# How Many System Architectures?

**Wayne Wolf,** Princeton University

power operation. InfoPad, one of the first portable networked multimedia devices, included specialized hardware for graphics, video, and wireless operations, leaving the CPU free to perform housekeeping functions. A special-purpose unit can perform an operation using less energy than a general-purpose CPU—just eliminating the instruction fetch and decode logic saves significant energy. Further, a special-purpose unit can be shut down

Previously, I talked about CPU architectures and their 15 minutes of fame ("Whither Warhol's Law," *Computer*, Sept. 2002, pp. 96-97). Quite a few instruction sets vie for attention in the embedded systems marketplace today. However, instruction sets provide only one piece of the design puzzle. An embedded system also requires input and output, memory, and, frequently, parallel processing. How best to structure this overall system is the fundamental architectural question in embedded computing.

Developers often call an embedded system's hardware architecture a *platform*. The idea is that you select a hardware platform, then write software to fit onto it. If only things were that easy. The platform provides the computing resources required to run the embedded application, and the decisions about what hardware should go into the platform often depend on what sort of software you want to run.

**Constant pressures on the cost and power consumption of embedded systems will likely continue to spawn a diversity of uniprocessor and multiprocessor platforms.**

## PLATFORM CHOICES

Choosing a platform is a critical decision in embedded system design: Too much hardware and you're wasting money and probably power, too little hardware and you can't run the application. Fitting the application onto the platform means more than just making sure you have all the right I/O devices. Embedded systems have strict performance goals, such as real-time deadlines. If you don't have enough computing power of the right type available to meet the deadline, the system fails. Real-time operation requires more than just having enough computing power on average—it requires enough power for the worst-case scenario.

We must also consider power consumption when choosing a platform. When confronted with real-time performance problems, developers tend to respond reflexively by overdesigning the platform and providing anywhere from a little to a lot of extra computing power. However, that computing power consumes electrical power. Battery-operated devices are always highly sensitive about energy consumption, but even systems plugged into the wall must watch their consumption because power dissipates as heat. Fans to cool the electronics not only cost money, but they can also be noisy and unacceptably annoying in many situations. For example, would you buy an MP3 player with a fan?

Building a platform with some specialized hardware to support common operations in the application can save significant amounts of energy. The InfoPad project at UC Berkeley in the 1990s clearly demonstrated the importance of specialized hardware for low-power operation. InfoPad, one of the first portable networked multimedia devices, included specialized hardware for graphics, video, and wireless operations, leaving the CPU free to perform housekeeping functions. A special-purpose unit can perform an operation using less energy than a general-purpose CPU—just eliminating the instruction fetch and decode logic saves significant energy. Further, a special-purpose unit can be shut down when it is not needed. This is especially important today, when leakage currents constitute a large fraction of a unit's total energy consumption.

## GROWING THE CHIP ECOLOGY

The push toward higher VLSI levels has added new urgency to the question "How many platforms are necessary?" The number of platforms required to support embedded systems translates directly into the number of different chip types that semiconductor manufacturers need to supply.

### Why more is better

Having more platforms offers advantages for semiconductor manufacturers, particularly as the industry consolidates. More platform options mean more market niches for chip companies, just as a richer ecology spawns more species to occupy its many environmental niches.

But it takes a lot of customers to make a new chip type economically viable. Designing a chip takes great engineering effort, a cost that must be spread over its manufacturing life. Even the cost of the masks used to manufacture chips has become a major

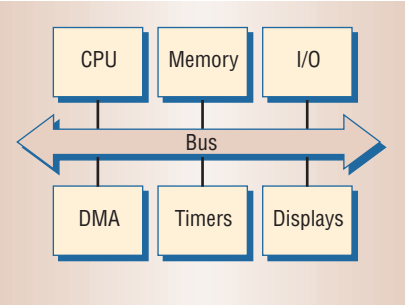


***Figure 1. A simple bus-based architecture for embedded systems.***

factor in overall cost. Masks for a chip manufactured in a 0.25-micron linewidth process a few years ago cost about $120,000. Today, the masks in the latest 90-nanometer process cost $1 million. Given that the typical consumer-grade chip sells in the $10 range, you have to sell a whole lot of chips to avoid having the design's fixed costs drive the chip's unit cost to an unacceptably high level.

However, today's technology has created several large chip markets. For example, consumers buy about 30 million CD players every year—that's just audio CD players, not computer CD drives. While this may seem surprising, remember that CD players are standard equipment on boom boxes and other audio systems. Increasingly, automobiles also ship with CD players installed. Consumer audio and video components alone provide a long list of high-volume devices: DVD players and recorders, digital still cameras, digital video cameras, digital set-top boxes, and more. PDAs constitute another major segment of consumer electronics. And there are plenty more major markets both within consumer electronics and in the industrial and military markets.

## Fewer is simpler

The other side of the debate argues that having fewer platforms makes life simpler for both semiconductor manufacturers and their customers. The winners in the battle to provide the standard embedded systems platform would open up enormous markets.

The PC has been a huge cash cow for the chip and computer industries. Vast numbers of PCs have been sold, all based on a relatively small number of different parts. Customers and small companies have been left the task of making those PCs useful for individual applications. Quite a few people in the electronics industry would like to see another killer category like the PC to sustain the industry for another decade or two. That means finding a common hardware architecture that can serve all sorts of markets efficiently enough for economies of scale to overwhelm any inefficiencies the platform engenders for a particular operation.

