IEEE micro

November/December 2010

Being Geek
p. 80

The magazine for chip and silicon systems designers

RETHINKING DIGITAL DESIGN

IEEE computer society

http://www.computer.org

Rethinking Digital Design: Why Design Must Change

Because of technology scaling, power dissipation is today's major performance limiter. Moreover, the traditional way to achieve power efficiency, application-specific designs, is prohibitively expensive. These power and cost issues necessitate rethinking digital design. To reduce design costs, we need to stop building chip instances, and start making chip generators instead. Domain-specific chip generators are templates that codify designer knowledge and design trade-offs to create different application-optimized chips.

Ofer Shacham
Omid Azizi
Megan Wachs
Wajahat Qadeer
Zain Asgar
Kyle Kelley
John P. Stevenson
Stephen Richardson
Mark Horowitz
Stanford University

Benjamin Lee
Duke University

Alex Solomatnikov
Amin Firoozshahian
Hicamp Systems

Over the past two decades, chip designers have leveraged technology scaling and rising power budgets to rapidly scale performance, but we can no longer follow this path. Today, most chips are power limited, and changes to technology scaling beyond 90 nm have severely compromised our ability to keep power in check. Consequently, almost all systems designed today, from high-performance servers to wireless sensors, are becoming energy constrained. Years of research have taught us that the best—and perhaps only—way to save energy is to cut waste. Thus, clock and power gating have become common techniques to reduce direct energy waste in unused circuits. But power is also wasted indirectly when we waste performance. Higher performance requirements necessitate higher-energy operations, so removing performance waste reduces energy per operation. Using multiple simpler units rather than a single aggressive one, therefore, saves energy when processing parallel tasks. At the system level, this observation is driving the recent push for parallel computing.

Looking forward, the best tool in our power-saving arsenal is customization, because the most effective way to reduce waste is to find a solution that accomplishes the same task with less work. By tailoring hardware to a specific application, customization not only results in energy savings by requiring less work but also improves performance, allowing an even greater reduction of the required energy. As a result, application-specific integrated circuits (ASICs) are often orders of magnitude more energy efficient than CPUs for a given application.

However, despite the clear energy-efficiency advantage of ASICs, the number of new ASICs built today is not skyrocketing, but decreasing. The reason is simple: nonrecurring engineering (NRE) costs for ASIC design have become extremely expensive, and very few applications have markets big enough to justify such costs. This uneasy

0272-1732/10/$26.00 © 2010 IEEE Published by the IEEE Computer Society

status quo is reminiscent of chip design problems in the early 1980s, when all chips were designed by fully custom techniques. At that time, few companies had the skills or dollars to create chips. The invention of synthesis and place-and-route tools dramatically reduced design costs and enabled cost-effective ASICs. Over the past 25 years, however, complexity has grown, creating the need for another design innovation.

To enable this innovation, we must face the main issue: building a completely new complex system is expensive. The cost of design and verification has long exceeded tens of millions of dollars. Moreover, hardware is only half the story. To be useful, new architectures require expensive new software ecosystems. Developing these tools and code is also expensive. Providing a designer with complex intellectual property (IP) blocks doesn't solve this problem: the assembled system is still complex and still requires custom verification and software. Furthermore, verification costs still trend with system complexity and not with the number of individual blocks used. To address some of these design costs, the industry has been moving toward platform-based designs, where the system architecture remains fixed.[1] Nevertheless, although such strategies address design costs, they don't provide the desired performance and power efficiency.

Therefore, we propose rethinking how we approach design. The key is realizing that, although we can't afford to build a customized chip for every application, we can reuse one application's design process to generate multiple new chips. Every time a chip is built, we inherently evaluate different design decisions, either implicitly using microarchitectural and domain knowledge, or explicitly through custom evaluation tools. Although this process could help create other, similar chips, today we settle on a particular target implementation and record our solution. Our approach embeds this implicit and explicit knowledge in the modules we construct, enabling us to customize the system for different goals or constraints, and thus to create different chip instances. Rather than building a custom chip, designers create a template—*a chip generator*—that can generate the specialized chip, similar to the way Tensilica's tools create customized processors (http://www.tensilica.com).

Creating flexible, domain-specific chip generators rather than single chip instances lets us preserve and enhance knowledge from past designs. We can then use that knowledge to produce new customized solutions by capturing, not only the final design, but also the tools and trade-off analysis that led to this design. In this approach, the chip generator uses a fixed system architecture (or *template*) to make both software development and verification simpler, but its internal blocks are flexible, with many parameters set by either the application designer or through optimization scripts. Thus, when application designers port their applications to this hardware base, they can customize the software and hardware simultaneously. In addition, the template contains highly parameterized modules to enable pervasive hardware customization. The application developer tunes the parameters to meet a desired specification. Then, this information is compiled, and optimization procedures are deployed to produce the final chip. The end result of this process is a chip with customized functional units and memories that increase computing efficiency.

Technology scaling and the cause of the power crisis

When considering technology scaling and the semiconductor industry's growth over the past few decades, it's almost impossible not to begin with Moore's law. Introduced by Gordon Moore in 1965, this law stated that the number of transistors that could economically fit on an integrated circuit would increase exponentially with time.[2] Although this law was an empirical observation, history has shown Moore's prediction to have been remarkably accurate. In fact, today, Moore's law has become synonymous with technology scaling.

Moore successfully predicted the exponential growth of the number of transistors on a chip. However, explaining how such scaling would affect device characteristics took another decade: Robert Dennard addressed this in his seminal paper on metal-oxide semiconductor (MOS) device

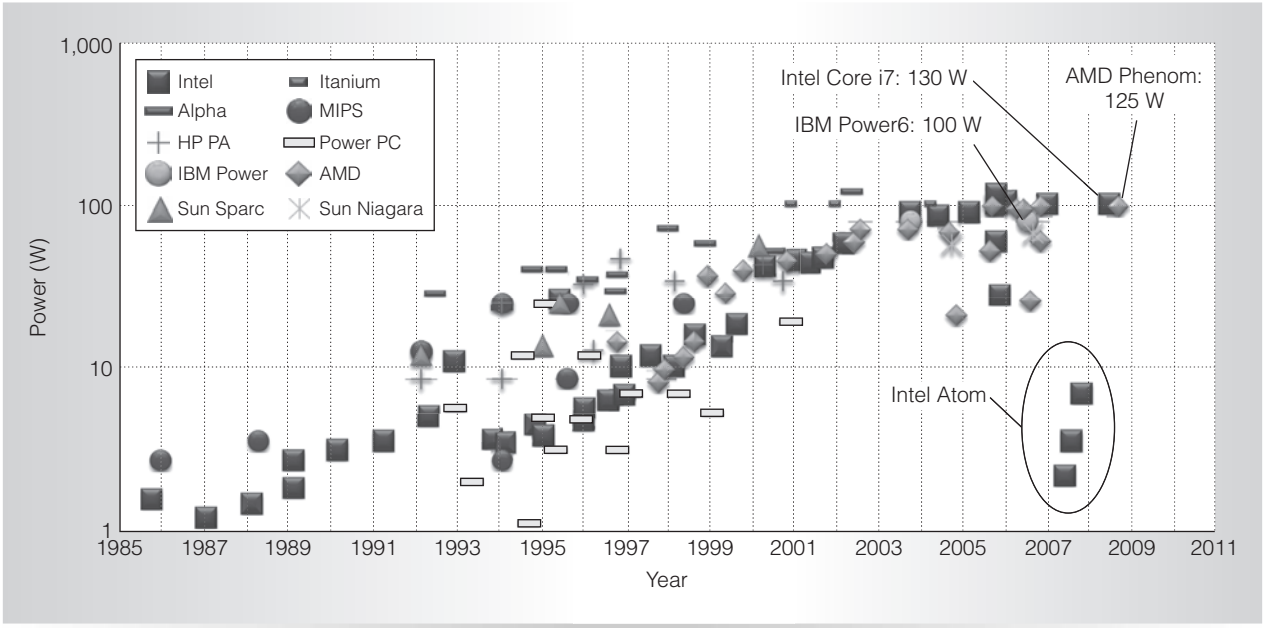

Figure 1. Historic microprocessor power consumption statistics. Power consumption has increased by over two orders of magnitude in the past two decades. However, as evidenced by the Intel Atom, there is a recent trend of lower power processors for the growing market of battery-operated devices. (PA: Precision Architecture.)

scaling in 1974.[3] In that paper, Dennard showed that when voltages are scaled along with all dimensions, a device's electric fields remain constant, and most device characteristics are preserved.

