VOL. 10, NO. 10

SOMETHING COMPLETELY DIFFERENT

The Case for Reconfigurable Computing

Will Intel Become a Victim of Its Own Success?

by Nick Tredennick

I tried to explain the case for reconfigurable computing at the Microprocessor Forum in the fall of 1994. I failed completely. Perhaps I failed because I had only 10 minutes to make the case before an audience to whom the concept was new. Perhaps I failed because I'm just not articulate. Give me another 10 minutes at the next conference and I'll probably fail again. Part of the fault lies with the commercial introduction of the microprocessor in 1971 and the subsequent explosion of microprocessor sales from almost nothing at introduction to a worldwide market exceeding four billion units just 25 years later. So how's that a problem that's partly responsible for my inability to get a concept across? Let me explain.

Computers Introduce a Method

The invention of the computer started us down a particular path with respect to problem-solving methods. Before the commercial introduction of the computer, engineers solved problems by building custom hardware from discrete components to implement both the computational resources and the processing algorithm. After the invention of the integrated circuit, engineers solved problems more efficiently by designing with TTL macro functions. The state sequencer and the computational resources were implemented in hardware (TTL macro functions).

The computer changed the basic problem-solving method, because the computer implemented fixed computing resources and variable algorithms (in memory). Implementing algorithms in memory allowed the cost of expensive hardware computing resources to be amortized across a range of applications. It also allowed the computer to iterate to solve a problem, which amortized fixed computing resources in time: trading time for hardware. With a computer, the engineer could solve problems too large to be affordably tackled with custom hardware—provided there was time to wait for the answer. Before the computer: fixed computing resources and fixed algorithms. After the computer: fixed computing resources and variable algorithms.

Microprocessors Entrench the Method

Twenty-five years of phenomenal growth in the microprocessor market have produced a design community in tune with solving digital control problems using microprocessors. Development systems, programming tools, debuggers, peripheral chips, experience, folklore, data books, legacy systems, microprocessors, microcontrollers. It's all there, it works well, and it's comfortable. The microprocessor is a tool for solving problems. Since the microprocessor's introduction, we've had generations of engineers solving problems with microprocessors.

But the tool set influences the method for solving the problem. The paving method used by a road crew equipped with hand tools differs from the method used by a road crew equipped with graders, rollers, and power paving machines. The microprocessor is more than a tool with fixed computing resources and variable algorithms: it drives the way engineers think about how to solve problems. Problem-solving engineers map algorithms into the microprocessor's fixed resources, adding to their accumulated experience and to the sophistication of application methods with each new challenge.

As engineers improve application methods, microprocessor "architects" improve the tool. The microprocessor's architects are increasing clock rates, considering VLIW (very long instruction word) processing, experimenting with multiprocessors, and adding new instructions (Intel's MMX instructions for the x86, for example). These ideas are direct attacks on the tool's primary limitation: the instruction execution bottleneck. The microprocessor's architects work to improve the microprocessor's rate of instruction execution, just as the tool architects for the first road crew would work to improve the efficiency of hand tools.

A New Tool and a New Method

The microprocessor is a great tool, with its fixed computing resources and variable algorithms, but what if engineers were given a tool with variable computing resources and variable algorithms? They'd ignore it, just as the first paving crew would ignore a power paver. Engineers have 25 years of accumulated microprocessor-based problem-solving experience. There's an entrenched way of solving digital design problems. Engineers and pavers are paid to solve problems and pave roads, respectively; they aren't paid to experiment with problem-solving or paving methods. The companies they work for pay them to get the work done and expect them to leave design methodology (my first non-pompous application of this term) to university professors. Engineers would get on with their work using familiar tools at hand: microprocessors.

Tools with variable computing resources and variable algorithms, however, have compelling advantages in some applications. The SRAM-based programmable logic device

(PLD) is such a tool. A PLD can be viewed as a two-layer device. The first layer is logic elements and interconnect wires. The second layer is memory. Ones and zeroes in the memory determine how logic elements and wires connect together to form circuits. Need a different circuit? Reconfigure. Rewrite the memory to reconnect wires and logic gates as needed.

To see how this might provide an advantage, imagine an Excel spreadsheet. Values in cells are either direct entries or computed functions of other cell entries. Using PLDs, it would be possible to build a "live" spreadsheet. The PLD memory could be used to construct the necessary computational hardware connections behind each cell. Input cells need only storage, while output cells might require sophisticated computational hardware, such as a floating-point multiplier, 27-input accumulator, or cosine generator. Recalculation in the spreadsheet would be fast relative to the alternative of executing Excel object code on the PC's microprocessor.

You could build a PLD-based accelerator for the PC that could be configured for audio and video decompression processing. The PLD-based accelerator could be configured to decode MPEG-1 or MPEG-2 video streams or Dolby AC 3 audio. A custom ASIC and a DSP would be cheaper and just as fast. But how good would your ASIC and DSP solution be for a newly defined MPEG-4? And how good would your ASIC-based accelerator be for encryption and decryption? The PLD-based accelerator could be reconfigured to track the development of new methods, definition of new standards, or variation in application requirements. Reconfigurable computing is a powerful and useful concept. It's got only two major drawbacks: it's expensive, and it requires a new, and therefore unfamiliar, design method.

