DFT rule checkers glue design together

By Rohit Kapur
Scientist

Chris Allsup
Marketing Manager for
Test Automation Products

Synopsys Inc.

With increasing chip design complexity, more cost-effective ways to manage the challenges of high-quality test are being sought. Firms have gradually shifted from high-maintenance in-house test tools to commercial test automation solutions for test synthesis and pattern generation. With such EDA products, developing and maintaining specialized functions is not considered an organization's core competency. Their functionality tends to be generic to require a degree of customization for supporting a range of design styles and test methodologies. Design-for-test (DFT) rule checkers, it turns out, "glue" everything together.

Test automation aims to generate high-quality test patterns for designs. This is done by using design analysis and algorithms application to generate test patterns for hypothesized failures within reasonable computed constraints. Known as ATPG, this capability is difficult to achieve without having to modify the design in various ways so that the algorithms can achieve acceptable results. In fact, DFT modifications, such as adding scan chains to a design, are needed to overcome inherent limitations of current ATPG technology. Thus, it can be argued that design rule checks (DRCs) for DFT ensure that the minimum conditions for the ATPG tool are met to obtain quality results.

However, design teams develop additional rules for their own requirements so that the evolved design rule checks are not as ATPG-focused as described. Designers adopt test methodologies and determine design restrictions so that commercial ATPG tools can routinely achieve high fault coverage. They may extend the scope of DRCs to check for good (not necessarily essential) design practices for test.

Consequently, the use of test rule checkers throughout the design flow has expanded to meet all desired constraints before running DRCs on the design netlist to extract information for the ATPG. A design house uses different commercial test-automation and in-house tools so that a DFT rule checker used in earlier design phases is different from the one used with the sign-off ATPG tool.

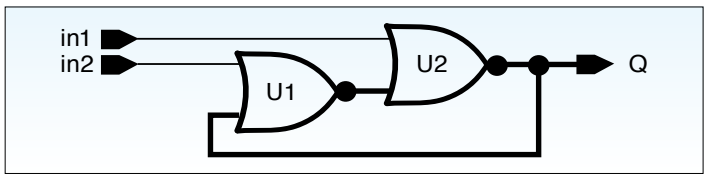

Figure 2: Combinational feedback loop increases the difficulty of controlling values on paths.

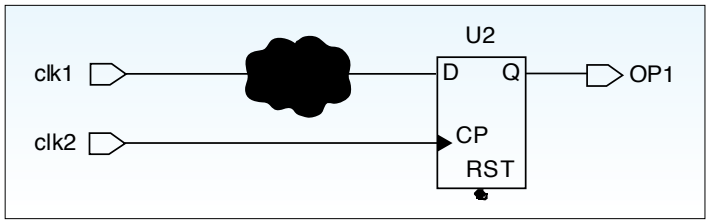

Figure 3: A clock affects data input of a register.

While the outcome of simple rule checking correlates across tools, it gets complex when specialized testing functions are incorporated into designs. Under such conditions, it is important that the same rule checker be used across all phases of the design flow. Using different tools for DFT implementations can increase the unpredictability of the test creation process and even lead to overcorrection of the design.

Types of checks

A large amount of engineering time and effort is spent on debugging DFT rule check violations. Commercial DFT rule checkers do not get the respect they deserve despite their high return-on-investment in design organizations. These tools routinely check DFT rule violations and automatically fix them.

Generally, DFT rule checkers should confirm that proper logic structures exist in designs, alerting the designer to any vio-

