

Intelligent Use of Hierarchy in Formal Equivalence Checking

Formality Hier-IQ™ Verification Technology Overview

February 2002

Today's complex SoC designs present many verification challenges for design teams. Historically, to ensure design integrity throughout the implementation process, engineers used a bottom-up, hierarchical equivalence checking methodology to reduce the size and complexity of the verification. This methodology offered good coverage and was much faster than gate-level simulation. However, to meet tight timing and area constraints for SoC designs, engineers need to use more advanced synthesis and routing techniques that complicate the bottom-up verification flow, making it impractical.

Figure 1 shows an example of a design that has undergone boundary optimization. Hierarchical verification of Module M in the reference design REF 1 to Module M' in the implementation design IMPL 1 would yield a failing result. However, if you view the same module in the context of the entire design, you can see that the inverters at the top level of REF 1 make the designs functionally equivalent.

Figure 2 shows a similar design that has undergone routing optimizations. Functional verification of Module L in REF 2 with Module L' in IMPL 2 fails when performed on the lower-level modules. However, if you consider both modules in the context of the entire design, port B is connected to OUT 1 and port C is connected to OUT 2. A flat verification (one that considers the entire context) would succeed for this design.

Design Transformations That Complicate Bottom-up Verification Flows

- **Boundary optimization** – such as inversion pushing – that moves logic between levels of hierarchy results in verification failures if changes are not considered.
- **Block-name changes** throughout the design flow adversely affect module-name mapping.
- **Clock trees** with clock ports that cross hierarchical boundaries cause unmatched clock pins, producing compare-point mapping errors.
- **Periodic cell flattening** at different design stages that eliminate hierarchy. With periodic cell flattening, the user must debug a failing verification run to determine whether flattening a block in the reference design will yield a passing result. In some cases, the entire design must be flattened to yield a passing result, eliminating the advantages of hierarchical verification.

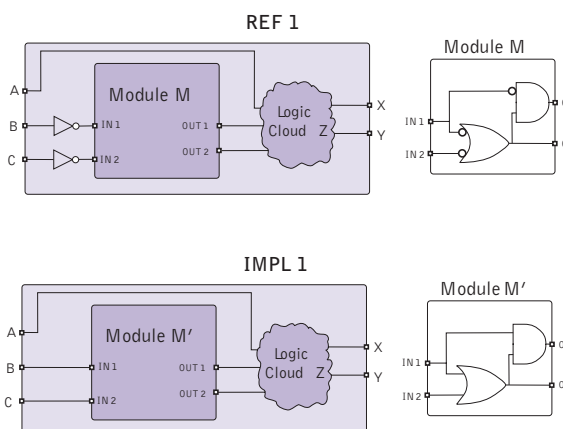


Figure 1: Boundary Optimization.

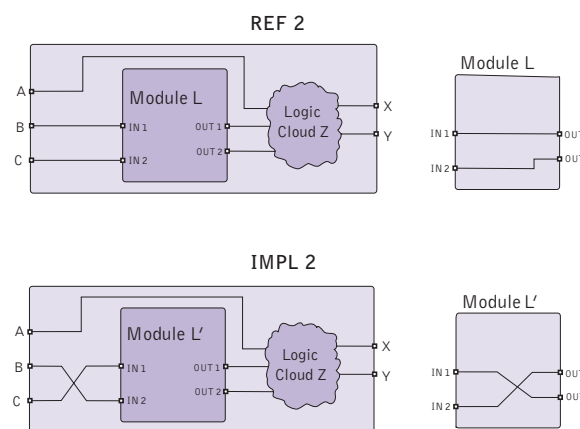


Figure 2: Routing Optimization.

What Makes Flat Verification Inefficient?

- Large, complex logic cones:
 - Are difficult to verify conclusively
 - Significantly increase verification run time
- Requires a great deal of memory

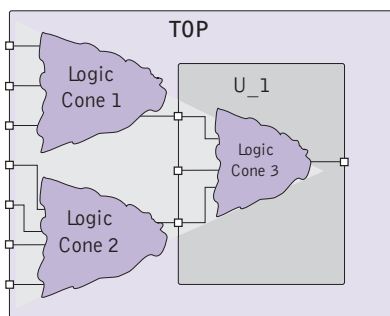


Figure 3: Flat Verification.

Laboriously documenting transformations such as those shown in Figures 1 and 2 can aid verification engineers in the proper setup of equivalence checking constraints. For example (see Figure 2), hierarchical verification would pass if a constraint were set to match port IN 1 of Module L with port IN 2 of Module L' and another were added to match port IN 2 of Module L with port IN 1 of Module L'.

However, engineers are often given the RTL and the netlist of a design without any knowledge of its block-level functionality. Information about inter-hierarchical transformations, such as the examples in Figures 1 and 2, is rarely available to the verification engineer.

To circumvent the problems arising from hierarchical verification, a design may be verified as a single flat entity. The advantage of a flat verification is that all logic between primary inputs and outputs are considered in a single pass; therefore, flat verification requires minimal setup. However, while flat verification is easier to run, it is slow and inefficient for full-chip and large designs.

