

ESL Synthesis: Silver Bullet or Power Tool?

John Sanguinetti
CTO, Forte Design Systems

There is a wide diversity of opinion about the role of ESL Synthesis in future hardware design methodologies, ranging from outright disbelief that it will have any role to a belief that it will completely replace the hardware designer. To discuss this meaningfully, we need to define ESL and ESL Synthesis. For our purposes, we will use the following definitions:

ESL design

ESL, or Electronic System Level, design is the activity of designing a system consisting of hardware and software together, beginning with a software description of the entire system as an executable specification.

ESL Synthesis

ESL Synthesis is a process which turns software into silicon predictably. That is, it takes a general-purpose language description of an algorithm and produces a scheduled, allocated RTL (or lower level) representation. The primary activities of ESL Synthesis are operation scheduling and resource allocation.

Typically, we expect ESL design to include modeling activities like virtual prototyping and system analysis of the complete hardware-software system. We also expect ESL design to include implementation activities, which is the role of ESL Synthesis. ESL Synthesis is the automated process of producing a hardware implementation, usually in RTL, from a software description. The question, of course, is can it do that? If the answer is yes, then what need is there for hardware designers? In the same way that assembly language programmers were replaced by programmers in higher level languages (FORTRAN, C, C++), won't hardware designers be replaced by programmers?

The answer to this question is clearly no, for a very good reason. ESL Synthesis is not a silver bullet. It does not take a general description and produce exactly what you want with no further guidance. It is better described as a power tool to be used by the hardware designer to produce better hardware more efficiently and predictably.

High-level source code, the starting point for ESL Synthesis, can be used to create hardware with vastly different physical characteristics, including latency, frequency, area, and power. There is always going to be some additional information provided to a synthesis tool to guide it to produce a design with the desired characteristics. Providing that additional information is called hardware design. The task of a hardware designer is deciding what needs to be built, and then guiding the design tools to produce it. ESL Synthesis is simply a powerful tool for doing that.

On the flip side, we often hear the argument that general purpose programming languages, specifically C, are not well-suited for designing hardware. Again, those making this argument are assuming that ESL Synthesis is supposed to be a silver bullet. That is, the synthesis program is supposed to take a program written in C or C++ and automatically produce the desired hardware implementation. Since it is pretty clear that current synthesis programs

cannot do that without being guided by additional information in the form of constraints and directives, this is taken as an indictment of high-level synthesis as a whole.

However, if we look at ESL Synthesis as a power tool for the designer, the criticism of C or C++ as a starting point becomes much weaker. In fact, C has a number of deficiencies that have been pointed out by various commentators (see Stephen A. Edwards, "The Challenges of Synthesizing Hardware from C-Like Languages", IEEE Design & Test of Computers, Sept.-Oct., 2006). Those deficiencies have been remedied by a combination of SystemC, a C++ class library which provides a hardware abstraction layer in C++, and modern synthesis tools. SystemC provides semantic means of expressing concurrency and bit accuracy, while newer synthesis tools provide pragmas by which design constraints and requirements can be specified.

ESL Synthesis consists of two basic capabilities, scheduling and resource allocation. Scheduling is simply determining the order of each of the logical operations that make up the desired computation and creating the state machines to realize that ordering. Resource allocation is the mapping of the logical operations of the computation to physical hardware resources. Taken in combination, it is easy to see that there is a wide latitude of implementation for any non-trivial computation. The resulting implementation can have many resources (meaning a lot of silicon area) and few states (meaning low latency), or it can have few resources (low area) and many states (high latency). There are many more design choices that can be made – the use of pipelining, loop unrolling, how arrays are mapped to memories (if they are at all), aggressiveness of resource sharing, clock gating, etc. For any given design, there is no one best way to implement it. The best way is dependent on the requirements of the overall system. Thus, the role of ESL Synthesis is to give the hardware designer a tool to allow him to experiment with different design constraints and criteria.

As it becomes feasible to implement more complex algorithms in hardware, this design exploration becomes more and more important. It is simply beyond the capability of a human to come up with the "best" hardware implementation of an algorithm a priori, when measured along multiple axes (latency, area, power). One needs to experiment, relying on the power tool to give you a good implementation at any given design point. Ultimately, much of the exploration of the design space can be automated, but it is unlikely that it will be free of hardware designer input anytime in the foreseeable future.

The change from RTL design to ESL design will occur much like the change from schematic capture to RTL design. Writing RTL looked suspiciously like "writing software" to a lot of sparkys. It turned out there was just as much hardware design to be done at RTL as there was at gate level, it's just that the tools were better and the hardware designer could be more productive. The same experience will play out in the shift to design at ESL. Hardware designers will discover that their hardware insights will still be required, but their tools will be more powerful, allowing them to produce better hardware.