

**CONFIGURABLE PROCESSORS
CAN BE A COST-EFFECTIVE
INGREDIENT IN YOUR DESIGN.
BUT THEY MAY ALSO BE AN
EVOLUTIONARY ABSTRACTION.**

Stir

coverstory *By Robert Cravotta, Technical Editor*

A

SK ENOUGH PEOPLE what defines a configurable system, and you will realize that each definition depends on the person's assumptions about how to abstract the system components. For embedded systems, the abstraction usually applies to the system's hardware components and any accompanying software. A common thread for defining a configurable system is that you have the flexibility to change a system's features and behavior without materially changing the system platform. What differs between each definition of configurability is the interpretation of what materially changing the system platform means (see sidebar "Configurable perspectives").

Your application's cost and flexibility requirements drive the need for and value of each type and level of configurability. Costs can manifest themselves in your project as first-to-market-opportunity costs; upfront, nonrecurring costs; recurring bill-of-materials costs; and development-support costs. Platform flexibility allows you to change your design and add capabilities to it without scrapping your work and incurred costs. Flexibility allows you to more easily reuse your work

At a glance42
Configurable perspectives42
Supporting your legacy44

Illustration by Colin Johnson

across multiple designs, accommodate changing requirements, correct design errors, and "future-proof" standards-based designs, such as for communication protocols.

Using an ASIC to implement a fixed function in your design makes the most efficient use of silicon. An ASIC lets you balance and optimize your design for higher performance with lower recurring costs and lower power requirements. Offsetting these ad-

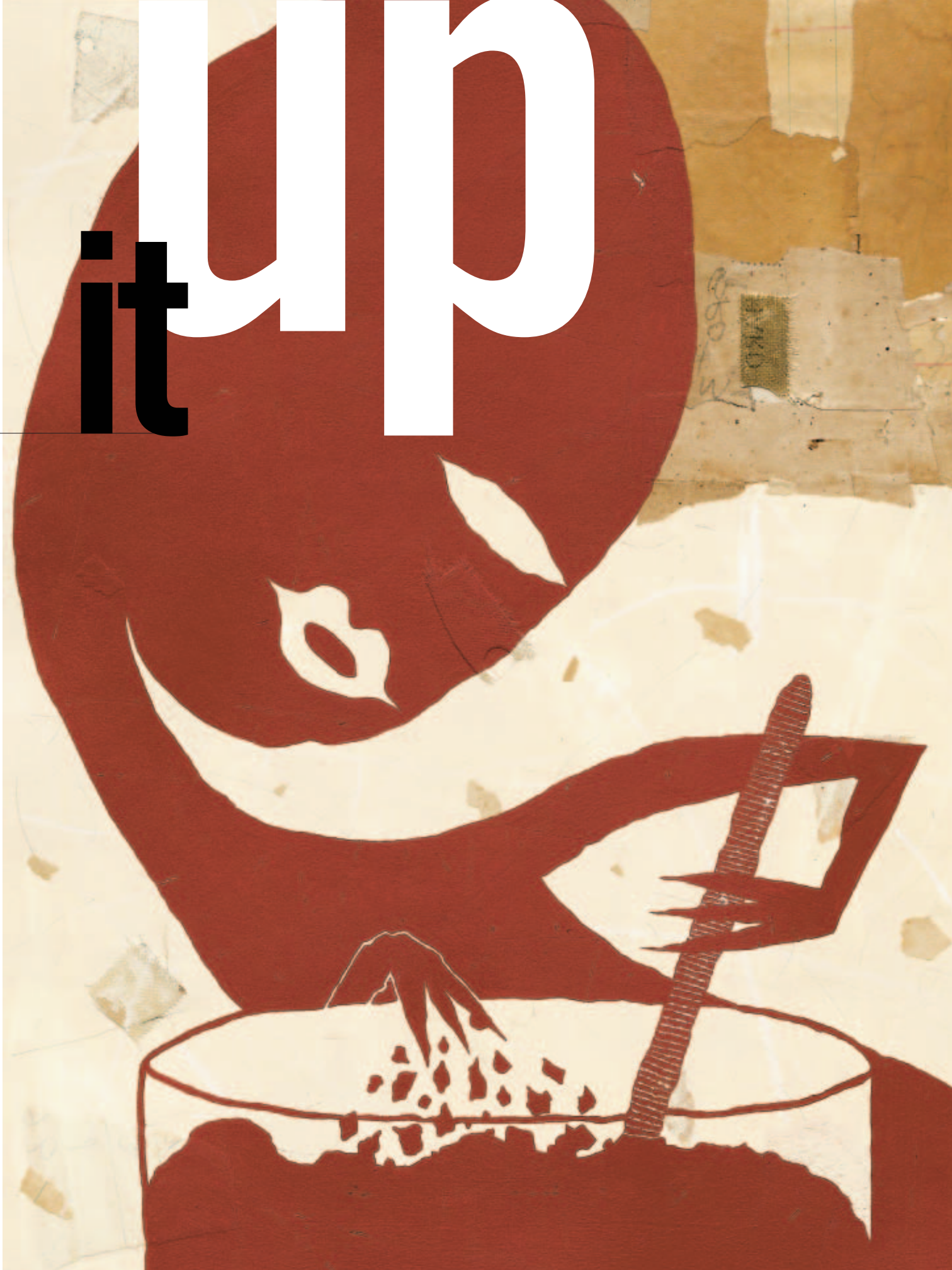
vantages is the fact that the high, upfront, nonrecurring costs you incur to build an ASIC necessitate that you amortize those costs across large volumes. A fixed-function ASIC has poor flexibility that can eliminate your ability to amortize a single ASIC design across multiple projects and can require you to perform a redesign and again incur the nonrecurring costs to add new capabilities to your application. Finally, ASIC development has a longer cycle time than other approaches, because you must design and test your circuit, fabricate the wafers, and validate your implementation.

