



# The FPGA industry

## STRIVES TO KEEP ON SCALING

Count Xilinx's chief technology officer among the pioneers forging a path toward the next process node.

SENIOR TECHNOLOGY LEADERS in FPGA companies occupy a unique role in the engineering community. As developers of a ubiquitous means of implementing digital circuits, FPGAs have become virtually the lingua franca of digital design in some applications. Just as important, as the earliest adopters of new digital-CMOS processes, the SRAM-based FPGA vendors are pioneers, among the first to scale mountains, survey gorges, and categorize the wild creatures in the new country of the next process node. So, it was with considerable interest that *EDN* spoke with Ivo Bolsens, chief technology officer at Xilinx.

➤ **What do you see as the most important question facing FPGA developers and users today?**

The same question faces all semiconductor companies: How do we keep on scaling? Today, this question isn't so much about process technology. At Xilinx, we see at least another couple of process generations on our road map. The real question is the scalability of the design process itself: the gap between understanding the problem and producing the implementation.

➤ **And what are your thoughts on this question now?**

The most critical thing today is for us to invest in usability. When you create an implementation in an FPGA today, the search space is enormous; you have millions of LUTs [look-up tables] available,

and that number keeps increasing. We have to shield users from irrelevant variables and let them focus on their functionality. Otherwise, there are not enough field-applications engineers on the planet to allow an FPGA company to grow substantially further.

Our latest devices have 1.2 billion transistors. To put that number in perspective, we will be able to put a few thousand MicroBlaze CPUs in a 32-nm FPGA. We solved the deep-submicron part of the problem by throwing transistors at it. So, you can assume abundance: You don't have to account for every LUT, and that makes the user's problem a little easier. If you can capture the complexity of the problem, you can map it onto a generic logic fabric.

That mapping is much

easier if there are templates—overlays that focus on specific sets of application challenges to limit the scope of the problem. For instance, a designer might have, in effect, the hardware equivalent of an API [application-programming interface] that implements functions for digital-signal-processing applications or for packet-processing applications. This would allow designers to think about their architectures in terms they already understand: FIFOs, queues, and so forth. In a way, these APIs would be the equivalent of reusable-code IP [intellectual property] in the software world.

➤ **So, you are using these templates to encapsulate the complexity of the FPGA and protect the user from it? This approach sounds almost like a software application-specific standard product.**

In a way, yes. The way we think about using FPGAs keeps evolving. In the beginning, you programmed in terms of switches and LUTs. Today, you think in terms of netlists and synthesis. Tomorrow, people will work as if they were programming a specialized CPU; they won't necessarily realize it's an FPGA down there at all. They won't have to know about LUTs

or deal with timing closure.

This will happen first in some very specific application domains, but it will spread gradually to more general applications. And the role of specialists—who understand switches, LUTs, and netlists but also understand the applications—will become the role of creating the APIs. This is very much the way the silicon-IP industry has evolved for ASICs. These IP houses will be the vendors of the templates for the users.

To achieve this, we have to find a way of abstracting hardware-level issues, such as timing analysis and functional debug. But here, too, we can borrow a lot of ideas from the software world.

And there will be costs. For us, this abstraction will mean more transistors. For some users, it will mean more constraints: more discipline imposed on the implementation process, such as, in effect, firewalls between functional blocks to make debugging feasible. In a way, this [approach] is similar to the concept of restrictive design rules we see in the SOC [system-on-chip] world, but it is about raising the level of abstraction, not about restricting the details of implementation.

—Ron Wilson