Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

MIPS EMBRACES CONFIGURABLE TECHNOLOGY

Pro Series Processors With CorExtend Compete With ARC and Tensilica

By Tom R. Halfhill {3/3/03-01}

Ever since ENIAC, Colossus, and even the Difference Engine, the trend in computer evolution has been toward higher levels of abstraction above the hardware. It is most visible on the software side: wire patchboards and toggle switches soon led to machine language and assemblers, then to compilers, interpreters, emulators, and virtual machines. Today, it's possible (and common) for skilled programmers to be blissfully ignorant about the underlying hardware.

Abstraction is rising on the hardware side as well. Engineers once designed logic by manually drawing wires and gates. In the 1970s came the first hardware description languages (HDL), followed by synthesis compilers. Over time, HDLs have evolved to the point where engineers can design processors in much the same way programmers design software: by creating a functional description independent of a hardware implementation. Indeed, the latest trend is to adapt software programming languages for hardware design. Eventually, this convergence will topple the wall that separates hardware and software engineering. In the future, someone will write code to solve a problem, and the end product will be gates, bits, or the optimum combination of both—perhaps generated by a single compiler.

HDLs accelerate the design process and make today's complex microprocessor designs more manageable. But a side-effect of hardware abstraction is that it also makes the softened hardware more malleable, like software. Soon after companies began creating synthesizable models of microprocessors, they realized that the flexibility software programmers had enjoyed since the 1950s was now within the reach of hardware designers.

One of the first companies to build a business around this discovery was ARC International (formerly ARC Cores, an offshoot of U.K.-based Argonaut Software). ARC was an early supplier of synthesizable intellectual property (IP) to embedded-chip developers. According to company lore, an ARC engineer grew weary of designing minor variations of the same basic microprocessor for different customers. In 1993, he created a general-purpose 32-bit microprocessor core designed to be optimized for specific applications by the customers themselves. This innovation freed up his weekends and in 1996 led to the first openly licensed user-configurable microprocessor.

In 1997, some former MIPS engineers helped found Tensilica, to compete with ARC. Tensilica created a 32-bit microprocessor core with a graphical configuration interface and integrated tool chain—all designed from the start to be user customizable. The company's first Xtensa processor appeared in 1999.

Both ARC and Tensilica have established the validity of the configurable-processor concept, which is actually a principle long familiar to software programmers. By identifying and optimizing the most critical parts of an application, it's possible to achieve huge gains in performance. Programmers do it with careful coding or in-line assembly language. Now, chip designers can do it by optimizing a configurable microprocessor core with custom instructions and other enhancements. It really works, and ARC and Tensilica have the certified EEMBC scores to prove it.

MIPS Technologies is the latest company to endorse the concept. At Embedded Processor Forum 2002, MIPS

introduced the M4K Pro synthesizable processor core, which implements a revision of the MIPS32 instruction-set architecture (ISA). Along with that revision, MIPS opened the door for customers to add application-specific instructions to the M4K. More recently, MIPS announced that all soft processors in the Pro Series are user extendable, thanks to a technology MIPS refers to as CorExtend. Initial Pro Series cores, in addition to the M4K, are the 4KSd, 4KEp, 4KEm, and 4KEc.

It will be no surprise if other companies jump on the bandwagon. A soft microprocessor core should be more than just a synthesizable version of a hard core; intuitively, something soft should be more flexible than something hard. Eventually, a licensable soft-IP microprocessor core that is not customizable will seem as limited as a text editor without a macro language or an operating system without control panels.

The ability to customize a processor is particularly valuable in the embedded-systems market, where the diversity of applications argues against a one-size-fits-all solution—or even a fairly large number of off-the-shelf solutions. Using customizable processor cores, chip designers can create a virtually unlimited number of solutions. (See *MPR 11/12/01-01*, "Configurable vs. Fixed Instruction Sets.")

Small Step, Big Leap

The first obvious question about MIPS's new CorExtend technology is whether it matches the more-mature configurable-processor technology from ARC and Tensilica. The short answer is no. At this time, MIPS Pro Series cores have fewer configurable options than do the ARCtangent-A5 or Xtensa V cores. Furthermore, ARC and Tensilica offer better tool-chain integration. Table 1 summarizes the features of these configurable processor cores.

Although MIPS Pro Series cores are less broadly configurable than the ARCtangent and Xtensa cores, they do have the most important feature: an extendable instruction set. This allows designers to leverage the familiar 80/20 rule, which holds that 20% of a program usually does 80% of the work. Software programmers use profiling tools to identify those "hot spots," then optimize the critical algorithms or inner loops with tighter code or assembly language. Hardware

