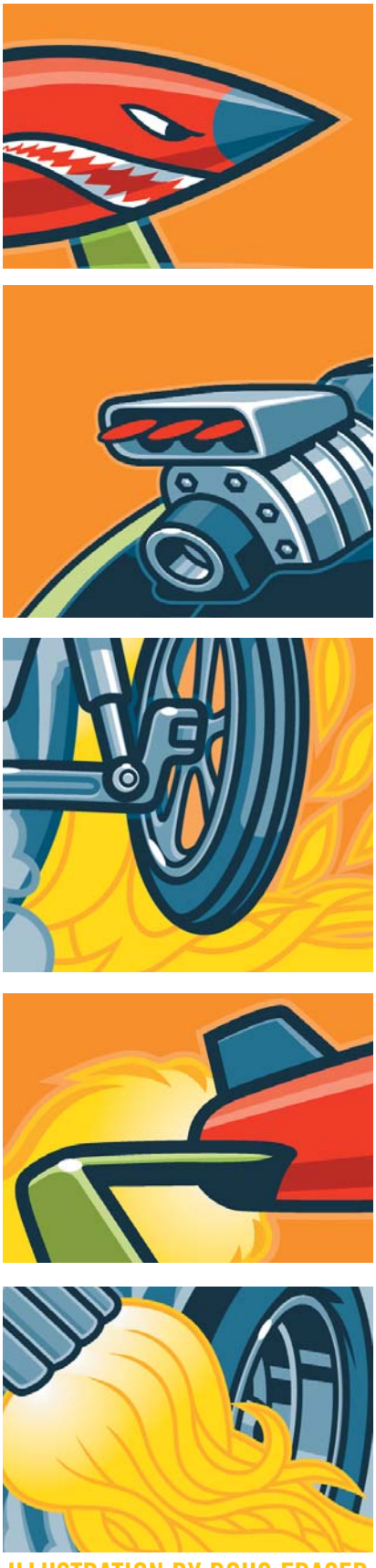

ILLUSTRATION BY DOUG FRASER

PART 2

Automate your ACCELERATION

THE SECOND INSTALLMENT IN A TWO-PART SERIES EXPLORES HOW AUTOMATION ENABLES DESIGNERS WITHOUT HARDWARE BACKGROUNDS TO CREATE CUSTOM HARDWARE ACCELERATION.

A GROWING NUMBER OF PRODUCTS are automating the translation of algorithm models and software code into hardware, enabling designers without detailed hardware backgrounds to implement custom hardware acceleration. These tools do not change the design or tool flow but rather automate some of the manual steps in the process (**Figure 1**). Their goal is to reduce the time it takes to implement a design over alternative methods, allow designers to explore more design configurations, enable entire design teams to work from a single source during a project's duration, use automated correct-by-construction methods to improve process reliability, and ensure that the verification process ties back to the original specification.

As Part One of this two-part series explained, the options for creating custom hardware as instruction or coprocessors to accelerate the performance of software are evolving (**Reference 1**). However, the process that Part One described assumed that a hardware engineer would manually perform the translation from software to hardware. Although a new crop of tools can automate that process, most of the tools that generate RTL (register-transfer language) from a software description require designers to understand at least some hardware considerations (see **sidebar** "Porting software to hardware").

Creating hardware from an algorithmic or software specification is not without challenges. The skills a designer needs to explore and implement an algorithm as software differ from those necessary to implement that algorithm in an FPGA or ASIC. Software designers rarely need to consider system-clock skew, bus capacitance and loading, analog-signal processing and characteristics, temperature, power consumption, and electromagnetic interference, which all affect how a hardware designer would optimize the design. Software employs a sequential computational model, and hardware generally deals with state machines and combinatorial logic that a designer can implement with high levels of parallelism.

Traditional EDA tools approach how to abstract hardware at a higher level to improve designer productivity. Tools that automatically generate RTL from software code have a different focus. According to David Stewart, chief executive officer of CriticalBlue, "The challenge is how to abstract software concepts, so that the tool can automatically implement them as hardware." These tools target designers without a background in hardware, rather than hardware designers proficient with Verilog or VHDL, and they allow a software-development methodology to persist throughout the design process. They enable the algorithm model or software code to remain a golden source throughout the project, so that changes made to it quickly, reliably, and automatically propagate to the downstream activities.

