The transition from instruction-based to reconfigurable circuits won't be easy, but has its time come?

The Inevitability of Reconfigurable Systems

NICK TREDENNICK, GILDER TECHNOLOGY REPORT
BRION SHIMAMOTO, INDEPENDENT CONSULTANT

The introduction of the microprocessor in 1971 marked the beginning of a 30-year stall in design methods for electronic systems. The industry is coming out of the stall by shifting from programmed to reconfigurable systems. In programmed systems, a linear sequence of configuration bits, organized into blocks called *instructions*, configures fixed hardware to mimic custom hardware. In reconfigurable systems, the physical connections among logic elements change with time to mimic custom hardware. The transition to reconfigurable systems will be wrenching, but this is inevitable as the design emphasis shifts from cost performance to cost performance per watt. Here's the story.

SCRATCH-BUILT CIRCUITS

Until the 1940s, solving problems meant building hardware. The engineer selected the algorithm and the

hardware components, and embedded the algorithm in the hardware to suit one application: fixed hardware resources and fixed algorithms. The range of applications amenable to hardware solutions depended on the cost and performance of hardware components.

THE COMPUTER

The computer happened in about 1940. It was a breakthrough in problem solving that separated the algorithm from the hardware. The algorithm resided in memory as computer instructions. The instructions configured the computer's hardware to mimic functions that would otherwise be custom hardware. And the computer's hardware could be shared across a range of applications: fixed hardware resources and variable algorithms. The computer, much less efficient than a custom circuit, did nothing to extend the range of electronic applications toward higher

The Inevitability of Reconfigurable Systems

performance. But, by placing the algorithm in cheap memory and by conserving expensive logic, the computer greatly extended the range of affordable applications. The pool of problem solvers that had been limited to logic design engineers now encompassed programmers. The expanded pool accelerated electronic systems development.

THE IC AND MOORE'S LAW

The integrated circuit (chip) happened in 1959. In a 1965 issue of *Electronics*, Gordon Moore observed that the number of transistors on a chip could double every year. The observation became Moore's law. It's called a law because it's the pace of improvement that the industry sets for itself. For the past 30 years, this Moore's-law pace has doubled the number of transistors on a chip about every

18 months. (For the first few years, transistors per chip doubled every year, but the pace slowed.)

Semiconductor makers don't set this pace for the sake of progress; they set it to make money. It costs the same amount to process a wafer irrespective of the size of the chips and transistors (just as printing doesn't depend on font size). For an existing design, shrinking the transistors shrinks the chip, and, therefore, more chips fit on the wafer (see figure 1). That increases the manufacturer's profit. For a given-size chip, more of the smaller transistors fit on the chip, so the chip reaches higher-value applications and sells at a higher price (see figure 1).

Integrated circuits accelerated growth in the semiconductor market by raising the productivity of design engineers. Before integrated circuits, engineers designed with discrete components: transistors, diodes, inductors, capacitors, and resistors. Each component had to be selected for the particular circuit. With integrated-circuit "families" of snap-together "logic macros," the engineer no longer needed to size individual transistors and other discrete components. Integrated-circuit families proliferated and logic macros grew more complex at the Moore's-law rate.

THE MICROPROCESSOR

Eventually, enough transistors fit on one chip to be a computer. The microprocessor, a chip-oriented version of the computer, brought the computer's programming model to hardware design. The microprocessor stopped the proliferation of logic macros and focused manufacturing on microprocessors, memories, and peripheral chips. The microprocessor fit the broad range of applications for which the primary considerations were low cost and adequate performance. Broad application meant high-volume production that reduced the microprocessor's unit cost, further extending its application. Microprocessor shipments grew from almost nothing in 1971 to billions of units per year. For the past eight years, manufacturers have been shipping more microprocessors annually than there are people on the earth. The average microprocessor for embedded (noncomputer) applications costs less than \$6, and many sell for less than \$1.

In a direct parallel with how the computer developed, the microprocessor had two other consequences. The microprocessor, memory, and peripheral chips were most of the hardware needed to solve problems. For the first time, engineers only had to choose the procedure (program); they didn't have to choose the resources and design the control structure (as they did when designing with logic macros). Problem solving became programming. The

microprocessor also extended the pool of embedded-systems designers from logic designers to logic designers *and* programmers.

