cadence®

Choosing the Right Superlinting Technology for Early RTL Code Signoff

By Pete Hardee, Cadence

No one can afford to go through weeks of verification only to discover problems in the register-transfer level (RTL) code that might not be functionally wrong, but do not follow established rules for successful implementation. Traditional lint tools have become ineffective in evaluating RTL code for today's larger, more complex designs. However, superlinting technology, such as the Cadence® JasperGold® Superlint App, brings together linting with automatic formal checks to provide an efficient way to assess RTL code quality early on, well before verification and implementation. This paper discusses key care-abouts for the right superlint solution and discusses the benefit of effectively bringing formal to the designer's desktop.

Contents

Introduction

During early verification, when problems are discovered in the code that could hamper implementation, big problems can ensue. In such situations, you must make changes to your RTL and redo the verification effort. This wastes valuable time and resources, putting you at risk of missing key project deadlines.

Years ago, line interpreters emerged as tools that could interpret code and provide warnings about issues in the syntax of that code. In the RTL world, linting has surfaced to provide a set of coding style and structural checks to assess the quality, without requiring any verification environments. As designs have grown larger and more complex, linting alone has proven insufficient. Instead, augmenting these simpler checks with checks using true formal technology—superlinting—has shown great benefit in terms of finding more complex issues earlier with minimal effort. Superlinting provides an effective way to eliminate common coding style, structural, and functional design errors before verification begins.

Because of aggressive project schedules, RTL designers often find themselves performing basic verification as soon as enough RTL code is available. There isn't always time to wait for testbenches to become available and to hand the designs over to verification specialists. However, writing assertions in standard languages like SystemVerilog Assertion (SVA) or Property Specification Language (PSL) calls for specialized knowledge and can be time-consuming even for seasoned experts. Superlinting makes this process easy by automatically extracting properties from RTL and enabling designers to verify these with formal technology.

Integrating Formal into Linting

Traditional lint checks cover a variety of areas without requiring testbenches, including:

- Naming and file format: Assesses naming convention for design elements and formatting of the RTL source file
- Coding style: Evaluates semantics and reusability and portability of the code
- Simulation/synthesis mismatch: Identifies unsynthesizable constructs and pre-synthesis/post-synthesis mismatch; for example, a signal missing from the sensitivity list of a combinational always/process block
- Structural: Pseudo-synthesizes RTL description to obtain structural view; assesses feed-throughs, combinatorial paths, shoot-throughs, and more
- Finite state machine (FSM): Evaluates compliance to FSM modeling styles
- Race condition: Detects all simulation race conditions as statically possible (read-write, write-write, trigger propagation)
- Normal-mode and test-mode design for test (DFT): Detects DFT issues that make design nodes uncontrollable and unobservable; checking is done both with and without the test circuitry present in the design

Automatic formal checks use formal intelligence to evaluate the quality of the code, bringing into the mix automatically created assertion-based checks for:

- Code reachability: Analyzes pieces of code that do not appear reachable from the current inputs, examining areas including:
  - Block: If all branches in the RTL are reachable
  - FSM: If all branches of FSM are reachable
  - Toggle: If all signals can be toggled
- X assignment: Checks whether X assignments are reachable
- FSM livelock/deadlock: Assesses whether there's a scenario where the design either can't get out of a working state or hits a dead end
- Bus contention: Determines whether multiple drivers are attempting to write to the same bus at the same time
- Pragma: Checks whether synthesis pragmas are honored; this is designed for simulation only and is not to be synthesized
- Range overflow: Evaluates whether range overflow can occur for array accesses
- Arithmetic overflow: Assesses whether overflow can occur in the case of an arithmetic operation
- Unwanted X sources: Determines if there are places in the design from where X can propagate and can cause different results at the gate level

Together, traditional RTL linting and automatic formal analysis enable you to do more structured verification and find design bugs earlier in your process. The JasperGold Superlint App brings these two technologies together in one platform, enabling designers to debug the violation and provide advanced waiver and reporting capabilities.

