

Embedded microprocessors and DSPs: considerations for converging technologies

If you're considering a change from a traditional embedded processor to a DSP device, you're not alone.

Chris Bernard, Agilent Technologies

Introduction

ANYONE WHO HAS EVER BEEN TO A COCKTAIL PARTY can relate to the following analogy. Picture yourself standing in the middle of a room surrounded by the roar of hundreds of conversations occurring simultaneously. Now, pick a random person far across the room and design a system that allows you to isolate that person's seemingly indistinguishable voice from the roar of all the other voices in the room. Does this sound like some impossible graduate project at MIT designed to make even the brightest student sweat?

Actually, it is strikingly close to the problem that wireless engineers are facing every day. Wireless connectivity is just one of the many applications fueling the explosion in DSPs and the convergence between them and traditional microprocessors. If you are an engineer debating whether to move from a traditional embedded processor to something with better DSP and mathematical functions, you are not alone. The following paragraphs clear up some confusion in this debate by presenting some common DSP applications, a comparison between DSPs and traditional microprocessors, and topics for designers to consider when moving to a DSP.

Converging technologies

DSPs have long been the cornerstone of math-intensive computing applications, such as audio and image processing, as well as telecommunications. With the explosion of wireless communications and Internet connectivity, DSPs suddenly find themselves in the mainstream of embedded-systems technology. In the last couple of years, DSPs have begun to look more like embedded microprocessors by adding modern C++ compilers and multithreaded OSs, but embedded processors have in some cases added DSP-like functions, such as multiply-accumulate instructions and vector engines. Market forces are precipitating an ever-increasing convergence of these two technologies, thereby causing engineers to re-evaluate which of the two is better for their application.

Applications driving DSP growth

What is fueling the growth in DSPs? To answer that question, you simply need to look inside your pocket at your mobile phone or Palm Pilot. DSPs are becoming mainstream primarily because of wireless communications, in particular because of new technologies, such as wideband code-division multiple access (W-CDMA). To grasp the computational power and complexity required to make a modern mobile phone operate, consider the cocktail-party analogy. W-CDMA spectrums basically have a white Gaussian-noise distribution, much like that of a large, noisy room with lots of conversations and echoes. Identifying a specific speaker within the noise is deterministic thanks to the code but still requires sophisticated algorithms to extract. The DSP is the mathematical workhorse that you use to disseminate the desired signal from the noise distribution.

In the future, any device that connects onto a wireless network will require a hefty processor to transmit and receive voice and data. In the wired future, everything from

your wrist watch to a refrigerator may have a connection to a wireless network; consequently, silicon vendors are moving their DSPs and embedded processors toward this convergent future.

Architectural differences

DSPs differ from typical embedded controllers in a number of ways. First, most high-performance DSPs contain multiple execution engines, which allow for an excellent performance/efficiency ratio. The assembler tags instructions for execution in one of the pipelines. Most often, each pipeline has its own task, such as load/store, add, multiply, or shift. Most embedded controllers have one execution engine. Even if multiple execution pipelines exist, a dispatcher generally assigns instructions to a pipeline, and these instructions execute in the same way no matter which pipeline the dispatcher assigns them to. Coding a DSP at the assembly level can become fairly sophisticated when you have to deal with the different execution pipelines—a necessary task to maximize performance.

These days, most embedded controllers are RISC (reduced-instruction-set-computing) processors. DSPs, on the other hand, are beginning to move to a VLIW (very-long-instruction-word) architecture. A DSP may have 10 times the number of instructions of a comparable RISC processor to account for every conceivable mathematical operation. In addition, DSPs have many more addressing modes to allow for optimal data handling.

High-performance DSPs often require adherence to fixed-point mathematics to attain optimal performance. Although fixed-point DSPs tend to provide better raw computational power, the complexity in programming compared with similar floating-point processors makes them somewhat less desirable for inexperienced users. Compilers are making great strides to mask the fixed-point math complexity by optimizing common mathematical functions in prepackaged C routines that are available to programmers. However, as with many other DSP features, maximum performance requires increased complexity.

Architectural similarities

Although a number of differences exist between embedded controllers and DSPs, there are many similarities. First and foremost is the memory architecture. Like embedded controllers, DSPs are heavily cached and often have on-chip memory controllers to support modern high-performance memory systems. DSPs are also beginning to incorporate peripheral systems that have long been a part of embedded controllers. DMA; serial ports; and even full-blown programmable communications controllers, such as the Motorola StarCore, are showing up with a DSP as the core.