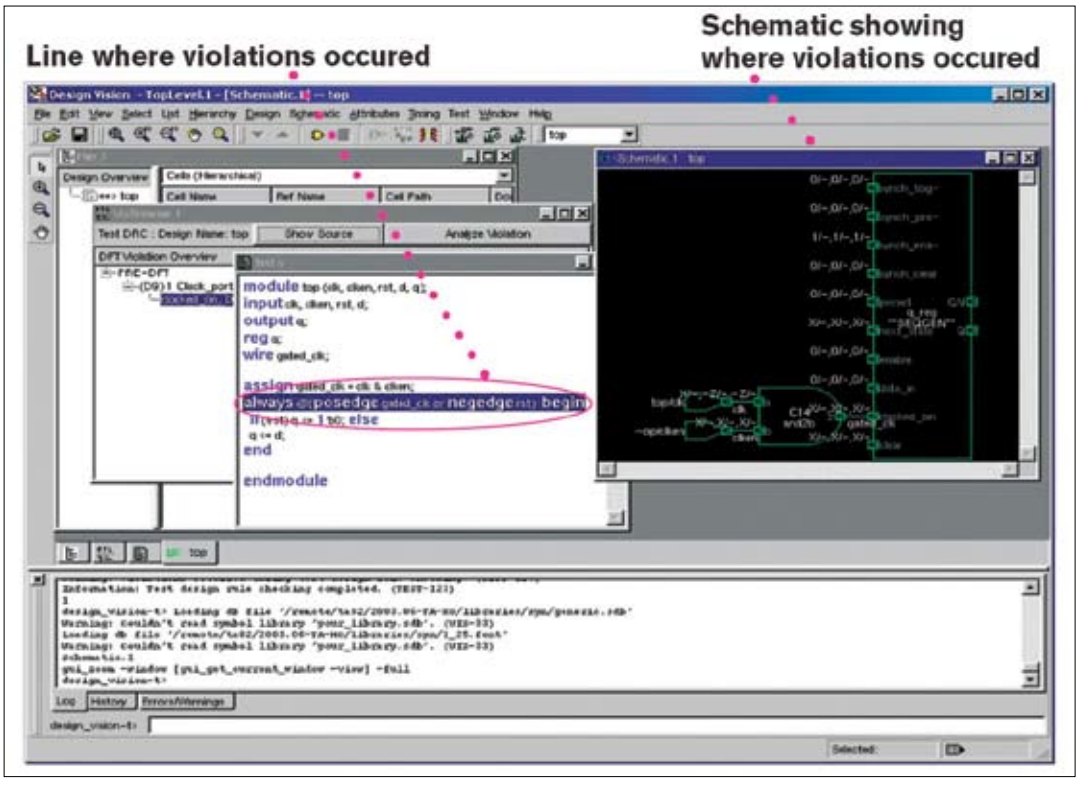

Figure 1: The GUI-enabled test rule checker integrates RTL test DRC violations and schematics.

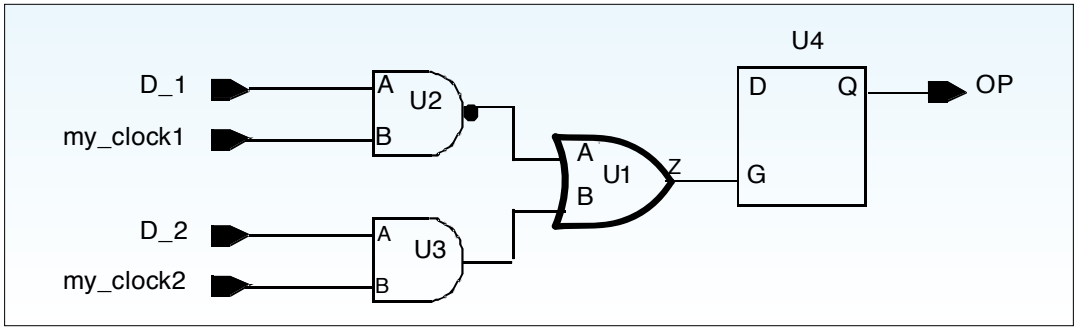

Figure 4: A valid latch clock-gating must contain an AND gate with one clock input and one or more data input, and an OR gate with clock or gated clock input.

lations. It should also assign data constructs, which the ATPG tool will use later. For example, it tells the ATPG tool that a flip-flop's state after scan shifting is unknown if not properly controlled. Moreover, it should optionally correct certain DFT violations automatically.

Meanwhile, DFT rules have several distinct categories:

- Test pattern generation-oriented rules—ATPG tools make assumptions about zero-delay combinational designs. Thus, many design constraints are needed for predictable, high-quality results.
- Fault coverage-oriented rules—These basic rules guarantee both controllability and observability of all logic portions. These are developed to test the final IC with highest possible coverage.
- Test methodology-oriented rules—Based on the type of test methodology, such as scan, logic BIST, I_{DDQ} testing, boundary scan or compression, several checks are run to ensure successful implementation or reuse of the test method.
- SoC reuse rules—Designs built on smaller modules demand DFT rigidity and discipline, which are not mandatory for complete chips. For example, bounding I/Os of a hierarchical block isolates the hierarchy from the embedded environment and successfully creates a more predictable test-pattern set.
- Test environment-oriented rules—Test patterns created by ATPG tools are applied on testers (ATE equipment). Tester limitations are considered beforehand to make sure that the test pattern works well.

All DFT rules are not created equal. Some are more critical to the IC creation process than others, so a severity level is associated with each. Designers may overlook non-showstopper violations without fear of adverse effects on the design's test operations or functionality. Thus, DFT rule checkers are used in most design-flow stages. In fact, earlier DRCs identify problems, and the earlier such problems are resolved, the less impact they have on project schedules.

