Written by
Nick Tredennick

DynamicSilicon

Published by
Gilder Publishing, LLC

The Investor's Guide to Breakthrough Micro Devices

Dynamic Logic vs. Computing

The desktop computer dominates the electronic systems market. It is a consumer item, which means it is built at the lowest possible cost. We're familiar with the legendarily low margins in the personal computer business. Manufacturers compete on performance, so the PC is also built for the highest possible performance. Even though this market segment contains many more devices than the desktop computer, we'll call it the *desktop computer segment*. That's the situation today; that situation is changing.

The world is splitting into tethered and untethered devices. Tethered devices support the global information grid: computing, access ports, data transport, and storage. The earth will look like an infinite source for on-demand computing. It will provide access ports everywhere (untethered devices need only enough bandwidth and transmitter power to reach the nearest access port). The earth will provide data transport from any port to any destination. And the earth will be an infinite repository.

Untethered (mobile) devices of the future are the *producers and consumers* of data. They have the sensors that collect information and they have the actuators and displays for the consumption of data. Mobile devices will adapt to the situation with computing, functions, and storage that are necessary to meet demands that change with time. They are consumer-oriented devices with high computational requirements. Because they are consumer products, they must be low cost (consumer markets are highly sensitive to cost). Because they are mobile products, they must have long battery life. Because they must calculate and communicate in a real-time environment, they must be built for performance. Consumers want low price, long battery life, and instant answers from their portable devices. This market is the intersection of the consumer, low-power, and performance segments. It is the most challenging and most rewarding segment to reach. It also leads the market as an indicator of the future—components in high-cost, low-volume products of today will migrate to the low-cost, high-volume products of tomorrow. I call this segment the *leading-edge wedge* (discussed in *Dynamic Silicon* Vol. I, No. 1).

In my career, I've been struck by lightning three times. The first time was in 1977. I was an assistant professor at the University of Texas when Tom Gunter walked into my office and asked me to design a microprocessor at Motorola. Actually, he asked if I would design the on-chip cache for MACS (Motorola Advanced Computer System—later changed to MC68000). I joined Motorola only to be immediately diverted from cache design into the microprocessor's logic design and microcode—"just until we can find someone competent to take over the logic design." I finished the logic design before anyone competent replaced me. It was a two-year, labor-intensive, pencil-and-paper design. I translated the English-language descriptions of instructions into equations and I translated those equations into the microprocessor's logic gates and microcode. The magic of engineering is its transformation of plain-language behavioral descriptions into working engines. I learned about microprocessor design.

In the late 1970s, transistors were scarce and electrical power was free. Speed and function were important. We wanted the fastest, most capable chip we could get within the transistor budget. I spent months of engineering time improving the functional efficiency of the design (to get the maximum benefit from every resident transistor). Speed was important: our target was 8 MHz. (I thought Tom was daft when he said that someday the MC68000 would reach 25 MHz.) We didn't want people using MC68000s as hot plates for coffee cups, but we weren't worried about power; we assumed the microprocessor's host system would be plugged into the wall.

Our goals: speed, performance, and capability—more is better. Our assumptions: transistors are scarce, design must be efficient (labor-intensive), and within reason, power is free. In the years since, the goals have stayed the same, but semiconductor progress has changed the assumptions. Chips are bigger and transistors are smaller. Today, there's an abundance of transistors and a shortage of designers (transistor design can't be as labor intensive as it was). Today's "more is better" leading-edge microprocessors operate above 1,200 MHz and dissipate a coffee-warming 50-60 watts.

As the world goes mobile, the microprocessor is trying to turn the corner by adding "low power" to its "speed, performance, and capability" goals. Vested interests will mount heroic efforts, but it's a losing cause. As we'll see, speed and low power conflict. The microprocessor is starting in the wrong place and it cannot get where it needs to go. The microprocessor's dilemma is an opportunity for a disruptive technology. I found a disruptive technology the second time I was struck by lightning.

In 1993, Bob Hartmann, founder and VP of Business Development at Altera, walked into my office and asked if I wanted to be Chief Scientist. I did. Altera's circuit designers built its business on programmable logic devices (PLDs), integrated circuits that let the engineer "program" logic circuits. Semiconductor progress had carried integrated circuits past a million transistors. Who would be the customers for these and future parts? I learned a lot about the programmable-logic market. Programmable logic has the potential to be a disruptive technology. **Altera** and **Xilinx** had (and still have) the lion's share of the programmable-logic market. Could PLDs from Altera and Xilinx be the disruptive technology that displaces the microprocessor in the mobile world of the future? No. Altera and Xilinx are growing as fast as they can to support customers they already have. These companies have a great future, but they are trapped by their customers and, therefore, are unable to take advantage of this new opportunity. New companies will exploit the opportunity. Their disruptive solutions will create enormous wealth for themselves and for their backers. I know how this might happen, which brings us to the third time I was struck by lightning.

I got interested in microelectromechanical systems (MEMS) and had been collecting clippings for a couple of years when George Gilder asked if I wanted to write a newsletter on breakthrough micro devices. I did. Microprocessors were breakthrough micro devices. PLDs were breakthrough micro devices. Microelectromechanical systems are today's breakthrough micro devices. Because I have been struck by lightning at the right times, I have both the necessary experience and the platform to tell you where we are in microprocessors, where we're headed, how we're getting there, why that won't work, and what disruptive technologies will be creating enormous wealth in the near future. Here's a sample.

The microprocessor's commercial introduction in 1971 disrupted and displaced the established market for integrated-circuit macro functions. In thirty years, the microprocessor grew from nothing to billions of units annually, making Intel one of the most successful companies of all time. The microprocessor market will continue to expand. In the future, however, derivatives of PLDs will disrupt established microprocessor market segments. They will do so because increased demand for portable devices will change the goal from performance to efficiency. Programmable logic devices that conserve clock frequency will displace microprocessors that waste it.