For embedded system designers, knowing that a small number of platforms can adequately cover the design space leads to comfort. Rather than searching high and low for the right combination of hardware and performing extensive experiments to measure performance and power consumption, knowing that the choice lies in two or three platforms makes choosing one of them straightforward. Because a platform can be chosen quickly, work on the system can swiftly proceed to software design and implementation. Further, the characteristics of a platform—its speed for various operations, power consumption, and so on—can be more thoroughly characterized. Better characterization leads to easier and faster software development because the implications of a software design decision can be predicted by looking at the code, rather than performing a series of custom-designed experiments.

## DEFINING THE NICHES

What sorts of platforms might we envision to support embedded applications? The simplest such platform is what I call the *PC architecture* because it resembles an early personal computer. This architecture, shown in Figure 1, has a simple bus-based organization. A single CPU is supported by some memory, timers, a direct memory access (DMA) controller, some I/O, and perhaps a display driver. This block diagram closely reflects the design of the early PCs. Modern PCs are considerably more complex, with graphics operations, audio, and networking commonly offloaded from the main CPU into separate specialized processing units.

## Uniprocessor simplicity

The uniprocessor architecture's biggest advantage is simplicity. Its design means that the hardware is fairly straightforward—the bus, caches, and other components are simple to design and manufacture. More importantly, the software is simple. To develop an application for this platform, programmers need only a straightforward programming model. The programmer doesn't need to worry about synchronizing data, handshaking, or any of those other complexities that make parallel programs harder to debug.

Uniprocessing is attractive if it works. In the embedded world, that means the CPU must be fast enough to perform the real-time operations with enough capacity to provide a reasonable operating margin. As VLSI technology improves and CPUs run faster, more applications will fall into this category.

The hardware block diagram of a basic PDA is not much more complex than that shown in Figure 1—we need only a little extra hardware to read the touch screen. A single CPU performs handwriting recognition and the database functions associated with the PDA applications. A flash-based MP3 player is also simple—basically the architecture shown in Figure 1 with a headphone amplifier for an I/O device. A single CPU is more than fast enough to decode MP3 audio in real time.

## Moving to multiprocessors

However, consumer appetites continue to grow. As people see more, they want more. Not only does this mean a demand for more functionality, such as a move from audio to video, it also requires a device that does more things at one time. PDA applications are de-

signed along a traditional model in which one thing at a time happens—the CPU doesn't have to worry about spreading itself among multiple simultaneous deadlines. However, imagine an audio player that receives its music input from a network. That device must now juggle the audio and networking loads simultaneously. Because both these tasks run at relatively high rates, the CPU operates under a much greater burden.

As another example, consider the lowly CD player. Compact disc technology is an amazing triumph of sophisticated electronics and signal processing over cheap, low-quality components. Even a traditional CD player that plays audio CDs without decompression requires a DSP to perform real-time servo algorithms, along with a separate unit to perform the modified Reed-Solomon decoding required to correct errors. All this is in addition to an analog front end that performs the truly high-rate signal processing. If you want to play MP3 files, you need to add another CPU for the audio decode. You now have a fairly sophisticated multiprocessor running in real time.

Video applications require even more powerful multiprocessors. Philips' Viper chip provides a digital television platform that contains a MIPS CPU along with a Trimedia processor. The Trimedia is a very-long-instruction-word CPU that can execute five operations per clock cycle. Each of these CPUs has its own bus, and a third bus feeds video data from bulk memory to the two processors. In addition, dozens of devices and accelerators support various aspects of video operation.

## CRITICAL INSTANTS

Some have argued that only two or three platforms will survive to serve the embedded systems market. Others claim that the diverse needs of embedded applications require many more platforms. Who is right? The answer depends in part on marketing, politics, and other unforeseen forces. But a little bit of science can give us some insight into how many platforms we may need to make designers' lives a little easier.

I've covered rate-monotonic analysis before ("Household Hints for Embedded Systems Designers," *Computer*, May 2002, pp. 106-108). This little piece of theory provides the foundation for real-time systems and helps us predict how efficiently a CPU can meet real-time deadlines. A key concept in rate-monotonic analysis is the *critical instant*: the combination of events that leads to the highest processor load. The critical instant occurs when the deadlines line up so that all the real-time processes are ready to run. This means that all the higher-priority processes must finish before any lower-priority processes can run. Further, a high-rate process can have several deadlines in the same period as a single, slower-rate process.

All this means that juggling several tasks at once imposes a significant load on a processor. Rate-monotonic analysis also teaches us that we can't always utilize 100 percent of the processor if we want to meet all our deadlines. This means that splitting up a complex set of tasks across several smaller CPUs often makes it easier to meet performance goals. We've also seen that using several specialized processors can improve power consumption.

The division into multiple processors is application-specific. It might be possible to come up with a fairly generic architecture that would let a designer turn on and shut down parallel processing elements as necessary. But we'd also need good virtual I/O to turn this sort of architecture into a general platform. It is likely that we will see a diversity of embedded systems platforms for quite some time to come. ■

***Wayne Wolf*** *is a professor of electrical engineering at Princeton University and author of* Computers as Components: Principles of Embedded Computing System Design *(Morgan Kaufman, 2000). Contact him at wolf@princeton.edu.*