Following Dennard's scaling theory, chip makers achieved a triple benefit: First, devices became smaller in both the x and y dimensions, allowing for $1/\alpha^2$ more transistors in the same area when scaled down by α, where $\alpha < 1$. Second, capacitance scaled down by α (because $C = \varepsilon(LW/t)$, where C is the capacitance, ε is the dielectric coefficient, L and W are the channel length and width, and t is the gate-oxide thickness). Thus, the charge Q that must be removed to change a node's state scaled down by α^2 (because $Q = CV$). The current also scaled down by α, so the gate delay D decreased by α (because $D = Q/I$). Finally, because energy is equal to CV^2, energy decreased by α^3.

Thus, following constant field scaling, each generation supplied more gates per mm^2, gate delay decreased, and energy per gate switch decreased. Most important, following Dennard scaling maintained constant power density: logic area scaled down by α^2, but so did power: energy per transition scaled down by α^3, but frequency scaled up by $1/\alpha$, resulting in an α^2 decrease in power per gate. In other words, with the same power and area budgets, $1/\alpha^3$ more gate switches per second were possible. Thus, scaling alone was able to bring about significant growth in computing performance at constant power profiles.

Nevertheless, power and power density have continually increased. Figure 1 shows a dramatic increase in processor power over the past 20 years. The reason is a combination of designers' not following constant field scaling exactly and creating more aggressive designs, thereby increasing performance more quickly than Dennard predicted.

Figure 2 shows clock frequency growth. By the early 2000s, clocks were running 10 times faster than expected by Dennard's rules. Some of this performance increase came from technology tuning; below 0.25 micron, channel lengths became shorter than feature sizes. Designers also optimized circuits to make faster memories, adders, and latches, and created deeper pipelines that increased clock frequency beyond what Dennard had prescribed. These strategies increased both power and power density,

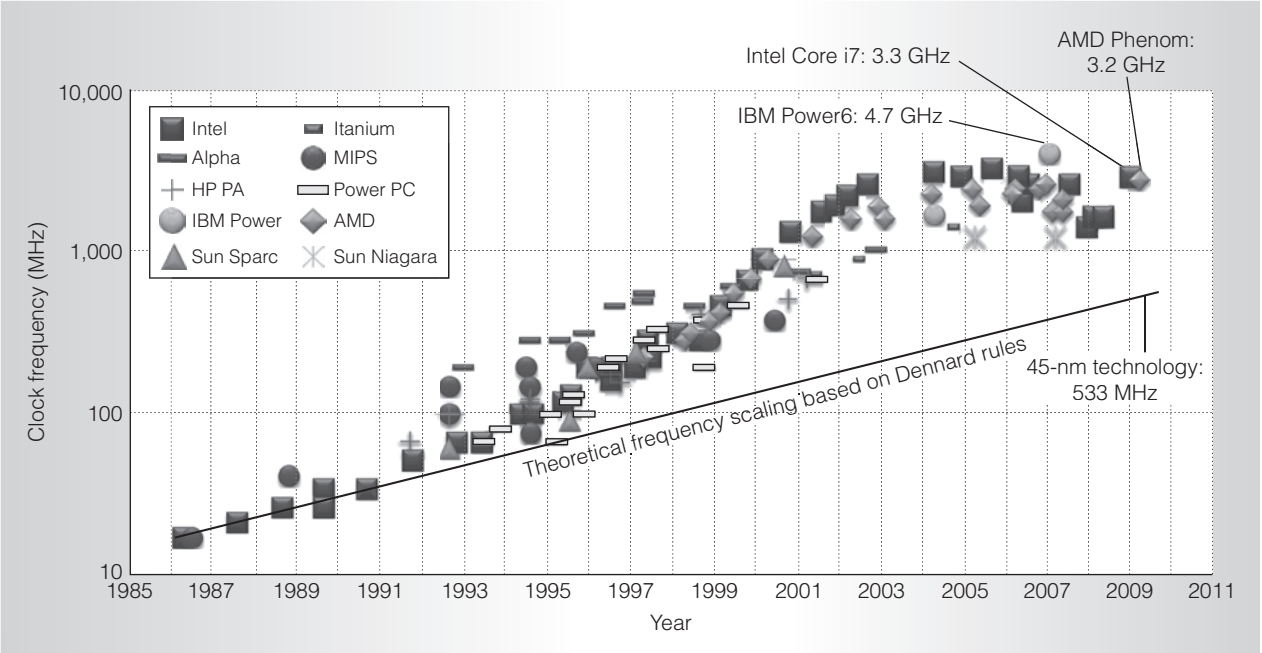

Figure 2. Historic microprocessor clock frequency statistics. The theoretical frequency scaling line shown results from applying Robert Dennard's scaling guidelines to the chronologically first processor in the graph, which would have resulted in a clock frequency of 533 MHz at 45-nm technology. The industry exceeded Dennard's predictions by an order of magnitude.

but, because designs were not power constrained at the time, this increase was not a problem. During this period, computer designers wisely converted a "free" resource (extra power) into increased performance, which everyone wanted.

In the early 2000s, however, high-performance designs reached a point at which they were hard to air-cool within cost and acoustic limits. Moreover, the laptop and mobile device markets—which are battery constrained and have even lower cooling limits—were growing rapidly. Thus, most designs had become power constrained.

Although this was a concern, the situation could have been manageable, because scaling under constant power densities could have continued as long as designers stopped creating more aggressive designs (by maintaining pipeline depths, and so forth). However, at around the same time, technology scaling began to change as well. Up until the 130-nm node, supply voltage (V_{DD}) had scaled with channel length. But at the 90-nm node, V_{DD} scaling slowed dramatically. Transistor threshold voltage (V_{TH}) had become so small that a new problem arose: leakage current. Faced with exponentially increasing leakage power costs, V_{TH} virtually stopped scaling. Because scaling V_{DD} without scaling V_{TH} would have had a significant negative effect on gate performance, V_{DD} scaling nearly stopped as well, and it is still around 1 V for the 45-nm node.

With constant voltages, energy now scales with α rather than α^3, and as we continue to put $1/\alpha^2$ more transistors on a die, we are facing potentially dramatic increases in power densities unless we decrease the average number of gate switches per second. Although decreasing frequencies would accomplish this goal, it isn't a good solution, because it sacrifices performance.

Design in a power-constrained world

In the power-constrained, post-Dennard era, creating energy-efficient designs is critical. Continually increasing performance in this new era requires lower energy per operation, because the product of operations per second (performance) and energy per operation is power, which is constrained.