Figure 3 illustrates how flattening can make verifications difficult. To verify this design hierarchically, you would first verify U_1, "black box" it, and then verify TOP. When using an hierarchical approach, Logic Cones 1, 2, and 3 are verified independently. Flattening and verifying the entire design results in one large logic cone, complicating verification. The size and complexity of combined cones can negatively impact verification runtime and significantly reduce the capacity. Also, if one or more of the logic cones includes arithmetic, flattening nearly always results in an inconclusive verification.

The Optimal Solution

Instead of flattening the entire design, what if a verification tool adopted an intelligent approach to hierarchical verification? For example, an engineer can quickly conclude that the verification of Module M and Module M' in Figure 1 will pass if the inverters in the upper level are considered as part of Module M, because these are all that is needed to verify equivalence. Logic cloud Z can still be treated as part of a separate logic cone. On the other hand, the design in Figure 3 can be verified hierarchically, so there may be no need to look at the context.

If verification tools offered an intelligent approach to hierarchical verification by only looking beyond boundaries when necessary, engineers would have the best of both worlds: small logic cones for fast memory-efficient verification with an environment that requires minimal setup. This is, in general, how Formality's Hier-IQ technology operates.

Formality Hier-IQ Verification Technology at Work

Instead of verifying blocks upwards through a design, Formality's Hier-IQ verification technology recognizes logic beyond hierarchical boundaries to eliminate false-negative results without the need for additional setup.

Formality verifies every compare point considering its entire (flat) context, and limits the size and complexity of logic automatically by considering intermediate boundaries. The intelligent selection of intermediate points results in the performance of a successful hierarchical verification, with the accuracy and ease-of-use of a flat verification (see figure 4). In addition, Hier-IQ technology employs proprietary methodologies to limit memory consumption to the levels normally associated with hierarchical verification.

The Formality Hier-IQ Advantage

- **Simple setup:** Requires no additional parameters for boundary optimization, module name changes, clock port name changes, or cell flattening.
- **No false-negative failures:** Reduces debug efforts by removing errors that result from missing setup parameters.
- **Performance runtimes:** Equivalent to successful hierarchical verification times.
- **Capacity/memory advantages:** Handles design sizes normally associated with hierarchical verification.

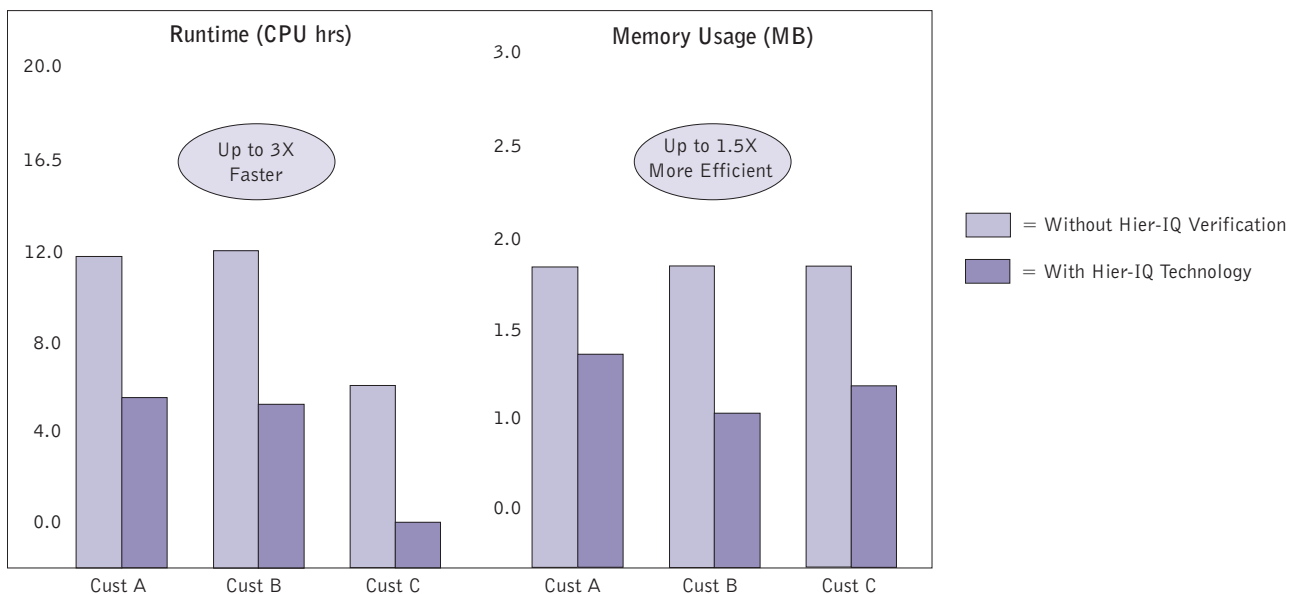


Figure 4: Hier-IQ technology contributes significantly to Formality's performance and capacity improvements.

SYNOPSYS®

700 East Middlefield Road, Mountain View, CA 94043 T 650 584 5000 www.synopsys.com
For product related training, call 1-800-793-3448 or visit the Web at www.synopsys.com/services

Synopsys, the Synopsys logo and Formality are registered trademarks and Hier-IQ is a trademark of Synopsys, Inc.. All other products or service names mentioned herein are trademarks of their respective holders and should be treated as such. All rights reserved. Printed in the U.S.A.
©2002 Synopsys, Inc. 2/02.PS