

Toward Disciplined Design

Brian Greskamp

October 8, 2007

At the seventh annual Workshop on Complexity Effective Design (WCED), microarchitects met to discuss — well — complexity: How can we measure it, predict it, design around it, and make it do our laundry? The discussion featured perspectives from John Shen (Intel), Dennis Abts (Cray), Dan Leibholz (AMD), Victor Zyuban (IBM), and José Renau (University of California Santa Cruz).

1 Disciplined Design

One of the grand visions that drives WCED is “disciplined design”, a move toward formalizing the design process and removing subjective judgments. In current design processes, architects rely heavily on intuition and experience when making early decisions about product features and implementation schedules. By contrast, in the ideal *disciplined design*, the correct features and schedules would be obtained mechanically as the result of a well-defined optimization problem. For a high-performance processor, the objective might be to maximize performance under constraints on the final product (eg. power consumption, die area, cost per unit) and on the development process (eg. time to market and engineering cost). Early in the process, designers would use thumb-rules to obtain estimates of some variables. Later, as more data becomes available, they would refine their estimates, re-run the optimization, and order changes to stay on an optimal trajectory.

Obviously, the ideal described above is a long way off. The most conspicuous problem is the need for high-level thumb-rules to estimate quantities early in the design. Microarchitecture performance simulators coupled with cache and power modelling tools have facilitated early performance, power, delay, and area estimation. However, other quantities like time to market and development cost are much harder to estimate. How can we estimate them before the project is finished, let alone early on? That is the important question, and since almost every response invokes “complexity” sooner or later, let’s get that out of the way first.

2 Defining Complexity

What is “complexity”? To a computer scientist, it probably means *computational complexity*: the asymptotic limits on a program’s running time or storage requirements. To a hardware designer, the connotation varies. For example, Palacharla *et al.* [3] give die area, number of transistors, cycle time, and total power dissipation as possible complexity metrics. To a verification engineer, *complexity* may refer to the size of the state space or the number of tests that will be required. For an engineering manager, it may be the amount of engineering effort (person-months) that will be required to complete the design.

Not only are these definitions different, they are mutually contradictory. For a high-performance processor design, reducing power dissipation does not decrease the design or verification effort (in person-months). On the contrary, clever power management schemes add corner cases and make the design more difficult to understand, yielding an *increase* in design and verification time.

We’re unlikely to come to an agreement on a single definition for “complexity”, but there are a couple things we can do to lift the fog. It helps to always prefix “complexity” with a qualifier that concisely describes the specific quantity or measure under consideration. For example, we can discuss “area complexity” or “power complexity” and make precise statements like “The area complexity of the router is $O(n \log n)$ in the number of processors”. But in that usage, we might as well banish the beast and refer simply to “area”.

The real problem is the use of “complexity” to represent some nebulous and unmeasurable but supposedly intuitive quantity. These statements often crop up in discussions on the human factors of design. Our human intelligence is the primary determinant of how long design and verification take, but how can we measure the “understandability” of a design? We can not, but there is a simple, direct, easy-to-measure proxy that we’ve already alluded to. That proxy is *effort*, essentially the number of person-months required to complete a task. In most other branches of engineering (including software engineering), rigorous effort measurement and estimation procedures are already established. In many cases, microarchitects can shun the ambiguity of “complexity” by discussing the associated *effort* instead.

3 Factors Influencing Design and Verification Effort

Design and verification effort then are the quantities we really want to estimate, and that is a difficult task. Current estimates rely heavily on intuition, although work is progressing on objective estimators (for example [1]). Perhaps a good early step to developing estimation methods is to identify, from experience, microarchitectural features that strongly influence effort. The WCED panelists, some of the industry’s most experienced practitioners, were ideally positioned to do just that. All of the features they identified impacted effort in one or more of the following three ways:

Growing State Space An increase in the size of the state space of the control logic usually makes the design more difficult to verify. Deep pipelining is a specific feature that

grows the control state space. When the pipeline is deep enough that global stall, squash, and exception signals can not reach all stages in a single cycle, additional control states are needed to correctly handle late and overlapping arrival of global signals. Dynamic power and thermal management (DPTM) also increases the size of the state space by making the latencies of operations dependent on the power saving state. A floating point add may have a latency of two cycles or ten cycles depending on whether the FPU is asleep when it issues. Finally, a classic example of state space explosion occurs whenever protocols (especially cache coherence protocols) are present.