Figure 3 illustrates the new design optimization problem. Each point in this figure represents a particular design. Some design points are inefficient, because the same performance is achievable through a lower energy design, whereas other designs lie on the energy-efficient frontier. Each of these latter designs is optimal because there are no lower energy points for that performance level. To find these points, we must rigorously optimize our designs, evaluating the different design choices' marginal costs in terms of energy used per unit performance offered, and then selecting the design features that have the lowest cost. In this process, we trade expensive design features for options that offer similar performance gains at lower energy costs.

Clearly, the first step is to reduce waste in the design. Clock gating prevents a logic block's gates from switching during cycles when their output isn't used, reducing dynamic energy with virtually no performance loss. Power gating goes further by shutting off an entire block when it's unused for longer periods of time, reducing idle leakage power, again at low performance costs.

After removing energy waste, further reducing energy generally has performance costs, and these costs increase as we exhaust the less-expensive methods. When an application requires greater performance, a more aggressive, and more energy-intensive, design is necessary. This results in the relationship between performance and required energy per operation shown in Figure 3, and the real processor data of Figure 4.

In the past, the push for ever better performance has seen designs creep up to the steep part of this energy-performance trade-off curve (for example, Pentium IV and Itanium in Figure 4). But power considerations are now forcing designers to reevaluate the situation, and this is precisely what initiated the move to multicore systems. Backing off from the curve's steep part enables considerable reductions in energy per operation. Although this also harms performance, we can reclaim this lost performance through additional cores at a far lower energy cost, as Figure 3 shows. Of course, this approach sacrifices single-threaded performance, and it also assumes that the application is parallel,

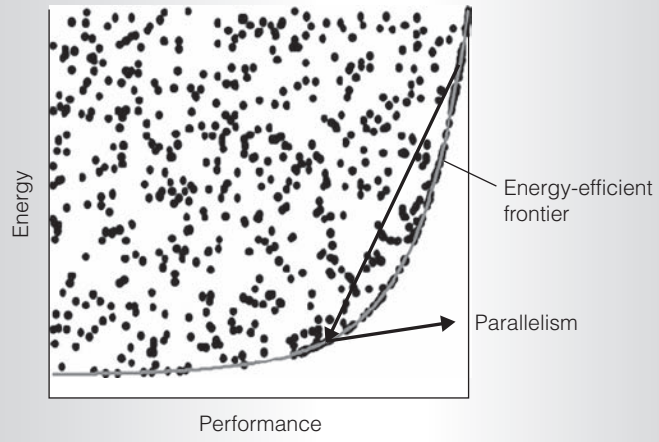

Figure 3. The energy-performance space. The Pareto-optimal frontier line represents efficient designs; no higher performance design exists for the given energy budget. The recent push for parallelism advocates more, but simpler, cores. This approach backs off from the high-performance, high-power points (see the arrow pointing down) and uses parallelism to maintain or increase performance (see the arrow pointing to the right).

which isn't always true. However, given the power constraints, this move to parallelization was a trade-off that industry had to make.

Unfortunately, though, we can't rely on parallelism to save us in the long term, for two reasons. First, as Amdahl noted in 1965, with extensive parallelization, serial code and communication bottlenecks rapidly begin to dominate execution time. Thus, the marginal energy cost of increasing performance through parallelism increases with the number of processors, and will start increasing the overall energy per operation. The second issue is that parallelism itself doesn't intrinsically lower energy per operation; lower energy is possible only if sacrificing performance also yields a lower energy point in the energy-performance space of Figure 3. Unfortunately, this follows the law of diminishing returns. After we back away from high-power designs, the remaining savings are modest.

Further improving energy efficiency requires considering another class of techniques: hardware customization. By specializing computing platforms for the specific tasks they perform, customization can not only result in significant energy savings but

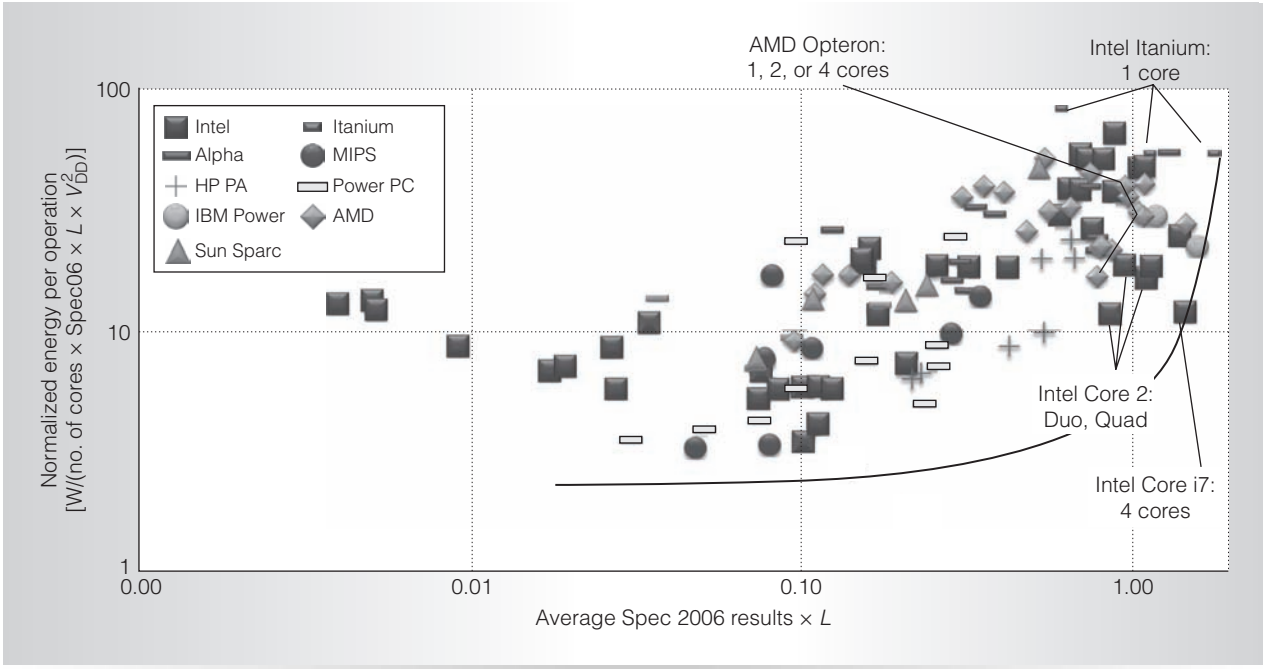

Figure 4. Historic microprocessors' power per SPEC (Standard Performance Evaluation Corp.) benchmark score vs. performance statistics. Performance numbers (*x*-axis) are the average of the single-threaded SPECint2006 and SPECfp2006 results.[4] To account for technology generation, we normalized the numbers according to feature size *L*, which is inversely proportional to the inherent technology speed. The *y*-axis shows power per SPEC score, which is a measure of energy per operation. Energy numbers are normalized to the number of cores per chip and to the technology generation; since $E = CV^2$ (where E is energy, and C is the capacitance), E is proportional to LV^2. Note how the move to multicore architectures typically sacrifices single-threaded performance, backing off from the steep part of the curve.

also reduce the time to perform those tasks. The idea of specialization is well-known, and is already applied in varying degrees today. The use of single instruction, multiple data (SIMD) units (such as streaming SIMD extension [SSE], vector machines, or graphics processing units) as accelerators is an example of using special-purpose units to achieve higher performance and lower energy.[5] To estimate how much potential gain is possible through customization, we need only look at ASIC solutions, which often use orders of magnitude less power than general-purpose CPU-based solutions, while achieving the same or even greater performance levels.