Using ASSPs (application-specific standard parts) in your design allows you to reduce your development time and trade upfront ASIC costs for higher recurring costs, because the device vendor has already incurred those costs. If an appropriate ASSP is available, it is unlikely that you are the first to market. You must therefore consider differentiation challenges, because your competitors have access to the same device. Like ASICs, ASSPs are not flexible platforms; they are inherently specific and optimized to a target application. According to iSuppli (www.isuppli.com), the 2002 ASIC market was down 28% from the 2000 market high, primarily because of weakening demand for wired communications and partly because of an overall shift to standard products, such as ASSPs, which showed a growth of 1.6% in 2002.

ASSPs (application-specific programmable parts) or SOCs (systems on chips) improve on an ASSP's inflexibility by sacrificing some performance, silicon (cost), and power efficiency for software programmability. These devices differentiate themselves by integrating an optimized set of peripherals, memories, interface controllers, and application-specific hardware accelerators with a DSP, a microcontroller core, or both.

Standard general-purpose, software-program-

it up



mable devices, such as DSPs, microcontrollers, and microprocessors, do not integrate application-specific hardware accelerators, and they integrate a more general set of peripherals, memories, and interface controllers than do ASPPs. They sacrifice some combination of higher performance, lower power, and lower cost to support a wider variety of applications, but, because they are useful for a variety of applications, their development tools and industry-support infrastructures are more mature than those of ASICs and ASSP devices. There is also a larger pool of experienced software developers for these instruction-set-processing platforms.

All of these devices employ fixed architectures that offer a spectrum of options to maximize and minimize system performance; power; upfront, nonrecurring costs; recurring costs; development time; and flexibility. To increase your sys-

AT A GLANCE

- ▶ Configurable processors provide mechanisms to configure hardware resources, implement custom instructions, or do both.
- ▶ Custom instructions effectively convert software into hardware.
- ▶ Configurable architectures shine for applications with evolving and high-performance signal-processing requirements.
- ▶ Reconfigurable architectures may parallel the evolution and adoption of DSPs.

tem's performance with these devices, you can redo how you implement your algorithms, change your processor choice, or increase the clock speed. The workload requirements for applications with heavy digital-signal processing can

exceed the maximum workload your processor can perform at the fastest clock speed. Your processor choice and application specifics may allow you to scale your performance by operating multiple processors together, but adding processors to your design can complicate it by introducing communication requirements to coordinate between the processors. Adding processors or fixed-function accelerators to your design can push your power consumption, bill-of-materials cost, communication latencies, and board-space requirements beyond what you can afford.

