Managing the Transition from Complexity to Elegance: Knowing when you have a problem

CHARLES MOORE
crmoore@cs.utexas.edu

In this first installment of a short series, former IBM Power4 chief engineer Chuck Moore takes a humorous look at how it's easy for complexity to take a design team by surprise.

He bases this column on his keynote speech for the Workshop on Complexity-Effective Design, an event held as part of ISCA 2003.

—Pradip Bose, Editor in Chief

One of the interesting things about complexity is how sneaky it can be. No one working on a microprocessor design project wakes up in the morning and thinks, "I'm going to design the most complex set of mechanisms on the face of the planet." But nonetheless, we often find ourselves, and our designs, mired in complexity at some point in the design process.

If only it were possible to spot the first symptoms of complexity and to refine the design early on to minimize as much extraneous complexity as possible. Toward this end, I've come up with a list of "top 10" indicators that your design project might have complexity issues. Perhaps the most interesting thing about this list is that I've actually encountered every single one in real projects that I have lead or managed—I'm not just making these up!

Indicator 10: You have a daily meeting to discuss new requirements

Feature creep is a familiar concept to hardware and software developers alike. It is the tendency to accept new features into a project throughout the project's course. I believe that a key discipline for managing complexity in projects is to learn how to say "No!" If you find yourself having daily meetings to discuss new requirements, you probably are not saying "no" loud enough.

Although such meetings might be appropriate very early in the design process, their presence later on is a sure sign of trouble. Early requirements contribute to setting in place the guiding principles and baseline structure of the design. New requirements that don't fit into that existing structure will typically lead to undesirable complexity.

Indicator 9: Your architects outnumber your verification people

If your architects outnumber your verification people, this is an indication that you are probably out of balance when it comes to managing complexity. Architects like to invent and create—it is in their blood. And every time they invent or create, there's a verification burden associated with the innovation. It is actually harder to verify a design than to create it, so I think the appropriate ratio of architects to verification people is actually about 1:2. In other words, for every architect, there should be two verification people. Unfortunately, that's often not the case in industry projects these days.

Indicator 8: You find large triple-nested "case statements" in the HDL

The hardware description language (HDL) is a key representation of the design. It is critically important that it is both a functionally clean and a structurally accurate representation of the actual design. Triple-nested "case statements" in the HDL are an indicator that the design probably isn't really very well structured. Chances are, the designer isn't really thinking of this as a representation of hardware, but rather, as chunk of software code. This code-based mindset is very dangerous because the HDL must not only express the functionality, but also the inherent hardware structure that will ultimately be built. Furthermore, this situation often produces *spaghetti code* that tends to escalate the complexity over time because as the design team finds bugs, the fixes tend to be patches rather than clean structural improvements.

continued on p. 86

Published by the IEEE Computer Society

continued from p. 88

Indicator 7: Most "design fixes" result in one or more new bugs

In some sense, an ill-conceived design is a form of complexity. Such designs are difficult to understand and difficult to fix. If every time a designer tries to fix a bug, it creates new bugs, this is an indication that you've either got an ill-conceived design, a spaghetti code representation of it, or both. In any case, chances are the designer really isn't on top of the design, and this is probably because it involves more complexity than he can handle. Left unmanaged, this situation tends to be a vicious cycle and can easily spin out of control—the sort of scenario that actually kills projects.

Indicator 6: Designer says, "Let's get the function right first, then worry about those *other things*"

When designers say, "let's get the function right first, then worry about those *other things*," they are forgetting that those other things—timing, testability, clocking, debug, and reset just to name a few—lead to very important structural aspects of the final design. Usually when designers start talking like this, it means that they are challenged by the details of the functionality and are unable to balance functionality with the requirements that come from these other aspects of the design. What you really want are designers who are capable and willing to design functions multiple times so that they can select the solution that yields the best *physical* instantiation in the end.

If a designer is having difficulty just getting the functionality off the ground, chances are the design is going to be seriously disrupted when you bring these other factors into play. If the project leaders allow situations like this to continue, they risk running into big problems later on when it comes time to close down the design.

Indicator 5: Design team has five different phrases for talking about the same thing

Using multiple names for the same thing is classic engineer-style communication, but it tends to lead to big problems. When you talk about an interface or a function, and you're trying to communicate with other designers, it is very important that everyone understand the discussion in the same specific way. Effective communication among designers is just as important as effective communication among the circuits on the chip. Unless the team can talk the same language, you're in big trouble. So whenever I see multiple interpretations emerge from some set of terminology, I force people to say what they mean, zeroing in on the term they should use. It sounds silly, but this sort of social discipline within the team can avoid lots of problems and, in the end, it can avoid a lot of expensive misunderstandings.