Engineering problems

The microprocessor, and the computer more generally, represented a fundamental breakpoint, a historical demarcation dividing engineering history into two eras, the first lasting tens of millennia, and the second now barely in its fifth decade.

Until fifty years ago, all engineers worked in the same way: they solved problems by building the solution, "mapping the application" directly into hardware. Any intelligence required by the solution was manifested in the physical structure of the device. At first blush this seems obvious.

Imagine graphing all the engineering problems of the world onto fig. 1, below, which rates problems and their solutions in two ways: by the magnitude of the problem (horizontal axis) and by the speed required to get the solution (vertical axis)—an answer that arrives too late to apply is no solution. In today's terms, electronics for blenders and toasters would be close to the origin, while systems for real-time weather modeling and for aircraft flight control would be far from the origin both in performance and in problem size.

DynamicSilicon

Editors	**Nick Tredennick**
	Brion Shimamoto
Publisher	**Lauryn Franzoni**
Associate Editor	**Keri Geiger**
Designer	**Julie Ward**
President	**Mark T. Ziebarth**
Chairman	**George Gilder**

DynamicSilicon is published monthly by Gilder Publishing, LLC. Editorial and Business address: P.O. Box 660, Monument Mills, Housatonic, MA 01236. Copyright 2000, Gilder Publishing, LLC. Editorial inquiries can be sent to: bozo@gildertech.com Single-issue price: $50. For subscription information, call 800.229.2573, e-mail us at dynamic@gildertech.com, or visit our website at www.dynamicsilicon.com

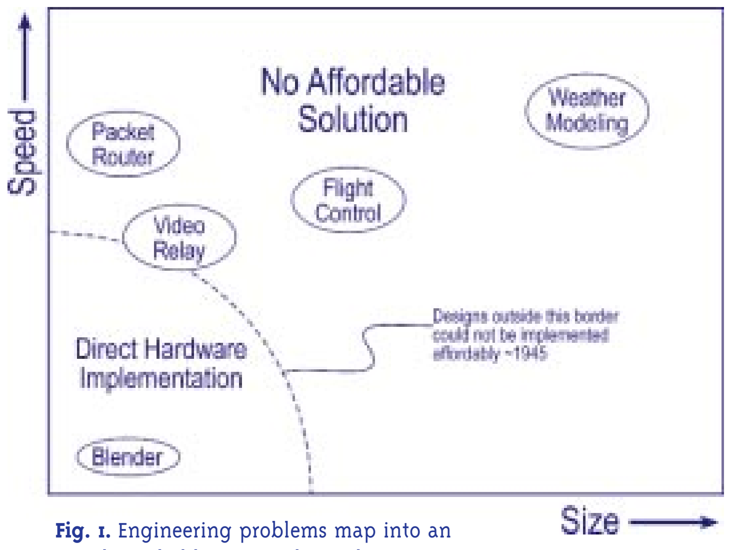
No Affordable Solution
Weather Modeling
Packet Router
Flight Control
Video Relay
Designs outside this border could not be implemented affordably ~1945
Direct Hardware Implementation
Blender
Speed
Size

Fig. 1. Engineering problems map into an area bounded by size and speed.

For most of history the only problems engineers could solve were clustered down and to the left, because with hardware only we usually cannot solve very big problems.

Where we are

The invention of the computer was a breakthrough in problem solving. The computer separated the algorithm (rigorous steps that solve a problem) from the hardware and broke the affordability constraint. The computer amortized expensive hardware resources over time by iterating to solve the problem. A single computer could solve a range of problems (amortizing the cost of expensive hardware across applications). The computer could solve a problem of any size—provided you had the time to wait for the answer. The computer opened a previously inaccessible region of engineering problems: large problems with limited performance requirements (down and to the right in fig. 2).

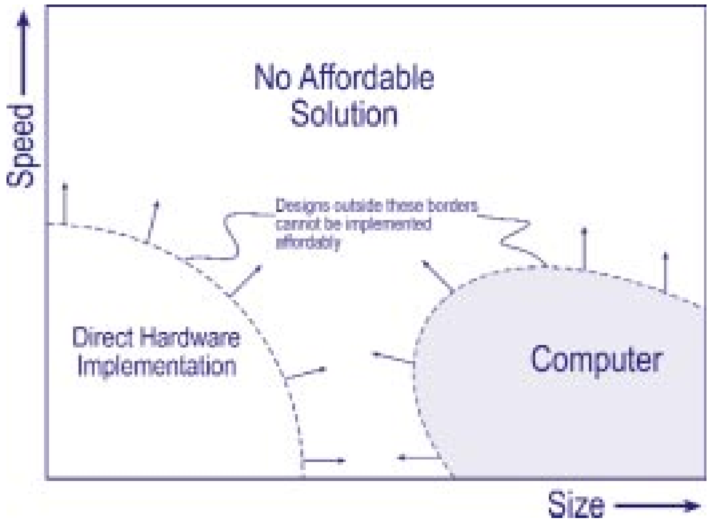
No Affordable Solution
Designs outside these borders cannot be implemented affordably
Direct Hardware Implementation
Computer
Speed
Size

Fig. 2. The computer is a breakthrough problem-solving method. It amortized expensive hardware over time to solve large problems.

In 1943, IBM's founder Thomas J. Watson, Sr., predicted that five large computers would satisfy the world need. Invention of the solid-state transistor and the integrated circuit made that one of the worst predictions ever made. With billions of microprocessors shipping every year, Tom's projection was off by at least ten orders of magnitude. What an interesting world where an expert's prediction could be off by that much!

The transistor and the integrated circuit (IC) accelerated the design of electronics and fostered rapid penetration of electronics into consumer markets. ICs are prepackaged circuits on a single piece of silicon (chip). The integrated circuit made the engineer's job easier because integrated circuits made building blocks. Think of them as Lego blocks for electronic systems: the engineer didn't have to think about each individual transistor, resistor, and capacitor in a design. IC macros, most notably the transistor-transistor logic (TTL) family, quickly displaced discrete components in electronic systems. TTL began as a small family of elemental logic chips (inverters, latches, flip-flops, Boolean gates, half-adders, etc.)—the building blocks of complex logic systems. As semiconductor fabrication improved, the TTL family grew in complexity and diversity, enabling the designer to build more complex logic systems from fewer chips.

