Vol 15, Issue 53 December 31, 2001

General Purpose Architecture

By Max Baron

It's no longer possible to assume you can design a processor without having a good idea of the system architecture in which it will be used and the workload it will execute. A large number of designs have become application specific; they can provide higher performance with less of a power requirement than equally priced general-purpose engines.

What happened to general-purpose architecture? Did it become extinct a long time ago with the introduction of floating-point coprocessors? Did it disappear when OS support, such as shadowing registers and register windows, was added? Was it always a myth? Can architectures still be called "general purpose" if they must add MMX, SSE, 3DNow, DSP, Java enhancements, Viterbi instructions, and other extensions?

Recently, I was told about the introduction of a new chip that boasts multiple general-purpose processing cores. As always, I asked the microprocessor vendor what applications the new chip was targeting. The marketing person's answer was not the only one of its kind: "We don't know what software they (our customers) will be running," he said, "but if they use our multiple on-chip processing cores they can get better performance than with only one core."

This general-purpose, application-indifferent approach may have been sufficient several years ago, when it was very hard to design and make processors. Just a few processor architectures were around. The differences between an embedded processor and a desktop processor were price, simplicity of implementation (microarchitecture), and performance. The embedded processor was designed to be inexpensive. Its modest performance was adequate for tasks that were less demanding than those required of the desktop processor.

Today, to survive and prosper, both desktop and embedded processors have to track applications. They are evolving in different ways and following different roadmaps. They employ different architecture and microarchitecture enhancements, conditioned by the technologies they can afford to use.

Desktop microprocessors use state-of-the-art semiconductor technology and design to get the highest possible performance through enhancements to frequency and microarchitecture. Their evolution is driven by the need to present a simple, system-independent programming model to a large base of software applications writers. However, increased frequency doesn't always help, and vendors need to make periodic architectural enhancements as applications become more demanding for improved performance and have the need to process new data types.

Embedded microprocessors, limited as they are by price-acceptable ASIC technology, use both enhancements in architecture and additional processors. What the embedded processor can't obtain through raw frequency and multiple pipe stages, it achieves via special ALU units, additional instructions, accelerators, and hardwired logic. Complex chips that use multiple microprocessor (MPU) and digital-signal processor (DSP) cores and accelerators can deliver performance at lower frequencies and with low power consumption. To top it off, with processing resources in place, designers can increase a chip's chances of success by adding application-specific peripherals. The peripherals count in some chips runs into the tens.

The resulting software development target is complex; programmers must juggle the management and control of multiple on-chip resources. However, the software is created by only a relatively small group of programmers, for whom documentation and support are within the capabilities of an OEM.

One look at processors such as Motorola's Dragonball, Intel's IXP1200, and some of TI and ADI's DSP chips proves that processors are designed with specific applications in mind. Quite a few companies have introduced configurable or extensible instruction-set processors. ARC and Tensilica let designers add special-purpose hardware to speed up processing. Proceler takes a different route: it can call "soft" hardware into existence to accelerate parts of a workload that require high performance. Altera, Xilinx, Triscend, and others provide microprocessor cores and raw hardware that can be programmed at boot time, or, if time permits, at run time. LSI Logic can now combine programmable and fixed cores on one chip. Chicory (now merged with Parthus) has made the argument that an accelerator can do a better job than an instruction-extended processor from the viewpoint of power dissipation. It makes sense that power management may lower the frequency of a central core while accelerator-based activity using less power is taking place.

In the embedded space, with today's demands and available technology, targeted processor architectures, accelerators, and peripherals have become solutions that, given sufficient engineering time and testing, will produce better results than unchanging general-purpose architectures and higher frequencies. Variations, extensions, and special-purpose architectures will continue to be introduced for different reasons: the workload needs high performance; an OEM may have inherited or acquired a binary or source software that needs to be speeded up; or there is a need to efficiently execute an interpreted language, such as Java.

The new embedded chip is very far from being a "don't care" general-purpose processor. Designers should know what applications they are targeting for their chips and what software will be running to execute those applications. And as hardware and software design tools evolve, embedded architectures will more closely reflect the system workload.

Two years ago Keith Diefendorff wrote an editorial entitled "Are There Too Many Processors?" Keith expressed the hope that the laws of natural selection would eventually reduce the number of architectures, and that just a few good ones would survive. I keep thinking of the wonderful world we live in, with its unbelievable number of different living organisms, each configured to fit into its habitat and to succeed there. Why so many processor architectures? I think the answer is "Because there are so many different workloads."