ADD A DASH OF CONFIGURABILITY

Configurable hardware architectures can help you when available fixed-architecture approaches cannot meet your requirements or when device-level flexibility is important. Configurable architectures allow you to balance perform-

CONFIGURABLE PERSPECTIVES



A configurable system allows you to specify or build in,

within some level of granularity, the features or capabilities you need. It aims to meet your needs and lower your costs by avoiding charges for capabilities that you do not need. As an example, when you are buying an automobile or a desktop computer, the options you choose affect the final price; choose more options, and your cost is higher. The system platform and basic function remain unchanged, but the options can make the system performance better meet your needs.

However, the vendor offers a limited set of options; it may not offer all the features you need, or it may even cause you to pay for features you do not want.

The automobile and desktop-computer examples are analogous to the situation that occurs when you are selecting a standard processor device for your embedded system. Limiting your processor selection to devices

within the same family allows you some board-level configurability that you can use across multiple designs. Devices within the same processor family share the same execution engines, and they usually differ in the types and amounts of integrated memories, peripherals, and external interface support and maintain software compatibility and pin-compatible packages. The cost of each device correlates with the amount of integrated resources; however, your choice of configurations is limited to the processor vendor's offerings.

Software programmability extends below board-level configurability, is a fundamental capability for instruction-set-architecture-based processor platforms, and allows you to use the same processor device across multiple designs. Software programmability provides the flexibility to modify your design until the production configuration. And, if you are using the proper memories, such as flash, you can fix and upgrade your system in the field with a soft-

ware download. This ability allows you to better accommodate evolving requirements without scrapping your entire design.

Software programmability provides a high level of system configurability and flexibility, but the instruction-set architecture your choice of processor supports can significantly impact the effectiveness of real-time signal-processing code based on the processor's inclusion or exclusion of accelerated mathematical instructions and execution resources. Hardware-configurable processors extend configurability below software programmability to offer cost, power consumption, and performance alternatives when the available set of standard processors cannot cost-effectively support your design requirements.

Configurable processor architectures provide mechanisms for configuring the processor resources, extending the instruction-set architecture, or both. Besides configuration and instruction-extension support, you can further classify config-

urable processors into processors that are configurable until silicon-fabrication time, processors that are reconfigurable during system-loading time, and processors that are dynamically reconfigurable during runtime (Table 1).

IP (intellectual-property) processor cores are configurable until silicon-fabrication time, and they do not support reconfigurability unless you explicitly integrate mechanisms for it, such as FPGA blocks. Processors that are reconfigurable at system-loading time can change configuration but often do so only as a function of a design update or as a discrete, operational-mode change, such as switching from processing one codec or protocol algorithm to another. A dynamically reconfigurable processor, on the other hand, can adjust the hardware configuration as frequently as on a clock-by-clock basis to implement only the portions of a state machine or an algorithm that it needs at that moment.

ance, power consumption, and the cost of silicon area, and it can provide higher performance, lower power consumption, and lower cost, because you need fewer clock cycles and less logic to compute the same result as your programmable, fixed-architecture implementation. Configurable architectures provide mechanisms for configuring the system resources, extending the instruction-set architecture, or both.

Configuration mechanisms may allow you to tune your system to application requirements by changing the processor resources, such as by adding, deleting, or resizing various memories, such as caches; changing bus widths; creating special registers and buses; duplicating execution units, such as ALUs and MACs, to improve instruction-level parallelism; integrating custom peripherals; and even creating multiprocessor systems. This flexibility allows you to tune your system and address system bottlenecks for higher performance; however, a configurable architecture can lengthen the design process when the software-development tools cannot easily use the additional resources for performance optimizations.

