hands-on project

By Robert Cravotta, Technical Editor

ACCELERATE YOUR PERFORMANCE

Illustration by Doug Fraser

The options and development tools available to designers for implementing custom hardware acceleration are evolving. Custom hardware acceleration is an increasingly viable method for implementing parallel processing that balances performance, power consumption, and cost and brings these features to a new threshold. Albert Wang, chief technical officer of Stretch, explains that the processor vendor recently introduced a software-programmable processor with an integrated reconfigurable hardware-acceleration technology, "because time to performance is critical in meeting system design requirements for today's leading-edge compute-intensive applications."

This first of a two-part hands-on series provides an overview of the project effort, options, and tools for hardware acceleration, as well as the process for analyzing the software, identifying what to accelerate, and implementing the hardware acceleration. Part two, which will appear in *EDN*'s Dec 7, 2004, issue, will focus on hardware-acceleration tools that provide an additional layer of abstraction by operating directly on system models or the software source.

Design efficiency and development productivity are paramount concerns for embedded-system designers. Design efficiency, in the context of this article, encompasses how well a design handles its processing-performance requirements and meets all of its timing, power, and cost constraints. Embedded designs that can surmount complex performance and constraint requirements present an opportunity for a development team to identify and deliver differentiated, value-added features, because the complexity can act as a barrier to competitors. It

At a glance 52
Comparing performance 52
Reconfigurable acceleration....54
For more information 56

THIS TWO-PART SERIES EXPLORES THE INCREASING OPTIONS FOR HARDWARE ACCELERATION AND HOW DESIGNERS WITHOUT HARDWARE BACKGROUNDS CAN BENEFIT FROM EVOLVING DEVELOPMENT TOOLS.

is difficult to offer unique, value-added, differentiating features when a design does not tackle and incorporate sufficiently complex requirements, because competitors can quickly duplicate and incorporate the best ideas into their own product offerings.

However, that complexity barrier is temporary, because competitors will focus their efforts to compete with your best ideas. You will lose your first-to-market advantage if you cannot quickly and adequately adjust your product's features and costs to stay ahead of your competitor's efforts. Development productivity balances out design efficiency. It encompasses not only the time and resources required to complete your current design and implementation effort, but also the reusability of the results in the inevitable follow-up projects. If your development processes and tools cannot abstract and automate nondifferentiating portions of your design effort better than your competitors' processes and tools can, you may open the door for those competitors to outdo you in innovation. You may have to concentrate more valuable development time and resources than your competitors on those nondifferentiating details.

An ASIC is a semiconductor device that you can optimize to an application by implementing only those performance features an application needs to meet requirements. An ASIC can best balance performance, cost, and power, but its implementation incurs a longer development cycle and substantial nonrecurring engineering costs that are difficult to recover when there are rapidly evolving feature requirements.

Programmable platforms reside at the other end of the spectrum. They emphasize productivity over efficiency and allow both reusability and the flexibility to quickly modify or substitute code or logic to implement a new feature or function. Software-programmable systems can efficiently implement sequential processing but can be inefficient for parallel processing. Programmable hardware systems excel at implementing parallel operations but are generally less efficient than software-programmable systems at performing sequential operations. Software also excels at incorporating layers of abstraction that hardware cannot for cost efficiency and reusability. For example, you might redesign hardware to accommodate a different but functionally equivalent hardware component, but software for such a system may have to operate on both the old and the new configurations. As a result, the software abstracts a small variation in the hardware that is not functionally obvious.

The development tools for software- and hardware-programmable systems fundamentally differ. The tools that software and hardware engineers use provide appropriate but different visualizations for the system behavior because of the predominant sequential or parallel nature of each type of programmability. Jim Hwang, director of DSP-design tools and methodologies at Xilinx, contends that HDL designers represent a small portion of the number of developers working on signal-processing applications. Most designers in the DSP world, he explains, use C and Matlab. According to Hwang, development tools, such as System Generator for DSP, "lower the barrier to use FPGA-based products by application programmers and systems architects who do not know VHDL." Many of the companies that support custom hardware acceleration are investing significant development resources to offer tools and devices that allow nonhardware engineers, such as system designers and software engineers, to implement parallelism in software as hardware acceleration.