It is easier for a designer to see the big picture working at the algorithmic level than at the hardware level. And because algorithmic and software simulators can characterize the relevant system behaviors over a significantly wider time frame than RTL-level simulators, designers can explore and profile more use scenarios. Identifying high-level behavioral optimizations can yield dramatically better performance improvements in the total system performance than low-level hardware optimizations. Automating the translation of software to hardware also empowers designers to explore more alternative silicon implementations, because the tools simplify the effort and reduce the time it takes to evaluate the effects of a design change.

These tools usually support analyzing and performing trade-offs between processing speed and area through reporting, scheduling, and explicit mapping of resources. In addition to generating the RTL code, they save designers time during the verification process, because they automatically produce a testbench that uses the stimulus from the algorithm-exploration tools. Few of these tools are stand-alone; when targeting specific FPGA devices or process technologies, they interface with or integrate directly into the silicon-vendor tools.

Just because some tools can automatically generate RTL code from software does not mean it makes sense to use them for all code. In general, a hardware implementation cannot perform a complex sequential operation at a better cost and performance mark than software. Algorithms or functions that are good candidates for translating into hardware are those that perform parallel operations and can benefit from running multiple instantiations, those that can take advantage of nonstandard data-bit widths, those that can be pipelined, and those that can perform data processing with an execution time that is disproportionately larger than the data-transfer time.

AT A GLANCE

- Automation tools improve designer productivity and introduce more reliability into the design cycle.
- Standard programming languages do not capture concepts such as parallelism, timing, pipelining, and synchronization.
- Designers still need some understanding of hardware to use automation tools.
- Automated software-based synthesis tools offset the increasing complexity of differentiated embedded designs.

SOFTWARE TO RTL

AccelChip offers algorithmic synthesis and verification tools and services that support top-down DSP design for FPGAs, structured ASICs, and ASICs. This year, the company introduced the AccelChip DSP Synthesis tool, a second-generation replacement for AccelFPGA that can automatically generate synthesizable RTL models directly from the MathWorks' Matlab tool for DSP-algorithm development, data visualization, and data analysis. It also introduced AccelWare, a library of parameterized DSP IP (intellectual-property) building blocks for common Matlab toolbox functions. AccelWare blocks use the same parameters as Matlab functions and Simulink blocks, and they allow designers to specify implementation parameters, such as timing, to meet design goals.

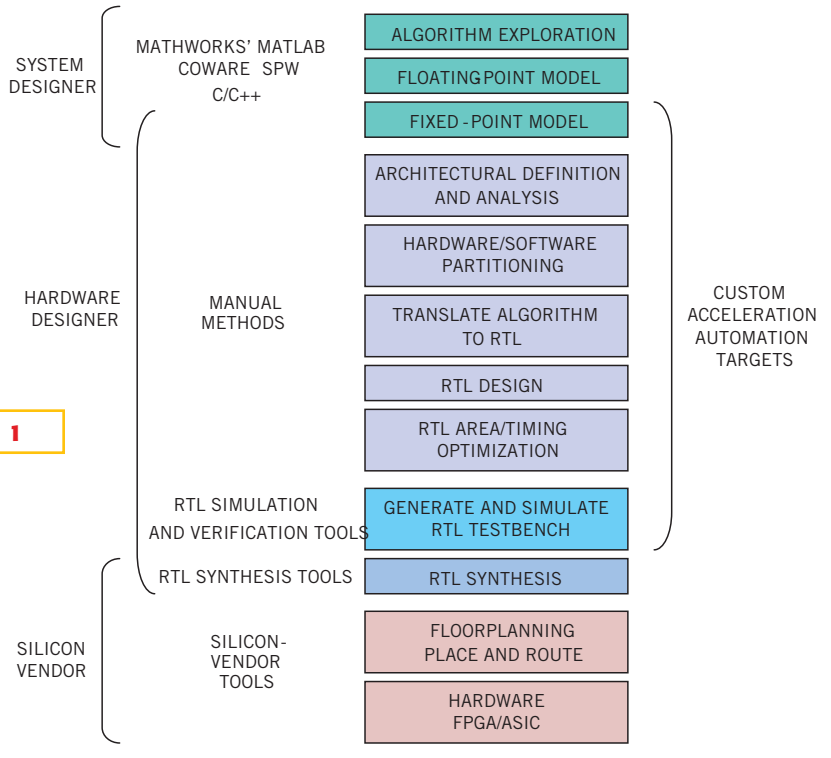

Figure 1

Certain steps in the design-tool flow for accelerating software with hardware are good candidates for automation. Some designer activities overlap in the tool flow, and unique tools support each portion of the flow.