The integrated circuit, together with continuing improvements in semiconductor fabrication extended the range of affordable solutions for both direct hardware implementations and for the computer. Computers became faster and more capable and they became smaller and cheaper. Mainframe computers extended the range of affordable solutions to problems requiring more performance. The minicomputer encroached on the low end of the mainframe's problem range and it brought affordable solutions to smaller problems. The arrows in fig. 2 indicate the expansion of the range of affordable solutions. New components got faster and more complex, extending the range of affordable solutions to larger problems and to problems with higher performance requirements. The same semiconductor fabrication progress that made integrated circuits faster and more complex also made old ones smaller and cheaper. Rapidly expanding applications led to higher component production volumes that drove the integrated circuit down the manufacturing learning curve to better performance and to lower cost.

Semiconductor fabrication improvements led to the microprocessor. The microprocessor was originally intended as a logic replacement device. That is, engineers built systems with a microprocessor, memory chips, and a few standard peripheral chips instead of building them with TTL blocks. The microprocessor displaced TTL for the same reason TTL had displaced discrete components: it made design easier. The microprocessor brought the computer's problem-solving methods to the region of engineering problems that had previously been solved with TTL building blocks. Problem solving became programming (see fig. 3).

The microprocessor is not as efficient as TTL building blocks because the microprocessor is an indirect solution. TTL building blocks construct the function directly; the indirect solution programs general-purpose logic (the microprocessor's arithmetic and logic resources) to construct the desired function. The general-purpose microprocessor's solution is less direct and less efficient, but it leads to wider application. Wider application means higher production volumes: the microprocessor trades efficiency for production volume. This is a smart trade as long as the microprocessor's inefficiency isn't important to the application.

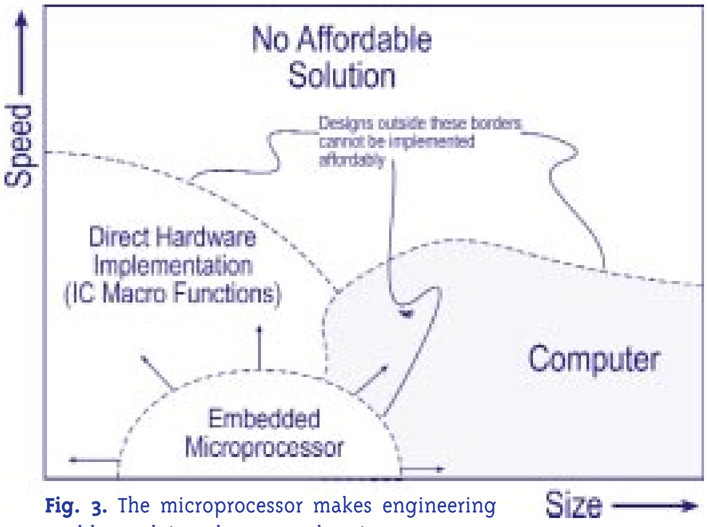
No Affordable Solution
Designs outside these borders cannot be implemented affordably
Speed
Direct Hardware Implementation (IC Macro Functions)
Computer
Embedded Microprocessor
Size

Fig. 3. The microprocessor makes engineering problem solving cheaper and easier.

For semiconductors, high-volume production leads to rapidly decreasing cost and to improved performance. Semiconductor fabrication improvements increase the microprocessor's speed to help overcome inherent inefficiencies in programmed solutions, leading to wider application. The microprocessor became fit for consumer applications (i.e., low cost and adequate performance).

With continuing improvements in semiconductor fabrication the microprocessor's performance increased to the point that it started to encroach on the domain of the minicomputer. The microprocessor made the transition from embedded systems to computers in 1974 (only three years after the microprocessor's commercial introduction). It took some time for the microprocessor to be accepted as the brain in computer systems, but that happened in 1981 when IBM introduced its Personal Computer based on the Intel 8088. With the IBM personal computer, microprocessor design transitioned from an emphasis on low cost and on adequate performance to an emphasis on performance.

How we're getting there

Design for the desktop-computer segment has dominated the industry. The leading microprocessors for desktop computers are designed for maximum performance. The microprocessor's performance on a program is proportional to its speed (clock frequency, f) times the number of instructions it executes per clock tick (instructions per clock, ipc), divided by the number of instructions (n) in the program:

$$\textbf{Performance} \approx f\cdot\textbf{ipc/n}$$

Engineers improve microprocessor performance by writing programs that use fewer instructions, by adding more powerful instructions, by increasing the number of instructions to execute each clock tick, and by increasing the clock frequency. Adding more powerful instructions and increasing the number of instructions to execute each clock tick increase the complexity of a microprocessor's design.

Engineers take advantage of the improvements in semiconductor fabrication by loading the design with more transistors and by increasing the clock frequency. With this combination of increasing circuit complexity and rising clock frequency, engineers have been able to double microprocessor performance every eighteen months. That works for the desktop-computer segment and it works for the performance segment. It's a good strategy for applications that connect to the power grid. This strategy (increasing complexity and raising clock frequency) isn't a good fit for leading-edge mobile applications. The requirement for long battery life gets in the way.

Power dissipation in digital systems is approximately one half times the capacitance times the square of the voltage times the frequency:

$$\textbf{Power} \approx {}^1/_2\cdot c\cdot V^2\cdot f$$

The compromise that engineers make, designing for mobile applications, is to lower the voltage. Since the voltage term is squared in the power-dissipation equation, a small change in the voltage means big savings in power. Lower the voltage by half and power dissipation drops to one-fourth of its former value. But lowering the operating voltage lowers the maximum operating frequency, which has the undesirable consequence of decreasing performance. The fastest microprocessors for laptop computers are a couple of speed grades below the microprocessors for desktop computers because the laptop microprocessors need to run at lower frequency and at a lower voltage to extend battery life. The latest designs run faster when the laptop is connected to an external power source.

