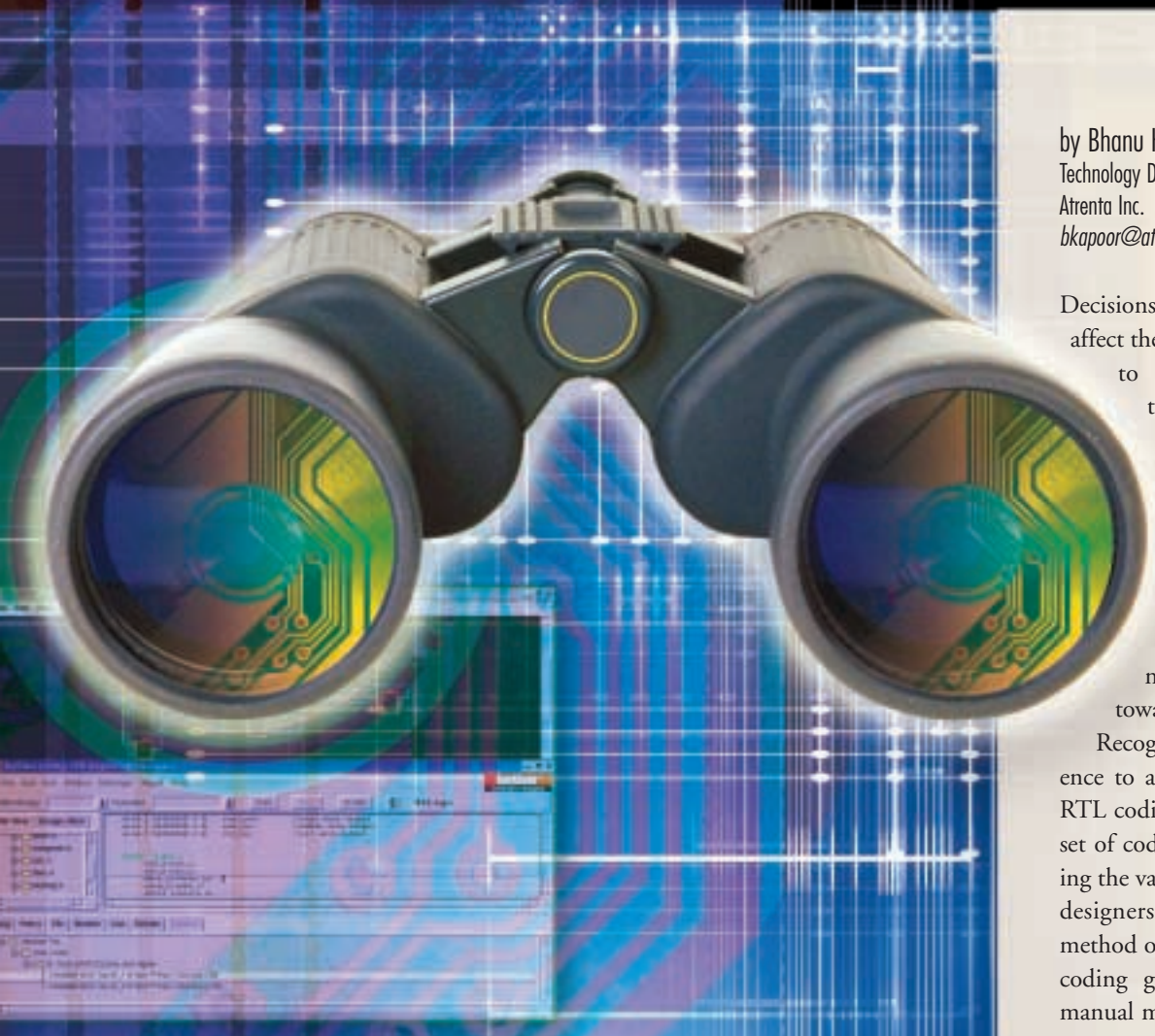
Use SpyGlass Predictive Analysis for Effective RTL Coding

Atrenta Inc.'s SpyGlass software uses a "look-ahead" engine based on fast-synthesis technology to help you identify potential problems – and fix them – early in the design process.



by Bhanu Kapoor
Technology Director
Atrenta Inc.
bkapoor@atrenta.com

Decisions made early in the design process affect the entire chip design process. Thus, to manage designs effectively, the tools you use for RTL (register transfer level/language) design must enable a stepwise refinement of the code by predicting likely downstream problems. For example, to achieve high performance, the guidelines for synchronous design must be met, as they play an important role toward meeting timing objectives.