Considerations for DSPs

Consider starting a new project that requires moving to a next-generation microprocessor. These days, it seems that every silicon vendor and IP house has a microprocessor that fits the bill. Vendor relationships, power consumption, price, performance, peripherals, tools, and more are all important considerations for selecting a microprocessor. Ignore these obvious criteria for the moment and explore simply whether to head down the traditional embedded-microprocessor road or to make the leap to a DSP.

Designers typically consider moving to a DSP when the application requires a great deal of mathematical computations. Theoretically, DSPs perform calculations faster and more efficiently than a similar embedded controller does. However, be aware that increased performance does not come for free. C code compiled for a PowerPC, which is

ported to a Texas Instruments C6000 DSP, may not show much—if any—of a performance increase at all. The key to getting the most performance from a DSP is taking full advantage of the features the architecture has to offer. Although compilers are making great strides in masking the complexity of a DSP's micro architecture from the general software engineer, even the best compiler falls short for a motivated engineer determined to squeeze just a little bit more performance out of the chip. DSP programmers have historically been very strong at assembly-level programming specifically for this reason.

Development tools

Hardware engineers have long viewed logic analyzers as the tools of choice when debugging embedded systems with microprocessors and DSPs. A logic analyzer is an excellent tool for characterizing performance, measuring setup-and-hold times, debugging boot code, and performing a host of other hardware-related measurements. Whether you use an embedded controller or a DSP, logic analyzers provide roughly the same level of functions for hardware engineers. Agilent Technologies' 16702B logic analyzer is one example of a system for debugging DSP-based embedded systems.

Software engineers have had a much different experience depending on whether they used an embedded controller or a DSP. For many years, most DSP code was written at the assembly level and didn't require a great deal of sophistication from the software-debugging tools. The complexity of the code was in the algorithms, not in the software infrastructure. Embedded microprocessors typically have large software architectures requiring a higher degree of sophistication in the debugging tools and compilers. Now that real-time OSs and C++ compilers support DSPs, DSP debuggers are beginning to look much like their embedded counterparts.

Real-time issues are becoming ever more important for all embedded-system designers. Emulator and silicon vendors for both DSPs and embedded controllers have to adapt to provide better real-time debugging. Embedded microprocessors have been somewhat slow to adopt real-time-debugging methodologies. DSPs, on the other hand, are paving the way for future real-time-debugging techniques. Texas Instruments' RTDX (Real-Time Data Exchange) is an example of a real-time-debugging technique. Users can import or export data from the application in real time to the debugging tools over the RTDX channel. This technique goes beyond simple JTAG debugging, which merely stops the processor and allows access to registers and memory. In the future, real-time, high-bandwidth, full-duplex debugging channels will be commonplace, allowing a range of new features, such as real-time statistical characterization and trace. DSPs have been designed from the beginning to provide real-time operation and currently have a distinct advantage in this area.

OSs and control applications

OSs and DSPs have traditionally been two terms that no one would dare use in the same sentence. However, OSs are becoming commonplace in DSP applications due to the ever-increasing complexity of the DSP's software architecture. Now that C++ compilers and real-time OSs exist for DSPs, have they completely bridged the gap to a general-purpose embedded controller? In some applications, the answer is yes. Depending on the complexity of the system, you can effectively use a DSP for general-purpose control. However, traditional embedded microprocessors shine in a few key areas where DSPs are unlikely to displace them. For simple keypad and LCD control, a DSP is overkill, explaining the emergence of multicore (ARM core plus DSP) ASICs in mobile phones. Using multicore ASICs is proving to be the most efficient technique for com-

binning DSP and control applications. The DSP can focus on incoming real-time data, and the small embedded processor provides the user interface and housekeeping. For complicated applications, such as those running on large ATM routers, embedded-microprocessor OSs, such as VxWorks, are much better at running large, multithreaded network-management applications and are unlikely ever to be replaced by a DSP.

Conclusion

The difference between embedded controllers and DSPs is quickly being clouded as the two begin converging on the same market and applications. Traditional DSP manufacturers are adding high-level language and real-time OS support, integrating peripherals and a host of other features targeting embedded-control applications. Traditional embedded-microprocessor vendors are also doing their part to close the gap by adding vector engines, custom instructions, and parallel operations, all intended to increase the efficiency for mathematical computations. All vendors are doing what they can to secure a piece of the enormous wireless-connectivity market. In the near future, a vendor's history—rather than the identifiable features that differentiate a processor—will largely determine whether a processor is called a DSP or an embedded controller.

Author biography

Chris Bernard is an electrical engineer with Agilent Technologies in Colorado Springs, CO, specializing in emulation and logic-analysis support for embedded microprocessors and DSPs. He received his BSEE from the University of Minnesota in 1995. At Hewlett-Packard, he spent three years designing emulators as an R&D engineer and two years as a product manager/field-applications engineer for HP emulation products supporting Motorola processors.