Because instruction-based hardware forfeits efficiency (speed and energy consumption) for engineering productivity, it fits applications where this performance is adequate. Circuit-based hardware fits performance-oriented applications. Moore's-law advances in microprocessor chips made them cheaper, or these advances made them faster and more capable, thus extending their application.

THE PC

Ten years after the microprocessor's debut, the IBM PC came to market with an Intel microprocessor in the role of the central processing unit. The PC split the microprocessor market into performance-oriented designs (bound for the central processing role in computer systems) and cost-oriented embedded designs.

The PC grew to dominate the semiconductor industry, consuming about 40 percent of all semiconductors. The PC dominated component sales and its microprocessor dominated both revenues and press coverage. Compared with embedded microprocessors, unit sales for PC-bound microprocessors are minuscule. PC makers sell fewer than 150 million units a year. PC-bound microprocessors, however, command high prices, averaging almost \$100 (against less than \$6 for embedded microprocessors). These high prices mean that PC-bound microprocessors represent almost half the revenue of all microprocessors in spite of small unit sales. It's no wonder that when

people think of microprocessors, they think of the brains of a PC.

When the PC came out, its performance wasn't good enough. Its performance improved with time at a rate close to the Moore's-law improvement rate of its underlying components. Leading-edge PCs offered leading-edge performance at premium prices. Ordinary consumers bought leading-edge PCs as soon as they were available. Expectations of the PC's early adopters continued to rise, but the population of users spread to include late adopters with lower performance expectations. Demand for performance thus rose and spread. The supply of performance rose at a Moore's-law rate, eventually outpacing demand (see figure 2).

THE VALUE PC

After 20 years of improvement in the supply of performance, and after 20 years of spreading demand, the PC now satisfies most users. The nerd community still buys leading-edge PCs, but most consumers are satisfied with "value PCs." The value PC offers *good-enough performance* at an attractive price. Next year's value PC will have better performance than this year's value PC, so its definition isn't static.

While the PC dominated the semiconductor industry, the industry allocated engineering resources to it. The value PC causes these resources to shift to the design of (more lucrative) untethered systems. That will be a big change for the industry—moving from the PC's cost-performance orientation to the cost-performance-per-watt orientation of untethered systems.

UNTETHERED SYSTEMS

Untethered systems, such as cellphones, share the PC's need to be a low-cost consumer item—and they share its requirement for performance—but untethered systems must be energy efficient. The microprocessor, with its instruction-based processing, won't do as the workhorse for untethered systems. It has to do with design objectives. PC microprocessors are designed for performance; those for

The Inevitability of Reconfigurable Systems

embedded applications are designed for low cost. Microprocessors could be designed to balance performance, cost, and energy conservation, but their instruction-based processing, which needs to be constantly configured to mimic hardware functions, isn't efficient enough for untethered systems.

For the past 30 years, the microprocessor has been the workhorse in embedded systems. The microprocessor has been the workhorse in computer systems for more than 20 years. Engineers increase the microprocessor's performance primarily by increasing its clock frequency. The PC's first microprocessor, Intel's 8088, an ancient relative of today's Pentium, ran at 4.77 MHz. Today (mid-2003), a top-of-the-line x86 microprocessor runs at 3 GHz—more than 600 times the speed of the original. Doubling clock frequency doubles energy use.

That's not a problem if the system plugs into a wall socket as the PC does, but it's a problem for untethered systems. To lower energy use, engineers lower the voltage. Halving the voltage lets the microprocessor run four times as fast on the same amount of energy. That sounds like a solution except for two things.

First, voltages have come down as clock frequencies have increased so that low-power microprocessors already run at less than a volt. They approach the threshold below which their transistors will cease operation.

Second, instruction-based processing uses too much energy. (All the energy spent setting up the microprocessor to mimic desired circuits is wasted.) Digital signal processors, cousins of the microprocessor adapted for signal processing, face the same dilemma. They are likewise running out of room to lower their operating voltage.

How about using the microprocessor for supervisory functions and using custom circuits for the heavy lifting?