Early on in this hands-on project, *EDN* had to make a decision about its scope.

AT A GLANCE

▷ The strength of software is to the strength of hardware as sequential operations are to parallel operations.

▷ Custom hardware-acceleration logic is but one of many ways to accelerate system performance.

▷ Hardware-acceleration tools are targeting and becoming friendlier to designers lacking a hardware background.

▷ Choosing between instruction extensions or custom coprocessor logic depends on a design's hardware needs.

COMPARING PERFORMANCE

During the early planning stages of this hands-on project, *EDN* considered comparing the acceleration difference of the same algorithm across each architecture. But we dumped the idea, because no application space is common to every architecture, and it would quickly become an apples-to-oranges comparison.

We next considered comparing and discussing the before-and-after performance metrics for each system. However, although this comparison might be useful, it would be subject to incomplete and inappropriate conclusions. For example, when accelerating the CRC (cyclic-redundancy-check) algorithm with Atmel, the hardware-implemented algorithm performed 23 times faster than the original software implementation. The software implementation was a loop-driven CRC, and comparing the acceleration difference with a table-driven CRC would have reduced the performance comparison. With more time and effort, we could have improved the performance of the hardware implementation. Analogous conditions exist for each acceleration effort.

Be careful not to compare a worst-case software implementation with a best-case hardware acceleration. When measuring hardware performance, it is a stand-alone execution unit, but software performance is part of a greater whole. For example, the memory organization or size of the processor cache or register files can affect the performance of an algorithm; therefore, some systems support configurability of these types of architectural features.

It is important to avoid implementing hardware acceleration to compensate for executing the software on an inappropriate processor architecture. For example, whether the processor includes a MAC (multiply/accumulate) unit or appropriate data-bus structure can significantly affect the software performance of a signal-processing algorithm. Our experience at Tensilica drove home this point when we implemented a five-operand adder that required more data than the bus structure could support in one clock cycle. Another example is implementing streaming DSP functions on a microprocessor that includes a hardware MAC unit but cannot handle the streaming-data load.

The project could have a narrow focus and deeply explore the process of accelerating an algorithm on a single platform, or it could approach hardware acceleration broadly (and consequently spend less time on each architecture) by examining several architecture and tool offerings. We chose the broader and shallower approach to better highlight the methods of the growing number of companies offering or supporting hardware acceleration. Just within the last few months, two companies announced their first products. Poseidon Design Systems introduced tools that synthesize application-specific hardware accelerators directly from standard ANSI C source code, and Stretch rolled out its catalog processor family and tool set, which enables hardware acceleration from ANSI C source code.