Reconfigurable Computing Faces Barriers

Reconfigurable computing is expensive because PLDs are expensive. PLDs are expensive because demand is rising and a few major suppliers control the market; in addition, the flexibility provided by the PLD's two-layer conceptual structure carries more overhead than the U.S. government. A million-transistor device nets about 50,000 transistors worth of logic. Prices have been coming down, so cost isn't a permanent problem. As the semiconductor process improves, PLD costs move down the learning curve: parts that cost $1,000 today will be in the range of $10 to $25 in two years.

Since the leading suppliers are essentially fabless, PLDs track semiconductor process with some lag, so it's easy to see their future by looking at today's leading-edge memory processes.

PLDs may need such features as rapid reconfiguration, partial reconfiguration, and background configuration loading to support reconfigurable computing. PLD configurations have been notoriously slow to load through a one-bit serial interface, but manufacturers configure and test parts rapidly, so faster configuration interfaces could be provided to the user. Similarly, there's no inherent barrier to partial configuration or background loading other than simple market demand to justify the features. Manufacturers are already beginning to support these features.

The key barrier to the rise of reconfigurable computing is that it requires the introduction of a new way of thinking about how to solve problems. Reconfigurable computing offers variable computing resources and variable algorithms. The engineer employing reconfigurable computing has the responsibility to determine and configure computing resources in addition to mapping the algorithm onto the resources. Reconfigurable computing adds another degree of freedom, dynamic resource creation, to the engineer's problem-solving method, which makes solutions more difficult but potentially more powerful.

Reconfigurable computing enthusiasts have been working to construct a useful and encouraging development environment for building reconfigurable computing applications. The idea is that if the industry creates a compelling, functional development environment, applications will follow. It isn't going to work. The barrier is not the lack of a development environment for reconfigurable computing, it's overcoming the engineer's entrenched microprocessor-based design method.

Reconfigurable computing has to offer incontrovertible proof of its value before engineers will adopt the necessary new design method. Incontrovertible proof can be provided by an installed base of applications with compelling performance advantages. This looks like a chicken-and-egg problem. How can we ever get an installed base of applications? Here's how I think it might happen.

Low-Margin x86 May Be on the Way

Systems using x86 microprocessors as the CPU dominate the personal computer market, with more than a 90% share. Intel provides most of these CPUs—the only high-margin component in the system. Intel has become one of the most profitable companies in America by riding the rise of personal computers from almost nothing in 1981 to worldwide sales of 60 million units in 1995.

The enormous size of the market and its potential for fat profits attract competition. Intel's x86 microprocessors face competition from AMD, Texas Instruments, Cyrix, SGS-Thomson, and IBM. Some competitors may be left behind as Intel moves the market from Pentium to Pentium Pro and beyond, but Intel already faces price competition from AMD and Cyrix with Pentium-class microprocessors.

Increasing investments in improving instruction-level execution in successive microprocessor generations may provide diminishing returns. Current-generation superscalar microprocessors effectively exploit available instruction-level performance from the installed base's object code. But the installed base is here to stay, so low-level instruction execution remains a performance bottleneck. Adding instructions may aid new applications in some areas, but it also adds to the compatibility baggage carried forward to the next generation.

Design costs double with each new processor generation. The $250 million to $400 million needed for a next-generation microprocessor design might return more performance if invested elsewhere in the system.

The system's point of performance leverage may move away from the CPU: one possibility is application accelerators. Windows dominates the installed base. Windows and its APIs (application programming interfaces) isolate applications from the underlying hardware. Application developers no longer write low-level drivers; applications request services from the operating system to manipulate system resources (hardware).

The transition from ISA to PCI, a move fostered by Intel, gives add-in boards much faster access to system resources. Application accelerators, with performance provided by access to system resources through PCI or even direct connection to the host CPU, can intercept OS service calls and subvert low-level instruction execution (in the OS service routine) by speedier delivery of equivalent function.

Rather than let the instruction-level OS routine draw a line on the display, for example, a graphics accelerator might just draw the line on the screen (by stealing the parameters from the OS service request). Dozens of application accelerators, some claiming performance improvements of 40× or more over base-configuration systems, are available. Today, these application accelerators are based on DSPs, microprocessors, and ASICs.

Economics may soon favor reconfigurable application accelerators based on PLDs. These reconfigurable accelerators will follow the user to accelerate compute-bound applications. If the user launches Photoshop, the accelerator will download configuration files from the hard disk and prepare to accelerate compute-bound filter functions. If the user launches a spreadsheet, the accelerator might wire the cells together with the appropriate equations to provide "live" answers to changes. The default configuration might accelerate Windows or Netscape.

If the point of leverage in system performance moves from instruction-level execution to application accelerators, the incentive to pay premium prices for marginally improved CPU performance decreases. The x86 processor may become a low-margin component in the system. Reconfigurable computing could gain the installed application base to demonstrate the power in the required new way of thinking about how to solve problems. Then we will have progressed from the fixed resources and fixed algorithms of TTL macrofunction-based designs to the fixed resources and variable algorithms of microprocessor-based designs to the variable resources and variable algorithms of PLD-based designs.

Reconfigurable computing isn't going to kill the microprocessor. The microprocessor is entrenched in countless designs where it is better suited to the application than a PLD would be. But the PLD should begin to displace the microprocessor in those designs where the ability to reconfigure computing resources offers a performance advantage. M

Nick Tredennick did the logic design and microcode for the Motorola 68000 and was later chief scientist at a PLD company. He can be reached at bozo@tredennick.com.