

nce upon a time, you verified a logic design for an FPGA by compiling it, loading it, and pushing the reset button on your evaluation board. But, as FPGAs have become larger, this “blow-and-go” verification style, as Xilinx’s director of software-product marketing, Hitesh Patel, terms it, has become counterproductive. The odds of creating a multi-million-gate design so close to perfection that

you could debug it from the pins on the package are vanishingly small. So, design teams have begun to employ software-based simulation of the design, much as ASIC teams have done for years.

VERIFYING FPGA DESIGNS: SIMULATE, EMULATE, OR HOPE FOR THE BEST?

SIMULATION IS A FACT OF LIFE FOR MANY FPGA USERS TODAY. BUT WHEN IS IT TIME TO STOP SIMULATING AND JUST DROP THE DESIGN INTO THE CHIP?

BY RON WILSON • EXECUTIVE EDITOR

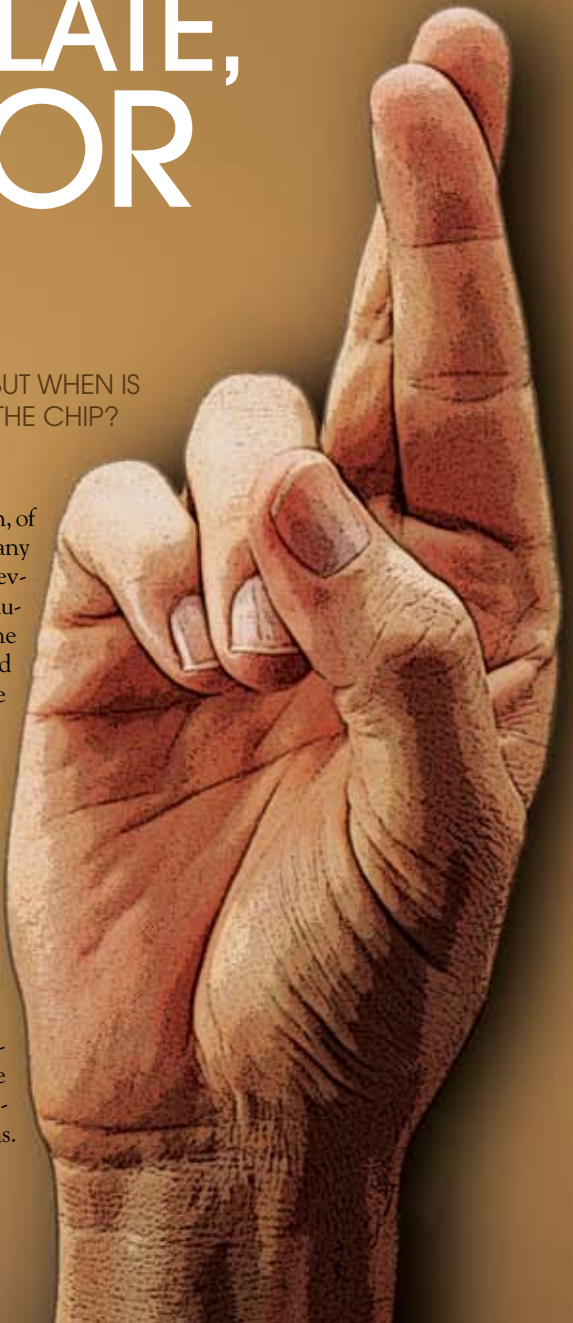
But this approach raises a series of important questions: Should the role of simulation in an FPGA design be the same as it is in an ASIC design? Should the verification team still, at some point, just put the design in the target FPGA and start testing it at speed? If so, when is that point? To find out what design teams are doing today, we asked some of the people who work most closely with FPGA users. And, for reference, we asked a few ASIC design teams who use FPGA prototypes in their verification processes for their views, as well.

PROS AND CONS

Most people begin discussing the question of verification flow by assessing the relative strengths and weaknesses of simulation versus in-FPGA verification. At the risk of boring the experienced, this article will follow the same formula.

The huge advantage of simulation, of course, is access. You can observe any signal in an RTL (register-transfer-level) design down to clock-cycle resolution. You can control the state of the design to whatever degree you find worthwhile. Your knowledge of the RTL and your skill with the simulation environment are the only limits to observability and controllability. You can work interactively on limited areas of a design, or you can set up grand experiments that may run for days. And the relatively speedy setup of simulation runs makes it possible to quickly try lots of things.

Another advantage of simulation is that most simulation environments today are friendly to the use of OVL (Open Verification Library) or SystemVerilog assertions.



Often, there is a straightforward way to transport these assertions into the simulation environment. As assertion-based verification becomes more common, this issue becomes important. In addition, the simulation environment allows you to keep your provisions for stimulating and observing the design separate from the design itself. This consideration might seem secondary, but it can become important in preserving the integrity of the design through intensive verification work.