Indicator 4: Designer asks, "*What knee of the curve?*"

This is sort of an *Engineering 101* principle, but it is amazing how often people choose to optimize by the seat of their pants. If you can imagine a plot in which a line has an upward climb for a while and then flattens out, the flattening point is typically called the *knee of the curve*. For example, in a plot of L1 cache size versus performance, you may be gaining benefit as you increase cache size, but once you hit the knee, the performance stays about the same even when you continue increasing the cache size. After that point, you're getting little—if any—return for the extra resources provided.

As it turns out, every aspect of the design has a *knee of the curve* associated with it, and it is part of the designers' job to understand and optimize around it appropriately.

If designers seem unaware of these inherent design sensitivities, chances are they've either undersized or oversized the resources, which can cause disruptions and problems later. If the design is undersized, you might have to add resources later, which is usually a messy proposition, to make the design work within performance specs. Or, if oversized, you might have created a larger, more expensive or more power hungry design than was actually necessary.

Indicator 3: The number of operational modes approaches the number of instructions

It sounds crazy, but I've seen designs where there were almost as many modes as there were instructions. That's because it's very tempting to add operational modes to a design. For example, it helps avoid making difficult decisions: "I couldn't quite decide whether it should do this or that, so I put a little mode in, and now software can select whichever is more appropriate in each case." This may be a well-intentioned and seemingly friendly gesture, but it is also quite expensive in the end.

For every mode that you put into a design, you're really adding a new dimension to the verification problem. In some sense, with each mode bit, you've doubled the state that the verification process must consider in evaluating the overall design. When a machine has as many modes as instructions, just imagine the huge cross-product matrix this situation creates! It becomes a very ugly, and complex, verification problem.

Indicator 2: Designer says, "*It is really simple; I just can't explain it to you*"

When designers can't explain their design clearly, I believe that the appropriate take-away message is that "It is too complicated for me to handle." An effective design element must be easily communicated so that others can interface with it and work with it. It must be broken down into a hierarchy of cooperating mechanisms and interfaces. I believe that if a designer can't describe their design in English or on a whiteboard, they have no business trying to describe it in the HDL. It is likely that this

person is deferring an understanding—a level of understanding—that is really important, and that this is almost certainly going to come back to haunt the project later on.

Indicator 1: Several individuals on your team have filed in excess of 100 patents

Patents are good, in general, and I like patents. I actually have a whole bunch of them. But, when people file a patent, they feel obligated to use it—to express it within their design. In reality, you cannot afford to put every good idea from your designers into the final design. You must decide what is really important and just say "no" to lots of other interesting ideas. Part of the problem here is that many companies offer incentives to people for filing patents, often making it highly lucrative. At the same time, however, companies want designs completed on time and according to specifications. So in effect, designers get a mixed message: "Be inventive, but also don't make the designs too complicated." It is important that the design process help strike a balance between these antagonistic goals.

A friend at IBM once suggested that companies offer counter-balancing incentive programs: They should reward people for patents and also for the successful removal of complex mechanisms from the design (he likes to call these "anti-patents"). I think he's on the right track, because the design space for a microprocessor starts with a very broad set of possible options, but must ultimately settle in on a set that meets the requirements as optimally as possible. The design process is all about *convergence*: making decisions and narrowing the scope to a point where the design strikes that right balance between complexity and elegance.

Hopefully, most of you can relate to at least a few of these indicators. Together, they point toward a dichotomy that, as architects and designers, we must solve. On the one hand, we are creative people who want to invent and to include clever new ideas in our design. On the other hand, we are trying to achieve an appropriate level of efficiency and elegance in the design, which demands that we eliminate as much complexity as possible.

That's why I've entitled this series *Managing the Transition from Complexity to Elegance*. Achieving elegance requires a ruthless attack on complexity and the sources of complexity. In future installments of this column, I will continue to describe these sources of complexity as well as a ruthless design process blueprint that helps to manage it.

Charles Moore is a senior research fellow at The University of Texas at Austin. Previously, he was the chief engineer on the IBM's Power4 and PowerPC 601 microprocessors.