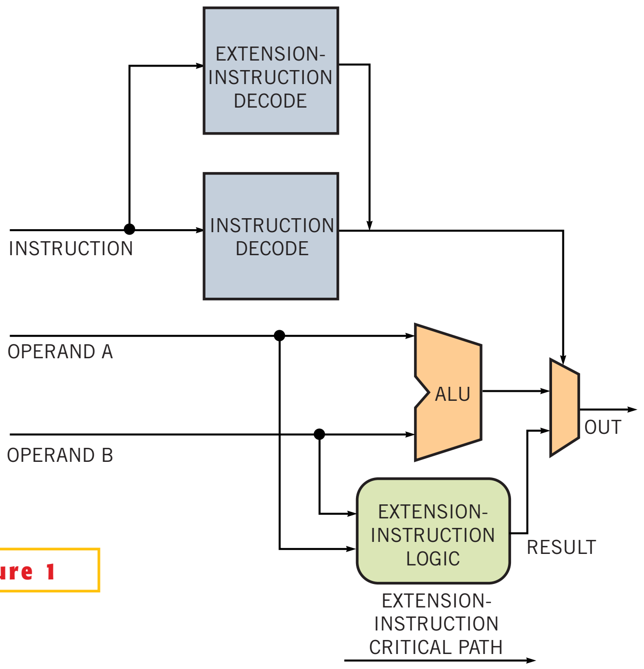

EXTENSION-INSTRUCTION DECODE
INSTRUCTION
INSTRUCTION DECODE
OPERAND A
OPERAND B
ALU
OUT
EXTENSION-INSTRUCTION LOGIC
RESULT
EXTENSION-INSTRUCTION CRITICAL PATH

Figure 1

A custom instruction is an extension of the processor's ALU (arithmetic-logic unit) and instruction decoder. The custom instruction logic may be a critical path that can affect the processor clock rate unless you implement pipeline-stall controls or instruction pipelining.

We spent time at Altera, ARC, Atmel, MIPS, Tensilica, and Xilinx, implementing hardware acceleration of algorithms of their choice on their workbenches. Performing the task at each site alleviated time constraints and simplified our effort to license and configure each development environment; it also meant an engineer could direct us through the process.

Each company chose an algorithm to accelerate without knowing what the other companies chose. Therefore, each company could choose the algorithm acceleration with which it had the most experience and that would demonstrate how the company's architecture and tools supported the analysis and implementation for each acceleration. By discouraging common algorithm acceleration between companies, the project effort could better focus on the process, tools, and architectural mechanisms and avoid the temptation to compare the efficiency of each acceleration effort (see **sidebar** "Comparing performance").

Altera provides programmable-logic devices, associated software tools, and IP (intellectual-property) software blocks. Its Excalibur devices integrate an ARM922T processor (that can operate at 200 MHz) with the Apex 20KE FPGA architecture. For this project, the programming and acceleration targets were the 32-bit Nios and Nios II soft-processor cores, which can reside in Altera's Stratix, Cyclone, and HardCopy FPGA devices. The Nios II core is available in three configurations, all of which support custom instructions. The standard-configuration Nios II/s balances

RECONFIGURABLE ACCELERATION

A reconfigurable platform is one way to reduce a design's size and cost, and it allows designers to realize the benefits of using hardware accelerators with software in an embedded design. Some systems, including one from Morpho Technologies, can reconfigure their logic in one clock cycle to accelerate application-specific tasks. Some FPGA-based systems can support runtime reconfiguration of the system; however, their reconfiguration requires many clock cycles.

The Reconf project (www.reconf.org) is attempting to demonstrate a transparent infrastructure for dynamic FPGA reconfiguration (**Reference** A). This effort addresses not only the actual dynamic reconfigurability of an FPGA device, but also the necessary development-tool support. Atmel's AT94K FPSLIC (field-programmable system-level IC) is the demonstration device for the current project. It supports runtime reconfiguration by relying on its ability to reconfigure part of the FPGA without affecting the operation of the rest of the device. Atmel's Figaro design-implementation tool does not natively support partial reconfiguration, so the project defines a special implementation procedure to generate the bit streams for all necessary coprocessor contexts.

To access and use each of the coprocessors (in this case, different floating-point operations), the application software calls a CallFPGA function. Because the FPGA is too small to simultaneously contain all the operations, the FPGA reconfigures the coprocessor as necessary within 50 msec, using an infrastructure that includes a reconfiguration controller in the FPGA.

Analogously, Stretch's ISEF (instruction-set-extension fabric) supports runtime reconfiguration by partitioning the ISEF into two dynamically loadable sections. This setup allows the device to reconfigure one section of the ISEF over many clock cycles while the rest of the device continues to operate normally.