ASICs are more efficient because they eliminate the overhead that comes with general-purpose computing. Many computing tasks, for example, need only simple 8- or 16-bit operations, which typically take far less than a picojoule in 90-nm technology. This is in contrast to the energy consumed in the rest of a general-purpose processor pipeline, which is on the order of hundreds of picojoules.[6] Efficiently executing these simple operations in a processor requires performing hundreds of operations per processor instruction, so the functional-unit energy becomes a significant fraction of the total energy.

Although their efficiency makes building customized chips preferable, designing them is expensive. The design and verification cost for a state-of-the-art ASIC today is well over $20 million, and the total NRE costs are more than twice that, owing to the custom software required for these custom chips.[7,8] Interestingly, fabrication costs, though very high, account for only roughly 10 percent of the total cost today.[7] That means high design, verification, and software costs are the primary reasons why the number of ASICs being produced today is actually decreasing,[8] even though they're the most energy-efficient solution.

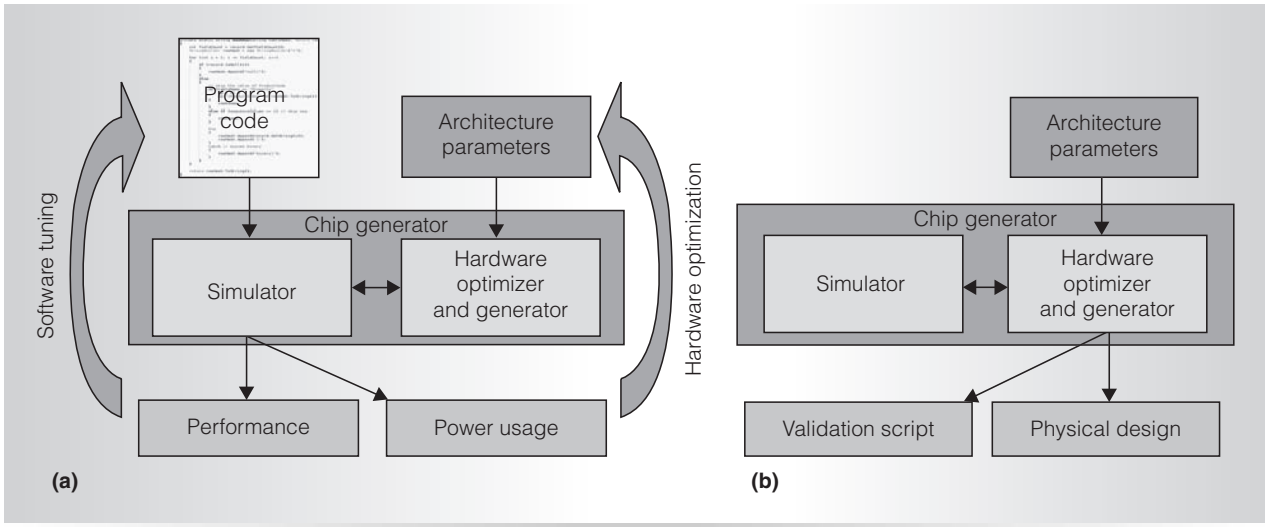

Figure 5. Two-phase hardware design and customization using a chip generator: phase 1 (design exploration and tuning) (a), and phase 2 (automatic generation of the RTL and the verification collateral) (b). In phase 1, the tight feedback loop of both application performance and power consumption enables fast and accurate tuning of the design knobs and the algorithm. In phase 2, the hardware is automatically generated to match the desired configuration.

Design chip generators, not chips

Part of the cost of creating new hardware is the creation of a new set of simulation and software tools, including a system-level architecture simulator and, often, additional design-space exploration tools for internal blocks. Only after the architectural and microarchitectural design trade-offs are well-understood do designers create optimized instances, and ultimately the final chip. In addition, new hardware typically requires new software support, such as compilers, linkers, and runtime environments. An optimized software stack is as important as optimized hardware, because a bad software toolchain can easily cause an order-of-magnitude loss in performance.[9] The importance of mature software also explains the need for software compatibility and why a few architectures dominate.

The importance of software today raises a key question: if the infrastructure to support a new architecture introduces a huge overhead, why do we treat it as disposable? In other words, if we spend so much time and effort on infrastructure to optimize components, why do we freeze the design to produce only one instance? Instead, we should be creating a system that embeds this knowledge, and these optimization tools, inside the design. Rather than record one output of the design and optimization process (the design produced), the process should be codified so that designers can leverage it for future designs with different system constraints. The design artifact produced becomes the process of creating a chip instance, not the instance itself. The design becomes a chip generator system that can generate many different chips, where each is a different instance of the system architecture, customized for a different application or a different design constraint.

A chip generator provides application designers with a new interface: a system-level simulator, whose components can be configured and calibrated. In addition, it provides a mature software toolchain that already contains compilation tools and runtime libraries because, even though the internal components are configurable, the system architecture is fixed, and some basic set of features always exists. Consequently, application designers can now concentrate on the core problem: porting their application code. Furthermore, they can tune both the hardware and software simultaneously to reach their goals.

Per-application customization becomes a two-phase process (Figure 5). In the first phase, the designer tunes both the

application code and the hardware configuration. The chip generator's system-level simulator provides crucial feedback regarding performance, as well as physical properties such as power and area. Application designers can, therefore, iterate and quickly explore different architectures until they achieve the desired performance and power or area envelope. Once the application designer is satisfied with the performance and power, the second phase further optimizes the design at the logic and/or circuit levels, and generates hardware on the basis of the chosen configuration. The chip generator also generates the relevant verification collateral needed to test the chip functionality (because all tools can have bugs).

A chip generator provides a way to take expert knowledge, and more importantly, trade-offs, specific to a certain domain and codify them in a hardware template, while exposing many customization knobs to the application designer. The template architecture provides a way to describe a family of chips that target different applications in that domain and/or that have different performance and power constraints. (Examples of application domains include multimedia and network packet processing. Examples of applications in the multimedia domain are H.264 and voice recognition. Examples of applications in the network-packet-processing domain are regular expressions and compression.) Thus, a chip generator lets application designers control their system's hardware and software with low NRE costs by reusing the entire system framework rather than only individual components (see the "Reconfigurable Chip Multiprocessor as a Limited Generator" sidebar).

In some sense, a chip generator turns the design process inside out from the system-on-chip (SoC) methodology. In SoCs, designers use complex IP blocks to assemble an even more complex chip. SoC design efficiency comes from using preverified IP blocks, and then reusing them for different final chips. However, the system architecture and the software toolchain could vary greatly from one SoC to another. This architectural variance afforded by SoCs exacerbates the verification challenge because verification complexity is related to system-level complexity.

In a chip generator methodology, on the other hand, rather than fixing the components and leaving the system architecture open, we highly parameterize the components and keep the system architecture fixed. Moreover, the flexible components can codify the trade-offs such that the application designer can set each knob's actual value later, and the chip generator would adjust the rest of the design accordingly. The result is that architectural variance is constrained at the system level, so the difficult verification problem can be amortized over many designs. All tools can be faulty, and thus it's impossible to guarantee bug-free hardware; however, the fixed system architecture lets a chip generator find bugs efficiently by either reusing or generating system-level verification collateral.

Is a chip generator possible?