Think about a camera's lens filter. It transforms an entire image in realtime. The filter transforms the image at essentially no cost in time or energy. To build the equivalent transformation for a digital image, an engineer programs the filter's function in a high-level language. A compiler translates the program into microprocessor instructions. The microprocessor runs tens of thousands of instructions on each of several million pixels to apply the filter function. Efficiency is lost with each step in the process. A custom circuit builds the equations of the filter's transformation directly.

The custom circuit can be efficient enough, and for some untethered applications it may be the answer. But for the most interesting emerging untethered applications, it still won't do. The most efficient custom circuit is the application-specific integrated circuit (ASIC), which builds exactly the circuit to do the application on a single chip.

ASICs lack the flexibility needed when requirements are still developing. This includes most of today's wireless consumer applications, such as cellular networks and wireless local-area networks. ASICs may also cost too much to develop and build.

If the microprocessor won't do and an ASIC won't do, then what will? The answer right now is: Nothing. But the story isn't at an end.

THE PROMISE OF PLDS

Let's look at another candidate that doesn't meet the cost-performance-per-watt requirements of untethered systems: programmable logic devices (PLDs). Unlike the other candidates, the PLD holds promise. Think of the PLD as two layers: one layer is logic blocks and wires; the other layer is personalization memory (see figure 3). Bits in the memory specify connections between the blocks and wires to build circuits.

One variety of PLD uses static random-access memory (SRAM) for the personalization memory. Altera and

Xilinx dominate the market in SRAM-based PLDs. SRAM personalization memory could enable reconfigurable chips, but so far that's not what customers want them for, so manufacturers don't build chips that reconfigure easily. PLD critics point to the enormous overhead in wires and in personalization memory (perhaps 20 overhead transistors to net one logic transistor) and to performance 10 or 20 times slower than an ASIC. Trade-press articles and technical conferences pit PLD advocates against ASIC advocates. The debates focus on chip size and circuit speed.

This reminds me of the long-running debates between assembly-language programming and high-level-language programming, which also focused on size and speed. Assembly-language programming advocates won the battles by proving better performance and better size. But they lost the war because they missed the significance of the shift to high-level languages. The critical shortfall was programming talent. Thus, the real battle was about programmer productivity and about what was good enough. High-level languages sacrificed efficiency for greatly increased programmer productivity. While the debates highlighted applications that demanded leading-edge performance, high-level languages were adequate for most applications.

Time was on the side of high-level-language programming. Each Moore's-law turn of the crank meant faster microprocessors running the programs and cheaper memory to store them in. Each semiconductor generation tipped the scales further in favor of high-level languages.

Debates between PLD advocates and ASIC advocates that focus on chip size and circuit speed miss the significance of the shift to PLDs. It's no longer about absolute size and speed; it's about what is good enough. We can apply the same supply-and-demand model that we used for the PC (see figure 2). The supply of chip size and circuit speed for ASICs and PLDs improves with Moore's law, but the supply curve for ASICs is well above that for PLDs. The demand for chip size and for circuit speed grows at some (difficult-to-measure) rate less than Moore's law, and the demand spreads out with time. Over time, the number of applications satisfied with the supply of chip size and circuit speed from PLDs grows, while the number of applications that demand ASICs diminishes.

The range of applications suitable for PLDs increases, but that doesn't mean that they are suitable for untethered applications. PLDs are still too slow and use too much energy, but they will improve. The general-purpose nature of today's PLDs will give way to faster, more efficient application-oriented PLDs. These don't need the ability to connect any logic element to any logic element

(general-purpose interconnect), nor do they need to connect to a broad range of chips. In addition, they might have application-oriented logic elements rather than general-purpose logic elements. SRAM personalization memory, however, is a problem; it's power hungry and retains its contents only as long as the power is on.

NEW MEMORY OPPORTUNITY

Efficient PLDs for untethered applications need the ultimate memory: nonvolatile (like flash memory), as dense as dynamic random-access memory (DRAM), and as fast as SRAM. Candidates for this Holy Grail actually exist. They include magnetoresistive random-access memory (MRAM), ferroelectric random-access memory (FRAM), and ovonic unified memory (OUM).