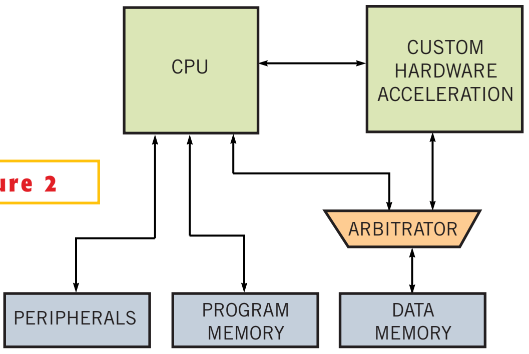
The Reconf demonstration and the Stretch product demonstrate how an application programmer can use many accelerated operations in a programmable-logic fabric tied to a processor without being familiar with the digital-circuit design.

Reference

A. Danek, M, P Honzik, J Kadlec, R Matsousek, Z Pohl, "Reconfigurable System-on-a-Programmable-Chip Platform," Seventh IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, April 2004.

performance and size (cost); the economy-configuration Nios II/e implements the smallest core—roughly half the size of the Nios II/s standard core—at the expense of performance and maintains compatibility with the Nios II ISA (instruction-set architecture); and the Nios II/f configuration maximizes the core for processing performance. Altera's SOPC (system-on-programmable-chip) Builder system wizard-driven development tool enables you to work with the MathWorks' Matlab and Simulink to develop, implement, and port algorithmic designs to HDL files for use with the Quartus II design software.

Figure 2



Custom acceleration as a coprocessor or peripheral allows the custom logic to operate loosely coupled to or independently of the processor architecture. The custom logic may also access system resources, such as memory, without processor intervention.

ARC licenses configurable and preconfigured processor and IP cores along with development software, profiling tools, and a real-time operating system. Its configurable processor cores support optional DSP extensions and scale from the four-stage-pipelined ARCtangent-A4 architecture to the seven-stage-pipelined ARC 700 architecture. The cores are synthesizable and configurable, and designers can extend the ISA.

This project targeted the 32-bit ARCtangent-A4 processor but also focused on how the project would differ using an ARC 600 processor (most notably a longer instruction pipeline and additional ALU/DSP extensions). The acceleration effort focused on implementing a packet-header-processing function as a custom instruction. The ARChitect configuration tool configures the cores and the Extension Instruction Automation tool suite to integrate Verilog instruction extensions to the pipeline. The tool automates the insertion of all control signals and structures to integrate your instruction to the pipeline and creates a library that the ARChitect tool suite can use.

At nearly all of the companies, we worked with a 32-bit architecture. However, at Atmel, we explored an 8-bit AVR microcontroller core integrated with an FPGA on an FPSLIC (field-programmable system-level IC) device. Hardware acceleration with an 8-bit device is more about lowering your processing costs than tackling a high-performance state-of-the-art algorithm. This project focused on implementing a CRC (cyclic-redundancy-check) function on a block of data as a fire-and-forget peripheral processor. The AVR core is not configurable, and the ISA is not extensible, but the FPGA enables designers to implement hardware acceleration as a coprocessor or peripheral to the AVR. Atmel's System Designer environment includes FPGA-development tools, AVR studio tools, and co-verification tools to allow concurrent hardware and software development and debugging.

MIPS Technologies licenses 32- and 64-bit processor architectures and cores. Its Pro Series processor core family features the CorExtend capability, which allows designers to add and integrate proprietary instructions and tightly couple hardware to the core. We focused on the MIPS32 24Kc Pro processor core, which includes the CorExtend capability. The session involved implementing and verifying a UDI (user-defined instruction) that performed computations on a block of data. To implement the UDI, designers build a CorExtend block in Verilog RTL and integrate it with the processor core. The MIPS software-tool kit includes the software-development environment and the MIPSsim software simulator. MIPSsim uses an API for creating customized CorExtend libraries, so you can add any UDIs to MIPSsim for functional and cycle-accurate simulations.

Tensilica's configurable, extensible, and synthesizable processor cores emphasize configuring predefined elements of the architecture and building new instructions and hardware-execution units. The Xtensa Processor Generator produces a software-development environment, including operating-system support, for each processor configuration. Its Xtensa V processor core was the object of interest. (The Xtensa LX core was not available in time to support the project.)