Extending the instruction-set architecture effectively converts software into accelerated hardware and abstracts the implementation into the software-instruction-set architecture. Application-specific instructions can improve by orders of magnitude the amount of work an application can perform per clock cycle and reduce the lines of software code

FPGAs ARE OTHER STANDARD DEVICE OPTIONS FOR IMPLEMENTING RECONFIGURABLE AND EXTENDABLE ARCHITECTURES.

it needs. These custom instructions usually simplify software development, debugging, and verification efforts at the expense of upfront profiling analysis that compares custom instructions with pure-software execution. Extending the instruction set can lengthen the development process if the software-development tools lack simple or automatic support to incorporate the custom instructions into the compiler and simulator.

Configurable architectures are available across a similar range of implementations to those of fixed-architecture approaches to maximize and minimize system performance; power; up-front, nonrecurring costs; recurring costs; and development time (**Table 1**). Configurable and extendable IP (intellectual-property) processor cores support configurability to the point of silicon fabrication and have similar advantages and disadvantages to those of ASICs. Development tools for these architectures

may include performance tools that can provide you with immediate estimates on the performance, die size, and power requirements of a design.

Configurable processors can support field hardware updates and operational-mode reconfigurability, such as switching from processing one codec or protocol algorithm to another, by integrating a programmable logic block with a hard processor core and a minimum set of peripheral blocks. A number of available standard processor products include this level of integration. They allow you to customize a peripheral set and implement custom instructions, but they do not support reconfiguring the base resources of the processor core. You gain the extra flexibility of these systems at increased cost, because the reconfigurable portion of the device requires more silicon than it would if you had implemented it using fixed components.

FPGAs are other standard device options for implementing reconfigurable and extendable architectures, and developers successfully use them as algorithmic coprocessors to standard processor cores. Standard FPGA devices can also include soft- or hard-processor cores to support software programmability and to expand their use by software developers. FPGAs' high-flexibility and foundry-independent logic makes them strong candidates for prototyping configurable designs and for delivering low-volume, high-margin applications. These devices avoid the upfront costs of ASICs, but they consume more silicon space, have poor power efficiency, and have a high recurring cost that can be a concern for high-volume applications.

Dynamically reconfigurable architectures can drive configuration flexibility to the extreme by supporting single-clock-cycle reconfiguration and replace the concept of instruction sequencing with configuration sequencing for data flow and task-level parallelism. Dynamically reconfigurable architectures can implement functions needed at a given moment and then reuse the same block of gates or computational units for a different function at the next moment. This flexibility entails extra cost and complexity for handling more power-consuming, fast-loading circuits and higher complexity for switching methods and circuits. Currently, a small set of appli-

SUPPORTING YOUR LEGACY



One potential issue with configurable architectures is whether future systems will have to maintain backward compatibility. Implementing custom instructions will increase your need to accommodate backward compatibility to support the legacy code on future platforms. Such a provision increases the amount of documentation and engineering skills you need to maintain future legacy code beyond purely managing the software code.

You will need someone that can cross traditional skill boundaries not just to maintain the software code, but also to coordinate and maintain the associated hardware and the tools. If you must hand-generate or modify your tools to support your custom instructions, you may end up dedicating additional internal resources to maintain your tools; however, doing so may be acceptable if those tools provide you with a competitive advantage.

Another consideration for legacy support is how well your custom hardware, software, and tool extensions integrate into your system-configuration-management tools. When managing these custom components, you need to consider an additional set of error sources that users of fixed architectures can generally assume that the hardware and tool vendor have handled.

cations can benefit from this level of reconfigurability, because it enables a level of gate-reuse efficiency that can allow you to implement the application in a smaller silicon area with lower system-power consumption than other approaches.

SOFTWARE IS STILL THE SPICE

Software development influences most current design efforts. A configurable architecture's flexibility and productivity are limited without software programmability, especially for applications that

support multiple evolving and emerging standards or protocols. According to many vendors of configurable, programmable architectures, developers often first approach them to explore using the configurable architecture as a stopgap hardware accelerator with legacy design and software assets that cannot scale up performance to meet new processing workloads.