The AccelChip DSP Synthesis tool generates RTL for the complete algorithm, including AccelWare blocks.

The AccelChip tools enable Matlab to remain the golden source throughout the design process by supporting system-level verification via the MathWorks' Simulink interface and by integrating into Synopsys' ASIC and Xilinx's System Generator tool flow. The AutoQuantizer tool automates the conversion from floating-point models to a fixed-point design by using the original floating-point source and the design model to determine the dynamic range of each variable. The AccelChip tools produce the files required to generate Simulink S-functions from bit-true, cycle-accurate, fixed-point Matlab models. The AccelChip DSP Synthesis tool automatically creates a simulation testbench that designers can use to verify the postlayout and fixed-point design.

Synplicity's synthesis, verification, and physical-implementation software tools target FPGA, structured- and platform-ASIC, and cell-based- and COT (customer-owned tooling)-ASIC implementations. The company this year released Synplify DSP, which generates synthesis-ready RTL code from Matlab and Simulink models. The Synplify DSP tool uses algorithms such as system-level retiming to optimize a Simulink model at the system level before RTL generation, thus improving DSP-implementation performance. A designer can perform what-if analyses on thread capacity by automatically generating a multichannel system from a single-channel specification. The tool can share on-device resources, such as multipliers, within a specified performance budget, and it supports trade-off analyses for area and performance before implementation to reduce design iterations.

Celoxica's DK design suite of tools helps designers explore hardware and software partitioning, automatically generate hardware representations from C descriptions, and verify system functions. The technology integrates into Altera's design flow with the DK Accelerator design tool for Altera SOPC Builder. The DK Design Suite for Xilinx EDK comes bundled with Xilinx's tool flow and supports cosimulation of hardware and software across multiple high-level languages, including C, C++, SpecC, SystemC, and Handel-C. The DK3 release supports tighter cosimulation with Matlab and Simulink.

Translating a C-based algorithm with the DK tool set involves porting the C source to Handel-C, a dialect of C, and the use of #pragmas for hardware and software partitioning. Jeff Jussel, Celoxica's vice president of marketing, points out, "Generating RTL from Handel-C, rather than from what-if analysis models, enables the designer to capture and express enough details required for efficiently implementing complex algorithms in hardware." This year, Celoxica extended the DK design-suite capabilities beyond performing only Handel-C synthesis by introducing the Agility compiler, which supports SystemC synthesis with a flow nearly identical to that of the Handel-C tools.

SystemC is a modeling language based on C++ to facilitate system-level design. The SystemC library of classes layer on and extend standard C++ to provide for specifying concurrent behavior, time-sequenced operations, data types for describing hardware, structure hierarchy,

PORTING SOFTWARE TO HARDWARE

Porting software to a hardware implementation is analogous to porting software between processor architectures—an exercise in details at best and a trouble-shooting nightmare at worst. The C software-programming language is good for capturing the behavioral and functional specification for a system, but it fails to capture hardware concepts, such as parallel constructs, buses, and timing. The constructs that a software designer might use, such as dynamic memory allocation or using a pointer to pass data between modules, also fail to translate cleanly to hardware. Some automated synthesis from software tools support a derivative of C code, such as ANSI C, C++, SystemC, Handel-C, or proprietary extensions of ANSI C.

The motivation for using these C derivatives is to eliminate the need for a designer to manually rewrite all of the C-based system behavioral descriptions into RTL code. These C derivatives provide mechanisms for a designer to capture hardware concepts in the source code, but they do so without breaking the C specification. It is critical that, even after many hardware-specific refinements, the C code remains an executable specification that a designer can compile and execute at any time to verify the system's function. The extensions include directives for specifying parallelism, timing constraints, communication and synchronization between modules, and bit-wise data sizes; mapping operators to hardware; and mapping aggregate data structures, such as arrays, and their operations.