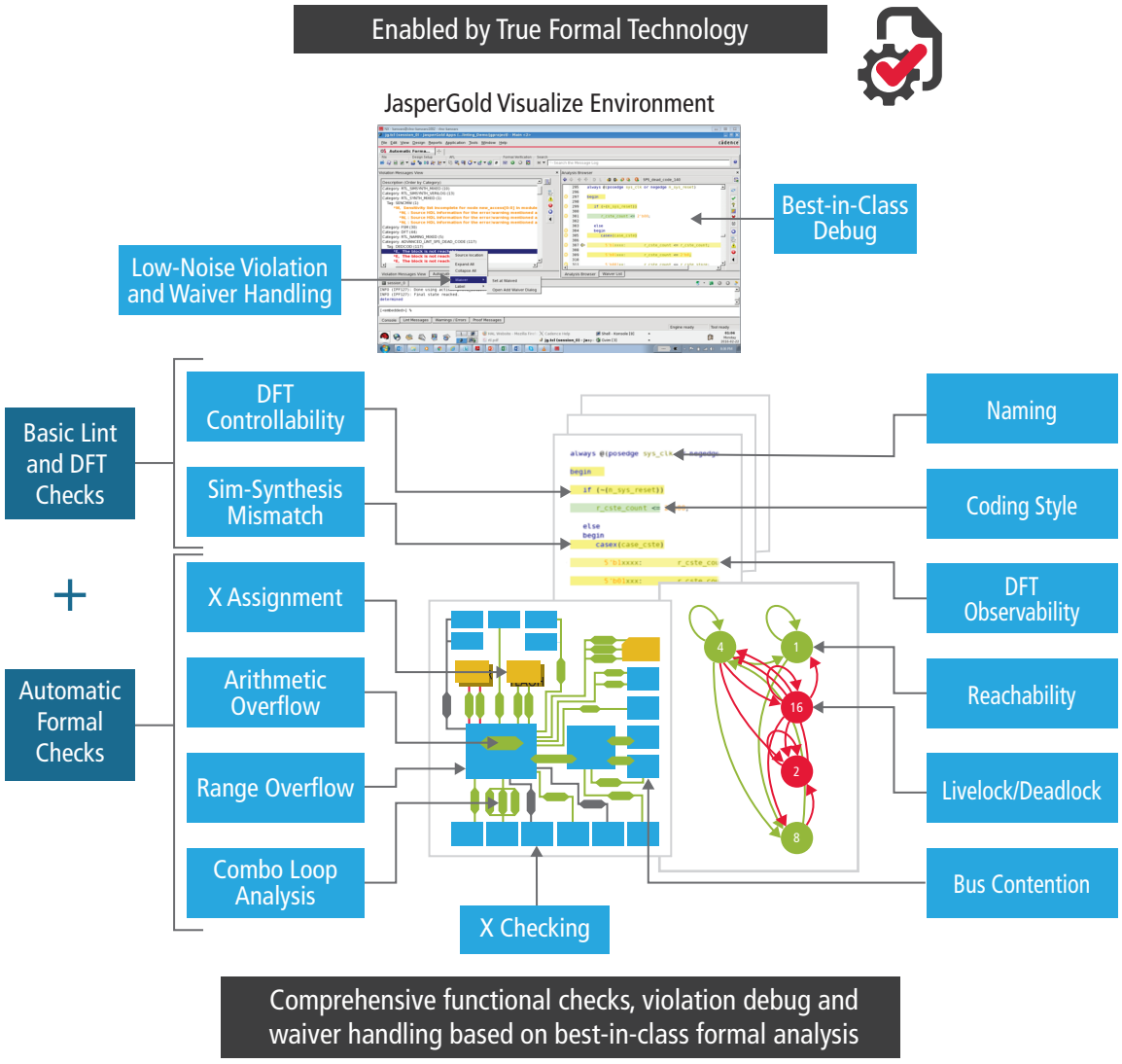

Fig 1: JasperGold Superlint App: Hand off robust reusable RTL

When to Use Superlinting Technology

If your designs are typical, they likely incorporate off-the-shelf and internally developed intellectual property (IP) blocks that are designed to use in multiple systems on chip (SoCs). As such, you'll want to perform as many checks as possible at the RTL and IP levels to demonstrate that you've got robust, reliable IP. IP signoff typically encompasses linting, DFT, and low-power checks. Once the IP passes these tests, there may be some violations that you can resolve at the source or waive, if they pose no consequence for the intended implementation, but the block is essentially ready to go into your RTL design repository. Then, when the implementation teams are ready to integrate the IP blocks into different chips, they should ideally be at a stage where they need to make either minimal or no changes to the design. The IP should already be clean.

Superlinting should be used during RTL development where it performs extensive checks and generates automatic formal checks to help increase design quality. It can be used to explore and clean the design for issues such as deadlock, livelock, range overflows, and simulation mismatches. The resulting code is then ready for you to hand off for implementation and verification, with confidence that it confirms to your set of rules.

Designers can also use superlinting technology to perform incremental checking during design development and integration. For example, you can use the superlinting solution as a regression baseline for ongoing validation. If you're a verification engineer, you can use superlinting to complement a bug-hunting flow based on formal property verification, or by taking advantage of its automatic formal checks.

RTL Signoff Flow

Superlinting is now part of an overall RTL signoff strategy where the momentum is to move all possible checks to the RTL design and IP levels. This trend has emerged in part because IP blocks are often reused in many SoCs. At the same time, particularly with larger organizations, IP and SoC design teams are often separate. So, it has become pragmatic to perform checks early on. The combination of robust IP and a good methodology can help minimize late surprises and encourage much easier netlist signoff checks.

