ABSTRACT SPECIFICATION RATHER THAN HDL PROGRAMMING

William Bricken
November 2002

Here's a summary leap-of-concept: We can offer "C-to-hardware". Not a parser which converts the C language to an HDL, but a parser which converts C programming into Comesh configuration files. OK, to be more accurate, the C language would be good only cause it is crafted for particular microprocessors, which are not Comesh. More correctly, we want "Java-to-Comesh", but not through the Java virtual machine. Some (to me wonderful) qualities of this business strategy;

-- It is easy to write an HDL to Java parser, offering total compatibility.

-- Java has been developed explicitly for this kind of abstraction; our parser would be supported by the language constructs. (E.g., Java has garbage collection as its memory reclamation mechanism, which means that memory management is not part of the programmer's job. HDL has no memory management concepts either.)

-- Java is a superb language for stating abstract performance specifications. It bypasses the clumsiness of HDL, and perhaps more importantly, it sends a clear message that we do not want "your Mother's HDL" as an input language to Comesh. This is obvious since we want to toss 80% of the HDL mechanism. It mediates the culture shock.

-- Java has extensive machine-level design and support features. It also has huge public libraries of other programming tools. An example: Java-to-Comesh would allow co-development of product interfaces (physical or software) which are from-the-beginning integrated with hardware performance resources.

-- Java-to-Comesh turns every software programmer into a hardware programmer, and gives hardware programmers a tool which is far easier to use than HDL.

-- Java-to-Comesh clearly defines a territory and a strategy which is consistent with a disruptive technology, while bypassing many of the awkward transitional issues of asking users to change their skill sets.

-- Java-to-Comesh puts us in a virgin territory strategically. We are not competing with EDA, we are redefining it.

-- Java-to-Comesh solves our two-cultures problem, since the software rewrite will be the same as the internal hardware tools development.

-- Java-to-Comesh solve the training/retraining problem, we do not have to take on any specialized skill set instruction, not even teaching hardware designers how to write less.

-- Java is quite customizable. We can build an object library which is tailored to Comesh, and the customization would be invisible to the specifications programmer.

BIG CAVEAT: This idea is not as smooth as it may sound, since Java is still designed for the microprocessor, so we'd need to, for e.g., throw out 80% of it too. I just couldn't bring myself to say "we need a brand new language".

But several sub-points are accurate:

-- We want a software language interface (or even a graphic design interface) rather than an HDL as a specification language for Comesh. We want a *specification language*, not an HDL language designed for telling fab houses how to construct circuits, not a software language designed for telling microprocessors how to run instructions set.

-- We can always be backward compatible with existing HDL specifications. The question is one of encouraging wrong behavior in the future. We can always use the formal specification aspects of HDL such as state machine description, logic equation description, and look-up tables.

-- In any event, we need to provide a customized shell language which pre-filters undesired language constructs, rather than taking anything that anyone writes and sifting it later. We basically will need a filter/compiler at the Verilog/C/Java level which says "This specification is valid for Comesh". The question is: do we want Synopsis to be making this decision?

-- REITERATION: The Comesh hardware/software suite *does not need* an engineer to address optimization, timing, or place&route, which is about 90% of what EDA tools do.

-- I want to be very clear that this perspective applies to the reconfigurables market, a very small $3B segment of the $175B annual semi-conductor market. We are not moving the Earth, we are simply providing a better tool and (I'm suggesting) asking people to use it to simplify their design tasks.