To effectively use a configurable architecture, you need to expend extra effort during system analysis to explore how to best partition your hardware and soft-

ware. The system-analysis, software-development, and system-verification tools that support your architecture choice heavily impact the effectiveness of exploring partitioning options. The system-analysis tools should help you identify underused hardware resources that are candidates for elimination; resources you could add or reorganize to improve application-specific performance; and code that is a good candidate for custom instructions, because it consumes high percentages of processing cycles. The software-development tools should sim-

TABLE 1—CONFIGURABLE AND RECONFIGURABLE ARCHITECTURES*

Company and Web site	Class	Architecture	Notes
Altera, www.altera.com	FPGA—reconfigurable logic	Fine-grained reconfigurable	Hard-(ARM) and soft-processor cores
ARC, www.arccores.com	IP—von Neumann architecture	ARCTangent, ARCLite	
Atmel, www.atmel.com	Chip and IP—reconfigurable logic	AVR core with fine-grained FPGA-like reconfigurable	
Broadcom, www.broadcom.com	IP—internal	Coarse-grained reconfigurable	Acquired Silicon Spice Calisto, Element 14
ChipWrights, www.chipwrights.com	IP—von Neumann architecture	SIMD/Vector DSP	
Context Inc, www.contextdrl.com/	Dynamic reconfigurable logic	IP—Dynamic reconfigurable logic	Coarse-grained—dynamic reconfigurable logic start-up
Cradle, www.cradle.com	Chip—dynamic reconfigurable logic	Coarse-grained reconfigurable	Adaptive technology
Cypress Microsystems, www.cypressmicro.com	Chip and dynamic reconfigurable logic	Programmable-system-on-chip, 8-bit core with dynamic reconfigurable-logic peripheral blocks	Reconfigurable digital and analog peripheral blocks
Elixent, www.elixent.co.uk	IP—dynamic reconfigurable logic	Coarse-grained reconfigurable	Licensed to Toshiba's MeP
Equator, www.equator.com	Chip—von Neumann architecture	VLIW DSP	Media processing
Improv, www.improvsys.com	IP—von Neumann architecture	Jazz DSP	VLIW
Infineon, www.infineon.com	IP—prototype chip	Scalable-array architecture	Acquired MorphICs flexible array
Intrinsity, www.intrinsity.com	Chip—von Neumann architecture	MIPS with 2-GHz Matrix Math Engine	Math co-processor
IP Flex, www.ipflex.com	Chip—reconfigurable logic	DAP/DNA coarse-grained reconfigurable	NPU—targets networking
MIPS, www.mips.com	IP—von Neumann architecture	MIPS Pro Series, MIPS-3D ASE	
Morpho Technologies www.morphotech.com	IP—dynamic reconfigurable logic	Coarse-grained reconfigurable	RDSP cores
Motorola, www.motorola.com	Chip—wireless chip set	StarCore, PPC, coarse-grained rCF	Using Morpho Tech's MS1-16 RDSP core
Pact, www.pactcorp.com	IP—dynamic reconfigurable logic	XPU family	
Philips, www.philips.com	IP—internal	Coarse-grained reconfigurable	Acquired Systemonics
picoChip, www.picochip.com	Chip—dynamic-reconfigurable-logic adaptive array	Coarse-grained reconfigurable	Working silicon and demo systems
PMC-Sierra, www.pmc-sierra.com	IP—internal	Flexible DSP	Acquired Malleable MECA family
Protean Devices, www.proteandevices.com	Chip—dynamic reconfigurable logic	ASSPs with coarse-grained dynamic reconfigurable processor	
QuickLogic, www.quicklogic.com	FPGA—reconfigurable logic	Fine-grained reconfigurable	Hard- (MIPS) and soft-processor cores
QuickSilver, www.qstech.com	IP—Dynamic reconfigurable logic	Coarse-grained reconfigurable	ACM (Adaptive Computing Machine)
SandBridge Technologies, www.sandbridgetech.com	Chip—von Neumann architecture	Multithreaded VLIW DSP core	
Systolix, www.systolix.co.uk	IP—Dynamic reconfigurable logic	PulseDSP—systolic array	Licensed through and to Analog Devices
Tensilica, www.tensilica.com	IP—von Neumann architecture	Xtensa	
3DSP, www.3dsp.com	IP—von Neumann architecture	SP-5Flex	
Toshiba, www.mepcore.com	IP—Custom systems on chips	Media embedded processor	Licenses Elixent's technology
Triscend, www.triscend.com	Chip and IP—reconfigurable logic	ARM core with FPGA-like fine-grained reconfigurable	New IP direction, tie to www.insytec.com
Xilinx, www.xilinx.com	FPGA—reconfigurable logic	Fine-grained reconfigurable	Hard- (PPC) and soft-processor cores

