Applications and Architectures

Wayne Wolf, Princeton University

How many different types of chips do we need? That has been one of the big debates in the semiconductor industry over the past few years. It's also a topic that we've considered in this column before ("How Many System Architectures?" Mar. 2003, pp. 93-95).

I'm beginning to have second thoughts about my earlier prediction that we will have many different computing platforms for the foreseeable future. The reason for my change of heart is—in a word—software.

MANUFACTURING COSTS

Chips are getting more expensive both to design and to manufacture. That makes it tempting for semiconductor companies to push more functions onto a single chip, using software to customize it for specific applications.

Each chip design needs a different set of manufacturing masks, and the cost of masks is increasing exponentially. For today's 90-nanometer processes, the set of masks for a new chip costs about $1 million (although most manufacturers, eager to maintain their reputations as bargain shoppers, say they don't pay retail). If a manufacturer sells 2 million chips over the course of a year, the masks add about 50 cents to the cost of each one. When the chip sells for $4 or $5, that's a big bite out of profits.

Are complex applications pushing us toward more general-purpose embedded platforms?

Using software to customize a few platforms for a variety of applications spreads the mask costs over more chips. It also saves the time of switching from one mask set to another. Time is money, so reducing the variety of parts that require fabrication helps reduce costs in other ways as well.

As a result, chip manufacturers try to design platforms that can support a range of products. Customers who buy the chips to build systems use software to customize the functionality.

COMPUTATIONAL EFFICIENCIES

The traditional reasons for using application-specific chips are cost and power consumption.

A CPU is seldom the most efficient vehicle for performing a particular, well-defined function. CPUs use extra gates to fetch and interpret instructions; the logic isn't as small as a state machine designed to do only one thing. Furthermore, some functions don't fit neatly into the CPU's word width; they might require several instructions to perform an operation that a specialized piece of logic could complete in one clock cycle.

Accordingly, hardware designers began building systems composed of several different processing elements. In fact, some of those elements might be CPUs, but many systems include specialized processing elements to perform functions unique to an application. Because the collection of elements is irregular, the architectures need irregular memory systems and irregular interconnections.

As more system-on-chips (SoCs) go into battery-powered products—cell phones, portable media players, and so on—energy and power consumption become critical design factors as well as critical selling points for the chip. Replacing a general-purpose processor with a specialized piece of logic can significantly improve a system's battery life. And that can mean the difference between winning and losing the competition to select a chipset for that new mobile device.

STANDARD COMPLEXITIES

So we have a collision between two undeniably important aspects of chip design. On the one hand, specializing an SoC to an application has undeniable advantages. On the other hand, it is becoming wildly expensive to make those specializations.

The jury is still out as to the number of platforms that system designers will have available to them. Manufacturing costs and application requirements are pushing chip designers in somewhat different directions. Complex applications push us toward general-purpose processors or multiprocessors and away from highly application-specific chips, but some straightforward hardware tricks can speed up some applications quite a bit.

I don't know yet what the final answer will be, but in contrast to my earlier column, I see more and more applications that require more regular, highly programmable architectures.

Important application algorithms are becoming extremely complex. While more sophisticated algorithms can produce higher functioning products such as cell phones, video players, or cameras, they also make these systems much harder to build.

Consider video compression, for example. The key to video compression has always been to use several different algorithms together: the discrete cosine transform to encode certain key frames, motion estimation to provide higher compression rates for the moving objects in frames between the key frames, and Huffman encoding to reduce the size of representing all this data.

But recent standards address many more modes and features. The new Advanced Video Coding standard, also known as H.264, is the latest in a line of video-compression standards. H.264 covers the full spectrum of video platforms, ranging from cell phones with very small screens running at 15 frames per second to HDTV with high-resolution displays running at 60 frames per second.

In addition, cell phones operate at different resolutions and have varying amounts of bandwidth available. These variables require somewhat different approaches to video compression.

Encompassing all these complexities requires newer standards like H.264 to be bigger and support many more modes of operation.

Relative motion

A prime example is motion estimation for multiple reference frames. Motion estimation tries to find a piece of one frame that has moved to a new position in another frame. Those pieces are generally squares, but the piece shapes have nothing to do with the object shapes in the picture.

As an object moves through the scene, it can cover and uncover background objects. If a video system chooses one reference frame to define what is in the scene, it might not find a good match between the motion-estimation block and the content in successive frames.

At issue is the complexity of the software as much as its volume.

By defining motion estimation relative to several reference frames, a video system can find better motion-estimation matches, thereby reducing the number of bits required to encode that motion.

The window shapes used for motion estimation are also important. Early MPEG standards used one shape, a 16 × 16 element known as a macroblock. Its size represented a compromise between image quality and computational effort, but a macroblock is too large for some of today's small video displays. H.264 provides a variety of ways to segment macroblocks, including shapes other than a square when they would better fit the window to the object that the motion estimation is tracking.

Not just transmitters

These changes generally require more complex receivers as well as transmitters. Television broadcasting is designed to allow simpler, cheaper receivers at the cost of more complex transmitters; but videoconferencing and cell phones must balance compression and decompression requirements since terminals must generally do both. That means designers must figure out how to support both functions in low-cost, low-power systems.

If the authors of standards add features destined for a high-end broadcasting studio, the system designers have more techniques available to implement those features and fewer constraints on their designs. When they must put those features into consumer items, not just professional-grade systems, the designer's job is much harder.

FEEDING HABITS

According to designers at ST Microelectronics, it takes more than a million lines of code to support the audio features of high-end consumer electronic products today. If this sounds surprising, consider all the product logos on your audio equipment. Dolby, for example, requires licensees to implement multiple standards to provide compatibility with older material. DTX is a competing standard. MP3 is a must for audio compression, but there are others as well.

Each of these standards is complex in itself. Putting them all together yields a very large chunk of code.

Similarly, implementing a major video standard takes well over a million lines of code today, and the number will increase to three or four million soon. Consider a digital hub that could play video from your mini-DV camera on your HDTV or record it on your DVD drive. As with audio, products must support multiple standards to be useful. The combination of standards requires a huge software load.

At issue is the complexity of that software as much as its volume. If we can efficiently implement key routines in hardware, most software could still run on a CPU with hardware assist. But as the volume of code grows, so does the complexity of standards functions—as the motion-estimation example illustrates. And not all functions are good candidates for hardware speedup. If the memory access patterns are highly data-dependent, for example, software may be a more natural way to capture the algorithm's behavior.

To be fair, there are reasons to keep designing new platforms. For example, designing hardware for fairly generic functions can significantly improve system performance. Specialized hardware for graphics operations, text operations, and other operator-rich routines can run faster than equivalent code. Multiple processor cores can support some task-level parallelism.

When standards capture these types of operations, we know they won't change quickly, so designing a chip to perform them is often cost-effective and power-efficient.

Only time will tell how many different chips we finally need to support our computing habits. But to understand platform requirements, I do think it's time for a more substantial dialog between application designers and architects.

As applications become more complex, application designers become more tempted to try methods that don't implement well. They need some early warning before those methods end up in standards. Embedded system architects can help them judge their options by giving them platform models to evaluate the characteristics of the algorithms they propose.

Building such models won't be trivial, but better models will represent a major step forward in computing-system development. ■

Wayne Wolf is a professor of electrical engineering at Princeton University. Contact him at wolf@princeton.edu.