Each C derivative addresses these directives slightly differently. For example, SystemC relies on explicit clocks to specify timing, and Handel-C uses rules to imply the clock; for example, performing an assignment requires a single clock cycle. For specifying hardware data types, Handel-C introduces hardware data types; SystemC uses C++'s type system. Handel-C now preserves the C model for pointers, and SystemC uses the rendezvous primitive.

In addition to extending the C standard, these tools may impose restrictions on the code that are the source of automatically generated RTL code. These restrictions, such as a ban on floating-point arithmetic, are necessary because the tools are more complex and expensive than analogous approaches, such as using integer or fixed-point arithmetic. It is often necessary to rewrite recursive functions to determine the maximum depth of the recursion or to rewrite the function as iterative code.

Although these C derivatives preserve the executable integrity of the C specification for functional and behavioral analysis, they are not directly portable between each other for generating RTL code. A designer's choice of C derivative depends on the output quality of the tools and the available supported targets. With enough tool options available now and probably more to come in the near future, it is unclear whether any of the C derivatives will emerge as the clear choice.

and simulation support. SystemC tool support consisted of only modeling and simulation until Celoxica and Forte Design Systems this year introduced SystemC synthesis tools.

Forte Design Systems, which develops behavioral-synthesis technology, enables designers to use a higher level of design abstraction by offering an automated path from high-level algorithms to RTL, including synthesis, verification, and cosimulation. Its Cynthesizer automatically builds fully timed RTL implementations from untimed SystemC models and a set of directives or constraints that the designer provides, such as clock speed, latency, pipelining, and loop unrolling. By applying different sets of directives to the same design source, a designer can explore and analyze the trade-offs between processing performance and die area. Cynthesizer's automated behavioral-synthesis functions include operation scheduling, cycle timing, state-machine implementation, control and datapath design, resource allocation, and RTL generation.

EDA company Mentor Graphics puts forth a broad range of software- and hardware-design tools, including tools focusing on C-based design. Its Catapult C Synthesis algorithmic-synthesis environment automatically generates RTL from untimed C++ and is compatible with SystemC models. Catapult C Synthesis does not require a designer to embed interface timing in the C++ source. The tool generates implementations based on constraints the designer provides, allowing the exploration of a range of hardware interfaces, such as streaming, single- or dual-port memory, handshaking, and FIFO. The architectural-constraints tool presents a graphical view of all ports, arrays, and loops in the design and allows the designer to explicitly apply loop unrolling or merging; pipelining; RAM, ROM, or FIFO array mapping; resource allocation; memory-bit-width resizing; and merging memory resources.

The Catapult C Synthesis tool employs symbolic analysis and optimization techniques, such as sequential constant propagation, variable lifetime, loop-boundary and -array index analysis, and memory-bandwidth optimization, so it can automatically reduce operator bit widths and share components and resources. To assist designers analyzing implementations for the best performance and area trade-offs, the tool graphically displays the results in x-y plots, bar charts, tables, or schematic views. The hierarchical Gantt chart is an algorithm- and microarchitecture-analysis tool that acts as a schematic viewer for the algorithm; it enables the designer to examine data-flow, component-usage, and loop-execution profiles to identify in the C++ source memory-bandwidth limitations, loop dependencies that prevent parallelism, and data dependencies that prevent optimal scheduling.

C SYNTHESIS EMPLOYS SYMBOLIC ANALYSIS AND OPTIMIZATION TECHNIQUES, SUCH AS SEQUENTIAL CONSTANT PROPAGATION, VARIABLE LIFETIME, LOOP-BOUNDARY AND -ARRAY INDEX ANALYSIS, AND MEMORY-BANDWIDTH OPTIMIZATION.

Impulse Accelerated Technologies' CoDeveloper system generates RTL from Impulse C source, which is standard ANSI C extended with a number of C-compatible library functions to express parallel operations. CoDeveloper Universal allows designers to develop mixed FPGA and processor platforms but does not require that the design include an embedded processor. CoDeveloper for Altera or Xilinx performs C-to-RTL compilation, targeting the FPGAs, and interfaces with the appropriate tool set from each company.

