

Complex FPGAs Require Equivalence Checking

Synopsys' Formality equivalence checker proves identical functionality.

by Lonn Fiance
Director, Strategic Alliances
Synopsys
lonn@synopsys.com

Robert Vallelunga
Product Manager
Synopsys
robertv@synopsys.com

With the latest process technologies, complex high-performance systems – once the exclusive domain of ASIC and custom chips – can now be created with FPGAs. Although this benefits both consumers and companies, it also creates significant new design and verification challenges that must be addressed to meet time-to-market requirements.

As a designer, you use system-level analysis and simulation to evaluate different architectural alternatives. The result of this architectural exploration is the RTL (register transfer language) specification, which forms the definition of your device's functionality.

Depending on the product requirements, you can realize this functionality in a number of forms: using Xilinx® devices as FPGAs designed into the final product or prototypes for verification of complex ASICs.

The RTL description defines the functionality, so maintaining equivalent behavior for all implementations under all circumstances is critical. As design sizes have increased, the ability to prove equivalence by exhaustive simulation has disappeared.

ASIC designers recognized this issue several years ago and turned to equivalence checking (EC) to maintain functionality. When you design one of today's high-complexity FPGA devices, whether used in the final product or as an ASIC prototype, you are clearly facing these same challenges. As a result, EC is quickly becoming a mandatory verification methodology in FPGA design flows. This methodology becomes even more important if you have selected the EasyPath™ low-cost FPGA solution for your production needs.

In this article, we'll provide an overview of EC technology and the benefits it brings to FPGA design. Using the Synopsys Formality equivalence checker in your FPGA design flow enables you to quickly verify the implementation of your device, giving you the freedom to focus your efforts on other design tasks.

Introduction to Equivalence Checking

EC is a formal, static verification technology that uses mathematical techniques to determine if two versions of the same design, at different stages of development, are functionally equivalent. The power of this technique is its ability to compare between:

- Two RTL versions
- An RTL description and a gate-level netlist
- Two gate-level netlists

EC flows consist primarily of four stages:

1. Read: the EC tool reads RTL descriptions or netlists for the reference and implementation designs and segments the logic in each design into smaller components called logic cones (Figure 1). Logic cones are simply small groups of logic bordered by registers, ports, and black boxes. The end points for each logic cone are known as compare points.

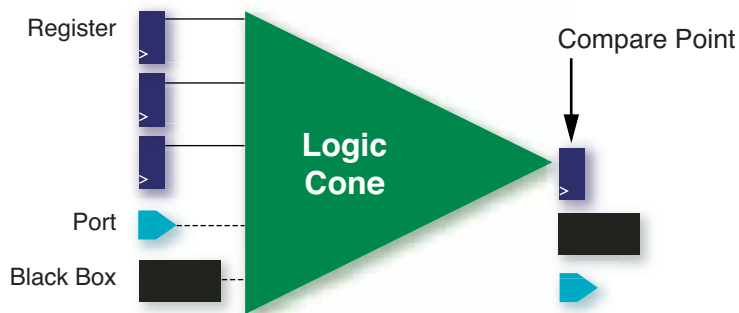


Figure 1 – Logic cone with compare point

2. Match: using a variety of techniques, the equivalence checker maps corresponding compare points between the implementation and reference designs (Figure 2). The fastest of these techniques is name-based matching, but differences between the RTL and gate-level representations can lead to highly dissimilar names, making advanced matching algorithms necessary. Registers may have been duplicated, merged, or otherwise optimized away such that the logic cones between design versions may be quite different, making the matching process far from simple.
3. Verification: using mathematical techniques, each set of matched compare points is proven to be functionally equivalent or non-equivalent.
4. Debug: when non-equivalent logic cones have been identified, graphical debug techniques are available to isolate the logic causing functional deviations.

