

**T**he number of options for choosing the best processor for embedded-system designs continues to increase. In addition to microprocessors, microcontrollers, and DSPs (digital-signal processors), several unified microprocessors from companies such as Analog Devices, Infineon, Microchip, and Freescale have also emerged (Reference 1). These software-programmable devices offer unified-processor architectures that combine

features of microcontrollers and DSPs to better mix control and signal processing in a single instruction engine. More recently, new silicon products and more mature tools are available to embedded-system designers to better harness the power of programmable logic as custom accelerators of software algorithms for signal processing (references 2 and 3).

The increasing incorporation of digital-signal processing in electronic applications has led over the years to a number of claims and predictions by competing semiconductor providers. One claim is that, for applications with high computational loads, FPGAs are better than DSPs for digital-signal processing and that designers can eliminate the need for DSPs in their designs by adding processor cores in the FPGA. A similar claim for applications that require some signal processing states that a microprocessor architecture that incorporates integrated digital-signal-processing extensions can replace DSPs.

Claims such as these could suggest that FPGAs are encroaching on the domain of DSPs from the computational high end and hybrid microprocessors from the computational low end. However, the replacement of DSPs in embedded-system designs by FPGAs and unified microprocessors is analogous to the bid of 32-bit processors to replace 8-bit processors. In other words, embedded-system designs are

# PROCESSING OPTIONS

BY ROBERT CRAVOTTA • TECHNICAL EDITOR

CHOOSE THE RIGHT  
MIX OF PROCESSING  
TECHNOLOGIES FOR  
EMBEDDED-SYSTEM DESIGNS.



## AT A GLANCE

▣ The requirements for consumer applications are driving the adoption of mixed-processing designs.

▣ Each processor option addresses a different sweet spot in the range of processing requirements.

▣ Software implemented as hardware is a growing opportunity for designers.

▣ Tools supporting mixed-processing designs need to preserve and incrementally evolve the programming model.

not eliminating the use of DSPs; for analogous reasons, 8- or even 4-bit processors have not become extinct. In fact, the message among semiconductor vendors has been making an important shift over the previous year or two from

a position of compete and replace to a stance of coexist and complement. This recent shift is most visible among suppliers of DSPs and FPGAs.

Semiconductor vendors were late in realizing this shift in distinction; their customers understood all along. Different processing architectures best suit different design constraints. As high-volume consumer applications continue to incorporate new features and converge multiple functions into the same end system, designs are employing a mixture of processing architectures in various topologies. A possible positive outcome of this shift from a compete to a complement stance is the eventual availability of development and debugging tools that better simultaneously support mixed-processing options in the same tool.

By mixing heterogeneous processing architectures in the same design, developers can lower the system cost, power consumption, design complexity, and time to market. A mixed topology can

also afford a development team a simpler maintenance environment that supports a shorter development time between iterative and derivative design cycles. This situation is possible because each type of processing architecture supports a different sweet spot in the spectrum of processing requirements (see sidebar “Processing sweet spots”).

## SWEET SPOTS

General-purpose microprocessors trade off cost and power efficiency to maximize and deliver high processing performance and the most flexibility of all the processor architectures. To achieve the highest processing performance in the face of uncertain processing behavior, microprocessors often employ additional complexity in their circuitry to offset the impact of unpredictable changes. This additional circuit complexity can take the form of superscalar architectures with deep pipelines, multiple levels of on- and off-chip cache memory, branch-predic-

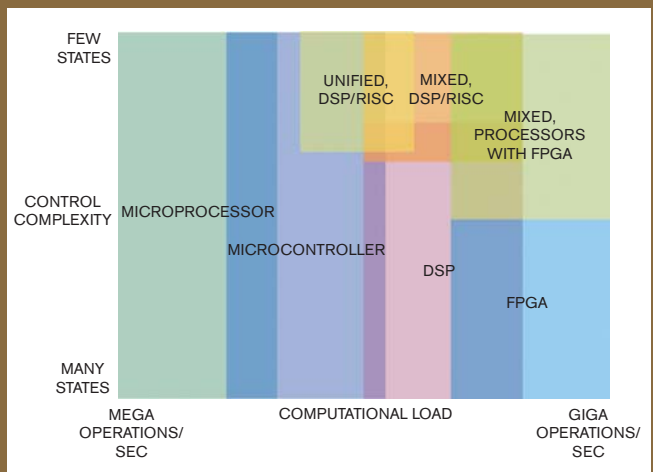
## PROCESSING SWEET SPOTS

Each type of processing architecture fits a sweet spot from the total range of processing complexity (Figure A). All of the sweet spots overlap other sweet spots to provide full coverage of the possible processing scenarios. The assumption is that, if the field of possibility contains any uncovered space, an alternative specialized architecture would exist. As application requirements further push the processing requirements on the right of the processing spectrum of the computational load, you see the introduction of unified and then mixed-processing topologies.