A designer needs to port computationally intensive software processes to Impulse C to be able to automatically create a parallel hardware implementation of the software block. The Design Assistant supports importing individual C processes, accepts input/output requirements, and automatically generates template source files. The Impulse C programming model supports streams, signals, and shared memory models for inter-process communication. CoDeveloper generates a consistent interface for interprocess communication so that designers can develop a common library of interfaces to Impulse C to meet project requirements.

Synfora tools and IP, which the company introduced this year, are the result of developing Hewlett-Packard's patents for PICO (program in, chip out) Algorithm-to-Tapeout synthesis. PICO Express combines configurable IP and exploration and configuration tools that

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

Altera
1-408-544-7000
www.altera.com

Celoxica
+44-0-1235-863656
www.celoxica.com

CoWare
1-408-436-4720
www.coware.com

Critical Blue
1-408-467-5091
www.criticalblue.com

Forte Design Systems
1-408-487-9340
www.forteds.com

Impulse Accelerated Technologies
1-425-576-4066
www.impulsec.com

MathWorks
1-508-647-7000
www.mathworks.com

Mentor Graphics
1-503-685-8000
www.mentor.com

Open SystemC Initiative
www.systemc.org

Poseidon Design Systems
1-770-937-0611
www.poseidon-systems.com

Stretch
1-650-864-2700
www.stretchinc.com

Synfora
1-650-314-0500
www.synfora.com

Synopsis
1-650-584-5000
www.synopsis.com

Synplicity
1-408-215-6000
www.synplicity.com

Tensilica
1-408-986-8000
www.tensilica.com

Xilinx
1-408-559-7778
www.xilinx.com

explore and build RTL directly from algorithmic C descriptions. The IP comprises a configurable pipeline of processor arrays for computationally-intensive operations and a configurable VLIW (very-long-instruction-word) architecture for control-intensive operations or for use as a programmable processor. The exploration tool creates a Pareto optimal set of implementations from C that meets different performance and area points and graphically displays the results. The configuration tool maps algorithm descriptions, expressed as sequences of nested loops, to the pipeline of the processor array architecture. PICO Express supports multiple loops with streaming data and creates a single, rate-matched RTL block.

PICO acts on C algorithms and constraints on area performance and cycle time that the designer specifies and then creates a set of alternative implementations with different degrees of parallelism to make the trade-off between performance and cost. The tool determines and keeps only optimal Pareto implementations. At the end of the exploration, the designer can examine the implementation alternatives that represent the best area implementation at each performance point.

Poseidon Design Systems recently announced its Triton Tuner and Builder tools for automatically generating RTL from ANSI C. The company provides electronic-system-level tools for embedded-system design, including hardware/software analysis, cosimulation, and hardware synthesis for designing systems on chips and on FPGAs. The Triton Tuner cosimulation tool uses a SystemC simulation environment based on transaction-level models of the system components to collect performance measurements; it also profiles the software at the instruction, function, and inner-loop levels. The tool models the entire system-memory hierarchy, including cache memories, write buffers, RAMs, and flash memories, allowing designers to configure memory subsystems to improve performance.

After the designer partitions the system between hardware and software, the Triton Builder tool generates the drivers, accelerator RTL, system bring-up support, and test conditions; it also automatically modifies the source code to use the accelerator block. When the Triton Builder tool modifies the source code, it tacks a driver onto the affected code block to configure, communicate with, and invoke the acceleration block. To support the accelerated logic block, the accelerator-block architecture can include local memories and dedicated data-transfer logic, an adaptable bus interface, a programmable high-speed controller, a single-step controller, and a system bring-up to peek at and poke memory.

LISATEK AUTOMATES RTL GENERATION FROM AN ARCHITECTURE-SPECIFICATION LANGUAGE.