Static verification has two primary benefits: it is orders of magnitude faster than dynamic verification and provides 100% verification coverage. EC can prove that different versions of a design are equivalent (or not) in a matter of minutes, rather than the hours or days required for dynamic simulation. Equivalence checking also eliminates

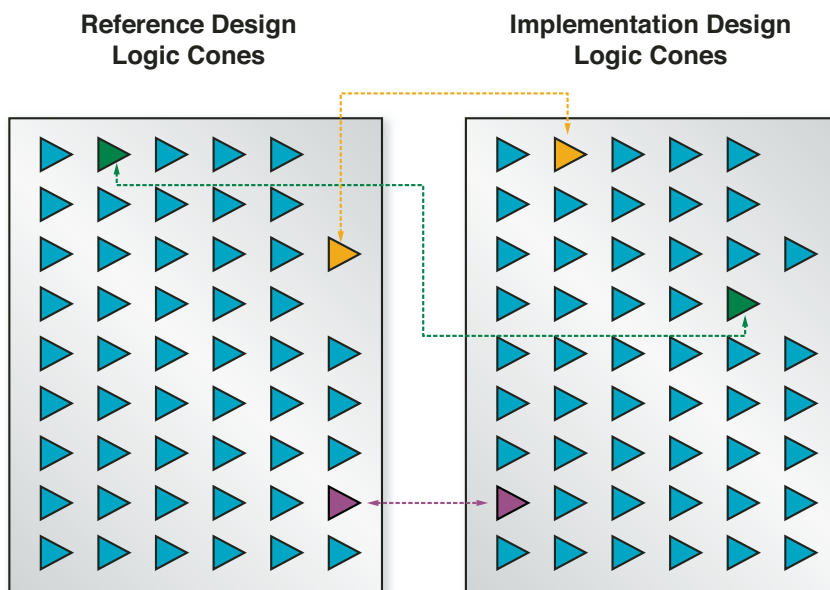


Figure 2 – Matched (or mapped) compare points between design versions

Today's very large programmable devices may contain hundreds of thousands of logic cells, making equivalency checking a mandatory component of the modern competitive design flow.

the need to create extensive sets of test patterns in an attempt to demonstrate equivalence by automatically identifying compare points and then applying sophisticated verification techniques. Dynamic simulation remains an important part of your verification strategy, but its primary use is to ensure proper RTL behavior.

Equivalence Checking in FPGA Flows

Because the value of EC is well understood by ASIC designers, its initial application to FPGA prototyping is an obvious choice. In the prototyping context, you want to prove that your FPGA prototype implements the RTL functionality that will be used to create an expensive ASIC.

The FPGA verification usually happens in two phases: first by proving the equivalence of the reference RTL to the

post-synthesis netlist, and second by proving the equivalence between the post-synthesis netlist to the post place and route (PAR) netlist. You can perform a similar check between the reference RTL and the ASIC implementation netlist(s) to ensure that your final ASIC matches the functionality proven in the FPGA prototype, as shown in Figure 3.

The benefits of EC also apply when you intend to use the FPGA as the final implementation of your design. The ability to quickly and exhaustively prove the correspondence between your reference RTL, the post-synthesis netlist, and the post-PAR netlist significantly reduces simulation time or multiple programming iterations, neither of which can conclusively prove the equivalence between various versions of your design.

Block-level design elements like digital clock managers (DCMs) and block RAMs, inherent features of the Xilinx architecture, pose interesting challenges in FPGA designs. When they are instantiated – as is common in FPGA prototyping flows – they are treated as “black boxes” for synthesis and EC purposes. The contents of a black box are not verified, but the signals at the periphery of the “black-boxed” element are proven to be equivalent.

RAMs can also be inferred from the RTL. These inferred RAMs can be directly verified by Formality as long as the inferred RAM does not become excessively large. Large memories can be fully verified using memory-specific verification tools.

Using Formality in a Xilinx Design Flow

Formality, the proven equivalency checking tool from Synopsys, allows you to verify designs 10-100X faster than simulation-based techniques – and identify and correct logic errors 5-10X faster – because of its graphical schematic debug capabilities. This performance advantage enables you to run Formality at every stage of the implementation flow and catch errors when they are first introduced, greatly reducing the cost of correcting them.