On the RTL design side, the goals toward achieving robust, reusable IP include:

- Performing lint, DFT, and automatic formal checks
- Performing checks to assure clock domain crossing (CDC)-clean RTL
- Easy debug and waiver handling

The primary goal of RTL implementation checks is to achieve SoC integration signoff via:

- Signoff checks with SoC-specific constraints
- Full chip-level capacity and performance
- Reuse of IP-level metadata/waivers to reduce noise

An integrated RTL signoff flow, therefore, should involve debugging and fixing RTL at the source. Then, any issues in SoC integration or implementation should be resolvable by constraint changes. Cadence provides the JasperGold Superlint App, providing a unified tool that delivers RTL code quality checks, automatic formal checks, and differentiated debug/waiver handling. The Superlint App automatically generates IEEE standard SVA properties based on your RTL, without requiring any testbenches, stimuli, or specialized knowledge of SVAs. Generated properties can be ranked, pre-classified, and output in standard SVA, and can then be proven using formal technology or used in any assertion-based verification flow to further increase functional coverage and reduce debug time.

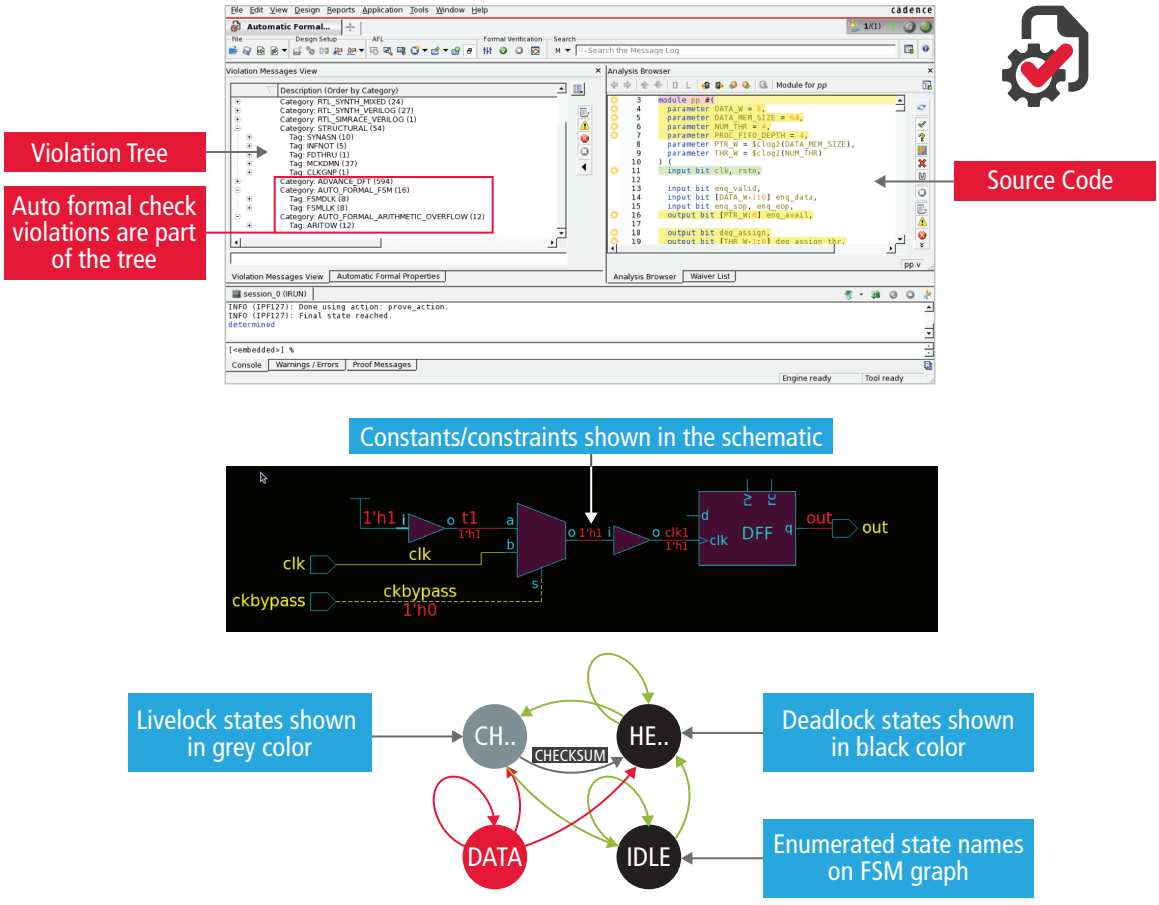

Fig 2: Debugging violations and creating waivers

Competitive solutions are considered to be noisy—violation alerts continue to pop up even after the issue has been resolved. With the JasperGold Superlint App, once a violation has been addressed and turned either into a waiver or resolved, it can be filtered out. The solution works with the JasperGold Visualize™ interactive debug environment, which generates reports and provides a view of violation messages, advanced waiver capabilities, and more. In this environment, you can create formal traces to debug without actually executing the design. You can also link back to the source line in the source code that is triggering the violation.

Summary

Using the JasperGold Superlint App can help shave off weeks of tedious work. By eliminating common design rule and functional errors and ensuring your RTL code is clean before you hand it off for functional verification and implementation, you can improve design quality while also meeting your time-to-market commitments.

Further Information

To learn more about the JasperGold Superlint App, visit www.cadence.com/ [URL TBD]