From the above formulas for performance and power, we can see the microprocessor's dilemma. To increase performance, the microprocessor wants to run at higher frequencies, but to limit power dissipation it wants to run at lower frequencies.

How we're getting there (from engineering designs for the present to designs for the future) derives from the methods engineers use. Engineers design with microprocessors because microprocessor-based design methods are simpler and lead to cheaper solutions than other methods and because that's what they learned in school. Selecting a microprocessor and programming a solution is more efficient for the designer than direct hardware implementation, conserving the critical resource (the engineer's time). But microprocessor-based designs are significantly less efficient than pure hardware-based ones.

I'm sure you've heard some variation of the joke about the engineer and the physicist: they are both trying to reach some goal, but at each iteration are only allowed to cover half of the remaining distance. The physicist gives up knowing that it will never be possible to reach the goal, since some distance will always remain. The engineer gets pretty close and declares, "that's enough for practical purposes; I've reached my goal." This illustrates an essential difference between an engineer and a scientist. The scientist may strive for the perfect solution. The engineer, knowing the enormous cost of achieving a perfect solution, is satisfied with a good solution. It is a tradeoff between effort and efficiency. A good engineer knows how much inefficiency the solution can afford in minimizing the design effort.

Direct hardware implementation leads to efficient designs (see fig. 4). The engineer converts the application to an algorithm that is implemented directly in hardware. There's little loss of efficiency in that process, but it is more taxing for the designer. A process based on a microprocessor is simpler. It's simpler to write a high-level language program than it is to design custom hardware and a custom state sequencer to accomplish the same task and it's easier to correct errors too.

Direct-Hardware Implementation

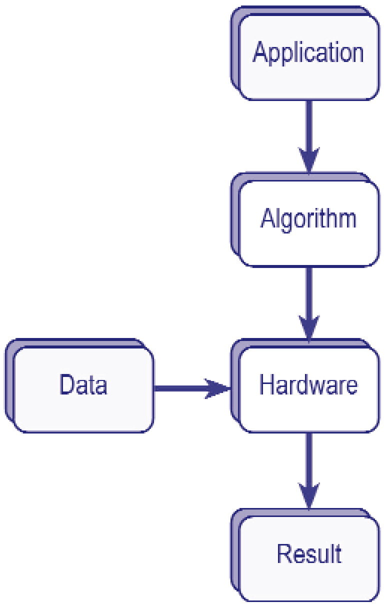

Fig. 4. Direct-hardware implementation leads to efficient designs.

Microprocessor-based implementations lead to less efficient designs (compare the steps in fig. 4 with the steps in fig. 5). In a microprocessor-based application, the engineer must consider the microprocessor's architecture in selecting the algorithm for implementing the application (a floating-point-intensive algorithm won't run well on a microprocessor that doesn't have floating-point instructions). The engineer then codes the algorithm in a high-level language such as C++ or Java. A compiler converts the high-level language into binary instructions the microprocessor will understand. Efficiency is lost at each stage of the process.

Computer Implementation

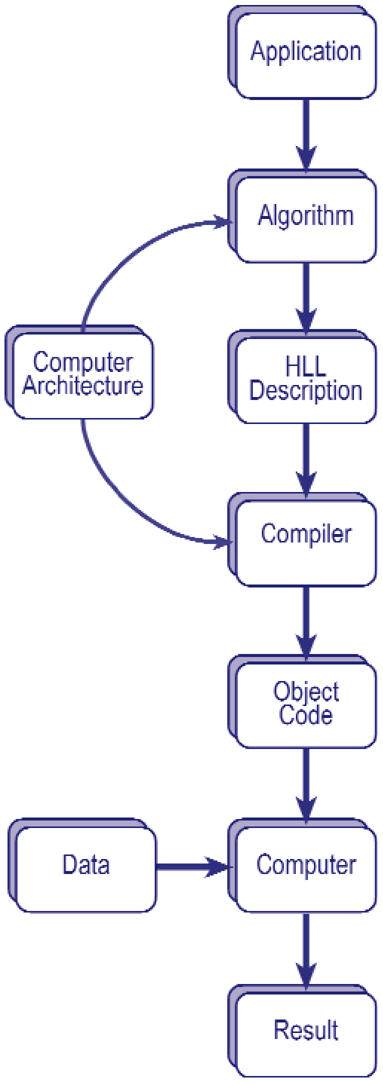

Fig. 5. Microprocessor-based implementations forfeit efficiency in translations.

Today's designs for leading-edge mobile applications are based on microprocessors and on digital signal processors (DSPs). Designs can't be based completely on microprocessors and DSPs, however, because the implementation would use too much power to get the necessary performance, so designers compromise by moving some functions into hardware. Application-specific integrated circuits (ASICs) implement compute-intensive and power-hungry functions, so the microprocessor or DSP can run at a lower voltage and at a lower frequency, saving power.

Why that won't work

Consumer expectations are rising. We want our cell phones and other wireless access devices to work anywhere. We don't want to be concerned with frequency bands and protocols. We want to be able to make and receive calls anywhere. We want universal paging. We want real-time stock quotes. We want wireless access to email. And we want it today. Since the expensive, difficult-to-change installed base is a dog's breakfast of protocols and frequencies, it's left to the engineers to implement multi-band, multi-mode handsets. That requires more performance from the microprocessor and it requires a separate ASIC for each protocol (and sometimes different DSPs for different protocols), increasing the cost of the design and adding to the number of components in the bill of materials. More components and more functions burn more power.