But simulation is slow. “If you are doing 2 [million]- or 3 million-gate blocks, simulation is great,” says Lauro Rizzatti, vice president of marketing at hardware-emulation vendor Eve. “But, at the multiple-block level, simulation can slow down until it is no longer useful.”

The complexity of the design is not the only limitation. Phil Simpson, senior manager of technical marketing at Altera, points out that, if the design inherently requires huge amounts of data for validation, simulation may become impractical even at the block level. He offers the example of a video codec, which has so much internal state that a bug may appear only in the middle of a 15-minute video clip. But simulating the compression and decompression of 15 minutes of high-definition video could take most of a career.

THE IN-CIRCUIT ARGUMENT

The strengths and weaknesses of in-FPGA verification precisely complement those of simulation. To begin with the obvious, the FPGA is fast. Often, you can run the design at full speed, although, in some cases, this approach may mean more fussing with timing closure than you would want to do early in the design. And, unlike simulation, the FPGA doesn't necessarily slow down as you integrate more blocks into the design. So, it becomes possible to test the entire design rather than individual blocks and to run tests with large real-life data sets instead of surgically crafted test cases.

Because of the speed and the fact that the FPGA has the actual I/O cells that the design requires, you can also test the design in-system: either in an FPGA-development board that you have lashed into the target system or in the target PCB (printed-circuit board) if it is ready to go. Such testing eliminates the lingering uncertainty about whether the test cases really reflect the operating environment of

AT A GLANCE

- ▶ Large-FPGA designs require an ASIC-like design flow.
- ▶ Blending simulation and FPGA-based emulation in a verification flow is necessary.
- ▶ There are no established guidelines for blending simulation.
- ▶ A quick survey suggests a consensus on a verification flow for advanced-FPGA designs.

the design. Also, testing the design in its actual board can uncover I/O-related issues—electrical problems, signal-integrity issues, or incompatibilities in high-speed serial protocols, for example—that would be virtually undetectable in any other way. And in-system test creates a software-test platform as a side benefit.

These benefits all pertain to system-level verification. But Altera's Simpson points out that there are some interesting advantages for in-chip debugging of blocks, as well. “Once you get a block into the FPGA, you can use an embedded-processor core, such as Nios, to assist in the debug process,” Simpson observes. “The core can move test data on and off the chip, for instance, and it can sequence the tests. In this way, you can test a block in isolation before the circuitry around it is ready.

“In our own IP [intellectual-property]-development group, we have written transactors to run on a Nios core to generate pseudorandom tests,” Simpson continues. “I don't know that this practice is common among customers yet, but it can be quite valuable.”

With all the advantages of FPGAs, you might wonder what's wrong with just dropping a newly coded core into the FPGA, coding up a test fixture around it, and starting to test. The answer to that question lies in the FPGA's disadvantages.

FPGA WEAKNESSES

The first and most obvious problem is visibility. In principle, every logic element in the FPGA is visible through the chips' debugging interface. But—rather surprisingly, given the power the internal debugging ports offer—vendors estimate that only about half of FPGA users actually synthesize the debugging interfaces into their designs and use them for verification. Xilinx's Patel thinks that num-

ber may be growing as FPGAs get larger.

So, for the most part, if you want to observe a signal in your design, you have to route it out to a pin so that you can put a logic analyzer on it. Given the nature of logic analyzers, you may have to route out a number of other signals, such as internal clocks, as well. This approach means extra work, and it may mean having to reduce the clock frequency on the FPGA if you are trying to observe a fast signal that's nowhere near the I/O blocks. Hence, some managers say that it is vital to include your requirements for observability of FPGA signals in your original verification plan.

The added design work to get access to the signal is one disadvantage. But another problem with stimulating and observing internal nodes in the chip is that it requires you to change the design, rebuild, and resynthesize, risking a clean separation between the design and

AS THE USE OF ASSERTION-BASED VERIFICATION GROWS IN THE SIMULATION WORLD, THE SITUATION CRIES OUT FOR A SIMILAR ASSERTION-BASED TOOL ON THE FPGA SIDE.



the testbench. Without careful separation of debugging code from design code and fanatical version control, you can lose track of these changes, risking the equivalent of leaving surgical tools inside the patient.

Then, there is the disadvantage of setup time. Synthesis times for large designs are not trivial, and the time to insert instrumentation, rebuild, resynthesize, and remap the design can become a factor in whether to perform a particular experiment. Incremental-synthesis tools can help here, but a 20 million-gate design can mean an overnight build-and-synthesis process.