Rule-check samples

The following examples represent DFT rule checks available with Synopsys' Galaxy Design Platform for Test, which includes RTL and gate-level DFT rule checking along with automatic repair of test DRC violations. The rule checker uses the same TetraMAX ATPG engine used for chip-level ATPG sign-off. In addition to generating batch reports, the test rule checker integrates DRC analysis inside the Design Vision GUI (**Figure 1**). Note that in case of such violations, a message is prompted containing information about the violation.

Example 1: Feedback loops—An active or sensitizable feedback loop reduces the fault coverage that ATPG can achieve because it increases the difficulty of controlling values on paths that make up the loop (**Figure 2**). An oscillating loop, for example, causes severe problems for both ATPG and fault simulation.

Designers break these loops by placing test constraints on the design, which deactivate the feedback loop. DFT rule checkers should not report violations on loops broken by setting constraints. If feedback loops are used as latches, designers must convert the combinational elements of the loop into its ASIC-based equivalent latch.

Example 2: Clocks on the input—A clock that affects the register's data input can reduce fault coverage, as some ATPG tools pulse only one clock at a time, keeping all other clocks off. Without this restriction, timing hazards can occur unless clocks are properly constrained for ATPG. In such cases, the rule checker will generate a message for it.

If this condition occurs for multiple flip-flops with different clocks, no timing relationship would allow the ATPG to capture all possible clock-as-data pulses. The safest approach is to configure the logic feeding the register's data input to eliminate clock-dependency.

Example 3: Multiple clocks—A latch can be used as part of a scan chain, if one clock or a clock ANDed enables it, with data derived from sources other than the clock. Multiple and gated clocks must be ORed together so that any clock can capture data. Since ATPG tools force all but one clock off at any time, latches that capture data from more than one clock must capture data with only one clock active.

For example, if a design has an AND gate with more than one clock input, the AND gate output will not generate a clock pulse if ATPG activates only one clock per unit time. For an OR gate with clock and data input, the output clock of the OR gate will have extra pulses based on the data input. Both cases are potential violations. To create valid clock-gating logic for latches, the circuitry must contain an AND gate with only one clock input and one or more data input, and an OR gate with clock or gated clock input.

Example 4: Black boxes—Logic that drives or is driven by black boxes drastically reduces fault coverage (**Figure 5**). This violation cannot be tested because the logic that surrounds the black box is unobservable or uncontrollable.

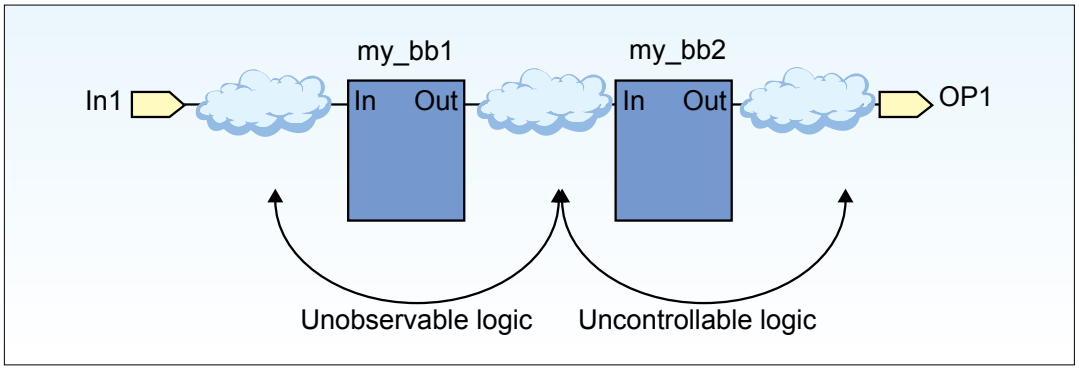

Figure 5: Logic surrounding the clock box is non-observable or uncontrollable.

Figure 6: Synopsys' test-rule checker shows the simulation waveform viewer for debug of complex test protocols.

Key benefits of DRC test checking come from identifying DFT issues earlier in the design flow so that its impact is minimized. The scope of test-rule checking has expanded and now includes comprehensive DRCs for all aspects of testing a design. Such include compliance checking for DFT methodologies to achieve consistent and predictable high fault coverage. With more complex DFT methodologies being implemented, unified test-rule checking throughout the IC design flow becomes mandatory.

There is greater pressure now on rule checkers because more resources are being devoted to debugging DRC violations than on other phases of the DFT process. More sophisticated design rule checkers help to quickly isolate and correct root causes of DFT problems, which create project bottlenecks. For example, Synopsys' Design Vision GUI, includes a simulation waveform viewer that helps designers more easily debug complex test protocols and initialization sequences (Figure 6). Advances in DFT rule checking will likely continue as more designs depend on increasingly complex testing protocols. Test-automation tools will meet these with more specialized DFT rule sets, and with even higher degrees of interactivity and customization.