Designers can count on progress in semiconductor fabrication for some improvement, but we're running out of room lowering the voltage (some microprocessors specify less than a volt in low-power applications—from five volts a few years ago—getting close to where the chip will quit working). When we run out of voltage, we've lost most of our control in improving the performance of microprocessor-based designs. Performance increases directly with clock frequency, but so does power dissipation. In the world of microprocessors, more complexity and higher frequency have been the path to more performance. What we need is a way to improve the efficiency of the design so we can do the same work at a lower frequency.

The microprocessor began as an embedded element and was soon converted to a performance-oriented design (with the current designs appearing as the CPU in computer systems and the older generations relegated to embedded applications). It is as if we began a journey in San Francisco (the first embedded systems) attempting to optimize a path to New York (the ultimate-performance microprocessor). We have reached Kansas City (today's leading-performance microprocessor) and then changed our goal to Los Angeles (the ultimate in power efficiency). We need to optimize the path from San Francisco to Los Angeles, but we insist on keeping Kansas City (microprocessor-based design) on the route. There may be a better way.

New approach: dynamic logic

The conceptual model for a programmable logic device is that it is a chip with two layers. One layer is an array of logic elements and a general interconnect-structure (wires). The second layer is a personalization memory (see fig. 6). The bit pattern loaded into the personalization memory controls connections between the logic elements and the interconnect-structure. By loading different bit patterns into the personalization memory it is possible to connect or not connect logic elements to build arbitrary logic functions (within the limits of the PLD's capacity). In this sense, PLDs are programmable—the function that the chip performs is different depending on the ones and zeroes in its personalization memory. TTL built complex electronic circuits by physically connecting logic building blocks; PLDs build complex electronic circuits by programmably connecting logic building blocks. The design potentially has the efficiency of a direct hardware implementation, but the component is programmable. Being programmable, one chip fits a broad range of applications, so, like the microprocessor, the PLD can achieve the Holy Grail of the semiconductor business: the high volume that leads to cheaper, faster parts.

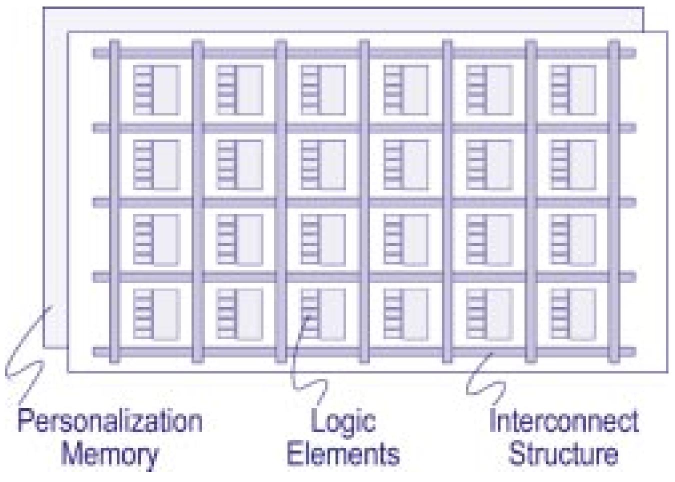

Fig. 6. The programmable logic device (PLD) has logic and wires on one level and personalization memory on a second level.

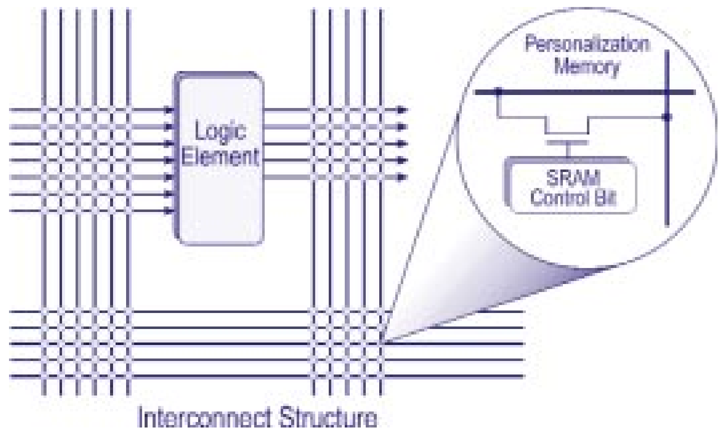

Fig. 7. The personalization memory forms the connections that build arbitrary circuits.

My diagrams, simplified as they must be for clarity, are a poor caricature of the devices themselves. High-end contemporary PLDs contain tens of millions of transistors. Personalization memory can be more than ten million bits (each bit implies a transistor switch controlling a part of the reconfigurable function as shown in fig. 7, above).

The bits in the personalization memory can interconnect more than fifty thousand logic elements (about 2.5 million logic gates plus a half-million bits of on-chip memory) such as that shown in fig. 8. The logic element itself can be configured into a variety of functions (the look-up table in fig. 8, for example, can be any function of four variables) and can be aggregated into larger functions such as adders, multipliers, encoders, decoders, and engines for direct encryption and decryption.

In the past, PLDs suffered from a number of seemingly crippling problems that prevented their use in leading-edge mobile applications. General-purpose PLDs had more overhead than the U.S. government—about twenty overhead transistors for every transistor doing real work. PLDs were slow. Their programming was all or nothing (no partial configuration and no background preloading of the next personalization pattern). It took a long time to load the personalization memory. They had limited capacity. They were expensive.

Semiconductor fabrication improvements solve many of these problems (speed, capacity, cost, and programming time). The first microprocessors, with a few thousand transistors, exhibited impressive capabilities. Each logic element in a modern PLD (see fig. 8) contains about a half as many transistors as the earliest microprocessors did. Think of the power of fifty thousand logic elements on a single chip—it's the raw logic equivalent of about 25,000 early microprocessors.

But, you will hear, "PLDs are slow and microprocessors are fast." Don't get hung up on the clock frequency. Comparing frequency of operation between a microprocessor and a PLD is like comparing blade speed on the Huffy Lawn King to screw speed on the QE II. Blade speed on the Huffy trims grass before it can bend out of the way; the QE II's screw pushes tens of thousands of tons of mass through the water. Frequency and blade speed do not measure the work that is being done.