FOR MORE INFORMATION...

For more information on products such as those discussed in this article, contact any of the following manufacturers directly, and please let them know you read about their products in *EDN*.

AccelChip 1-408-943-0700 www.accelchip.com	**ARM** +44-01223-400400 www.arm.com	**Critical Blue** 1-408-467-5091 www.criticalblue.com	**Morpho Technologies** 1-949-475-0626 www.morphotech.com	**QuickLogic** 1-408-990-4000 www.quicklogic.com	**Texas Instruments** 1-800-336-5236 www.ti.com
Altera 1-408-544-7000 www.altera.com	**Atmel** 1-408-441-0311 www.atmel.com	**The MathWorks** 1-508-647-7000 www.mathworks.com	**Pentek** 1-201-818-5900 www.pentek.com	**Stretch** 1-650-864-2700 www.stretchinc.com	**Xilinx** 1-408-559-7778 www.xilinx.com
ARC 1-408-437-3400 www.arc.com	**Celoxica** +44-0-1235-863656 www.celoxica.com	**MIPS Technologies** 1-650-567-5000 www.mips.com	**Poseidon Design Systems** 1-770-937-0611 www.poseidon-systems.com	**Tensilica** 1-408-986-8000 www.tensilica.com	

The hands-on session at Tensilica focused on implementing computational instruction extensions, including one that used more data than the data-flow structure could support in a single cycle. To create instruction extensions, engineers use the TIE (Tensilica Instruction Extension) language and compiler. The TIE language is a hybrid of Verilog and C that describes custom instructions, including multicycle and pipelined. The TIE compiler generates the files to customize the software-tool chain, extend the instruction-set-simulator and C-modeling environment, and estimate the hardware resources for processor configurations and custom instructions.

At Xilinx, which provides programmable-logic devices, advanced ICs, IP, and software-design tools, we worked with the 32-bit MicroBlaze soft processor core loaded in Virtex and Spartan FPGA devices. Xilinx also offers the 8-bit PicoBlaze soft core and the PowerPC 405 hard core. The acceleration effort focused on implementing functions for MP3 as a custom logic block or peripheral. The MicroBlaze does not support custom instructions extensions, but it does support user-defined hardware acceleration as a peripheral or coprocessor through an FSL (Fast Simplex Link) interface or the CoreConnect On-chip Peripheral Bus. The FSL interface provides a low-latency dedicated interface between the MicroBlaze register file and the custom acceleration logic. The embedded development kit and the Xilinx Platform Studio provide an integrated environment for creating MicroBlaze and PowerPC designs.

OPTIONS

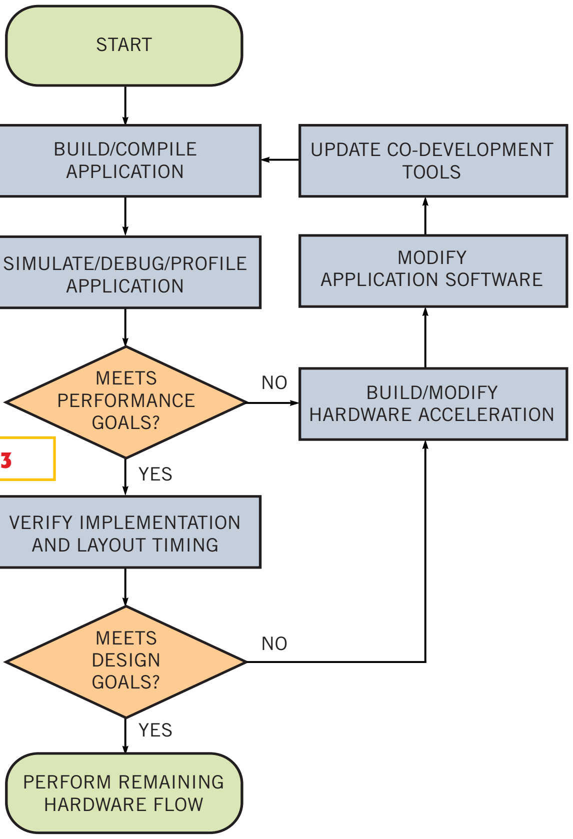
Sometimes, designers need a time-critical algorithm to execute more quickly than the software can handle. In such cases, they can try to accelerate the performance of the algorithm by restructuring it; throwing more data memory at the problem, such as by using a look-up table; or implementing the algorithm in hand-coded assembly. They can use faster processors, repartition the problem over multiple processing engines, or use application-optimized processor architectures, such as ASSPs (application-specific standard products) that integrate application-specific hardware accelerators. They may even consider creating their own custom instruction or hardware accelerator.

