

Join the "Cool" Club

You're not doing ESL yet? Loser!



by Kevin Morris
Editor – *FPGA Journal*
Techfocus Media, Inc.
kevin@techfocusmedia.com
www.fpgajournal.com

When the *Xcell Journal* asked me to write a viewpoint on ESL, I thought, “Hey, why not? They’ve got ‘Journal’ in their name and I know all about ESL.” ESL is the design methodology of the future. ESL will revolutionize the electronics industry. ESL will generate billions of dollars worth of engineering productivity while the people and companies that develop the technology fight tooth and nail to capture infinitesimal percentages of that sum.

ESL (electronic system level [design]) brings with it a pretentious name and the potential for manic marketing misrepresentation. Although there are excellent ESL products and efforts underway, the early market is a good time for the buyer to beware and for savvy design teams to understand that they are likely to be experimenting with cutting-edge, nascent tech-

nology. ESL is also currently an over-broad term, encapsulating a number of interesting but diverse tools and technologies.

In order to understand ESL, we need to hack through the typical layers of marketing hype and confusion to get down to the fundamentals. ESL is not about languages or layers of abstraction. ESL is about productivity. Electronic hardware design (this is important because ESL is a hardware designer’s term) is far too inefficient. Forget the fact that we have boosted productivity an order of magnitude or two over the past 20 years because of technology advances in electronic design automation. Moore’s Law is a much harsher mistress than that. Do the math. In two decades, the gate count of the average digital design platform has risen by a factor of something like 2K. Even if EDA has managed a 200x improvement in gates per digital designer day, they are falling behind at a rate of more than 5x per decade.

This effect was observed ad-infinitum in practically every EDA PowerPoint presentation given in the 1990s. Logarithmic graphs of gate counts were shown skyrocketing

upward exponentially while designer productivity lounged along linearly, leaving what the EDA marketers proudly promoted as “The Gap.” This gap (skillfully highlighted and animated by black-belt marketing ninjas) was their battle cry – their call to arms. Without the help of their reasonably priced power tools, you and your poor, helpless electronic system design company would be gobbled up by The Gap, digested by your competitors who were, of course, immune to Moore’s Law because of the protective superpower shield of their benevolent EDA suppliers.

This decade, however, The Gap has gone into public retirement. Not because it isn’t still there – it is, growing as fast as ever. This pause is because we found a killer-app use for all of those un-designable gap gates, giving us a temporary, one-time reprieve from the doom of design tool deficiency. Our benefactor? Programmability. It has been widely documented that devices like FPGAs pay a steep price in transistor count for the privilege of programmability. Estimates run as high as 10x for the average area penalty imposed by programmable logic when compared with custom ASIC technologies. Dedicate 90% of



your logic fabric transistors to programmability, throw in a heaping helping of RAM for good measure, and you've chewed up enough superfluous silicon to leave a manageable design problem behind.

By filling several process nodes worth of new transistors with stuff you don't have to engineer, you've stayed on the productivity pace without having to reinvent yourself. Now, unfortunately, the bill is coming due again. FPGAs have become as complex as ASICs were not that long ago, and the delta is diminishing. Design teams around the world are discovering that designing millions of gates worth of actual FPGA logic using conventional RTL methodologies can be a huge task.

In an ASIC, however, this deficiency in design efficiency was masked by the mask. The cost and effort required to generate and verify ASIC mask sets far exceeded the engineering expense of designing an ASIC with RTL methodologies. Boosting the productivity of the digital designer would be a bonus for an ASIC company, but not a game-changing process improvement. Even if digital design time and cost went to zero, cutting-edge ASIC development would still be slow and expensive.

In FPGA design, there is no such problem to hide behind, however. Digital design is paramount on the critical path of most FPGA projects. Are you an FPGA designer? Try this test. Look outside your office door up and down the corridor. Do you see a bunch of offices or cubes filled with engineers using expensive design-for-manufacturing (DFM) software, correcting for optical proximity, talking about "rule decks" and sweating bullets about the prospect of their design going to "tapeout"? Not there, are they? Now look in a mirror. There is your company's critical path.

With ESL the buzzword-du-jour in design automation, everyone in the EDA industry is maneuvering – trying to find a way to claim that what they were already doing is somehow ESL so they can be in the "cool" club. As a result, everything from bubble and block diagram editors to transaction-level simulation to almost-high-level hardware synthesis has been branded "ESL" and rushed to the show floor at the Design

Automation Conference, complete with blinking lights, theme music, and special, hush-hush secret sneak-preview demos served with shots of espresso in exclusive private suites.

FPGAs and ESL are a natural marriage. If we believe our premise that ESL technology delivers productivity – translating into a substantial reduction in time to market – we see a 100% alignment with FPGA's key value proposition. People need ESL techniques to design FPGAs for the same reason they turned to FPGAs in the first place –

FPGA platforms and ESL tools deliver dramatically increased flexibility.

they want their design done now. Second-order values are aligned as well, as both FPGA platforms and ESL tools deliver dramatically increased flexibility. When design changes come along late in your product development cycle, both technologies stand ready to help you respond rapidly.

With all of this marketing and uncertainty, how do you know if what you're doing is actually ESL? You certainly do not want all of the other designers pointing and laughing at engineering recess if you show up on the playground with some lame tool that isn't the real thing. Although there is no easy answer to that question, we can provide some general guidelines. First, if you are describing your design in any language in terms of the structure of the hardware, what you are doing probably isn't ESL. Most folks agree that the path to productivity involves raising the abstraction layer.

If we are upping our level of abstraction, how will the well-heeled hardware engineer be describing his design circa 2010? Not in RTL. Above that level of abstraction, however, a disquieting discontinuity occurs. Design becomes domain-specific. The natural abstraction for describing a digital signal

processing or video compression algorithm probably bears no resemblance to the native design tongue of the engineer developing packet-switching algorithms, for example. As a result, we are likely to see two different schools of thought in design description. One group will continue to pursue general-purpose specification techniques, while the other charts a course of domain-specificity.

In the software world, there is ample precedent for both ways of thinking. Even though software applications encompass a huge gamut from business to entertainment to scientific, most software engineers manage to develop their applications using one of the popular general-purpose languages like C/C++ or Java. On the other hand, some specialized application types such as database manipulation and user interface design have popularized the use of languages specific to those tasks. Software is a good exemplar for us, because programmers have been working at the algorithmic level of abstraction for years. Hardware design is only just now heading to that space.

The early ESL tools on the market now address specific hot spots in design. Some provide large-scale simulation of complex, multi-module systems. Others offer domain-specific synthesis to hardware from languages like C/C++ or MATLAB's M, aimed at everything from accelerating software algorithms and offloading embedded processors to creating super-high-performance digital signal processing engines for algorithms that demand more than what conventional Von Neumann processors can currently deliver.

These tools are available from a variety of suppliers, ranging from FPGA vendors themselves to major EDA companies to high-energy startups banking on hitting it big based on the upcoming discontinuity in design methodology. You should be trying them out now. If not, plan on being left behind by more productive competitors over the next five years. At the same time, you should realize that today's ESL technologies are still in the formative stage. These are nascent engineering tools with a long way to go before they fulfill the ultimate promise of ESL – a productivity leap in digital design methodology that will ultimately allow us to keep pace with Moore's Law. 