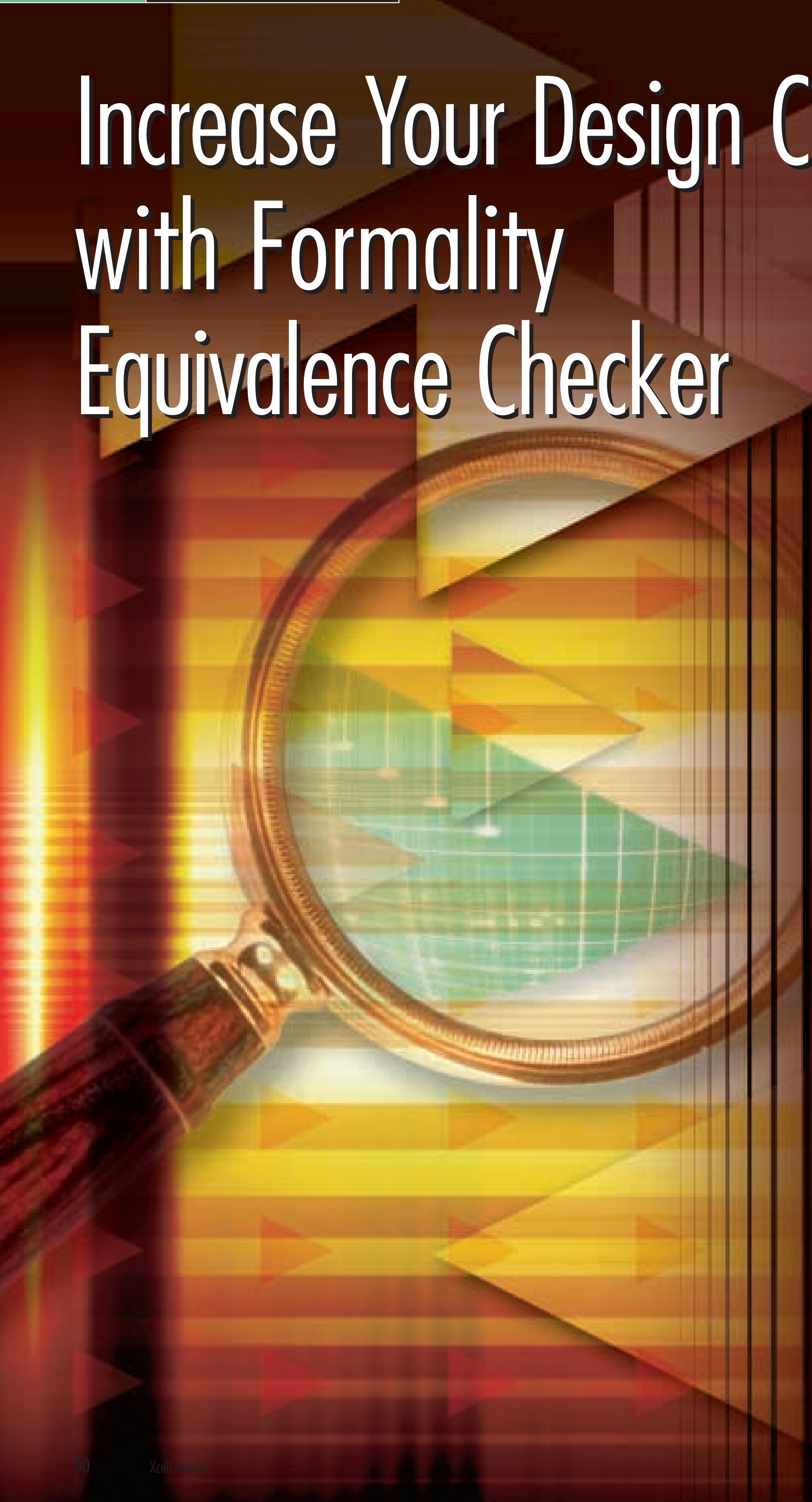
# Increase Your Design Confidence with Formality Equivalence Checker



Learn how to use Synopsys Formality Equivalence Checker with Xilinx ISE tools to verify complicated designs for Xilinx Platform FPGAs.

by George Mekhtarian
Technical Marketing Manager
Synopsys, Inc.
*georgem@synopsys.com*

Today's large, complex Platform FPGAs, such as the Xilinx Virtex™-II and Virtex-II Pro™ series, can exceed 10 million system gates and operate at speeds of 300 MHz or more. SoC (system-on-chip) designs targeting Xilinx Platform FPGAs are now subject to the same functional verification delays as large ASIC designs. Just as with ASICs, you must now employ a type of static verification technology known as equivalence checking (EC) to verify FPGA design logic and functionality.