MRAM aligns a tiny magnetic domain in one direction to store a 1 and aligns it in the opposite direction to store a 0. FRAM aligns the electrical polarity of a crystal in one direction to store a 1 and in the opposite direction to store a 0. OUM relies on the difference in resistance between amorphous and crystalline states of a polymer, similar to the way that bits are stored on DVD or CD.

Each of these candidates has impressive backers: Hitachi, IBM, Infineon, Motorola, and NEC for MRAM; Hynix, Oki, and Texas Instruments for FRAM; and Intel, Samsung, and STMicroelectronics for OUM.

Some of these candidates have been around for 15 years without making headway against the incumbents: flash memory, DRAM, and SRAM. So what makes me think the situation will change?

The PC began with only DRAM and read-only memory (ROM). DRAM was working memory and ROM held programs that initialized the chips on the system board. Flash memory displaced ROM because flash memory could be rewritten, which enabled updates of initialization programs in the field. When the PC was introduced, its microprocessor and DRAM were about the same speed. Over time, microprocessor developers improved the speed and DRAM developers improved the capacity. The result has been a widening gap between the speed of microprocessors and that of DRAMs. Today's microprocessors are more than 600 times faster than the microprocessor that powered the first PC. Today's DRAMs have 4,000 times the capacity of the PC's original DRAMs; however, they currently are only five to seven times faster than the PC's original DRAMs. In today's PCs, SRAMs attempt to bridge the speed gap between the microprocessor and the DRAM.

But the PC is at the point where improvements yield diminishing returns in system performance.

The Inevitability of Reconfigurable Systems

Flash memory, DRAM, and SRAM occupy unassailable niches in the PC. The PC takes advantage of the strengths of each memory type and isn't hurt by each one's weaknesses. Huge volumes in the PC market have driven down the cost of memory components. This combination of leveraged strength and low cost has made it impossible for novel memories to encroach.

Reconfigurable Future

MARK HOROWITZ, STANFORD UNIVERSITY

Predicting the future is notoriously hard. Sometimes I feel that the only real guarantee is that the future will happen, and that someone will point out how it's not like what was predicted. Nevertheless, we seem intent on trying to figure out what will happen, and worse yet, recording these views so they can be later used against us. So here I go...

Scaling has been driving the whole electronics industry, allowing it to produce chips with more transistors at a lower cost. But this trend is a double-edged sword: We not only need to figure out more complex devices, which people want, but we also must determine which complex devices lots of people want, as we have to sell many, many chips to amortize the significant design cost.

This push toward finding complex devices with large application classes was the driving force behind the creation of the early microprocessors. Remember, in the early 1970s Intel created the 4004 so it wouldn't have to create a new calculator chip for each company that wanted one. By programming the 4004, each company could configure the chip for its own application. For the past 30 years, the programmable processor model has been one of the most successful abstractions used in the semiconductor industry. Bear in mind that most of the processors sold (by more than an order of magnitude) don't go into PCs or other things we think of as computers; rather they are used to create some needed functionality in other electronic devices. The processor truly is a reconfigurable device; the configuration is done

SEA CHANGE

The appearance of the value PC changes everything. Engineering resources are being moved to untethered systems. Flash memory, DRAM, and SRAM all have shortcomings that make them unsuitable for untethered systems. DRAM and SRAM don't retain their data when the power is off. DRAM is slow and it leaks, requiring periodic reads and restores. SRAM uses too much energy. Flash memory is even slower and it wears out. And the memory hierarchy that worked for the PC uses too much energy for untethered systems.

So far, there isn't a novel memory chip that's worthy of being crowned a winner. What's important is that the incumbent memories are unsuitable for untethered

by executing a sequence of instructions.

As we have continued to scale technology, these processors have become increasingly sophisticated, consuming the available transistors to produce faster machines. This scaling led to the now famous corollary to Moore's law: Processor performance doubles every 18 months. What's remarkable about this performance growth is that the basic machine abstraction, sequentially executing instructions, remains constant. This stable programming abstraction makes it possible to run your old code on these faster machines, and to incrementally modify the code to create the desired more-complex systems.