In addition, many of programmable logic's shortcomings are not fundamental, but rather are an outgrowth of the requirements of mainstream applications for the devices. Programming times, for example, don't have to be slow; manufacturers put the chips in a special mode and quickly load configurations for testing. Configuration loading could be quick if there was demand from high-volume customers. Partial configuration and background configuration features are absent because there hasn't been great demand for them, not because they are difficult or impossible to provide.

The PLD market has been growing rapidly for at least the last fifteen years (at about thirty-five percent per year). The leading PLD manufacturers, Altera (1983) and Xilinx (1984), have been growing as rapidly as they can to support the available market. The PLD manufacturers are trapped by their customers, whose designs plug into the power grid (Altera and Xilinx are focused on the performance segment and are not aimed at leading-edge mobile applications). Altera and Xilinx are growing with their customers and they are moving up market to faster, larger components. There's an opportunity to exploit the potential efficiency of the PLD for leading-edge mobile designs.

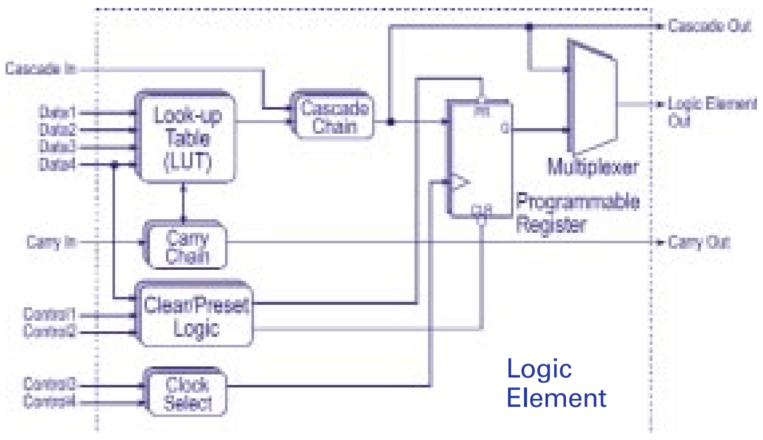

Fig. 8. The logic element in a PLD is the building block of arbitrary circuits.

Programmable Logic Implementation

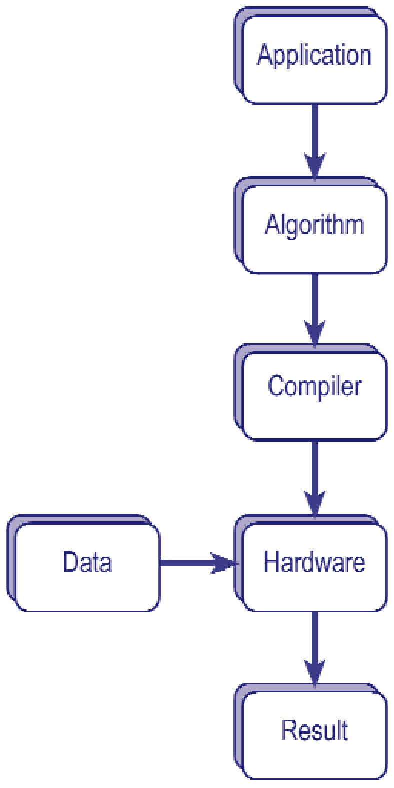

Fig. 9.
A programmable -logic implementation can be more efficient than one based on a microprocessor.

Imagine that you are in Florence Italy, looking east from the south bank of the Arno River, at the scenic sight of the Ponte Vecchio. There's a light fog, but you want a picture. If you have the right filter, you can get a good picture in spite of the fog. If you have my luck, however, as you lean out for that perfect shot your lens filter pops off and sinks in the Arno. You take the picture without the filter. You know you can later download it to a computer and fix it with Photoshop. This example illustrates the issue of energy efficiency: the lens filter processes the entire scene in real time for no net energy cost (the lens filter doesn't need batteries). If you forget the filter and process the picture in a computer, Photoshop will process each pixel in the image serially. In fact, Photoshop will look at a window of pixels around each objective pixel to calculate the value of each output pixel. It will take the program at least thousands of instructions to process each pixel. The PC will process several million pixels one at a time. This is not an efficient process and, as we have already demonstrated, the underlying microprocessor-based implementation is also inefficient.

The lens filter and the microprocessor-based implementation represent two extremes in the spectrum of information processing (the lens filter does a lot of work for not much evident effort while the computer exerts considerable effort for the same work). Transformation equations represent the effect of the filter on the image passing through it. Since we know the equations of transformation, we can build a direct hardware implementation of their representation using a PLD (to the limit of its capacity). This concept translates to the cell phone or to the intelligent mobile assistant. Appropriate logic is "paged" into the PLD to implement temporal, demand-driven functions. The dynamic-logic implementation may be able to do at 50 kHz what the microprocessor-based implementation needs 500 MHz to accomplish. Since power dissipation is proportional to the clock frequency, this translates into dramatically lower power in the dynamic-logic implementation. In leading-edge mobile applications, conservation of clock frequency becomes the first measure of efficiency.

Direct hardware implementations can be thought of as a combination of fixed hardware and fixed algorithms. Direct hardware implementations are the most efficient and are the least flexible. Microprocessor-based implementations combine fixed hardware with dynamic algorithms. Dynamic-logic implementations combine *dynamic hardware with dynamic algorithms*. Dynamic logic is a breakthrough problem-solving method just as the computer was a breakthrough problem-solving method. Dynamic logic is more efficient and more flexible than a microprocessor-based implementation.

