Practical Challenges for Industrial Formal Verification Tools

F. Erich Marschner
COMPASS Design Automation
5457 Twin Knolls Road, Suite 100
Columbia MD 21045, U.S.A.
Email: erich@compass-da.com

Formal verification of hardware design is rapidly becoming accepted as an alternative to the traditional strategy of verification via simulation. However, there are still major barriers to widespread acceptance of formal verification tools and methodologies. These barriers generally have very little to do with the robustness of the theory underlying formal verification; instead they are much more a function of the degree to which the formalism has been adapted to address practical concerns.

First of all, formal verification tools must be applicable to typical design styles. Altough simplifying assumptions may make a tool easier to build or give it better performance, they may also restrict a tool to the point where it is not useful enough to justify its expense. Even though synchronous design is often the rule, asynchronicity is nonetheless a universal exception. Similarly, while single-clock systems exist, multiple-clock systems are more and more common, especially in certain industry segments. A successful formal verification tool must be flexible enough to support the exceptional cases as well as the typical cases, otherwise it will not find a place in the mainstream design flow.

Equally important is the requirement to support standard interfaces, to fit into existing design flows. In the hardware design world today, this usually means support for the two IEEE standard hardware description languages, VHDL and Verilog - but it may also involve support for related standards, such as IEEE Std 1164 (standard multi-valued logic for VHDL), and IEEE Stds 1076.3 (Synthesis packages) and 1076.4 (VITAL packages for timing-accurate modeling of primitives). Support for pseudo-standards is also an issue, such as the de-facto standards created by the dominant vendors in the Electronic Design Automation (EDA) industry, or the library formats of the major semiconductor manufacturers. This is complicated by the fact that such languages and formats are not always well-defined - even those that are IEEE standards! Even so, a usable formal verification tool must be able to work with them.

Capacity and performance are always major issues in the EDA industry. The fact that formal verification technology provides a more robust verification strategy than simulation does not matter to a design manager if his designs regularly exceed the size or complexity that such tools can handle. And the technical challenges of handling various kinds of circuits are no excuse; the designer needs results no matter what the situation. Multiplication may be difficult to verify formally, but this just reinforces the need for a better solution. Use of hierarchy may be recommended for a formal verification methodology, but it may not be

possible in a given environment. A good formal verification tool must adapt to the designer's requirements as much as possible, and impose as few constraints as possible on the design process.

The biggest challenge for formal verification tools is providing good diagnostic information when problems are detected. Few hardware designers want to become experts in formal verification; they typically want to focus on solving the design problems instead. As a result, it is imperative that formal verification tools require the minimum possible input other than the design itself, and generate the most insightful reports possible to highlight potential errors. The fact that a formal verification tool cannot read the designer's mind does not lessen the demand for such a tool, or the expectations of evaluators of such tools.

Even the best formal verification tool cannot be successful until it has been adopted by a user community, and that involves a major barrier: fitting into an existing "design flow", or network of inter-operating design tools that has been shown to support the design process at least moderately well. The bulk of EDA tool sales involve replacement of tools already in the customer's design flow with ostensibly better tools that perform the same functions. But formal verification tools are not already present in most design flows, so adoption of a formal verification tool requires a much more ambitious change, a change to the design flow itself. While swapping individual tools within a flow involves nearly zero risk - one can always revert to the old tool, after all - adopting a new design flow can be much more dangerous, and design managers tend to avoid such risks for as long as possible.

Ultimately, the major barrier to the adoption of formal verification technology for hardware design is the difficulty of conveying to potential users the concept of formal verification as it applies to their particular needs. Every design team has different requirements, and every team uses somewhat different terminology to describe what they do. While formal verification tools are inherently general purpose, and thus can be adapted to meet various requirements, this in itself can make them difficult to grasp; design managers seem to feel more comfortable with focused tools tuned to specific applications. At the same time, it is economically infeasible to develop different tools for each application, because every design team has a unique application in mind.

Formal verification of hardware design has become a reality and will become a more and more essential part of the design process as advancing technology continues to enable more and more complex designs. It is certainly true that there are still open problems to address in the fundamental theory underlying formal verification methods. At the same time, there is just as much work or more to be done in the practical application of formal techniques. And unlike the theory, which has been under development for decades, this work is only just beginning.