As Nick Tredennick and Brion Shimamoto rightly point out in the accompanying article, "The Inevitability of Reconfigurable Systems," this dominance of the general-purpose processor is coming under pressure from a number of fronts these days, and it seems likely that other solutions will need to be constructed. (This does not mean, however, that the demand for simple processors will disappear. After all, how much computing does it take to run your microwave?)

As the authors note, one driving factor is power. Not only are we moving toward untethered systems, but also previous performance scaling unfortunately increased power dissipation along with performance. If you build the highest-performance solution you can think of in today's technology, you will likely be consuming more power than you can afford. Chips are now power-constrained rather than transistor-constrained, even in high-end desktop machines. When you look at general-purpose processors, they appear particularly power-inefficient compared with other approaches.

So now we face a true dilemma: What is the best computing platform going forward? Tredennick and Shimamoto claim that it will be reconfigurable systems. In some sense

systems. The waiting memory sockets in new untethered systems provide the investment incentive to develop new memory. It might be MRAM, FRAM, or OUM; it might be based on carbon nanotubes; or it might be something entirely new.

An interesting memory candidate from Axon Technologies, called *Programmable Metallization Cell memory* (PMCm), employs a solid electrolyte. Electrolytes transport electrons and ions (charged atoms) and are the working contents of most batteries. Axon sandwiches a solid electrolyte between two metal plates. One of the metal plates is silver, which has an ionization potential of 0.3 volts. Applying more than 0.3 volts across the plates ionizes silver atoms, which then migrate through the solid electrolyte away from the positive plate. When a silver

ion reaches the negative plate, it captures an electron. The migration of ions from the positive plate to the negative plate builds a physical bridge of silver atoms through the electrolyte, greatly reducing the resistance between the plates. The process takes about 10 nanoseconds and is completely reversible (reversing the voltage tears down the bridges by returning the silver ions to the silver plate).

I don't know which of the many Holy-Grail memory candidates will be the first to achieve volume production, but, with investment incentive and waiting sockets in untethered systems, it shouldn't be more than two or three years.

As stated earlier, even with application-oriented streamlining, SRAM-based PLDs were unsuitable for untethered applications. A new nonvolatile memory

they are right. Clearly having a large enough market requires systems that can be programmed for a number of different applications. Users will have to be able to reconfigure their hardware. The authors are also correct that the resulting hardware will be explicitly parallel. There will not be a single processor running a piece of code.

It is well known that if there is explicit parallelism in an application, doing that computation in parallel consumes less power for a given level of performance than performing the computation sequentially. While we know that future computing substrates will be parallel and reconfigurable, less clear is what the basic reconfigurable block will be. The current field programmable gate arrays (FPGAs) provide their customers with a field of logic gates that may be configured and reconfigured into complex logic. Use of these logic gates, however, is not optimal for a couple of reasons. First, the memory present to implement the reprogrammable interconnect creates a substantial overhead in power, area, and delay. Second, programmers of the FPGAs typically use a register transfer level (RTL) language such as Verilog or VHSIC (for very high-speed integrated circuit) Hardware Description Language (VHDL) rather than a programming language such as C, which would be much more familiar to most programmers. For reconfigurable systems to succeed, they need to develop a computation model that a language compiler may use.

Following this argument often leads people to conclude that the larger functional blocks should be processors, especially considering how the overall system will be configured/programmed. While we know how to program at the gate level by writing a hardware description language like Verilog or VHDL, which can be synthesized into logic gates, we don't want to force software programmers to work

at this level for the entire application. Clearly, programmers are going to need to work at a higher level of abstraction. Once programmers use these higher-level abstractions, they need to have a compiler that can translate these abstractions to configurable hardware. FPGAs are beginning to see higher-level abstractions through the use of libraries of larger components and Simulink-style block diagram editors. But it's not clear that gate-level FPGA is the best target for this type of programming.

Many people claim that the correct reconfigurable block is a processor as it matches the computation model the compiler understands. Yet it's up to the programmer to decide how all the processors can be used together to solve a large problem in this model—and history has shown that general parallel programming is not easy without sufficient tools.