Using the Formality® equivalence checker from Synopsys in a Xilinx Platform FPGA design flow allows you to verify equivalence quickly between RTL (Register Transfer Language) and the synthesized gate-level netlist – and between RTL and a post-Xilinx place-and-route (PAR) netlist as well. Formality EC increases confidence in functional integrity during design implementation, giving you the freedom to focus on debugging actual design problems.

## How Equivalence Checking Works

EC is a branch of static verification that employs formal mathematical techniques to prove that two versions of a design are functionally equivalent. In the first stage of the process, both versions of the design are read into the equivalence-checking tool. During the read process, each design is automatically segmented into manageable sections called "logic cones." Logic cones (Figure 1) are groups of logic bordered by registers, ports, or black boxes (BB). The output border of a logic cone is referred to as a "compare point."

Next, the tool attempts to match, or "map," logic cones from the reference design to the corresponding logic cones within the implementation design. This is called "matching" (Figure 2). Both non-function (name-based) and function-based matching methods are deployed to map compare points.

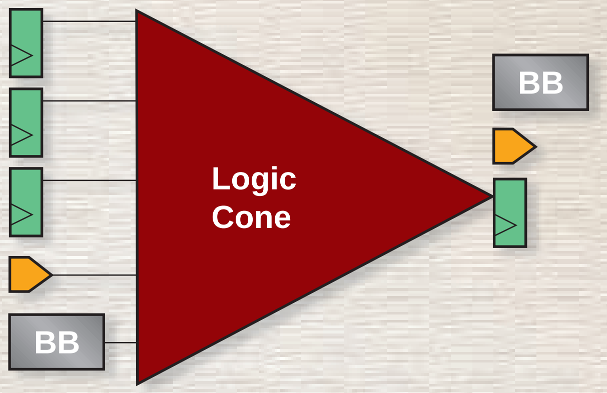


*Figure 1 - Logic cone*

Once the logic cones have been matched, the next step is to verify that the functionality of each matching cone is equivalent. Many solver (algorithm) technologies are available to prove the equivalence of logic cones: Formality EC uses SAT, BDD, Isomorphism, ATPG, and Arithmetic, among others. Once the verification step is completed, the tool produces a list of any compare points (logic cones) that are not equivalent. Formality EC also provides various debug and isolation capabilities to help isolate the implementation error.

## Equivalence Checking in FPGAs

In an FPGA flow, verification challenges result from transformations during design implementation. Synthesis, place-and-route, and other tools in the design flow can cause many types of design transformations, such as combinatorial reductions, sequential optimizations (retiming), FSM re-encoding, register merging, or duplication, as well as other place-and-route optimizations.

If your EC tools are not set up to account for these transformations, verification becomes cumbersome. Formality EC accounts for the transformations performed by synthesis and Xilinx ISE (Integrated Software Environment) tools (Map and PAR) through use of the following files and utilities:

- **Verification libraries:** Formality-specific models for Unified Simulation (UNISIMS) components and post-PAR Simulation components (SIMPRIMS)
- **Constraint file(s):** to inform Formality EC of the synthesis tool's register-merging (if enabled) and Mapper optimizations relating to:
  - – registers that turned to constant
  - – ports that were optimized away
  - – ports whose direction changed
- **Netlist:** a Formality-compatible, gate-level netlist.

In traditional FPGA design flows, simulation is used to validate the functionality of the gate-level netlist produced by synthesis and PAR tools. In modern flows, simulation is replaced by equivalence checking (Figure 3).

## RTL to Post-Synthesis Verification

You can use a number of synthesis tools to optimize designs during RTL operations. Xilinx supports the following synthesis tools for its Virtex and Spartan™ FPGAs:

- FPGA Express and FPGA Compiler II (FCII) from Synopsys
- SynplifyPro from Synplicity
- LeonardoSpectrum from Exemplar
- XST (Xilinx Synthesis Technology) from Xilinx.