Challenges to dynamic logic

There are seemingly insurmountable barriers to commercial success for dynamic-logic implementations. First, there's no dramatically successful commercial proof of concept for the proposition that a dynamic-logic implementation is substantially more efficient than a microprocessor-and-DSP-based implementation. This makes it hard for a startup to obtain funding. Venture capitalists are sheep: either all will invest or none will. There are few true pioneers in the venture community, their reputation notwithstanding, VCs are risk-averse. Further, the VCs' due diligence experts are likely to come from an application-related *mainstream* background with *conventional* design biases favoring direct hardware implementation or microprocessor-based implementation; they're not likely to understand or appreciate a radically different design method. And Wall Street? We may as well be talking about what kind of cheese the moon is made of.

Second, engineering skill in dynamic-logic design methods is scarce. Practicing engineers use direct hardware design methods and microprocessor-based design methods for new implementations. Universities train engineers in logic design and they train them in microprocessor- and computer-based design methods. (Ironically, universities use PLD-based development systems to teach both logic design and computer design, but they do not teach dynamic-logic design, so the PLD is contributing to the entrenchment of competing design methods.) There's no widespread understanding of dynamic-logic design methods, there's no expertise for instruction, and there's no apparent demand for the skill.

A dynamic-logic implementation will be expensive and time-consuming to develop (the long development time makes recruiting harder and its high cost makes funding harder). Building the partnerships for successful market penetration will be challenging because new designs will attempt to displace powerful entrenched companies.

How it might happen

Barriers to commercial success prevent overnight conversion to dynamic-logic design methods, but it can still happen. One company must collect one design team with dynamic-logic design skills and it must pioneer its way to a commercially successful proof of concept. If that company gains a substantial competitive advantage, others will follow. It's possible. A dynamic-logic implementation of a cell phone would be able to track protocol changes through over-the-air upgrades. Protocol changes would no longer mean another landfill of cell phones as the provider upgrades software and ASICs in its designs to accommodate transitions. That may be an incentive for service providers. The ability to adapt to different protocols and multiple bands efficiently would be attractive both to the consumer and to the service provider. Finally, the extended battery life of a dynamic-logic implementation would be attractive to consumers.

I met with Jaime Cummins and John Watson several years ago, before they started **QuickSilver Technology**, **Inc**. I met with them because I wanted to introduce them to some venture capitalists who I thought should be interested in funding their breakthrough idea (it was part of my education concerning sheep-like behavior and risk-aversion among venture capitalists). John and Jaime struggled for years looking for funding, until they encountered Gordon Campbell. "Gordy" is a venture capitalist, but he isn't an MBA-type venture capitalist; he's a visionary engineer who founded and ran Chips and Technologies (the first fabless semiconductor company) and now runs Techfarm. He knows a breakthrough idea when he sees one: Techfarm is funding QuickSilver.

QuickSilver is building a chip set for a next-generation cell phone using dynamic logic (QuickSilver calls this an Adaptive Computing Machine). Logic for each of the phone's protocols and functions can be "paged" into the chip's programmable logic, eliminating the need for a digital signal processor, ASICs, and possibly even the usual microprocessor. Functions that are not paged into the chip's gates do not use power. Efficiency improves because the implementation is more direct for each function than it is in a DSP-based implementation. The DSP-based implementation runs a variety of functions on a fixed set of resources, giving up efficiency for the sake of simplifying the programming and the hardware resources. The dynamic-logic solution gives up efficiency in "paging" functions into the programmable logic. QuickSilver's bet is that paging the logic into the chip will cost less power than having logic that is always resident but mostly idle. Compare the block diagram of fig. 10 with fig. 11, for example.

QuickSilver will circumvent the inefficiencies of PLDs designed for the commercial prototyping market by designing its own devices to suit just the anticipated range of applications. QuickSilver's Adaptive Computing Machine (ACM) will still be a PLD, but it will be designed for rapid partial reconfiguration to accommodate "paging" of its functions. QuickSilver's ACM will

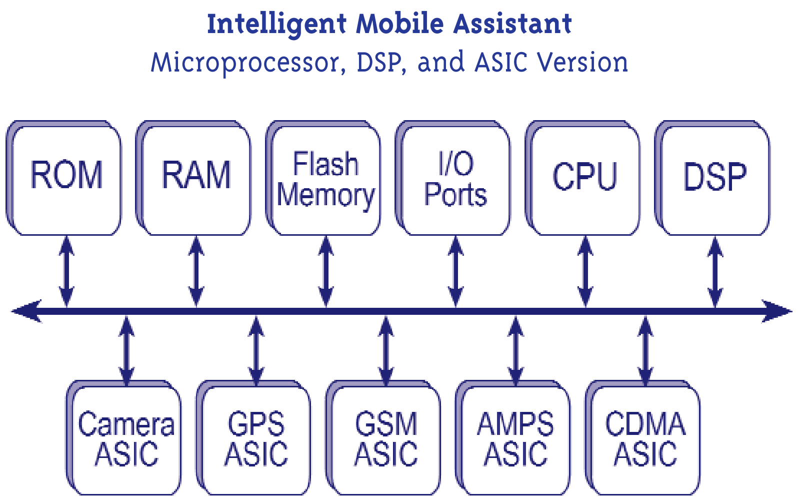

Fig. 10. All functions are resident in a microprocessor-based mobile assistant.

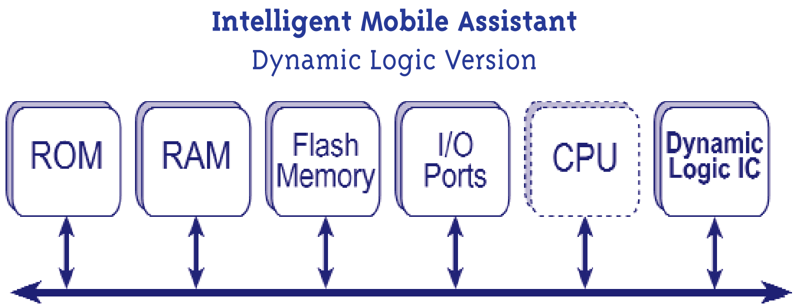

Fig. 11. The dynamic logic version of the mobile assistant "pages" functions as needed.

allow background reconfiguration while the device is operating. It may even cache high-use logic functions for more efficient paging (though this would cost valuable power). In addition, QuickSilver's ACM, since it is not being designed for general-purpose prototyping, can be designed with much less overhead than a commodity PLD, which typically has about twenty transistors of overhead for each transistor doing real work. In addition, the peripheral circuitry can be designed to suit a single system rather than being the general-purpose, universally configurable I/O pad ring required by PLDs for prototyping applications.