*Courtesy Forward Concepts

Other companies that provided information for this article include: Accel Chip, www.accelchip.com; Altium, www.altium.com; Analog Devices, www.analog.com; CoWare, www.coware.com; Forward Concepts, www.fwdconcepts.com; Pentek, www.pentek.com; Texas Instruments, www.ti.com.

ply or, better yet, automatically incorporate your configuration and instruction extensions. The speed, quality, and integration of your co-simulation, emulation, analysis, and profiling tools directly affect the number of partitioning configurations you can explore and compare.

Designing with configurable architectures changes how you explore system partitioning from fixed architectures, because you retain the flexibility to change your configuration until the last moment. A flurry of effort to explore partitioning configurations typically characterizes the first few weeks of a development project with a configurable architecture. Then, partitioning-exploration activity drops off, and system integration becomes the focus. The last few weeks of a design using a configurable architecture might see a resurgence of partitioning exploration for final tuning and optimization opportunities.

Configurable and instruction-extendable architectures effectively enable you to convert your software into accelerated hardware and allow a lot of freedom in how you can implement your system (see sidebar "Supporting your legacy"). Software developers, in general, are not experienced in implementing hardware-instruction acceleration, but software developers represent a large target market for configurable architectures. An apparent goal for configurable-architecture-development tools is to present a development environment that tries to make implementing hardware-instruction acceleration as intuitive as possible to software developers. Inherent in meeting this goal is the ability to easily or automatically incorporate custom instructions into the software-development tool chain.

CONFIGURABILITY AS A STOCK INGREDIENT

Flexibility is not free; so, is configurability cost-effective? A standard, fixed-architecture processor device is usually a more cost-effective implementation if it

exactly meets your design requirements. The catch is that the device needs to exist and be available when you need it. Will the device road map meet your needs at the next iteration of your application? Will you be able to continue leveraging your legacy assets? High-performance, application-specific signal processing does not constitute an entire

design, so a completely configurable implementation is impractical for most applications. A better approach for many applications is to mix fixed and configurable components as tightly coupled coprocessors to leverage the advantages of both approaches.

As persistent network connectivity becomes more pervasive, these same reconfigurable architectures provide the basis for field reconfiguration for remote monitoring, testing, and upgrades. Fixed-architecture processors with flash memories already support field reprogrammability. It is not too much of a stretch to expect that analogous hardware reconfigurability will become as important and pervasive for applications with variable and growing signal-processing requirements.

Besides being able to cost-effectively meet application-specific, high-performance, digital-signal-processing requirements, system-load-time configurable architectures are a strategy for implementing multiple product derivatives from a single design approach. For developers, many of whom lack a hardware background, important challenges and opportunities exist for tools to simplify how to directly implement algorithms as hardware.

Not that many years ago, programming for signal processing was an esoteric topic, but recent evolutions in tools have opened and continue to open the world of signal-processing programming to a wider audience. □

FIXED-ARCHITECTURE PROCESSORS WITH FLASH MEMORIES ALREADY SUPPORT FIELD REPROGRAMMABILITY.