Although a chip generator would be a great tool to have, constructing a useful one is quite challenging. Here, we describe some of our early research on exploring ways to address this challenge. This early research falls into several areas. First, we want to quantitatively understand how specialization improves efficiency. Unless the energy reduction is significant, customization isn't worth the effort. Second, if customization is worthwhile, we need to create a set of general optimization tools that the chip generator can use to make low-level design decisions. This is analogous to the logical mapping and optimization introduced by synthesis tools in the 1980s, but at a higher level.

Another challenge is how to actually construct a hierarchical generator. How do we integrate parameters into the lower-level design to write an individual flexible module? Next, how should we stitch these flexible modules together to create higher-level flexible modules? Then, once we have this elaboration framework, how can we efficiently evaluate the trade-offs of the various design knobs to automatically optimize the final design? Finally, our last research area examines how to create a validation framework for the chip generator that can test each produced instance.

Efficiency gains through customization

To better understand how customization can improve energy efficiency, we must first explicitly quantify the sources of overhead in general-purpose processors; then, we can explore methods to reduce them. As a case study, we examined H.264 video encoding (real-time 720p). H.264 has both highly data-parallel and sequential parts, making it an interesting application to study. In addition, there are both aggressive software and hardware implementations of the H.264 standard, which we can use as reference points.[10]

On a homogeneous general-purpose chip multiprocessor (CMP) architecture, our optimized software implementation used 500 times more energy per frame than an ASIC implementation. This energy overhead was roughly the same for commercial implementations using x86 processors.[11] Hardware customizations could narrow the gap, but we still wanted to know how much efficiency gain we could achieve using general-purpose parallelization techniques, and which gains we could achieve using only techniques highly tuned to a specific task. Thus, we classified our hardware customizations into two groups. First, we considered standard instruction-level parallelism (ILP) and data-level parallelism (DLP) techniques, including simple fused instructions. Second, we created specialized data storage and functional units.

In all, we compared five customization levels, including *RISC* (a simple RISC processor with no custom instructions added), *RISC+SIMD+VLIW* (the same processor with SIMD and VLIW capabilities added), *Op Fus* (all of the above, plus simple fused instructions), and *Magic* (which removes all the previous optimizations, and instead uses Tensilica Instruction Extension [TIE] language to add custom hard blocks).

The results for co-optimization of the H.264 code and hardware at different hardware specialization levels were striking.[12] General parallelization techniques improved both performance and energy by more than an order of magnitude. The results from this type of customization correspond to reported numbers from software implementations on programmable parallel processors such as Tilera and Stream Processors.[13] Despite these impressive gains, however, the energy required for real-time encoding was still 50 times greater than the energy required by an ASIC implementation.

The challenge in achieving ASIC-level efficiencies is that the narrow bit-width operations in these H.264 subalgorithms require very little energy; much of the processing energy is spent on overhead, even when using wide (8- to 16-operand) lanes. Our results show that we can recover about 200 times (of the 500 times more energy spent) by adding large specialized instructions, which couple specialized storage space and communication paths with specialized functional units. These units perform hundreds of operations per instruction, with all operands implicitly addressed from the local storage, thus also reducing data movement.

The inescapable conclusion is that, for computation on narrow data widths, specialized hardware is critical. (IBM's commercial wire-speed processor provides another example of a case requiring specialized hardware, with accelerators for cryptography, XML, compression, and regular expression handling.[14])

Creating chip generators

To create these customized, heterogeneous designs, starting with a flexible homogeneous architecture is better because it can make verification and software easier. To enable customizations, we must build the architecture's internals from highly parameterized and extensible modules, so that application designers can easily shape the components, creating heterogeneous systems tailored to specific application needs. However, initially creating these flexible, customizable modules is challenging.

The idea of creating flexible modules is, of course, not new. Both VHDL and Verilog (post 2001) use *elaboration time parameters* and *generate blocks* to enable more code reuse. Generate blocks let designers write simple elaboration programs, for which parameters are the input and hardware components are the output. Bluespec extended this concept, enhancing and merging programmability into the main code rather than limiting it to generate blocks. At the

Reconfigurable Chip Multiprocessor as a Limited Generator

A *chip generator* is a way to codify design knowledge and trade-offs into an architectural template to expose many knobs to this generator's user, an application designer. It is, therefore, to some extent, similar to a runtime-reconfigurable computing system in its ability to enable detailed control of how internal units function. Whereas a reconfigurable chip is actual silicon that is programmed at runtime, a chip generator is a virtual superset chip that is programmed long before tape-out, such that hardware can either be added or removed and only the required subset makes it to silicon.

One such reconfigurable chip multiprocessor is Stanford Smart Memories (SSM), which first made it to silicon in March 2009. SSM has a memory system flexible enough to support traditional shared memory, streaming, and transactional-memory programming models on the same hardware substrate.[1,2]

Figure A illustrates SSM's hierarchical structure, which connects two Tensilica VLIW cores[3] to 16 memory *mats*,[4] using a configurable processor interface and crossbar to form a *tile* (see Figure A1). Groups of tiles, along with a programmable *protocol controller* capable of implementing the different execution modes, form a *quad* (see Figure A2). Quads connected to one another and to the off-chip memories form a system of quads (see Figure A3). We achieved flexibility primarily by adding metadata bits to the on-chip memory mats, adding programmability at the processor interface, and most importantly, creating the microcoded programmable protocol controller.[5]

As a conceptual example, we repurpose the SSM design as a limited chip generator by leveraging the partial evaluation capabilities of commercial synthesis tools. Rather than dynamically writing values to memories and registers at runtime, we set the values and make them read-only at synthesis time. The synthesis tool then uses constant propagation and folding to reduce any unnecessary logic from the constants' cones of influence.

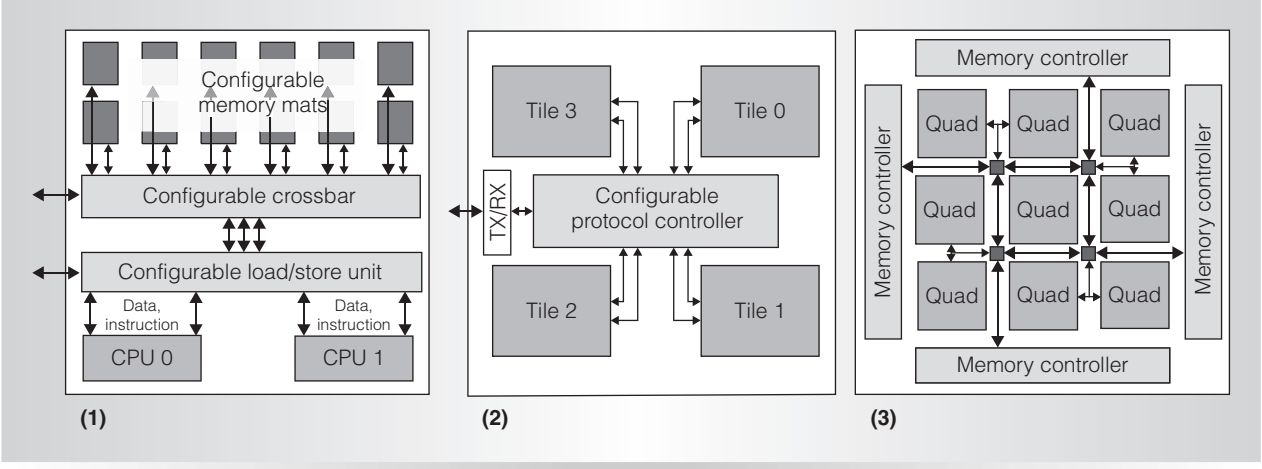

Figure A. The Stanford Smart Memories (SSM) chip multiprocessor architecture: a tile (1), a quad (2), and a system of quads (3). (Rx: receiver; Tx: transmitter.)

block level, commercially available products such as Tensilica's processors, Sonics' on-chip interconnects, and Synfora's PICO (Program In, Chip Out) system generate complete IP blocks for SoCs. However, although these methods are suitable for creating individual blocks, they aren't designed to produce full systems. The missing piece is how to compose these building blocks in increasingly larger blocks until reaching the system level.