One application for QuickSilver's ACM is a mobile device that could be a multi-mode, multi-protocol cell phone. This phone might allow roaming among protocols such as AMPS, CDMA, TDMA, and GSM and even among frequency bands. Since it is "adaptive," it could be updated to the third or fourth generation standards as they evolve. In addition to the cell phone functions, the device could accommodate a variety of other functions such as calendar, calculator, email, GPS, and MP3. It might ship with a standard set of functions that could be "paged" from ROM and a bank of flash memory to accommodate changes and for installation of new functions. These changes and new functions might be loaded over the air interface, keeping the device functional and current in the field much longer than devices based on ASICs and ROM programs.

In leading-edge mobile applications, such as QuickSilver's example mobile device, the electronic system must meet the conflicting demands of compute-intensive algorithms and of long battery life. The processing requirements of these applications can be demanding across a range of tasks. The cell phone, for example, must do call setup, call teardown, encoding, decoding, and a variety of protocol processing tasks. These applications typically require several application-specific integrated circuits, a digital signal processor, and a microprocessor. The world has not converged on a single cellular standard and is not likely to any time soon, so there is demand for multi-protocol cell phones. A separate ASIC typically supports each protocol. Increasing popularity of email and the demand for wireless connection to the Internet are driving these functions into the cell phone as well. Each of these added functions increases processing complexity and makes extending battery life more difficult. Derivatives of the programmable logic device offer a way out of this difficult situation. QuickSilver's Adaptive Computing Machine uses dynamic logic. That is, ACMs implement dynamic algorithms and dynamic resources.

QuickSilver's ACM can be significantly more efficient than a DSP-based implementation of the same functions. Resources on the chip can be allocated to the limit of availability for parallel calculation, since the resources are not dedicated to particular functions, as they would be in a microprocessor, DSP, or an ASIC. A large fraction of the fixed resources in a microprocessor or DSP may be idle at any particular time. DSPs generally work on data in multiples of a byte. Dynamic-logic implementations can work on any data width (the width can even vary with time to suit the needs of the problem).

As semiconductor fabrication improves, DSPs and microprocessors are built with ever more fixed resources and running at ever higher clock speeds, so they can tackle ever more complicated functions. But, while adding resources and increasing the clock rate improve computational ability, they don't improve the computational *efficiency* of these processors. Each of the functions "paged" into a dynamic-logic implementation makes efficient use of the resources it needs and is then overwritten by the next function "paged" into the chip. Computational efficiency is high, so power dissipation is low. A dynamic-logic implementation can improve power dissipation by a factor of two to ten (compared with a DSP-based implementation) and can improve performance by a factor of ten to one hundred.

[illegible] [illegible]

Nick Tredennick and Brion Shimamoto
7 March 2001

Dynamic Silicon Companies

The world will split into the tethered fibersphere (computing, access ports, data transport, and storage) and the mobile devices that collect and consume data. Dynamic logic and MEMS will emerge as important application enablers to mobile devices and to devices plugged into the power grid. We add to this list those companies whose products best position them for growth in the environment of our projections. We do not consider the financial position of the company in the market. Since dynamic logic and MEMS are just emerging, several companies on this list may be startups. We will have much to say about these companies in future issues.

Altera and Xilinx (ALTR http://www.altera.com) (XLNX http://www.xilinx.com)
Altera and Xilinx together dominate the programmable logic business, with almost 70 percent of the CMOS PLD market. Both companies are aggressive and competitive. Sixty-six percent of Altera's revenue comes from the rapidly growing communications segment (Telecosm companies) and an additional 16 percent comes from the electronic data processing (EDP) segment. Altera and Xilinx are positioned to be major suppliers in tethered applications such as the base stations that support mobile devices.

Analog Devices (ADI http://www.analog.com)
Analog Devices is a leader in analog electronics for wireless RF and communication, MEMS for automotive applications (accelerometers, pressure sensors, transducers), and in DSPs.

ARC Cores (ARK (London) http://www.arccores.com)
ARC Cores makes configurable processor cores. Configurable processors allow the application engineer to adapt the processor's instruction set to the requirements of the problem. Conventional microprocessors have fixed instruction sets.

Cypress (CY http://www.cypress.com)
Cypress Microsystems builds components for dynamic-logic applications. Cypress also builds MEMS and is a foundry for MEMS.

QuickSilver Technology, Inc. (* http://www.qstech.com)*
QuickSilver has the potential to dominate the world of dynamic logic for mobile devices (untethered). While many companies work on programmable logic and on "reconfigurable computing" for tethered applications, QuickSilver builds adaptive silicon for low-power mobile devices.

SiRF (* http://www.SiRF.com)*
SiRF builds RF GPS chips for the mobile market. It is a world leader in development of integrated GPS receivers.

Transmeta (TMTA http://www.transmeta.com)
Transmeta makes new generation microprocessors that use closed-loop control to adapt to problem conditions in an x86-compatible environment. This enables Transmeta's microprocessors to save power over conventional microprocessors from AMD and Intel. The base instruction set is not available to the application engineer.

Triscend (* http://www.triscend.com)
Triscend builds microcontrollers with configurable peripheral functions and with configurable inputs and outputs. Triscend helps consolidate the microcontroller market into high-volume, standard chips.

DynamicSilicon

Monument Mills
Housatonic, MA 01236
www.dynamicsilicon.com