Each synthesis tool employs its own combinatorial and sequential optimization, as well as retiming (if available) algorithms. Although the Xilinx/Formality flow as depicted in our model was validated using Synopsys FCII, the flow should work similarly with other synthesis tools.

### Creating the Post-Synthesis Gate-Level Netlist

The Synopsys FCII post-synthesis gate-level netlist contains UNISIMS components and is in EDIF (Electronic Design Interchange Format). The EDIF netlist is fed into Xilinx ISE for mapping and PAR. The UNISIMS components are LUTs, flip-flops, I/O buffers, and other available resources in the targeted Xilinx architecture. Xilinx ISE provides the capability to generate a Verilog™ netlist at any stage in the implementation process.

We chose the Verilog post-synthesis netlist because Verilog netlists are commonplace and are easily read into Formality EC. We then created a Formality-compatible netlist using the following methodology:

- Read the design and the CORE Generator's™ EDIF netlists into ISE using NGDBUILD. This step transforms the EDIF netlist(s) into the Xilinx database format. The CORE Generator block will be covered in a later section.
- Create a Verilog netlist containing SIMPRIMS components using the NGD2VER program in ISE.
- Process this netlist using the xilinx2formality.pl Perl script to generate a Formality-compatible netlist.

The post-NGDBUILD netlist represents the result of two transformations: synthesis and NGDBUILD. Because the netlist contains non-synthesizable constructs and "defparam" statements that cannot be read directly into Formality EC, Xilinx and Synopsys developed the xilinx2formality.pl Perl script to process the post-NGDBUILD netlist into a usable format (Figure 3). Future improvements will enable Formality EC to read the Verilog netlist generated from the ISE environment directly.

### UNISIMS and SIMPRIMS Libraries for Formality EC

Two special Xilinx verification libraries are needed for use with Formality EC:

- **UNISIMS:** The UNISIMS library contains the Xilinx primitives in RTL format. This library is required when the design contains Xilinx primitives, such as an instantiation of a DCM or block RAM.
- **SIMPRIMS:** The SIMPRIMS library contains the Xilinx primitives for back-annotated verification (Post-NGDBUILD, Post-MAP, Post-PAR).

These libraries must be read into their respective RTL and post-NGD containers within Formality EC during the design read stage. Xilinx provides specific unisims.fms and simprims.fms scripts to read the necessary models into Formality EC. Currently, the scripts read in the entire libraries. Synopsys is working with Xilinx to utilize Formality's read-library-on-demand feature – which will eliminate the need to read the entire UNISIMS and SIMPRIMS libraries and read only the components actually used in the design.

### Reading CORGEN Models

Xilinx provides a comprehensive set of IP (intellectual property) blocks through the CORE Generator tool. These blocks, which range from simple shift registers and memories to complex Reed-Solomon encoder/decoder blocks, can be customized. The CORE Generator software generates all the necessary models for the customized IP blocks, including a behavioral model for simulation and an EDIF structural netlist with UNISIMS components. Together, these elements represent the optimum implementation of the IP block using the available resources on the targeted Xilinx FPGA architecture. You can instantiate these IP blocks as black boxes in your RTL code.

FCII generates an EDIF netlist containing the black boxes. NGDBUILD then uses the optimized structural EDIF representation of the blocks to fill the black boxes in the post-FCII EDIF netlist. The post-NGBUILD Verilog netlist, created using SIMPRIMS, contains the complete structural representation of the design, including the content of CORE Generator blocks. During RTL to post-NGDBUILD verification, Formality EC needs the functional model for a given IP block in the RTL to match it with the post-NGDBUILD netlist. For this, Xilinx provides core2formal, a Perl script that reads in the UNISIMS-based EDIF structural netlist for the IP block. This creates a Formality-compatible SIMPRIMS-based Verilog netlist. The SIMPRIMS-based netlist is the functional model that Formality EC uses to verify the CORE Generator blocks (Figure 3).