Finally, there is the problem of transporting the testbench from the simulation environment to the FPGA environment. Stimulating a block now requires circuitry instead of a set of simulation commands. Observing a node requires circuitry and physical instruments, not just commands. And no one seems to have developed a way of systematically moving assertions from the simulation environment to the FPGA, despite the growing acceptance of assertion-based verification. "There is no solution today

SHINING SOME LIGHT ON THE COVERAGE GAPS

Everyone is in favor of the speed of in-FPGA emulation. But the difficulty of setting up, controlling, and observing experiments in an FPGA often forces laborious and time-consuming tests back into the simulation environment. In an ideal world, someone would put together a verification platform that combined the execution speed of FPGAs with the easy setup and excellent access of simulation. Not surprisingly, some vendors have targeted this ideal.

The first efforts, dating back to early in the ASIC era, were "big-iron" logic-emulation systems. These systems are, in effect, specialized mainframe computers in which either custom microprocessors or custom programmable-logic devices simulate or emulate, respectively, the operation of the logic. A representative of such systems would be the

Cadence Palladium. The systems offer many times the execution speed of simulation, with what the vendors would argue is at least equal access to the design under test. But they are limited in their capacity to not much more than the size block that you conveniently simulate—unless, that is, you have an impressive capital budget. These systems are major capital investments and therefore beyond the range of most design teams targeting FPGAs for the final design.

In recent years, a number of systems have entered the market—from companies such as Eve—that perform logic emulation in lean environments using commercial FPGAs. Such systems vary in personality from being mini-mainframe-emulation systems to being basically FPGA-evaluation boards with supporting debugging software. In all cases, the attempt is to

provide an FPGA-execution environment with less logic overhead in the design than would be the case with a big-iron emulation system. Because of their lower overhead, the FPGA-based systems can often run one to a few orders of magnitude faster than the mainframe-emulation systems. In general, the faster they run, the less convenience of simulation they can preserve. But they run into limitations when a design, including the debugging overhead, is too large for a single FPGA. Partitioning the design is complex and often involves multiplexing signals between FPGAs, which slows everything down.

These systems do offer the software support necessary to move testbenches and data back and forth between the FPGA system and the simulation environment. Eve, for one, reports being at work on a

way to import assertions into its environment, as well.

An interesting variation on this theme is the GateRocket system, which the company positions as either a simulation accelerator or an in-circuit emulator. As a simulator accelerator, the system attempts to just slip into the user's simulation environment, accelerating simulation of time-consuming pieces of the RTL (register-transfer-level) logic without disturbing the features of the environment. Assuming the 90/10 rule—that 90% of the simulation time goes into 10% of the code—this ability allows verification engineers to continue using the simulation environment further into the verification flow than would be practical without acceleration. GateRocket claims to support what it calls a synthesizable subset of assertions.

for automatically moving assertions to the FPGA, but we are getting more requests for this capability,” Simpson says.

Another weakness here is coverage metrics. Although simulation environments are developing sophisticated tools to measure verification coverage and fuse measurements from different kinds of tools, the notion of coverage barely exists in the FPGA world, and there are no established tools for measuring test coverage of a design and reporting that data back to a central coverage-closure system.

WATCH THE ASIC TEAMS

So there, in a nutshell, are the advantages and disadvantages of each approach. Given that information, how do experienced ASIC design teams—who often employ FPGAs during their own verification flows—balance simulation and FPGA-based testing?

One answer comes from video-processor vendor Ambarella. Executive Vice President Didier LeGall says, “Mostly, we do not use FPGA emulation at all. Our experience has been that, for emulation to work, you need to have very mature RTL. But, by that stage in the flow, [the process of] getting the design into an FPGA and setting up a testbench is a lot of work with little return.”

The application may condition LeGall’s view. Ambarella SOCs (systems on chips) process high-definition video and 10M-pixel still images at high frame rates and require fast internal clocks and complex algorithms. But LeGall follows his comments on FPGA emulation with an interesting point about the objectives of the whole verification process. “The secret of first-time-working ICs is not perfect verification,” LeGall says. “It is software”: That is, know where the risk areas are in your design, and plan for software workarounds from the beginning, not as afterthoughts. This strategy does reduce the value of much of the information that verification engineers can glean from extensive FPGA-based testing.

Another view comes from LSI Corp’s storage-components group. Bill Wuertz, vice president and general manager of the group, describes how that team does SCSI (small-computer-system-interface) and SAS (serial-attached-SCSI) controllers.

