In Search of an ESL Methodology

Gary Smith & Daya Nadamuni

During the period of December 2004 and June of 2005 Daya Nadamuni and I went on a search for an ESL methodology. We found that there were some prevailing myths about ESL design and we ended up disproving a few of the current ESL myths.

- Myth 1: The US doesn't do ESL design.
- Myth 2: Present ESL tools can be widely used across the whole ESL design spectrum
- Myth 3: ESL solutions are available in the market today.

While doing our interviews we found that ESL is alive and well, in the US, and that SystemC was the language of choice. When we discussed today's commercial tools the general input was that they were fine for Algorithmic design, but they didn't work for a majority of the design work going on in the US. And finally though vendors were claiming they had ESL solutions, in reality they had point tools that could solve point problems. That's when the importance of the design methodologies became obvious.

So the next step was to ask system designers how they started their system design. Keep in mind we are looking at this as far more of a black and white issue that it actually is. Many if not most systems combine several methodologies (described later) to create complete designs. What we were after was what was the most important design methodology needed of the one or more methodologies used. An example of this is the most complete methodology that is currently available: the Algorithmic Methodology.

EDA vendors have been producing tools for ES Level Algorithmic Design for nine years now. These tools specialize in datapath design. But to have a complete algorithmic solution, Control Logic must be included. So when asked, System Designers reported that they wanted very solid datapath design tools coupled with "good enough" Control Logic design tools and these were not available as full commercial.

How does it work today? The System Designers mostly addressed by ESL vendors today start with an algorithm. This is very common in Consumer Electronics and in some Datacom design work. This is the Algorithmic Methodology and has a tool flow that is coming close to being addressed. ESL vendors who have tools to support this flow primarily have Japanese and European customers. The reason these tools are popular in Europe and Japan is that a lot of consumer design work is done in these regions.

However, this isn't the case in North America. When we asked North American System Designers, how they started out their design, the answers were usually either C/C++ models that were not algorithmic, or State-charts. This then defines the two other methodologies, Processor/Memory Methodology and Control Logic Methodology. This was often followed by the complaint that the ESL vendors were not serving their needs and that they were forced to develop internal tools to get their job done.

The other question that was answered was the fact that we could talk to one group, from a European vendor, that was fairly satisfied with today's ESL tools and then talk to another group, in the same company, and get nothing but complaints. The satisfied group was doing Algorithmic consumer design. The other group was invariably the Automotive design group where control logic played a bigger part.

The Methodologies

From our discussions with semiconductor companies, system companies, vendors and academics, we now see we are looking at three ESL methodologies:

- Algorithmic Methodology
- Processor/Memory Methodology
- Control Logic Methodology

To complicate things further, the three methodologies appear to have two sub-methodologies: platform-based design and architectural design (see Table 1).

Table 1.1 ESL Methodologies

ALGORITHMIC METHODOLOGY	PROCESSOR/MEMORY METHODOLOGY	CONTROL LOGIC METHODOLOGY
Behavioral Level	**Behavioral Level**	**Behavioral Level**
HARDWARE	***SOFTWARE***	***PARTITIONING***
Architectural Level	**Architectural Level**	**Architectural Level**
1. Architectural Design	1. Architectural Design	1. Architectural Design
2. Platform-Based Design	2. Platform-Based Design	2. Platform-Based Design

(Gartner Dataquest May 2005)

Platform Based Design Issues

To keep the terminology simple, Gartner Dataquest defines a platform as a frozen architecture. The Architectural Design sub-methodology is more of a free-form, language-based design style rather than the model-based design style of the platform-based design sub-methodology.

Insufficient Competitive Differentiation

Both architectural design and platform based design have their place, but the anecdotal input coming from North American system designers suggests that Platform-Based Design doesn't provide sufficient competitive differentiation.