Introducing Nondeterminism Some features introduce uncertainty in the timing of microarchitectural events, making bugs uncovered during verification more difficult to reproduce and characterize. With nondeterminism, a bug in one run may not manifest in another run that starts from exactly the same initial conditions. Features that can cause nondeterminism are DPTM and multiple clock domains (MCD). In DPTM, units may be shut down unpredictably do to thermal or power constraints. With multiple clock domains, the clock edge at which a message is received into a clock domain may vary from run-to-run, depending on the relative frequency and alignment of the clocks in the transmitting and receiving domains.

Facilitating interaction Design and verification are simplest when a system decomposes into independent non-interacting modules. However, some features introduce non-obvious interaction between modules and prevent independent verification. For example, resource sharing between hardware threads in a simultaneous multithreading (SMT) processor can introduce subtle performance pathologies, livelocks, and deadlocks that are not present in standard CMPs. All of these factors reportedly contributed to the increased verification effort of the Pentium-4 [2]. Error detection and recovery is another source of interaction; recovery can require non-local operation to repair and restore consistent state.

Armed with five concrete examples of issues that exacerbate design effort, Table 1 shows which issues might be most important for various types of future designs. The four types of architectures in the table are (1) advanced uniprocessors (AUP) such as GALS and large window processors, (2) CMPs, (3) SMTs, and (4) fault-tolerant (FT) architectures. The categories are not mutually exclusive; a fault-tolerant CMP or SMTs includes the union of SMP, CMP, and FT issues.

4 Looking to the Future

The factors identified above are a start, but we have a long way to go before we can accurately and systematically estimate design and verification effort. This is an important open problem that deserves more attention. Contrary to popular belief, the CMP trend will not necessarily bring a decrease in design or verification effort. The effort is merely shifting from the core

	FT	SMT	CMP	AUP
Protocols			X	
Deep pipelining				X
Resource sharing		X	X	
MCD			X	X
DPTM			X	X
Error recovery	X			

Table 1: The left column shows characteristics that increase design and verification effort. The remaining columns show four microarchitectural categories: fault tolerant designs (FT), advanced uniprocessor core designs (AUP), CMPs, and SMTs. An **X** in a row signifies that the design exacerbates the corresponding characteristic.

into other areas of the design — into the memory hierarchy, the network, and subtle aspects of inter-core interaction.

Design and verification effort may *increase* in the near future due to the following specific factors: (1) Reliability concerns will motivate more and more error-recovery support, increasing interaction between components; (2) The drive toward *many core* CMPs will require scalable on-chip coherence protocols that grow the state space; (3) Sharing of on-chip cache, network, and I/O resources will invite unforeseen inter-core interactions; (4) Tight power and thermal budgets will require multiple clock domains and DPTM, adding non-determinism to the system.

Industry is proposing aggressive roadmaps for development of CMP products, but from a design and verification perspective, the road ahead is not necessarily any easier than the one we have known. A disciplined design process is just as desirable now as it ever was, and just as conspicuously missing.

References

- [1] Cyrus Bazeghi, Francisco J. Martínez, and José Renau. μ Complexity: Estimating processor design effort. In *International Symposium on Microarchitecture*, November 2005.
- [2] David Koufaty and Deborah Marr. Hyperthreading technology in the Netburst microarchitecture. *IEEE Micro*, 23(2):56–65, March 2003.
- [3] Subbarao Palacharla, Norman P. Jouppi, and James E. Smith. Complexity-effective superscalar processors. In *International Symposium on Computer Architecture*, pages 206–218, July 1997.