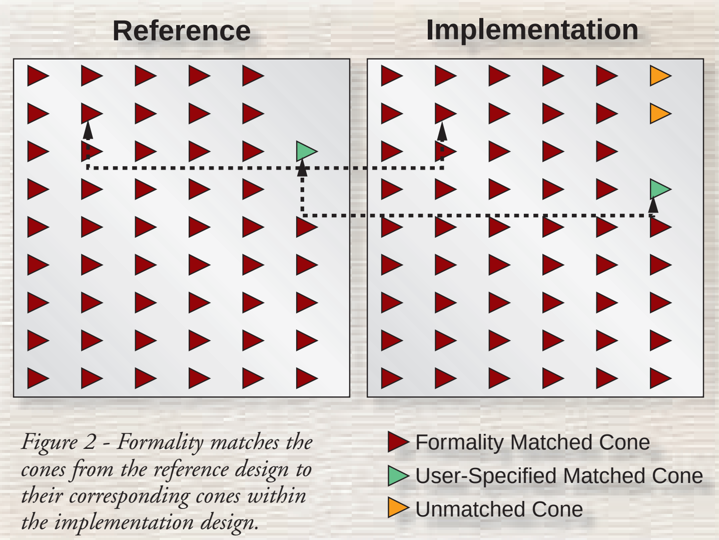


*Figure 2 - Formality matches the cones from the reference design to their corresponding cones within the implementation design.*

### Performing the Verification

The RTL2postNGDBUILD equivalence-checking flow is easiest when FCII synthesizes the design without using the following optimization options: register-merging, max fanout control (register duplication), and register retiming. However, without these optimizations, QoR (Quality of Results) may be compromised. Therefore, handling these transformations in an equivalence-checking flow requires some additional consideration.

For the register-merging option (on by default), Synopsys developed the makeconstraints.sh script. The script reads the FCII-generated report, which details the list of merged registers, and then produces a Formality set_constraint command file. This command file is then read into Formality EC prior to verification.

Formality EC offers a special feature for handling max fanout control using the register duplication option (off by default): To handle the transformation automatically, enable the verification_merged_duplicated_registers variable in Formality EC prior to verification.

When a design is synthesized with retiming, verification becomes more difficult. Formality EC supports sequential optimizations (such as retiming) when localized or limited to a block, but FCII generally performs retiming on an entire design. To perform a successful verification with such optimizations, the command set_parameter–retimed must be used on all blocks that have undergone retiming. If you're planning to use Formality EC, use retiming sparingly in FCII.

### RTL to Post-PAR Verification

Figure 3 illustrates the transformations that ISE applies to a synthesized netlist:

- **NGDBUILD:** Transforms the EDIF netlist(s) into Xilinx database format.
- **Map:** Packages the LUTs, flip-flops, SelectRAM, and other resources in the design into CLBs (configurable logic blocks), IOBs (input/output blocks), and so forth. Using the state-of-the-art Xilinx Mapper, you can apply certain transformations to the design, such as optimizing away constant registers, optimizing away ports that are no longer needed, and changing the direction of ports from bi-directional to output if warranted.
- **Place-and-Route (PAR):** PAR is the last step in implementing the design before creating the bitstream to program the Xilinx FPGA.

### Creating the Post-PAR Netlist

After PAR, a SIMPRIMS-based Verilog simulation netlist is created using NGD2VER, as shown in Figure 3. In the Xilinx design flow, this netlist, along with its accompanying SDF file, is used in functional and timing simulation to verify design integrity after Map and PAR. The same netlist, processed with the xilinx2formality.pl script, is read into Formality EC for functional verification.

### Performing the Verification

Before the RTL to post-PAR verification with Formality EC can be completed successfully, you must examine the Mapper's optimization of constant registers and some ports. Depending upon the target FPGA architecture and design constraints, the Xilinx Mapper uses special algorithms to identify:

- Registers that can be changed to a constant
- Ports that can be optimized away
- Bidirectional ports that can be changed to output only.

The Mapper performs these optimizations and records the result in the Mapper report.

These transformations must be accounted for during verification. The xilinx2formality.pl script reads the information relating to these optimizations from the placed-and-routed design database to produce a Formality constraint file. Reading this constraint file prior to verification enables Formality EC to account for these transformations.

### Conclusion

Effective verification of today's large, complex FPGAs requires a static verification flow. Xilinx and Synopsys have created a solution that uses the Formality equivalence checker to provide a fast, thorough functional verification methodology. You can benefit from this flow today using existing implementation technology. Synopsys is currently developing an improved, streamlined verification flow to handle next-generation FPGA implementation technologies.