Wuertz says that LSI uses a nearly parallel process, with one verification team

working in simulation with one set of objectives while another team works on FPGAs with a different set of objectives. “Early in the design, we create a step we call trial RTL,” Wuertz says. “This is the first point where we feel the RTL is basically functionally correct and that the blocks are connected to each other. At this stage, the verification splits into two tracks. The simulation team compiles the design for their tools and goes to work on the individual blocks. A separate team, the systems-engineering group, synthesizes the RTL for an internally developed FPGA board—we are

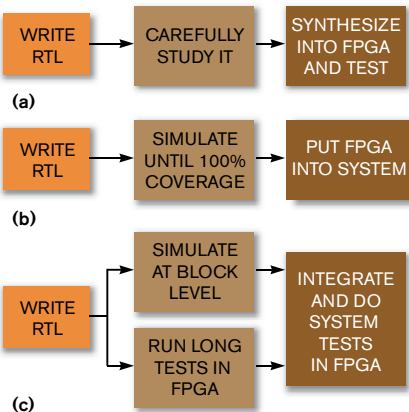


Figure 1 Three approaches to blending simulation with in-circuit debugging show the variety of verification flows teams are using today: traditional FPGA user (a), traditional ASIC designer (b), and the emerging blended approach (c).

on our fifth generation of the board design now—and begins exhaustive stress testing at the systems level.”

As Wuertz describes it, the two groups have different purposes. The simulation team is trying to ensure that the circuitry is correct. The systems team is mostly unconcerned with the circuitry but is verifying that the chip works in the incredibly varied and complex environments of storage networks. Wuertz says that the FPGA prototype may run multiday tests connected to a room full of disk and tape drives. “These tests have evolved over 20 years,” he says. “We’ve learned that it can take very long tests with a mix of different disk drives and tape units to generate just the odd timing alignment that will break the design.”

LSI has worked out its own internal tools for linking the two environments. These tools allow the system team to

capture a trace leading up to a failure and convert that data to a stimulus file for the simulation team, for example. Conversely, the simulation team can alert the system team to any risk areas it sees in the design. Establishing communications between the two verification teams in their different environments has been crucial to LSI's two-track approach. The two exchange data throughout the process, and, in the end, the design manager requires closure from both teams.

A CONSENSUS METHODOLOGY

From discussions with FPGA vendors and users, we can see a consensus on a verification flow that blends simulation with emulation (**Figure 1**). Such a flow begins with block-level simulation of the design—not the exhaustive, strive-for-perfection simulation of ASIC heritage, but more of a reality check. The objective is to verify that the block is functioning, that it is doing more or less the right thing at its pins, and that it can meet timing on an FPGA in a lab environment.

At this point, many teams move a version of the block to the FPGA and begin more exhaustive in-circuit testing. This case holds particularly true if the block, such as a video codec, requires long streams of high-speed data to verify correct functioning or if the block includes high-speed-I/O functions. In other cases, simulation work continues on the blocks until all are ready for integration.

Consensus suggests that, when the team begins to integrate blocks—the trial-system build—the FPGA really comes into its own. Here, the design may simply be too large for fast simulation. Or, with the known-working blocks, it may be more productive to troubleshoot integration issues on the FPGA than on the simulator.

But the consensus also suggests that the move from simulation to emulation is not a single irreversible step. Simulation work continues during system emulation, just as it runs in parallel with software development. And most teams use the FPGA emulation to capture and iso-

FOR MORE INFORMATION

Altera
www.altera.com

Ambarella
www.ambarella.com

Cadence
www.cadence.com

Eve
www.eve-team.com

GateRocket
www.gaterocket.com

LSI
www.lsi.com

Xilinx
www.xilinx.com

late bugs and then pass them back to the simulation team for diagnosis. It is just too painful to do detailed diagnosis on the FPGA.

That summary describes what appears to be happening today, and it points out several serious weak spots in the method. First, it is difficult to move testbench data back and forth between environments. There seems to be no automatic mapping from simulation directives that create a test to FPGA structures that implement the same test. Second, embedded-RISC cores, available from all major FPGA vendors, appear to be a vastly underused resource—able to manage data and control tests but isolated from the simulation testbench. In principle, the simulation team could move its testbench into C code for the embedded core, rather than into RTL for the FPGA. Third, there is no simple path from the data a team collects in an FPGA experiment back to the simulation bench. And, finally, as the use of assertion-based verification grows in the simulation world, the situation cries out for a similar assertion-based tool on the FPGA side.

One indication that these issues are valid is that vendors selling FPGA-based emulation systems address each of them (see **sidebar** “Shining same light on the coverage gaps”). Examples include systems from Eve; simulation accelerators, such as GateRocket; and “big-iron” emulation boxes, such as Cadence's Palladium. Whether this infrastructure will evolve for the ad hoc board-level emulations typical of the FPGA-verification world or whether it will remain the differentiation of big-ticket simulation accelerators and emulation systems remains to be seen. **EDN**

➤ Go to www.edn.com/090219df and click on Feedback Loop to post a comment on this article.

➤ For more technical articles, go to www.edn.com/features.

You can reach Executive Editor **Ronald Wilson** at 1-408-345-4427 and ronald.wilson@reedbusiness.com.

