tool.
- Best correlation with signoff timing tools: This allowed them to avoid surprises at the implementation and signoff phases.
- Good debug capability: The ability to analyze clock networks, data paths, unclocked pins, and report details of constant propagation allowed them to be more efficient in debugging their design.

Galaxy Constraint Analyzer Reduces Iteration Time

Another design team, involved in home entertainment electronics design, reported that the iteration time between geographically separated front-end and back-end teams was reduced from days to less than 30 minutes with Galaxy Constraint Analyzer. Using the tool, both teams were able to identify and collaborate on the problems very clearly, especially problems involving clocks. They found the tool most helpful in finding and fixing many constraint problems including:

- Missing generated clock definition at some potential clock pins.
- Impossible generated clock versus master clock edge relationships: Such problems previously were detected only at the signoff stage. Galaxy Constraint Analyzer allowed them to detect and fix such problems in the early phase of product development.
- All "–to" objects for an exception were invalid: Such invalid exceptions are ignored by timing analysis tools and may lead to larger optimization and timing closure effort. Galaxy Constraint Analyzer allowed them to understand the root cause of the problem and rewrite the exceptions.

Galaxy Constraint Analyzer Reduces Analysis Time

Lastly, a Graphics Processing Unit (GPU) design group at a large semiconductor company ran Galaxy Constraint analyzer on one of their 80+ million instance designs. Galaxy Constraint Analyzer finished the analysis in 3 hours using less than 70 GB of memory. Other commercial constraint analysis tools were not even able to complete the analysis. The group's constraint scrubbing turnaround time shrunk from days to hours. This group decided to run Galaxy Constraint Analyzer every time an engineer modified the constraints. Within two weeks of using the tool, the CAD team saw significant productivity benefits. Galaxy Constraint Analyzer was then deployed to other groups where it was quickly able to pinpoint conflicting case analysis definitions and many clock definition issues.

Conclusion

Constraint analysis is becoming a crucial step to ensure an efficient design implementation and signoff process. However, in order for a constraint analysis tool to be effective, it has to interpret and analyze constraint specifications in a manner that is consistent and correlated with implementation and signoff tools.

Galaxy Constraint Analyzer provides an extensive set of rule checks designed to maximize the efficiency of the Synopsys Galaxy Design Platform. Galaxy Constraint Analyzer uses technology based on the Synopsys golden PrimeTime timing engine to ensure correct interpretation and propagation of constraints. This gives designers a signoff-correlated view of the constraints ahead of each step of the design implementation process. The ability of Galaxy Constraint Analyzer to deliver comprehensive constraint analysis on 10 million gate designs in a matter of minutes, combined with a unique set of interactive analysis and debug capabilities, helps designers quickly identify and fix constraint issues within hours versus days.

To summarize, Galaxy Constraint Analyzer delivers Look-ahead Constraint Analysis technology that makes it a truly effective constraints analysis tool that boosts designer productivity.