There's no question that we can build reconfigurable hardware substrates, whether the blocks are processors or gates. The real question is how to program them. The key is to think about computation models, or programming abstractions, that fit a large class of applications and then find a computational substrate built out of transistors that they map well to. The programming abstraction most in use today is a synchronous data-flow model (sometimes called a stream computational model). This is the model that many of the tools mentioned in Tredennick and Shimamoto's article use; it is also the model used in the Simulink-style block diagram editors. The synchronous data-flow model works well for applications with large amounts of data parallelism, which is characteristic of many applications needing the higher-performance computation such as signal processing. Given this computation model, the question now is: What is the best computation substrate for a stream compiler? I don't think this will look like a reconfigurable system as we

The Inevitability of Reconfigurable Systems

will make application-oriented PLDs efficient enough for untethered systems. Replacing the PLD's SRAM personalization memory with the new nonvolatile memory will improve chip size, speed, and security. These application-oriented, nonvolatile PLDs will enable efficient reconfigurable systems: *variable hardware resources and variable algorithms on a generic chip.*

think about it today; nor will it look like a processor.

Many people are researching these types of machines, and my colleague, Bill Dally, is one of the leaders in this area. His proposal has a number of distributed simple processors, where the configuration looks more like a number of instructions merged together to form a very large instruction word (VLIW) program than FPGA bits [1]. Whether this turns out to be the "right" architecture for stream machines is still an open research question, but it clearly shows the solution may well look more like processors than FPGAs.

What will happen with memory devices is even harder to predict, but just as important to understand. While many exciting new technologies are on the horizon, displacing an existing standard is always difficult. The problem is that the standards for memory devices are extremely high. We expect dynamic RAM (DRAM) and erasable programmable ROM (EPROM) to have nearly 1 billion working memory bits on each device, dissipate less than one watt when active, and cost a few dollars. Getting any new technology to this point will take lots of money.

To make matters worse, designers have become clever about using a couple of devices to make the system look like it has a device with even better performance. For example, fast static-RAM (SRAM) caches mostly hide the fact that the DRAM is slow. In fact, they hide it so well that most people won't spend extra money/power to get faster DRAM. Similarly, you can use SRAM and EPROM to get a system that looks like it has nonvolatile SRAM. So keep your eyes open for new memory technology, but don't bet the farm on such new technology solving all your problems yet.

Clearly, in the future we will have chips that can be configured to perform a number of different functions. But like many projects, building the hardware is the easier part

RECONFIGURABLE SYSTEMS

The transition from instruction-based to reconfigurable circuits won't be easy. The industry has 30 years of experience in instruction-based circuits (embedded systems with microprocessors and digital signal processors) and programming as the way to solve problems. A huge installed base of development systems supports instruction-based implementations. The entire base of practicing engineers is experienced in and comfortable with instruction-based solutions. Universities teach instruction-based implementation. Corporations with billion-dollar businesses selling microprocessors and digital signal processors encourage instruction-based implementations.

The new design process itself is a barrier to the

of the problem. We know how to build these chips with the reconfiguration performed at the gate level (FPGA), at the instruction level (chip-level multiprocessors), and anywhere in between. Unfortunately, programming any of these chips to yield efficient solutions is still not solved, and it is the solution of this "software" problem that will eventually select the chip organization of the future.

NOTE

1. The VLIW instruction results from the compiler being able to statically schedule the many parallel execution units to operate concurrently each cycle. This approach is much simpler than the superscalar, out-of-order processor architectures of today's CPUs, which try to extract the parallel schedule dynamically in the hardware.

MARK HOROWITZ is director of the computer systems laboratory at Stanford University, and Yahoo Founder's Professor of electrical engineering and computer science. He received his B.S. and M.S. in Electrical Engineering from MIT in 1978, and his Ph.D. from Stanford in 1984. Since 1984 he has been a professor at Stanford working in the area of digital integrated circuit design. While at Stanford he has led a number of processor designs including MIPS-X, one of the first processors to include an on-chip instruction cache; Torch, a statically-scheduled, superscalar processor; and Flash, a flexible DSM machine. He has also worked in a number of other chip-design areas including high-speed memory design, high-bandwidth interfaces, and fast floating point. In 1990 he took leave from Stanford to help start Rambus, a company designing high-bandwidth memory interface technology.

