



# Imagining Verification Success

**Rajiv Kumar**

*Vice President of Engineering*

*Real Intent, Inc., Sunnyvale, CA, USA*

## Notice of Copyright

This material is protected under the copyright laws of the U.S. and other countries and any uses not in conformity with the copyright laws are prohibited. Copyright for this document is held by the creator — authors and sponsoring organizations — of the material, all rights reserved.

# DESIGN AUTOMATION CONFERENCE

## VIEWPOINT: Verification

# Imagining Verification Success

**Rajiv Kumar**

*Vice President of Engineering  
Real Intent, Inc., Sunnyvale, CA, USA*

**Abstract**— EDA developers need to have very active imaginations. They need to imagine becoming their own end users. Sometimes they need to become the designer, sometimes the verification engineer or perhaps even the design manager. This role play is essential to creating industry-leading tools that will be embraced by designers instead of becoming one-time wonders. For an EDA tool to earn an honored place in the designer's tool chest, it needs to have a very high usability quotient, and this can be achieved only with the proper role play by developers.

## I. The Specification Gap in EDA Tool Development

The EDA developer community has always been expected by the designer community to deliver great tools that will shorten design cycles. Most of the time, this expectation has translated into very clear goals such as creating tools that can do designs at smaller design nodes, tools that can deliver higher QoR, tools that parallelize algorithms for higher performance, etc. Such expectations placed on the EDA community are usually easy to specify and understand. EDA developers can easily sink their collective teeth into these challenges and deliver. On the other hand, expectations for providing greater usability, debuggability and other human-level metrics, though just as important to shrinking the design cycle, are very seldom clearly stated. For EDA developers, this part of the tool development remains an unspecified domain, and its evolution relies by and large on the developer's own imagination.

## II. Imagination to the Rescue

EDA developers need to have a very active imagination. They need to imagine becoming their own end users. Sometimes they may become the designer, sometimes the verification engineer or perhaps even the design manager. Each of these customers has different expectations. A verification engineer wants tools that will detect complex bugs and help efficiently diagnose the bugs found. For example, a verification engineer who is focused on verifying a priority arbitration protocol manager may want to use smart formal property checkers that will statically detect corner-case violations and find deep traces that might get missed in simulation. On the other hand, a design manager who is responsible for the tapeout may only be interested in tracking the verification closure, and wants actionable intelligence to assign verification resources to problem areas. EDA tool providers need to create tools that address these competing requirements, and an EDA developer needs to clearly understand the nuances of creating the tools that meet those expectations. But most likely, a developer has neither been a verification engineer or a design manager. She only has her imagination to rely on.

## III. Challenges in Building Verification Tools

Expectations are very high for verification tools. A good verification tool must not only find design-rule violations, but also lead the user to the origin of the violations and help debug it. The debug flow is where creativity on the part of the developer – in terms of how to lead the user intuitively through the mass of data – becomes essential.

Let us examine the case of finding and fixing metastability errors in the design due to multiple asynchronous clock domains. This problem was not a serious verification challenge even a few years ago, and designers could manage it through naming conventions and/or design reviews. Today's aggressive low-power designs, made essential by consumer applications, have turned this problem into a mainstream verification problem that almost every designer must deal with. And, it is not a trivial problem, either. While the basic 2-flop synchronizer cell has been in use for many years, creative designers have, over time, crafted numerous innovative and aggressive synchronization schemes to move data safely across asynchronous clock boundaries, including the use of two-or-more flop synchronizers, handshake-based data transfer, Gray coding, FIFOs, reconfigurable interfaces, and

more. It is typical for a variety of these schemes to be in use on the same chip. Designers expect verification tools to work on their chip, no matter the complexity.

Just as it has done before, the EDA community has responded to this new verification challenge proactively and created a class of tools called Clock Domain Checking (CDC) tools for this specific problem. At Real Intent, for example, our engineers looked at this problem from first principles, and by listening to the design community understood the various synchronization schemes that are deployed to move data and control signals safely across the asynchronous clock boundaries. The next step was for our developers to map this problem into an error model that could then be analyzed using both structural and formal algorithms.

Throughout this process, as tools were augmented to handle more and more complexity, the verification problem was analytically clear and specific, but the “*usability design*” largely fell into the laps of the EDA tool developers. This is where their imagination was vital to the development of a successful tool. For example, the Meridian CDC tool from Real Intent has created a classification of CDC errors that came straight out of the developers’ hypotheses of what would make intuitive sense to the end user. Our developers became their own end users in order to imagine the classification that would allow users to focus on the issues without being “noisy”. Similarly, our developers, via their designer role play, understood that Meridian CDC results were going to be only as good as the clocking and reset environment with which the tool was run. So, they decided to embark on a major R&D effort to ensure that clocking- and reset-related issues are weeded out first before the actual CDC issues are examined. As yet another example, an active imagination led to a natural order of examining the tool’s reporting which led to minimal false alarms regarding the actual design issues. This greater usability has resulted in a faster adoption of Meridian by its users.

#### IV. The Role of Product Engineers vs. Developers

Creating highly valued verification tools is all about closing the gap between the designer’s expectation and the developer’s creativity. Product engineers are normally the ones that serve as the middlemen. Their role is to translate user requirements into a product spec from which the developers can work. But despite a high-level spec being made available, when developers run against algorithmic or engineering walls, they must make adjustments on the fly to bridge the gap between the spec and what is achievable. Here is also where their imagination, creativity and role-play come in handy.

#### V. Debug Flow Creates Special Challenges

A verification tool’s usage in a design flow can nominally be broken into three distinct phases: (a) Setup; (b) Analysis; and (c) Debug. Setup is required just once per design. The right set of tools (for example, tools that help with the clock and reset setup) can make this a relatively simple phase. Analysis is done within the tool and is generally highly, if not completely, automated. The analysis phase should be designed to minimize the need for much user attention beyond the tracking of benchmark metrics such as performance, memory size, etc.

It is the debug phase where users spend most of their bandwidth. They must examine the output of the tool, combine their design knowledge with the tool’s analysis data, and quickly identify and repair the source of any detected issues. To accurately capture and automate this flow, developers must imagine how the users are going to interact with their tool. For example, does the tool present information in the proper terms and conventions of the language in use? Is the debugging output

organized consistently with the design structure? Is the tool effective at propagating bugs to observable points in the design? Is the debug environment able to reconstruct the faulty effect easily under user control? Effective organization of the output is essential to enable a user to view the results in ways that can be easily internalized. By a large margin, debug is the major factor in determining a verification tool's usability. An accurate understanding of the designer's desires and needs is a must in order to determine the most effective way of organizing the output in a clear and logical fashion. Developer imagination is a key part of this effort, as is real customer feedback to gauge how effectively the goal is being reached. Despite the availability of dedicated tools and methodologies for verification, users today are spending a lot of time tracking down bugs that should have been easily caught and debugged. Sometimes, the only difference between failure and success is just a little imagination on the part of the developer.

## VI. Conclusion

Building a great EDA tool demands creativity. Hard as it is to build the algorithmic engines that sit at the heart of such tools, building the level of usability that enables tools to become the preferred choice for design and verification engineers is even harder, and requires considerable imagination.