Lack of Standard Models and Reuse

Comments from all regions point out that the lack of ESL models and model standards are another problem plaguing Platform-Based Design. That's probably why we aren't seeing the expected amount of reuse per the ITRS roadmap. What we've seen, and it has been relatively flat for four years, is a 55 percent to 70 percent reuse. Part of the problem is also memory blocks. We seem to be stuck at an average of a little more than 30 percent of the die being memory. The

ITRS Design Working Group had thought that memory reuse would be closer to 50 percent by now.

Unfortunately, the power issue has kept down the use of large on-chip memory. This looks similar to the initial years of the register transfer level (RTL) methodology. We tended to code and synthesize 40 percent to 60 percent of the design and then do the rest using the gate level design. This is why we aren't seeing many very large designs. Once you start designing in straight RTL (that is, no reuse outside of design ware), your productivity drops to 25 engineers per million gates. With a reuse methodology it takes five engineers per million gates (ITRS).

Verification Problems

The most recent complaint, about Platform-Based Design, is the high cost of verification. Platform based design was expected to provide a significant decrease in verification costs as compared with free form Architectural Design. Today that doesn’t appear to be the case.

Power and Programmability

The two major "red bricks" (problems or challenges that must be resolved) in the design road map (in the context of the overall ITRS road map) are power and programmability.

Power

It's common to look at power in the context of a battery operated consumer appliance. Actually, the main power problem is reliability and performance. We literally are burning holes in some poorly designed integrated circuits (ICs). The ES Level is by far the best place to attack the power problem. At the behavioral level, power-efficient algorithms, state charts or power-efficient processor/memory architectures can be developed. The design can then be partitioned into the most-power-efficient hardware/software configuration.

Programmability

The other issue is programmability. Once we start outputting multiple processors on an SOC, we need to figure out how to program them. Programming is primarily a sequential process. A programmer can work on a parallel programming model using threads. But the majority of programmers find it difficult to do a complex design using more than four threads.

Some specific applications do exist, that take advantage of a massively parallel multiprocessor environment. Unfortunately, there are not too many of them. Let's face it: Supercomputers are almost always application-specific machines. Past efforts to develop general purpose multiprocessor systems have run out of steam at four threads. Very long instruction word (VLIW) processors have been an attempt to stretch the four-thread boundary. But concurrent programming is a challenge and most gains in efficiency currently are coming from hardware rather than software.

New Market Opportunities

Solving these issues will become the basis of a completely new market segment. This is neither EDA nor is it Embedded Software Design tools; it's something completely different. The new market will be larger than the EDA and Embedded Software Design tools market combined. The challenge is to develop new tools for the System Designer. During the most recent Gartner Dataquest Seat Count Survey, we discovered a new classification of System Designer. This is the Product System Designer who comes from the System Engineering ranks.

Our forecasted growth of the System Designers coming out of the Semiconductor Design ranks proved to be wildly optimistic. It just isn't happening. The total number of SoC System Designers is only expected to be a little over 3,500 this year. By far the largest number of System Designers comes from the Embedded Software world.

This is creating a lot of confusion in the market. The Embedded Software work is currently a lower price, fairly unattractive market for ESL vendors in terms of revenue per seat. This is really the Late Adopter segment of the Embedded Software pyramid. This leaves us with the Product System Designers. They are starting to spend money, not a lot yet, but already twice as much per seat as the Embedded Software System Designers. The survey showed a willingness to spend in the $30,000 to $45,000 a year for their tools. Secondly there also are more Product System Designers that SoC System Designer, almost 4,500 this year. This then could be the growth market for ESL design. These engineers are neither purely hardware engineers nor purely software engineers. They are System Designers and do whatever they need to do to solve their engineering problem whether it is in the hardware domain or software domain. That's exactly the kind of engineers we've been searching for. So as a vendor if you want to work on the next killer application, find out what the System Designer in Bulgaria or Brazil needs. Not the silicon ace in the US or the Software whiz in India.