transition from instruction-based systems to reconfigurable circuit-based systems. Instruction-based systems solve problems by programming. The original transition from circuits to instructions increased the pool of designers to include programmers with no expertise in logic design. A transition from instruction-based to reconfigurable systems, which shrinks the pool of designers to those with logic design expertise, will fail. A shift to higher abstraction, which *retains* programmers in the design pool, must accompany the shift. Product offerings from Accel, Celoxica, MathWorks, and others build reconfigurable circuit-based implementations from program-like specifications of system behavior. To keep the pool of designers, programs will evolve from being the (algorithm) procedure to being the circuit specification—that is, the role of the program will evolve from being instructions to being the specification that translates to circuit configurations on application-oriented PLDs.

The transition to reconfigurable systems is inevitable. The value PC marks the shift as engineering development follows the market to untethered systems. The design objective, however, changes from cost performance to cost-performance-per-watt. Microprocessors and digital signal processors have been exploited to their market limits in increasing performance and reducing operating voltage. Although they remain suitable for cost-performance-oriented systems, they lack the energy efficiency to meet the cost-performance-per-watt objectives of untethered systems. The microprocessor isn't going away; its role will change from workhorse to supervisor. The digital signal processor may not be so lucky. While signal processing requirements will continue to grow, instruction-based systems aren't efficient enough and will be replaced by reconfigurable systems.

Application-specific integrated circuits may meet the cost-performance-per-watt objectives of untethered systems (though this is open to debate), but ASICs fail because they are inflexible and too expensive. They also freeze the implementation and cannot adapt, so they cannot meet evolving requirements. Escalating costs for semiconductor equipment, development, and masks (the patterns for making chip layers) weigh against ASICs and favor PLDs. Generic in manufacture and customized in the field, PLDs meet the high-volume production requirements (which means low-cost chips) and the energy-efficiency requirements for untethered systems.

The shift to untethered systems breaks the incumbent memory components' lock on the market by offering new sockets for which the incumbents are unsuitable. The incentive to fill these new sockets is driving investment,

which will result in new nonvolatile memory. This new nonvolatile memory will enable a generation of application-oriented nonvolatile PLDs suitable for reconfigurable systems.

Long-running patterns encourage us to think they can go on forever. This is the engineers' view toward shrinking transistors, PC performance, microprocessor speed, and instruction-based solutions. In retrospect, we will someday view these times as engineering *phases* rather than as the institutionalized design approaches before us today. Future engineers will remember instruction-based circuits as a phase suited to cost-performance systems, and reconfigurable circuits as a phase suited to cost-performance-per-watt systems. Q

LOVE IT, HATE IT? LET US KNOW:

feedback@acmqueue.com or www.acmqueue.com/forums

NICK TREDENNICK is editor of the *Gilder Technology Report*. He is an advisor and investor in numerous pre-IPO startups and is a member of technical advisory boards for numerous companies. He is on the editorial advisory board for technical publications including *IEEE Spectrum* and *Microprocessor Report*. Dr. Tredennick was named a Fellow of the IEEE for contributions to microprocessor design. He has experience in computer and microprocessor design, holds nine patents, and has many technical publications. He was a senior design engineer at Motorola, a research staff member at IBM's Watson Research Center, and chief scientist at Altera. Tredennick was a pilot in the U.S. Air Force, in the Air Force Reserve, and in the Air National Guard. He has also been a Naval Reservist and a member of the Army Science Board.

BRION SHIMAMOTO has 30-some years of experience in computers, half of that in technical management. He's written realtime missile-tracking software for U.S. spy satellites and communications software at IBM. He has been an IBM systems engineer and he was a research staff member at the IBM's Watson Research Center in Yorktown Heights, NY. While there, he worked on fiber-optic I/O protocols and later managed the logic design of the first single-chip System/370 microprocessor. He convinced IBM to fund an entertainment startup, Digital Domain. Shimamoto was the vice president of technology at Digital Domain, a visual effects company. He has been director of Platform Technology Center at NCR, division manager at AT&T (IP networking services), and independent consultant. He coedits issues of the *Gilder Technology Report*.

© 2003 ACM 1542-7730/03/1000 \$5.00.