When creating a hierarchical chip generator, various types of design parameters must be determined. At the top level, the application designer specifies the high-level design architecture. Beyond these directly controlled parameters, however, many other parameters must still be determined. First, design parameters in different blocks are often linked. Thus, each flexible module must incorporate an elaboration program to specify how parameters in one block are computed from the parameters of another, whether higher or lower in the design hierarchy. This requires that the elaboration engine have significant software capabilities. Furthermore, there are many lower-level design parameters whose impact on the system

Figure B1 shows a generic SSM tile. Figure B2 shows the same tile configured for one active processor, with a small one-way instruction cache and a two-way data cache. Figure B3 shows a constant-propagated, generated version of this configuration. The configuration in Figure B3 also requires some specialized functional units (FUs) inside the processor—which our choice of Tensilica as the base processor facilitates.

References

1. K. Mai et al., ''Smart Memories: A Modular Reconfigurable Architecture,'' *Proc. 27th Ann. Int'l Symp. Computer Architecture* (ISCA 00), ACM Press, 2000, pp. 161-171.
2. A. Solomatnikov, ''Polymorphic Chip Multiprocessor Architecture,'' doctoral dissertation, Dept. of Electrical Eng., Stanford Univ., 2008.
3. R.E. Gonzalez, ''Xtensa: A Configurable and Extensible Processor,'' *IEEE Micro,* vol. 20, no. 2, 2000, pp. 60-70.
4. K. Mai et al., ''Architecture and Circuit Techniques for a 1.1-GHz 16-kb Reconfigurable Memory in 0.18-μm CMOS,'' *IEEE J. Solid-State Circuits,* vol. 40, no. 1, 2005, pp. 261-275.
5. A. Firoozshahian et al., ''A Memory System Design Framework: Creating Smart Memories,'' *Proc. 36th Ann. Int'l Symp. Computer Architecture* (ISCA 09), ACM Press, 2009, pp. 406-417.

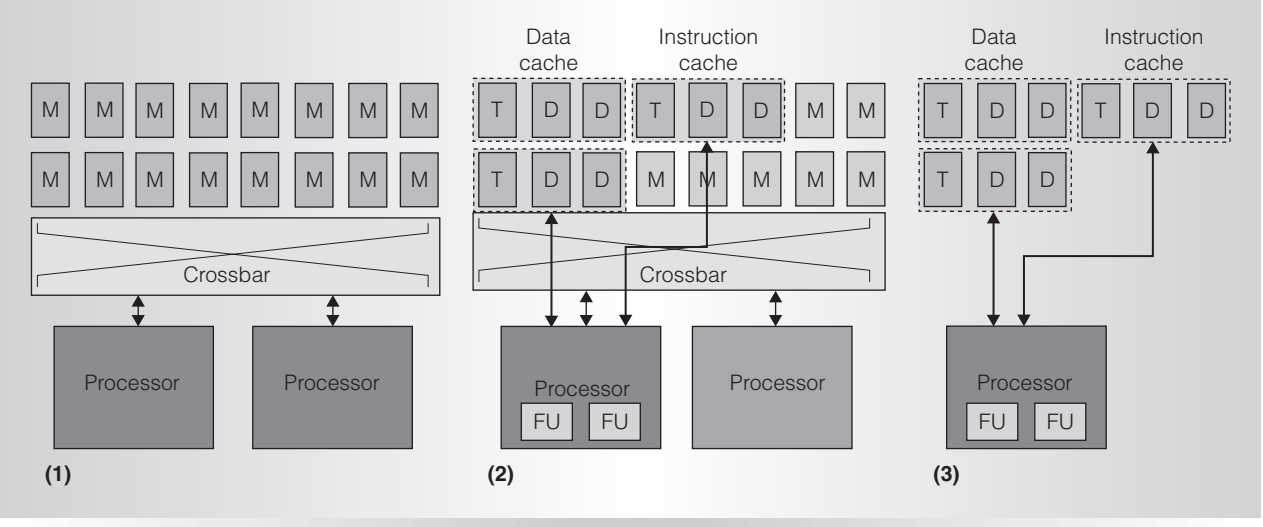

Figure B. Using the configurable SSM design as a limited chip generator: reconfigurable SSM (1); desired configuration, including a single-way instruction cache and a two-way data cache (2); and the generated hardware (3). (FU: functional unit; M: memory mat; D: memory mat used for cache-line data; T: memory mat used for cache-line tag.)

isn't functional but rather a matter of performance and cost (sizing of caches, queues, and so on). These parameters should be determined automatically through optimization procedures. To this end, after the application designer provides the chip program and the design objective and constraints (for instance, maximize performance under some power and area budget), the generator engine should use an optimization framework to select the final parameter values.

The challenge in creating the optimization framework lies in the huge space that must be explored. With even as few as 20 parameters, the design space could easily exceed billions of distinct design configurations. This is a problem because architects traditionally rely on long-running simulations for performance evaluation, so searching the entire space would take far too long. To make this problem more tractable, one powerful technique is to generate predictive models from samples of the design space.[15] With only relatively few design simulations, we can analyze the performance numbers and produce an analytical model.