Xilinx provides a comprehensive FAQ, application notes, and updated information for the Xilinx/Formality EC flow. Go to *http://support.xilinx.com/company/search.htm* and search for "Formality."

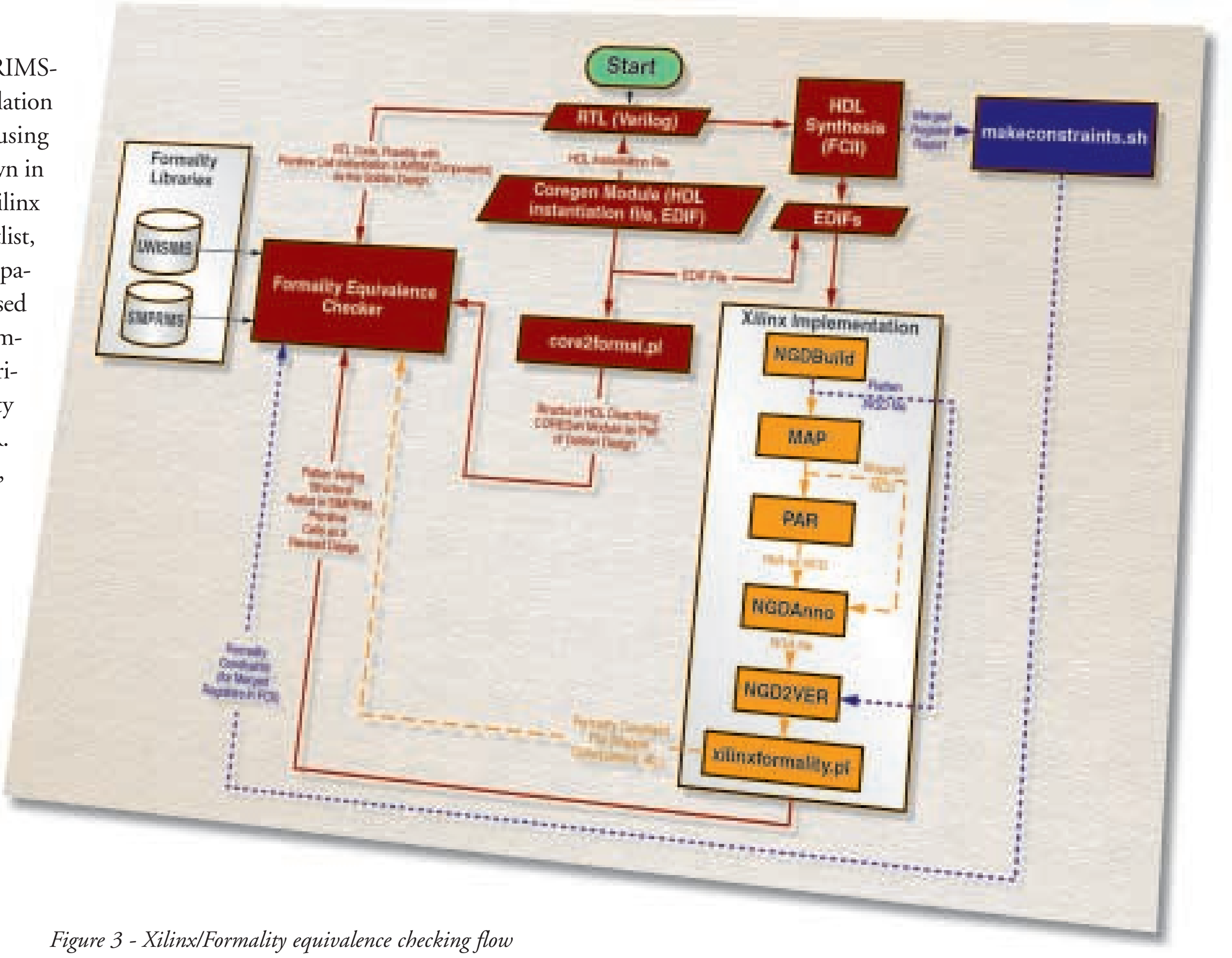


*Figure 3 - Xilinx/Formality equivalence checking flow*