DIFFERENT APPROACHES

Tensilica licenses the configurable, extensible, and synthesizable Xtensa V and Xtensa LX processor cores; however, its Xpres compiler takes a different approach to translating ANSI C to hardware: It automatically configures an Xtensa LX microprocessor core to more quickly execute the C code. The Xpres compiler works with the Xtensa C/C++ compiler to analyze software-performance-critical regions and then explores millions of possible processor configurations using a variety of acceleration techniques. These configurations represent a range of customized Xtensa LX processors that trade off application performance and area; they are made up of TIE (Tensilica Instruction Extension) code that the designer inputs into the Xtensa LX Processor Generator.

CoWare's LisaTek tool set automates RTL generation from an architecture-specification language. C/C++-based Lisa 2.0 specifies instruction-set architectures. The LisaTek tools can automatically generate RTL for VLIW, RISC, DSP, and SIMD architectures, as well create the custom software-development tools a designer needs to program the custom processing engine. The LisaTek profiling tools enable designers to optimize instruction sets, processor microarchitectures, and memory systems, including caches.

Stretch also differs from the aforementioned companies in that it provides both the software-development tools and the silicon target. The Stretch S5000 processors integrate a Tensilica Xtensa processor core with an on-chip ISEF (instruction-set extension fabric). Stretch based the software-configurable ISEF datapath on proprietary programmable logic that designers can use to extend the processor instruction set. The Stretch integrated development environment supports C/C++ functional development, performance profiling and tuning, system verification, and in-circuit debugging. The integrated Stretch C compiler compiles both extension instructions and application code.

Designers define new instructions by adding Stretch C extensions and compiler intrinsics in the code. Stretch C extensions include new data types and explicit data-movement and -manipulation instructions to accommodate the S5000's wide registers to the ISEF. A few restrictions are necessary for the compiler to accelerate a block of software. For example, the extension instructions cannot use floating-point data types, the compiler must be able to unroll all loops, and the compiler does not support accelerated divide and modulus operators unless both operands are known at compilation time.

The ISEF operates at 100 MHz, and the processor core operates at 300 MHz. These performance values not only lower total power consumption, but also allow the software time to read and write to the ISEF's wide register file without stalling the ISEF. The ISEF includes computation (arithmetic and multiplier units), routing, pipeline, and state-register resources, and it can support dynamic but not cycle-to-cycle reconfiguration. The designer can see how the total resource usage changes among configuration modifications, encouraging a focus on the higher level behavioral model and C code rather than a focus on the acceleration implementation, which would defeat the tool's productivity value as a higher level abstraction for software acceleration.

EDA-tool company CriticalBlue uniquely addresses automating the soft-

ware-to-hardware process from the other end of the software implementation. Its Cascade tool extracts parallelism directly from compiled object code and translates it into a custom microprocessor. This method is programming-language-neutral, because the software code is already at the target processor level. It requires no new language expertise, no new software-development tools, and no manual optimization of software code, and it supports the designer's established verification and debugging tools.

Rather than provide leading-edge performance, this approach can augment designs in which cost, power consumption, or schedule constraints preclude upgrading the main processor or adding a full instruction-set processor by offloading and accelerating the processing in an application-specific coprocessor. The tool automatically generates the coprocessor RTL, microcode, and testbench from the executable software code and then automatically modifies the executable code to operate with the coprocessor. Cascade allows designers to manually intervene to optimize the code and coprocessor and to deploy custom hardware units.

Over the previous year, many companies have turned their belief that automated RTL generation from software is a viable and value-added capability into useful new tools and products. In fact, these tools implement their automation technology in a variety of ways, ranging from using Matlab models; to C and C derivatives, such as ANSI C, C++, SystemC, Handel-C, and proprietary C extensions; and to object code. They also target their automated acceleration to a variety of implementations, including FPGA- and ASIC-tool flows, processor arrays, custom processors, and proprietary instruction-acceleration fabrics.

It is uncertain whether other approaches are currently in stealth-development mode and will be announced in the near future, but there will likely be more development activity and eventually consolidation for these types of tools. As with the evolution of software compilers, there is still room for improvement, including good resource estimation to support high-level exploration. Although these types of tools will likely become staples in many design-tool flows, they might never replace the need for handcrafting the most leading-edge, computationally intensive functions. After all, software compilers have not eliminated the need for handcrafted coding for the most leading-edge functions, either.□

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

Talk to us

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