Throwing processor resources, such as memory, at an algorithm can increase the algorithm's speed but requires a trade-off in hardware resources. Working with on-chip rather than off-chip memory can also make a material performance difference. Implementing the algorithm as hand-coded assembly allows a designer to stay with a fixed-ISA processor but can tie him closely to it. Using a faster processor can incur adverse side effects, such as increased power consumption. Repartitioning the design over multiple processing engines, such as by using a microprocessor with a DSP, allows designers to use the device that best suits each set of functions but increases complexity. ASSPs implement optimized peripheral and hardware-acceleration features for a specific application. However, using them can make differentiating a product challenging, because they are narrowly targeted, and competitors can access them. Licensing IP cores to accelerate algorithms can make sense, but, because these cores are licensable, competitors can also access them. Therefore, they are not a differentiating feature.

Custom hardware acceleration is an increasingly viable method of implementing parallel processing. And, for some parallel operations, it can improve performance by an order of magnitude or two over a software implementation. You can implement custom hardware acceleration with a discrete FPGA interfaced to a standard processor device. Several processor providers, including Altera, Atmel, QuickLogic, Stretch, and Xilinx, offer a processor integrated with an FPGA on a single device. A growing number of software providers, such as AccelChip, Altera, Celoxica, Critical Blue, Poseidon Design Systems, Stretch, and Xilinx, are offering software tools to make hardware acceleration approachable for designers lacking extensive hardware-design training. Incorporating the acceleration into the application software often consists of substituting the source code with compiler intrinsics or inline assembly.

Designers can implement custom hardware acceleration as a custom instruction that is tightly coupled with the processor architecture. Custom instructions are effectively extensions of the processor's ALU (arithmetic logic unit, **Figure 1**). The custom instruction logic couples to the processor clock rate and must provide control signals to stall the pipeline or implement a custom pipeline when the critical path of the custom logic is too long. Designers may need to multicycle or pipeline a custom instruction if there is a data dependency. They may be unable to efficiently implement

Figure 3


START
BUILD/COMPILE APPLICATION
UPDATE CO-DEVELOPMENT TOOLS
SIMULATE/DEBUG/PROFILE APPLICATION
MODIFY APPLICATION SOFTWARE
MEETS PERFORMANCE GOALS?
NO
BUILD/MODIFY HARDWARE ACCELERATION
YES
VERIFY IMPLEMENTATION AND LAYOUT TIMING
MEETS DESIGN GOALS?
NO
YES
PERFORM REMAINING HARDWARE FLOW

A generalized design flow for accelerating software highlights the performance-simulation and implementation-verification loops when building custom hardware acceleration.

a custom instruction for data-intensive algorithms if they need to access data beyond the capacity of the ALU datapath (typically, two operands and one result), unless the processor architecture supports local memories, registers, and data-access mechanisms,

Another approach to implementing hardware acceleration is as a custom logic block that loosely couples with the processor as a coprocessor or peripheral via a data interface, such as a memory controller (**Figure 2**). The coprocessor performs its logic outside of the processor, so critical path-timing issues need not directly affect the clock-to-clock operation of the processor core. However, they can affect your application's overall operation. A coprocessor may also be able to access other peripherals or memory in the system without processor intervention.