In looking at how to evaluate software complexity, this article identifies four measurements of processing. The com-

putational load is a measure of the number of repetitive computations the application needs to perform over a period of time. With the computational load, the control processing for the computations helps indicate when a multistate algorithm would become too large as a pure hardware implementation. This article uses these two measurements as the primary axes for mapping the processor sweet spots.

Application processing is analogous to computational load; however, the code does not repeat consistently and predictably. Application processing is code that would not lend itself to a parallel implementation. Last is the measure of the system’s response requirements. Of most



**Figure A** A conceptual mapping shows how the sweet spots of processing architectures can complement each other to cover the range of an application’s processing requirements (assistance from Gary Banta, chief executive officer of Stretch).

interest are the response requirements for real-time systems. This measure helps identify when the timing uncertainty in program execution—possibly from the operating

system or from using a cache—would exceed an application’s response requirements. It also helps identify the key application requirements that microcontrollers fill.

tion logic, multiple-instruction-issue engines, out-of-order execution, and speculative execution.

To maintain the highest level of flexibility, microprocessors rely on external memory, devices, and controllers to interact with the real world. They can run large, complex operating systems and receive support from a mature development-tool infrastructure that enables significant reuse of legacy software. Microprocessor-based systems are ideal for quick and dirty rapid prototyping and proof-of-concept exploration in which cost, power consumption, and system size can take a back seat to a shorter development cycle. An example of this situation is evident in the choice of processors for most of the entries in the DARPA (Defense Advanced Research Products Agency) Grand Challenge (**Reference 4**).

The processing sweet spot for general-purpose microprocessors is to support systems that exhibit high uncertainty in processing behavior and loads, such as when executing, managing, and multitasking disparate processing threads over the same processor resources. Desktop and handheld computers are common examples of these systems. The flexibility of microprocessors enables them to most easily support the development of user interfaces and high-level application code.

Microprocessors are available commercially in dozens of configurations. Common microprocessor architectures include x86, ARM, MIPS, and PowerPC, and each architecture can rely on support from a number of suppliers. In contrast, microcontrollers are available as dozens of architectures and in device configurations that number in the thousands. (**Reference 5** lists dozens of microcontroller suppliers and the devices they offer.)

Microcontrollers are specialized processors that trade away the flexibility and general processing performance of microprocessors to gain an advantage in cost and power efficiency. They accomplish this feat by delivering single device configurations that optimize on-chip resources to best meet the needs of their target applications. Each device integrates a processor core with on-chip memory, timers, I/O lines, and appropriate peripherals and device controllers. Examples of on-chip peripherals include ADCs; DACs; PWMs; and serial-com-



## RTOSs have fewer features than microprocessor operating systems, so they can better support deterministic system operation.

munication interfaces, such as I<sup>2</sup>C, SPI, and UARTs. In general, microcontrollers are self-contained processing and control systems that require significantly fewer external components than microprocessors to operate.

Microcontroller suppliers differentiate their products through features such as processing performance, integrated peripheral sets, on-chip memories, memory management, packaging options, power management, and development support. A single microcontroller family often comprises many similar and derivative configurations that offer designers a range of prices for varying amounts of processing and on-chip-resource support. Microcontrollers are more cost-efficient but less flexible than general-purpose microprocessors because they optimize board space, code size, and power dissipation for their target embedded-system applications.

Another difference between microcontrollers and microprocessors is the level of abstraction of the available resources they present to designers to better meet the constraints of real-time processing. Whereas designers using microprocessors usually rely on richly featured operating systems to abstract and manage the peripheral resources, designers using microcontrollers access the on-chip peripheral and controller resources directly or through homespun or commercially available RTOSs. RTOSs have

fewer features than microprocessor operating systems, so they can better support deterministic system operation.

The processing sweet spots for microcontrollers are systems that need to be able to deterministically respond to external real-world events, such as for machine and motor control, with tight latency tolerances. Microcontrollers are capable of rapid, frequent, and prioritized context switching. Many employ specialized hardware-interrupt processing that enables the system to sense and react to external events within a few clock cycles to provide a small and deterministic response window appropriate for real-time control applications. This fast, deterministic interrupt-response capability contrasts with the slower and less deterministic interrupt system of the microprocessor, which may route interrupt handling through the operating system.