Recognizing the importance of adherence to appropriate guidelines during the RTL coding process, Xilinx recommends a set of coding guidelines for designs targeting the various Xilinx device families. Most designers currently follow a manual method of design reviews to check to see if coding guidelines have been met. This manual method is both prone to error and time consuming. Automating adherence to

RTL coding guidelines, therefore, would greatly increase the timely success of projects targeting high-performance designs on Xilinx device families, such as the Virtex™-II series of platform FPGAs.

The SpyGlass™ Predictive Analyzer tool from Atrenta Inc. automates the process of meeting design guidelines through the use of an underlying predictive analysis technology. This approach performs detailed structural analysis on RTL aspects to check for coding styles, RTL-handoff, design reuse, clock/reset requirements, verification, timing guidelines, and much more. The SpyGlass tool employs a "look-ahead" engine based on fast-synthesis technology and a fast, built-in, cycle-based simulator to carry out such analyses. Such a look-ahead methodology only uses RTL code as input to the SpyGlass tool and does not require any vectors or assertions. It is therefore very easy to set up and run.

Policy-Based RTL Coding

The SpyGlass™ Predictive Analyzer is a comprehensive, policy-based system that defines, in a succinct and organized way, design policies that automatically point out time-consuming downstream issues. The SpyGlass tool then offers suggestions on ways to overcome these downstream issues during the RTL code development process. This helps you to meet your time-to-market target.

Policy

A policy is a collection of rules for a specific purpose, such as rules associated with a standard, a silicon vendor, or a specific design tool. Policies enhance user extensibility, allowing you to develop and manage customized groupings of rules more easily. The design methodology includes a set of policies that you can select during the process of RTL code development. Examples of such policies include lint, reuse, verification, timing, and testability.

Rule Groups

A collection of rules within a policy is termed a group. Typically, groups consist of rules addressing a particular area of interest in the RTL code. Groups are hierarchical, meaning that a group can contain other lower-level rule groups as well as individual rules. A group provides an additional level of modularity in applying policies to a given RTL design.

Figure 1 - Example of policy consisting of rule groups and rules

Rules

A rule is the most fundamental element in the policy-based management system. It describes a set of conditions that – when checked by the policy engine – result in an indication of a specific problem with the RTL code. Rules allow standard analysis of the RTL code. An example of a policy, rule groups, and rules in the SpyGlass system is shown in Figure 1. With this system, you can selectively turn on specific groups or specific rules within a group.

Policy Engine

The policy engine accelerates electronic product development by enabling development teams to capture, aggregate, distribute, and apply constraints and requirements early in the development cycle. The fast synthesis engine internally creates a structural view of the design and foresees downstream issues early in the development cycle, thereby eliminating errors at the earliest possible stage.

Although you can check many complex rules statically on the internal synthesized structural view, other rules require some understanding of the logic function of the design. This is particularly true for testability-related checks. In order to perform a testability check, you must use an evaluator. The evaluator in the policy engine is effectively a zero-delay, cycle-based simulator you can use to resolve functional design constraints, as well as carry out a simulation required to set up the design for testability analysis.

Policy implementation requires a traversal engine that works on the RTL netlist produced by fast synthesis. The SpyGlass engine generates basic primitives for rules that cover design traversal. The connectivity information, coupled with the traversal primitives, enable you to create rules that look for violations across the design hierarchy.

By applying a policy over a given RTL design code, you can obtain a wealth of information about the design – as well as any of violations of the defined rules. The SpyGlass interface highlights rule violations on the specific lines of RTL code. Not only does the SpyGlass predictive analysis tool offer an extensive help function to assist you in understanding the nature of the violation, but also, where