Functional verification using Formality is the clear winner in terms of performance and error isolation. Today's very large programmable devices may contain hundreds of thousands of logic cells, making equivalency checking a mandatory component of the modern competitive design flow.

Synopsys and Xilinx have created the Formality EC flow for Xilinx FPGA designs (shown in Figure 4). Design discrepancies can result from the numerous transformations that take place during synthesis and in ISE™ software (NGDbuild, MAP, and PAR). These transformations may change the design to

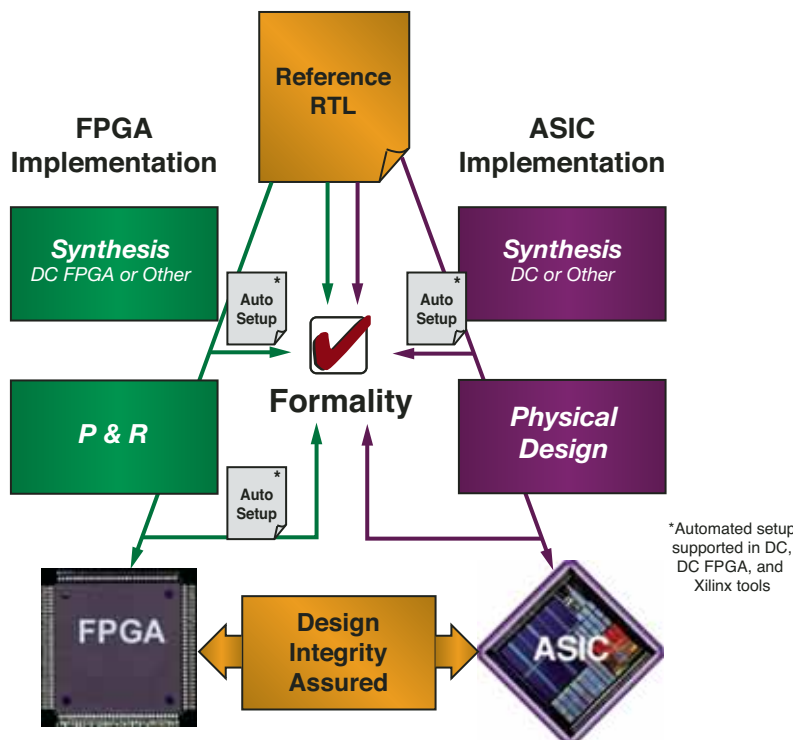


Figure 3 – Verifying an ASIC implementation against an FPGA prototyped implementation