Using this technique, we've created a hierarchical system trade-off optimization

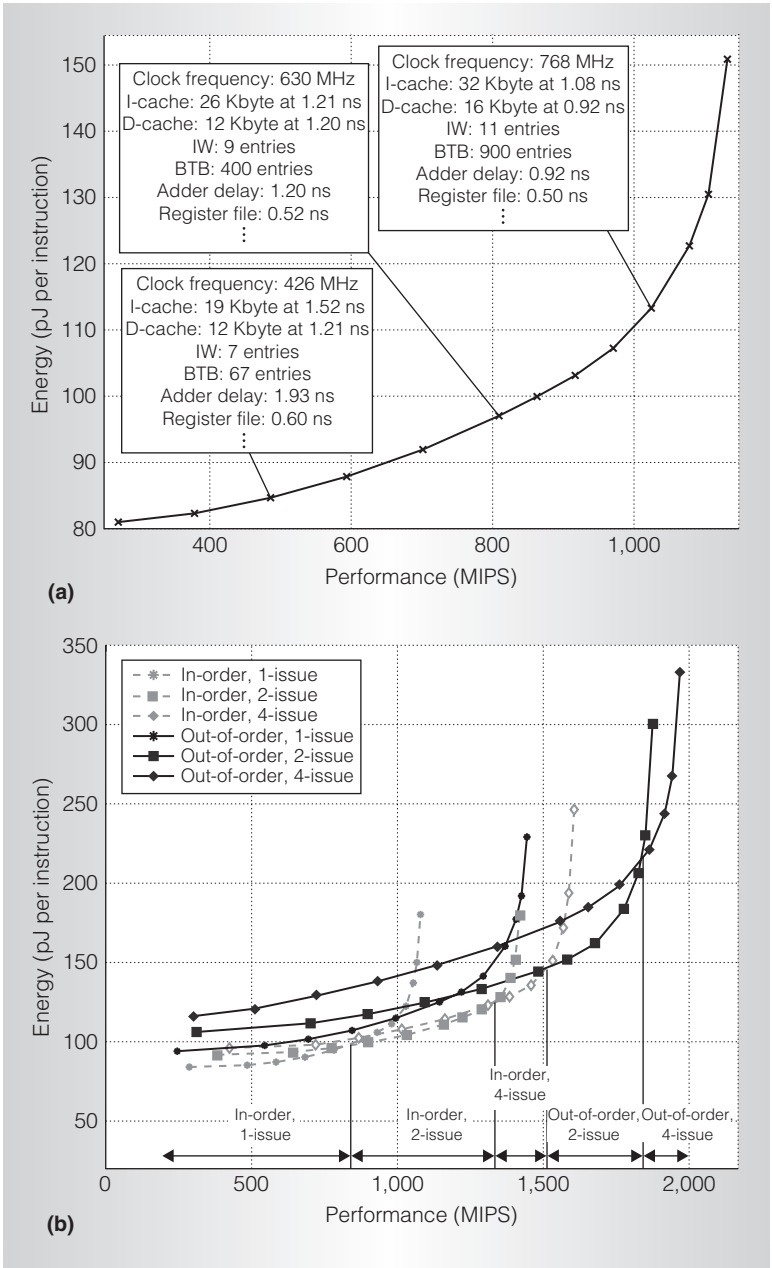

Figure 6. Exploring the energy-performance trade-off curve for processor design: Our optimization framework identifies the most energy-efficient set of design parameters to meet a given performance target, simultaneously exploring microarchitectural design parameters (cache sizes, buffer and queue sizes, pipeline depth, and so forth) and trade-offs in the circuit implementation space. As the performance target increases, more aggressive (but higher-energy) solutions are required. An example is shown using the optimization of a dual-issue, out-of-order processor (a). By repeating this process for different high-level processor architectures, and then overlaying the resulting trade-off curves, designers can determine the most efficient architecture for their needs; Pareto-optimal curves for six different architectures are shown here (b). (BTB: branch target buffer; IW: instruction window.)

framework.[16] Leveraging sample-and-fit methods, we've shown that it's possible to dramatically reduce the number of simulations required. With only 500 simulation runs, we accurately characterized large design spaces for processor systems that had billions of possible design configurations. Moreover, by encapsulating energy-performance trade-off information into libraries, we created a hierarchical framework that could evaluate both higher-level microarchitectural design knobs and lower-level circuit trade-offs. Finally, by using particular mathematical forms during the fit, we formulated the optimization problem as a *geometric program*, making the optimization process more efficient and robust. (Geometric programs are formalized optimization problems that can be solved efficiently using convex solvers. They are similar to well-known linear-programming problems, but they allow for particular nonlinear constraints and are, therefore, significantly more general.[17])

Figure 6 shows the results of using this framework to optimize a processor architecture. Each curve represents a particular high-level architecture with various underlying design knobs. As the performance requirement increases, our optimization framework allocates more resources to each design, resulting in higher performance, but also higher cost. Figure 6a shows some of the lower-level design parameters throughout the design space for one high-level architecture. Figure 6b compares various higher-level architectures. This optimization method is a great tool for tuning many low-level parameters. We're currently working on how to effectively encode and include higher-level, more domain-specific information into the framework.

Verifying chip generators

Even if we can successfully generate an optimized design, we must still address the validation problem because no tool, including ours, can be truly bug free. This is significant because design verification is one of the greatest hurdles in ASIC design, estimated to account for 35 to 70 percent of the total design cost. Our solution is for the chip generator to automatically produce significant portions of the verification collateral.

This verification collateral's main components include a testbench, design assertions, test vectors, and a reference model. Using a fixed system architecture with flexible components is advantageous: because the system architecture is fixed, and the interfaces are known, the same testbench and scripts can be used for multiple generated instances. This is similar to the verification of runtime-reconfigurable designs. In addition, we can apply the parameters used for the chip generator's inner components to create the local assertions.

Once the testbench is in place, a generated design would require a set of directed and constrained-random vectors. Generating directed test vectors is conceptually more difficult because they depend on the target application, although researchers have shown that automatic directed-test-vector generation can be very effective.[17] Most test vectors, however, are, the more easily generated, constrained-random vectors. Unfortunately, random vectors require a model for comparison, and accurate reference models of complex systems are difficult to create.

A traditional reference model, or *scoreboard*, accurately predicts a single correct answer for every output. This requires, however, that the model and design implement the same timing, arbitrations, and priorities on a cycle-accurate basis—a difficult requirement for any complex design, and an infeasible requirement for a generated one. The key to solving this problem is to abstract the implementation details from the correctness criteria. One example of this approach, TSOtool (Total Store Order tool), verifies the CMP memory system correctness by algorithmically proving that a sequence of observed outputs complies with TSO axioms.[18] Another recent method, the Relaxed Scoreboard,[19] moves verification to a higher abstraction level by keeping a set of possibly correct outputs, and updating this set according to the actual detected outputs. Thus, the Relaxed Scoreboard allows any observed outputs, as long as they obey the high-level protocol.

By not relying on implementation details, TSOtool and the Relaxed Scoreboard are suitable as chip generator reference models because they can be reused for all generated instances. Decoupling the implementation from the reference model also has a secondary advantage: it prevents the same implementation errors from being automatically duplicated in the reference model.

However, it's essential to understand that the validation efforts for our solution focus on verifying the resulting instance of the chip generator, not the chip generator itself. Moreover, verification engineers commonly tweak a design's components to induce interesting corner cases—for example, by reducing a producer-consumer first-in, first-out (FIFO) buffer's size to introduce more back-pressure scenarios. Leveraging the chip generator, verification engineers can quickly produce even more variations, introduce more randomness, and expose more corner cases.

When verifying instances produced by our prototype chip generator, we found that this technique resulted in a better and faster verification process for each of the design instances, because one generated instance often exposed a given bug more quickly than other instances. This synergy, coupled with our observation that generating the verification collateral wouldn't be significantly more difficult than creating a single validation infrastructure, is critical. First, it means we haven't made a very difficult task even worse. Second, the validation's effective cost decreases with each new instance produced, while the validation quality improves.

The integrated-circuits industry is facing a huge problem. When systems are power limited, improved performance requires decreasing the energy of each operation, but technology scaling is no longer providing the energy reduction required. Providing this energy reduction will require tailoring systems to specific applications, and today such customization is extremely expensive. Addressing this issue requires rethinking our approach to chip design—creating chip generators, not chip instances, to provide cost-effective customization. Our results have demonstrated the feasibility of our chip generator approach, as well as the potential energy savings it could foster.

Our current focus is on using these concepts to create an actual chip multiprocessor

generator that's capable of producing custom chip instances in the image-encoding and voice-recognition domains. If our success continues, we hope that people will someday look back at RTL coding the way they now look back at assembly coding and custom IC design: although working at these lower levels is still possible, working at higher levels is more productive and yields better solutions in most cases. MICRO

Acknowledgments

We acknowledge the support of the C2S2 Focus Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity, under contract 1041388-237984. We also acknowledge the support of DARPA Air Force Research Laboratory, contract FA8650-08-C-7829. Ofer Shacham is partly supported by the Sands Family Foundation. Megan Wachs is supported by a Sequoia Capital Stanford Graduate Fellowship. Kyle Kelley is supported by a Cadence Design Systems Stanford Graduate Fellowship. Finally, we thank the managers and staff at Tensilica for their technical support and cooperation.