Multiprocessor designs offer another emerging sweet spot for microcontrollers. Small microcontrollers, such as 8-bit devices, can operate at the lowest power consumption of all the processor options. These devices are capable of deep-sleep modes that draw extremely low current, and they can wake themselves up based on an external event or an elapsed time. This ability makes them ideal for handling periodic tasks, including health monitoring and system-bring-up sequencing, for systems with aggressive power-consumption targets or that exhibit long periods of idleness. In this case, the small microcontroller can offload these periodic or simple tasks from a larger or more complex processor that would consume more power to perform the tasks.

Similar to microcontrollers, DSPs are specialized processors that trade off the flexibility and general processing performance of microprocessors to gain an advantage in cost and power efficiency. However, DSPs differ from microcontrollers because they sacrifice strength at multitasking or handling context switching for optimizing the on-chip resources

### FOR MORE INFORMATION

**Actel**  
[www.actel.com](http://www.actel.com)

**Altera**  
[www.altera.com](http://www.altera.com)

**Analog Devices**  
[www.analog.com](http://www.analog.com)

**ARM**  
[www.arm.com](http://www.arm.com)

**Criticalblue**  
[www.criticalblue.com](http://www.criticalblue.com)

**Freescale**  
[www.freescale.com](http://www.freescale.com)

**Infineon**  
[www.infineon.com](http://www.infineon.com)

**The Mathworks**  
[www.mathworks.com](http://www.mathworks.com)

**Microchip**  
[www.microchip.com](http://www.microchip.com)

**MIPS**  
[www.mips.com](http://www.mips.com)

**National Instruments**  
[www.ni.com](http://www.ni.com)

**Stretch**  
[www.stretchinc.com](http://www.stretchinc.com)

**Texas Instruments**  
[www.ti.com](http://www.ti.com)

**Xilinx**  
[www.xilinx.com](http://www.xilinx.com)

to support continuous and extensive computational processing on a stream of data in real time.

DSPs support fractional-number types, and they employ specialized address generation for arrays, circular buffers, and bit-reversed algorithms. Their multiple bus and memory structures enable simultaneous memory operations that support single-cycle MAC (multiply-accumulate) execution. Their specialized registers minimize memory accesses and enable zero-overhead looping. The specialized structures that DSPs employ keep the computational units fed with new real-time data. The structures a DSP architecture employs depend on the target application. (**Reference 6** lists DSP suppliers and the devices they offer.)

DSPs differentiate themselves by their sustainable computational performance, code density, power consumption, and development support. Unlike with microprocessors and microcontrollers, signal-processing developers' focus on computational efficiency and on minimizing the amount of context switching in signal-processing applications leads to little or no need for RTOS support. Contemporary DSP architectures employ orthogonal-instruction-set architectures that enable efficient compiler support for most signal-processing code.

The processing sweet spot for DSPs is in delivering high and continuous computational performance at the lowest cost and power consumption. The upper limit of this sweet spot increasingly overlaps with FPGAs, except for highly computational algorithms that include a high level of computational control complexity. VOIP (voice over Internet Protocol) is an example of a computationally intensive algorithm that does not work well as an FPGA-only implementation, because the domain-specific knowledge for compressing VOIP data exhibits too many computational control states and decisions. In the VOIP case, the algorithm implements more cost-effectively as software. The lower limit of this sweet spot overlaps with processors containing unified architectures that combine control and signal-processing structures in a single instruction engine.

Unified-processor architectures are closer to microcontrollers and DSPs than they are to microprocessors. Within a sin-



**Because of the general availability of licensable or integrated accelerators, they will decreasingly be a source of differentiation for a design.**

gle instruction engine, they either extend microcontroller architectures to add support for signal processing, or they incorporate circuitry to handle control logic to DSP architectures. Unified-processor architectures often support running an RTOS. They are less optimized and specialized than microcontrollers or DSPs, because they must include extra circuitry to bridge the disparate behavior of control and computational processing. Because the same instruction engine does the control and signal processing, it may be easier to more tightly couple the shared data between the two processes.

The sweet spots for unified-processor architectures are systems with light control or signal-processing requirements, because they can alleviate the need for two heterogeneous processors within the design—lowering the system bill of materials. An advantage of using unified processors in a multiprocessor design is that, even if you partition the processing between the processors as control and signal processing, the designers of both processors are using the same tools; it may be possible to host and port code developed for one processor in the system to the other processor with minimal impact on the development effort.

## SOFTWARE AS HARDWARE

Another way to bring processing performance into a design at lower cost or power consumption is by employing application-specific hardware accelerators and coprocessors. Such accelerators may be available as discrete devices or as stand-alone IP (intellectual property) that an FPGA can host. If an accelerator

has proved useful for high-volume applications, a processor supplier may integrate it into some of its processor devices; for example, a Texas Instruments DSP integrates the hardware accelerator for the Viterbi decoder. Because of the general availability of licensable or integrated accelerators, they will decreasingly be a source of differentiation for a design. The value in these accelerators is that they offer the opportunity to offload processing capacity to add differentiating features to a system or to reduce the system cost or power consumption.