THE PROCESS

Despite the fact that we operated in what could have been a highly controlled, lablike demonstration environment, each session offered some level of troubleshooting fun. During one session, the workstation died, and none too gracefully, either. However, the malfunction did explain some of the intermittent behaviors we were experiencing. Another session included a mismatch between drivers and tool versions, but the blame here lay with porting design code to a new version of the tools and a different target.

DEVELOPMENT TOOLS ARE EVOLVING TO MAKE INTERCHANGING HARDWARE AND SOFTWARE MODULES AVAILABLE TO DESIGNERS WITHOUT A HARDWARE BACKGROUND.

The most common troubleshooting problems arose from manually entering changes into code or configuration files. Because these sessions dealt with "simple" examples, we did not enforce strict version control. However, this approach turned out to be a mistake. Building, executing, and profiling each configuration means using indirect support files, and it is easy to forget a single change here or there. Script files ultimately made the difference in keeping things straight.

The time constraints of examining different architectures and tool sets didn't allow the project to explore an entire EDA-tool chain. For example, we didn't have time to wait for the physical-synthesis tool to complete its operations, so we effectively talked through that portion of the process. The generalized hardware-acceleration design flow appropriate for this project focuses on the iterative loop for identifying, creating, and integrating the hardware acceleration into the cosimulation environment (**Figure 3**).

Each session began with implementing an algorithm solely in software. Simulation tools are essential components of any co-development-tool suite; by using the simulation and profiling tools, we could identify software bottlenecks. Good candidates for hardware acceleration include operations that allow the merging of multiple sequential opera-

tions to produce a single set of outputs from a single set of inputs and operations that let you execute parallel computations on a set of independent inputs to produce independent outputs.

After identifying what code to accelerate and deciding whether to implement the acceleration as a custom instruction or coprocessor, designers need to create the hardware design. Some tools, which Part Two of this project will explore, assist with or automate the creation of the hardware acceleration straight from the source code. If designers manually implement acceleration blocks, they will create VHDL or Verilog files. However, if they are using Tensilica's tool suite, they will create the acceleration instructions/blocks via TIE modules.

To use the new (or modified) acceleration logic, designers need to instantiate the acceleration logic in the cosimulation tools and modify the algorithm software. Using a custom instruction can involve inserting or substituting a compiler-intrinsic or inline assembly statement in the original code. To access a custom coprocessor or peripheral, they may need to add code to use a driver or API. After rebuilding the system, they cosimulate the system and profile the performance and behavior. They repeat this process, possibly improving on previously implemented acceleration logic, until the system design can meet the performance goals. The next steps in the process are validating the functioning of the acceleration-logic RTL with a verification-test program, followed by gate-level timing verification. If the testing shows that the design meets all of the functional and design requirements, the designer can proceed with the rest of the hardware flow.

You can reach Technical Editor Robert Cravotta at 1-661-296-5096, fax 1-661-296-1087, e-mail rcravotta@edn.com.

Development tools are evolving to make interchanging hardware and software modules available to designers without a hardware background. Balance decisions to use a better performing hardware implementation over a software implementation with the application needs. Designers may not want to incur the recurring hardware costs if simplifying the software algorithms is sufficient. For example, if they have unused memory in their systems, building a look-up table may prevent additional costs for resources to implement a hardware accelerator. On the other hand, they may be able to reduce the total system cost with hardware acceleration, and implementing a runtime-reconfigurable system may yield even larger cost benefits (see **sidebar** "Reconfigurable acceleration").□

ACKNOWLEDGMENT
Special thanks to the people at Altera, ARC, Atmel, MIPS, Tensilica, and Xilinx for coordinating their teams and hosting a hands-on session with us at each of their sites.

TALK TO US
Post comments via TalkBack at the online version of this article at www.edn.com.