References

1. A. Sangiovanni-Vincentelli and G. Martin, ''Platform-Based Design and Software Design Methodology for Embedded Systems,'' *IEEE Design & Test,* vol. 18, no. 6, 2001, pp. 23-33.
2. G.E. Moore, ''Cramming More Components onto Integrated Circuits,'' *Electronics,* vol. 38, no. 8, 1965, pp. 114-117; http://download.intel.com/research.silicon/moorespaper.pdf.
3. R.H. Dennard et al., ''Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions,'' *Proc. IEEE J. Solid-State Circuits,* vol. 9, no. 5, 1974, pp. 256-268.
4. ''SPEC CPU2006 Results,'' Standard Performance Evaluation Corp., 2006; http://www.spec.org/cpu2006/results.
5. P. Hanrahan, ''Keynote: Why Are Graphics Systems So Fast?'' *Proc. 18th Int'l Conf. Parallel Architectures and Compilation Techniques* (PACT 09), IEEE CS Press, 2009, p. xv.
6. J. Balfour et al., ''An Energy-Efficient Processor Architecture for Embedded Systems,'' *Computer Architecture Letters,* vol. 7, no. 1, 2008, pp. 29-32.
7. R.E. Collett, ''How to Address Today's Growing System Complexity,'' 13th Ann. Design, Automation and Test in Europe Conf. (DATE 10), executive session, IEEE CS, 2010.
8. D. Grose, ''From Contract to Collaboration Delivering a New Approach to Foundry,'' keynote, 47th Design Automation Conf. (DAC 10), ACM, 2010; http://www2.dac.com/App_Content/files/GF_Doug_Grose_DAC.pdf.
9. A. Solomatnikov et al., ''Using a Configurable Processor Generator for Computer Architecture Prototyping,'' *Proc. 42nd Ann. IEEE/ACM Int'l Symp. Microarchitecture* (Micro 09), ACM Press, 2009, pp. 358-369.
10. T. Weigand et al., ''Overview of the H.264/AVC Video Coding Standard,'' *IEEE Trans. Circuits and Systems for Video Technology,* vol. 13, no. 7, 2003, pp. 560-576.
11. K. Kuah, ''Motion Estimation with Intel Streaming SIMD Extensions 4 (Intel SSE4),'' Intel, 29 Oct. 2008; http://software.intel.com/en-us/articles/motion-estimation-with-intel-streaming-simd-extensions-4-intel-sse4.
12. R. Hameed et al., ''Understanding Sources of Inefficiency in General-Purpose Chips,'' *Proc. 37th Ann. Int'l Symp. Computer Architecture* (ISCA 10), ACM Press, 2010, pp. 37-47.
13. H. Li et al., ''Accelerated Motion Estimation of H.264 on Imagine Stream Processor,'' *Proc. Image Analysis and Recognition,* LNCS 3656, Springer, 2005, pp. 367-374.
14. C. Johnson et al., ''A Wire-Speed Power Processor: 2.3GHz 45nm SOI with 16 Cores and 64 Threads,'' *Proc. IEEE Int'l Solid-State Circuits Conf.* (ISSCC 10), IEEE Press, 2010, pp. 14-16.
15. B.C. Lee and D.M. Brooks, ''Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction,'' *ACM SIGARCH Computer Architecture News,* vol. 34, no. 5, 2006, pp. 185-194.
16. O. Azizi et al., ''Energy-Performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis,'' *Proc. 37th Ann. Int'l Symp. Computer*

Architecture (ISCA 10), ACM Press, 2010, pp. 26-36.
17. Y. Naveh et al., ''Constraint-Based Random Stimuli Generation for Hardware Verification,'' *Proc. 18th Conf. Innovative Applications of Artificial Intelligence* (IAAI 06), AAAI Press, 2006, pp. 1720-1727.
18. S. Hangal et al., ''TSOtool: A Program for Verifying Memory Systems Using the Memory Consistency Model,'' *Proc. 31st Ann. Int'l Symp. Computer Architecture* (ISCA 04), IEEE CS Press, 2004, pp. 114-123.
19. O. Shacham et al., ''Verification of Chip Multiprocessor Memory Systems Using A Relaxed Scoreboard,'' *Proc. 41st IEEE/ACM Int'l Symp. Microarchitecture* (Micro 08), IEEE CS Press, 2008, pp. 294-305.

Ofer Shacham is pursuing a PhD in electrical engineering at Stanford University. His research interests include high-performance and parallel computer architectures, and VLSI design and verification techniques. Shacham has an MS in electrical engineering from Stanford University.

Omid Azizi is a member of the Chip Generator Group at Stanford University. He is also an engineer at Hicamp Systems in Menlo Park, California. He is also His research interests include systems design and optimization, with an emphasis on energy efficiency. Azizi has a PhD in electrical engineering from Stanford University. He is a member of IEEE.

Megan Wachs is pursuing a PhD in electrical engineering at Stanford University. Her research interests include the development of hardware design and verification techniques. Wachs has an MS in electrical engineering from Stanford University.

Wajahat Qadeer is pursuing a PhD in electrical engineering at Stanford University. His research interests include design of highly efficient parallel computer systems for emerging applications. Wajahat has an MS in electrical engineering from Stanford University. He is a member of IEEE.

Zain Asgar is pursuing his PhD in electrical engineering at Stanford University. He is also an engineer at NVIDIA. His research interests include multiprocessors and the design of next-generation graphics processors. Asgar has an MS in electrical engineering from Stanford University.

Kyle Kelley is pursuing a PhD in electrical engineering at Stanford University. His research interests include energy-efficient VLSI design and parallel computing. Kelley has an MS in electrical engineering from Stanford University.

John P. Stevenson is pursuing a PhD in electrical engineering at Stanford University. His research interests include VLSI circuit design and computer architecture. Stevenson has an MS in electrical engineering from Stanford University.

Stephen Richardson is a consulting associate professor in the Electrical Engineering Department at Stanford University. He is also a cofounder of Micro Magic. His research interests include various aspects of computer architecture. Richardson has a PhD in electrical engineering from Stanford University. He is a member of IEEE.

Mark Horowitz chairs the Electrical Engineering Department and is the Yahoo Founders Professor at Stanford University. He is also a founder of Rambus. His research interests range from using electrical-engineering and computer-science analysis methods in molecular-biology problems to creating new design methodologies for analog and digital VLSI circuits. Horowitz has a PhD in electrical engineering from Stanford University. He is a Fellow of IEEE and the ACM, and is a member of the National Academy of Engineering and the American Academy of Arts and Science.

Benjamin Lee is an assistant professor of electrical and computer engineering at Duke University. His research interests include scalable technologies, computer architectures, and high-performance applications. Lee has a PhD in computer science from Harvard University. He is a member of IEEE.

Alex Solomatnikov is an engineer at Hicamp Systems in Menlo Park, California.

His research interests include computer architecture, VLSI design, and parallel programming. Solomatnikov has a PhD in electrical engineering from Stanford University. He is a member of IEEE.

Amin Firoozshahian is an engineer at Hicamp Systems. His research interests include memory systems, reconfigurable architectures, and parallel-programming models. Firoozshahian has a PhD in electrical engineering from Stanford University. He is a member of IEEE and the ACM.

Direct questions and comments about this article to Ofer Shacham, Stanford University, 353 Serra Mall, Gates building, Room 320, Stanford, CA 94305; shacham@stanford.edu.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.