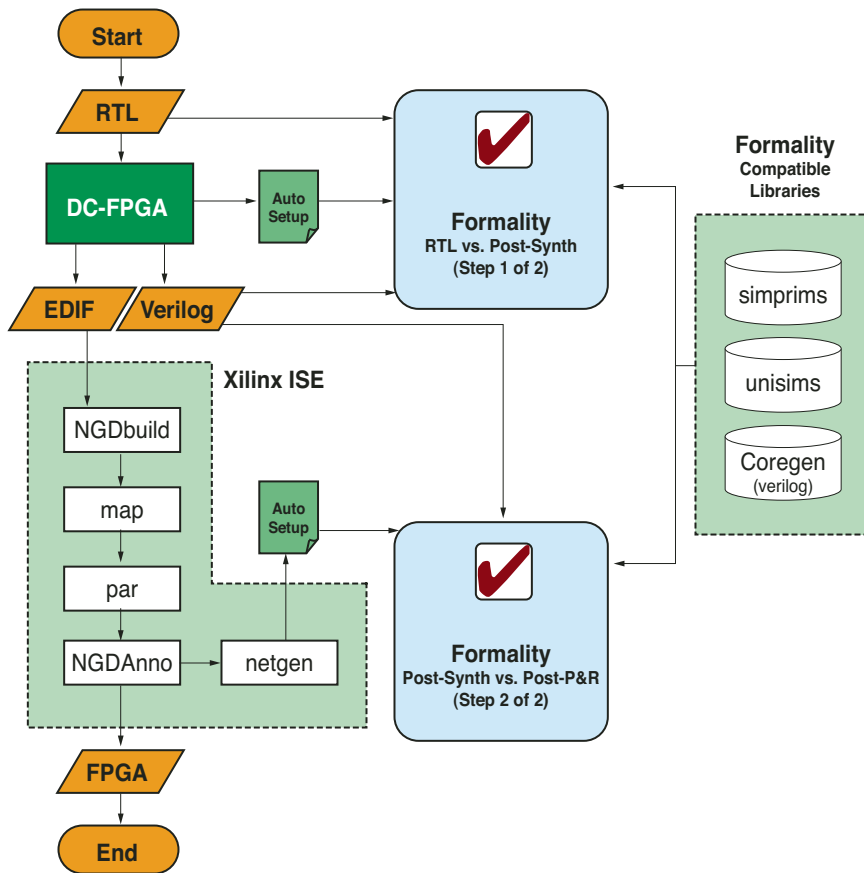


Figure 4 – Formality/DC FPGA/Xilinx verification flow

improve timing, reduce area or power, better match the architectural features of the Xilinx device, or meet design rules. Transformations such as combinatorial reductions, sequential optimizations (retiming), FSM re-encoding, register merging or duplication, and place and route optimizations increase the risk of unintended functional changes.

EC tools consider two versions of a design independently, without knowing how they were created. Therefore, Formality has no awareness of the transformations that occurred during design implementation. These transformations include changes to the logic, in net and instance names, in hierarchy, and to the number or meaning of state elements. These changes may impact the names of compare points, preventing the use of efficient name-based matching techniques and increasing the use of advanced but slower matching techniques.

Additionally, any changes to a bounding element of a logic cone results in a change to

the logic cone itself. For example, if you optimize away a register during synthesis, you remove the end point of a logic cone, forcing that logic cone to be expanded until another end point is found. The effect is that the matching of compare points may no longer be possible; even if a match is made, the logic between corresponding cones is no longer equivalent and verification will fail.

Tight Linkage Between Tools Improves Usability

In the past, EC tools required extensive tool setup to understand these logic optimization transformations. Fortunately, Synopsys Design Compiler FPGA and Xilinx PAR tools write out an automated setup file for Formality. This file contains information telling Formality which optimizations were performed in particular areas of the design, minimizing the manual setup information that you might otherwise need to provide. Formality uses this setup information to understand the differences between the two

versions and, after validating the information, uses it to more quickly complete the equivalence check. Linking multiple tools through the setup file helps you achieve the fastest possible time to results and eliminate errors that might be introduced in a manual setup process.

Complex FPGA devices are increasingly being designed using modular, or hierarchical, flows. Formality provides an ideal way to verify each individual module and ensure that they have been correctly stitched together at the upper levels of your design.

Formality is a mandatory step in the EasyPath methodology. EasyPath devices are not reprogrammable, so ensuring that the device is 100% equivalent to the reference RTL is essential. Formality's static analysis techniques prove functional equivalence, minimizing the risk of implementing incorrect functionality to help you reach volume production sooner.

Conclusion

Today's FPGAs have achieved the same level of functional complexity as some ASICs. The ability to realize very complex functionality in FPGAs has created tremendous challenges in the FPGA verification process. Formality helps you:

- Achieve 100% coverage
- Reduce runtimes 10-100X versus traditional dynamic verification
- Quickly isolate discrepancies between the RTL and implementation
- Reduce or eliminate the time spent re-simulating after a design change
- Verify that changes did not unintentionally impact other functionality

Synopsys and Xilinx have worked together to create a proven FPGA static verification solution centered on Formality, Design Compiler FPGA, and Xilinx ISE implementation tools. Formality provides a fast, thorough functional verification methodology for proving equivalence between multiple representations of your design. For more information on Formality or DC FPGA, visit www.synopsys.com or contact your Synopsys sales representative. ●●●