Last year saw an explosion in the number of tools publicly available to assist developers wanting to accelerate an algorithm in a hardware implementation (**references 2 and 3**). These tools are noteworthy because they enable designers to more easily create custom accelerators or instruction-set extensions. The ability to create custom accelerators can be a longer lasting differentiation, because the acceleration is generally unavailable to the public and may not become a commodity capability as quickly as generally available acceleration IP.

In most cases, these tools target the software-to-hardware implementation to an FPGA. The variety of methods for translating from models, source code, and, in the case of Criticalblue, object code to a hardware implementation suggests that industry participants will in the coming years expend much energy on this form of increasing processing performance. Some of these implementations are fixed-function RTL blocks, and others target a proprietary processor block that the tool configures to optimize the function performance.

The compiler tool from Stretch differs from the other tools because it targets the acceleration to its mixed-processor device, which combines in a single device a processor core with a tightly coupled, programmable fabric that is integrated with the processor pipeline. One goal of the compiler tool is to abstract the pro-

**MORE AT EDN.COM**

[+](http://www.edn.com/060119cs) Go to [www.edn.com/060119cs](http://www.edn.com/060119cs) to find this article's associated references.

[+](http://www.edn.com/microdirectory) For a comprehensive listing of microprocessors and microcontrollers, go to [www.edn.com/microdirectory](http://www.edn.com/microdirectory).

grammable logic so that the designer can focus on the algorithm and function rather than the implementation. Expect to see a resurgence of mixed-processor devices that integrate both a processor core and an FPGA, most likely with analogous tools to automate the translation of software to hardware.

Linking an FPGA containing an accelerator or coprocessor with a host processor or a DSP is becoming more common in high-performance consumer applications. The falling cost of FPGAs is enabling designers to lower overall design cost and power consumption by offloading the heavy-lifting computations to the FPGA and allowing designers to use a less powerful host processor or DSP. The drop in overall power consumption is possible for applications that can take advantage of the extreme parallelism that an FPGA affords—especially if the FPGA implementation allows the system to use a significantly lower clock rate than it would with a higher performance processor.

Algorithms that exhibit significant parallelism or that you can pipeline to meet tight timing constraints are the processing sweet spots for an FPGA used as a coprocessor to a host processor or DSP. However, in general, the FPGA cannot efficiently replace the entire host processor or DSP from a cost and power perspective, especially in systems that exhibit complex control structures. Recognizing this situation, FPGA suppliers offer proprietary processor cores that the FPGA can host. The tools from the FPGA vendors support configuring the processor core with standard peripherals as well as loosely or tightly coupling a design's custom accelerators with the processor core. In short, the tools are making it easier for designers to use the FPGAs as low-volume SOC (system-on-chip) substitutes.

## CHALLENGES

Challenges remain for designers who need to employ a mix of these processing options in their designs. A challenge for the tools that will support mixed-processing designs is how to preserve the programming model and allow designers to exploit the available parallelism. Maintaining a high level of productivity of the development team is essential for the adoption of these mixed-processing options and the tools that support them. It is not uncommon for a design team to

exclude candidate processor architectures from consideration because the team lacks experience with the development tools that support them.

A company's legacy code represents a significant amount of engineering and number of lessons learned about the design. Unless the design team is willing to rewrite the application code from scratch, the legacy code will need a repartitioning effort to exploit the available parallelism; however, this effort may be unable to yield clean and natural breaks. Poor repartitioning is analogous to poor data-structure implementation; a data structure that does not naturally and cleanly capture and represent the application requirements leads to troublesome coding and debugging sessions.

Key to preserving the productivity of developers with legacy code and adopting mixed-processing designs is more mature compiler controls. Unless the compiler has full visibility of all of the system resources, it has to make conservative assumptions about resource allocation and usage. To improve compiler performance, new skills are likely to evolve that allow developers to tighten and loosen rules and assumptions that the compiler uses to improve its ability to generate good results from legacy code.

The trend is toward processor suppliers' continuing to provide vertically targeted devices. These devices will integrate an appropriate set of processor resources, hardware accelerators, and programmable logic for target applications, but such devices will exist only for high-volume applications. For all the other applications, designers will have to put together their own mixed-processing configurations. It is becoming clear that using distributed- and multiple-processor topologies is the way to continue the relentless push for more processing performance. The challenge for today's tool providers is to create ways to better encapsulate tasks and implement support for mixed-processor configurations. **EDN**

You can reach  
Technical Editor  
**Robert Cravotta**  
at 1-661-296-5096  
and [rcravotta@edn.com](mailto:rcravotta@edn.com).

