



Managing the Transition from Complexity to Elegance: Design convergence

CHARLES MOORE

crmoore@cs.utexas.edu

..... In my last column, I poked fun at situations that can cause complexity problems in chip design. Although spotting such problems is relatively easy, the real challenge is identifying strategies for reducing complexity and achieving design convergence. To begin, it is important to define *complexity-effective design*. I like to define it as a design that

- embraces a relatively small set of overriding design principles and associated mechanisms, and
- has been ruthless in collapsing unnecessary complexity into these more fundamental and elegant mechanisms.

Basically, this means that you are very explicit about what goes into the design, and that you are committed to a rigorous refinement process. You carefully choose what design features to invest in and then become ruthless in collapsing other potential features in ways that make use of these investments. Rather than indulging in many small, low-cost, somewhat-useful features, you remain focused and say, "No, we're not going to do that." Instead, these features must fit within one of the existing mechanisms, even if it means a somewhat less-than-optimal implementation of that feature. (Remember, the fundamental mechanisms that you've already chosen are based on the most important requirements for the design.) It is interesting to note that to the

extent the design can support these new features, folding them into the existing framework actually furthers the value of the original design investments.

And so, the question remains, How do you do this? How do you choose what is most important? And, what's the process you use to collapse unnecessary complexity? Obviously, there is no set formula, but the following development philosophies have proven helpful in projects that I have led.

Understanding requirements

Today, more than ever, you don't just set out to design a chip, you set out to be part of a product development. Each product has some higher-level, first-order requirements that should strongly influence all the components that make up the product. Before architects begin discussing the nature of a particular chip or the mechanisms within it, they must understand these first requirements. It should be clear that missing key requirements can jeopardize the entire project, but it is also true that taking on unnecessary ones will increase complexity and jeopardize the project in other ways.

Furthermore, these first requirements are critical for early identification of the fundamental design principles and priorities for the design. As described earlier, appropriate selection of these fundamental design principles is important in managing complexity because the lower-level design details must build upon the frame-

work set up by these principles. Similarly, these requirements help to formally define the project priorities. It is very important to know the relative importance of factors such as scheduling, performance, cost, and power, to name just a few (and, no, they can't all be the first priority). Later, when the inevitable conflicts arise, the design team should be able to call on these established priorities to make the correct calls. By getting these things right early on and sticking with them, the project has an excellent chance of staying centered, even when some of the lower-level requirements are in motion.

For example, on the Power4 project at IBM, one of the primary requirements was to build a family of powerful server systems—not just a powerful microprocessor. As a result, it became clear that working out the system architecture's structure, scalability dimensions, and multiprocessing support was far more important in the project's early days than optimizing details of the microprocessor. In the end, my colleagues and I did develop a very powerful microprocessor. But as a clear indication of the weight of these first requirements, it is interesting to note that we implemented two microprocessors on the chip instead of a single more powerful one.

Staged development process

Once all the stakeholders understand and agree to the requirements and prior-

continued on p. 79

continued from p. 80

ities, you want a design process that helps manage the complexity creep that typically occurs during design. To do this, you want a staged development process, broken into phases. The phases should have milestones between them that call for the production of very specific items. These milestones and deliverables serve as forcing functions, driving the formalism and rigor of the design. Each phase has a theme, and progress through these phases serves to simultaneously confirm that everyone understands the requirements and that the design is indeed feasible. At the same time, working through these phases builds the leadership team and structures the design for a successful development.

Concept

The first phase, called the *concept* phase, is an opportunity to explore various possible design concepts that might be appropriate. It should be staffed with a small set of people who will ultimately be the project's key leaders. This group should simultaneously explore design concepts, understand past successes and failures, understand competitors, talk with customers, and refine the design requirements. By the end of this phase, you are really trying to achieve two goals: First, the group should have selected the design concept that best matches the requirements. Second, the project leaders should have agreed on an organization and share a common vision for the project. Teams sometimes overlook this second goal, but I believe it is especially important in the long haul because these are the people who will drive the design to completion.

High-level design

The next phase is called *high-level design*, and the theme here is "measure twice, cut once." At this point, you want to have brought in more of your team leaders and key engineers—the people who will ultimately own the design—but not the whole team. You start with the concept design, and the leadership team

refines that down to a set of fundamental structures and mechanisms. You want the leaders to really think through the design by drawing data flow diagrams on paper, writing performance models, partitioning functionality, budgeting resources, and defining the interfaces between subunits in the design. In many cases, it is actually desirable to have designers prepare two or three design alternatives for important sections before selecting the one that best matches the design's overall philosophy. Instead of accepting a collection of disconnected complexities, you spend time with these design alternatives, turning them into something more elegant by lining them up and making sure they fit together well.

Although some methodology trailblazing is usually appropriate, you really want to avoid committing any of this design work into the design system at this point. The time and effort invested during this phase is important for several reasons. First, it allows team leaders to grasp the overall design and learn how to communicate it before actually attempting to implement any of the individual pieces. In effect, this prepares them to become better and more-informed leaders for when the bulk of the team begins work on the implementation. Second, this phase structures the design and flushes out many of the interunit issues much earlier than in a traditional design flow. This early identification enables more thoughtful and elegant solutions, and also makes the overall back-end schedule more predictable. In the end, it is my experience that although this is an expensive investment in time, it is well spent and leads to better design decisions all around.

Implementation

The final presilicon phase is *implementation*. Presuming that team leaders did a reasonably good job during high-level design, the path through implementation to design closure is really only a matter of time and resources. Of course, new details and issues will emerge, but the design should be on a solid enough foundation to absorb many of these changes

without significant disruption. The second time you do something, it is always much easier and, in some sense, the implementation phase represents a second time around on the design. Your team members are already immersed in the design concepts, so they are more efficient in dealing with bugs, timing, test, power, and other physical design constraints.

In the interest of space, I have greatly oversimplified all of these phases—especially the implementation phase. The process of verification and design closure is enormous, but rather than attempting to treat those subjects in this column, I will wait for the next one, tentatively called "Getting it right."

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

you@computer.org

FREE!

All IEEE Computer Society members can obtain a free, portable email

alias@computer.org. Select your own user name and initiate your account. The address you choose is yours for as long as you are a member. If you change jobs or Internet service providers, just update your information with us, and the society automatically forwards all your mail.

Sign up today at

<http://www.computer.org>