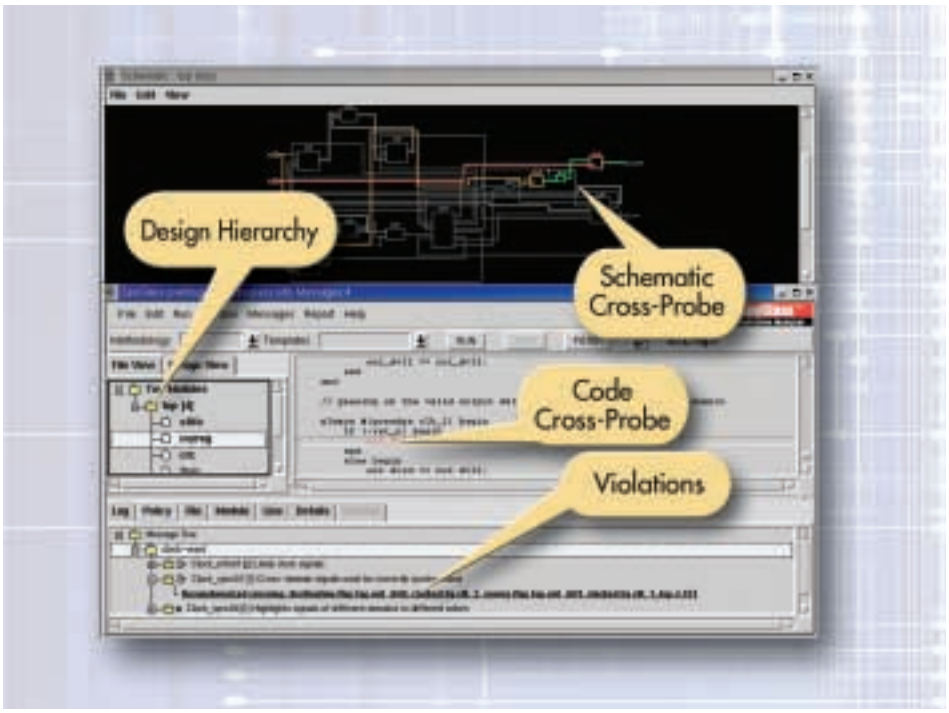


Figure 2 - Example of rule violation and associated debug windows

appropriate, it makes suggestions for resolving the problem.

Moreover, each violation is highlighted on a structural view of the design, which can give you additional debugging information for complex problems. And, as shown in Figure 2, cross-probing between the code and structural views reveals an example of a clock domain synchronization problem. Additionally, a comprehensive set of reporting mechanisms and violation management utilities – such as waiver to allow a specific rule violation in a given point in the design – further facilitate the RTL coding and assessment process.

Coding for Xilinx Devices

Xilinx recommends a set of coding guidelines for designs targeting the various Xilinx device families. These include general coding guidelines, as well as synthesizable HDL guidelines. Furthermore, synchronous design guidelines help you meet timing objectives in high-performance designs such as those targeting the Virtex-II FPGAs, which can accommodate as many as 8 million gates and operate at frequencies as high as 400 MHz.

Clock distribution also requires specific design code guidelines for successful implementation. For this reason, it is important to identify the clocks and resets in the design, as well as the clock-tree network, early in the code development process.

Many designs, especially in the networking domain, typically use multiple clock domains, a situation that presents challenging integration and verification issues. Signals that originate in one clock domain may be used in other clock domains. Correct chip operation can only be ensured if rules governing correct use of signals across asynchronous clock boundaries are followed.

The routability of design is another aspect that can benefit greatly by following appropriate coding guidelines. The number of I/Os, fanout of intermediate nodes, and widths of internal buses must meet specified limits for various parts of a given design.

Using the SpyGlass predictive analysis system, you can check for all of the above violations, problems, and issues during the RTL coding process itself. This saves many design iterations that might otherwise be needed to fix errors later in the design cycle.

Specifically, you can use the SpyGlass tool to check the following issues relevant to Xilinx FPGA designs:

- Single clock edge is used in the design to clock the data.
- Internally derived or generated clocks and set/resets are not used in the design.
- Appropriate synchronizers are used as signals cross clock-domain boundaries.
- The number of levels of logic between registers is less than a specified value.
- The fanout of any design node does not exceed a specified number.
- Unintended latch inferences are avoided in the code.
- If-then-else or case statements are not nested deeper than three levels.
- Naming conventions are followed while coding pipeline logic.
- Assess memory requirements and find out feasibility of conversion into regular flops, distributed memory, and block memory.
- For state machines, separate the next-state decoding and output decoding into two discrete processes or always blocks.
- Outputs from design units are registered.
- Identify dead code and floating nodes in the design.

Several of these checks are critical for meeting timing objectives in high performance designs.

Conclusion

We have described the elements of policy-based design methodology, enabled through SpyGlass predictive analysis, for effective RTL coding leading to efficient implementation of designs on Xilinx-based platforms. This approach performs detailed structural analysis on RTL code to check for coding styles, design reuse, clock/reset requirements, verification, timing guidelines, and more. SpyGlass software includes the most comprehensive coverage of Xilinx-recommended coding guidelines that are essential for reuse and efficient design implementation.

To learn more about SpyGlass predictive analysis, go to *www.atrenta.com.*