	MIPS M4K Pro	MIPS 4KEp Pro	MIPS 4KEm Pro	MIPS 4KEc Pro	MIPS 4KSd Pro	ARC ARCtangent-A5	Tensilica Xtensa V
Architecture (ISA)	MIPS32 32-bit RISC	MIPS32 32-bit RISC	MIPS32 32-bit RISC	MIPS32 32-bit RISC	MIPS32 32-bit RISC	ARCompact 32-bit RISC	Xtensa V 32-bit RISC
Target Applications	Gen-purpose embedded	Gen-purpose embedded	Gen-purpose embedded	Gen-purpose embedded	Smartcards, secure data	Gen-purpose embedded	Gen-purpose embedded
Synthesizable	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Core HDL	Verilog	Verilog	Verilog	Verilog	Verilog	Verilog, VHDL	Verilog, VHDL
User-Extension HDL	Verilog	Verilog	Verilog	Verilog	Verilog	Verilog, VHDL	TIE
Configurability	Medium	Medium	Medium	Medium	Medium	High	High
Custom Instr Slots	512	512	512	512	512	256	32,768
Custom Core-Reg Slots	0	0	0	0	0	28*	32
Multicycle Custom Instr	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Multiple Reg Files	Optional	Optional	Optional	Optional	Optional	Optional	Optional
Custom Instr Formats	0–3 operands	0–3 operands	0–3 operands	0–3 operands	0–3 operands	0–3 operands	0–4 operands
Standard Instr Length	16/32-bit	16/32-bit	16/32-bit	16/32-bit	16/32-bit	16/32-bit	16/24-bit
Custom Instr Length	32-bit	32-bit	32-bit	32-bit	32-bit	16/32-bit	24-bit
Graphical Config Tool	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Predefined Extensions	No	No	No	No	No	Yes	Yes
DSP Extensions	No	No	No	No	No	Optional	Optional
Config I/O Buses	No	No	No	No	No	Yes	Yes
Config Interrupts	No	No	No	No	No	Yes	Yes
Config Caches	Cacheless	0–64K	0–64K	0–64K	0–64K	0–32K	0–32K
Config Scratchpad RAM	No	Yes	Yes	Yes	Yes	Yes	Yes
Config Condition Flags	No	No	No	No	No	Yes	No
Config Endian Order	Yes	Yes	Yes	Yes	Yes	No	Yes
Config Peripheral IP	No	No	No	No	No	Yes	No
Fast Int Mult/Div Unit	Optional	No	Yes	Yes	Yes	Optional	Optional
FPU	No	No	No	No	No	No	Optional
MMU	No	No	No	Yes	Yes	No	Optional
Config Dev Tools	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Tool-Chain Automation	None	None	None	None	None	Low	High
Availability	Now	Now	Now	Now	Now	Now	Now

Table 1. The first configurable processors from MIPS don't match the competing technology from ARC and Tensilica, but they have the most vital capability: user-defined instructions. *ARCtangent-A5 has 32 extension-register slots, but 4 are reserved.

designers can do the same thing by creating custom instructions that execute the critical functions in logic. Doing this can often boost performance by an order of magnitude or more. In other words, a little configurability goes a long way.

The big selling point for MIPS is that customers can now license a configurable microprocessor core without having to adopt an alien CPU architecture. In the market for 32-bit embedded-processor cores, only ARM is more pervasive than MIPS, and ARM has not yet embraced configurable technology. Although ARC and Tensilica have very respectable CPU architectures from a technical standpoint, they are less "standard" than is MIPS, one of the seminal RISC architectures developed in the 1980s. Hundreds of universities worldwide use the MIPS architecture as a teaching tool, and it's strongly supported by third-party development tools and peripheral IP. Chips based on MIPS cores are found in everything from smartcards and home videogame consoles to high-performance network routers. It's a relatively straightforward architecture, easy to learn and use. The Pro Series cores with CorExtend are backward compatible with this architecture.

Another selling point for MIPS is a broader range of synthesizable processors. ARC and Tensilica each have only one 32-bit processor core (not counting older cores that are still supported but are being phased out). Although not all MIPS's processors currently implement CorExtend, among the first are most of the 32-bit processors, including the SmartMIPS 4KSd for smartcards and secure-data applications.

ARC and Tensilica have a good counterargument: because their processor cores are more customizable, they need not duplicate MIPS's closely spaced product line of 32-bit cores. One processor is enough. If chip designers were sculptors, ARC and Tensilica would be selling clay, not marble.

In the future, MIPS could broaden its product line beyond the range of configurability offered by ARC and Tensilica by adding CorExtend to its 5K-series 64-bit cores, which would make them the only 64-bit processors on the market with extendable instruction sets. Neither ARC nor Tensilica has a 64-bit processor core.

However, the 32-bit market is definitely the sweet spot. Today's embedded applications are less likely to use 64-bit integers than other, less traditional data types—such as 24-bit audio streams, 56- or 128-bit encryption keys, and 40-byte packet headers. A 32-bit processor is sufficient for those tasks.

In addition, there are other paths to higher performance, such as multiprocessing. ARC has customers integrating as many as 64 ARCtangent cores on a single chip. Tensilica says 59% of its customers are designing multiprocessor chips, averaging 5.1 Xtensa cores per design. For high-performance SIMD processing, Tensilica offers its optional Vectra DSP extensions, which have 160-bit-wide registers and datapaths. ARC has optional DSP extensions, too. For these reasons, neither ARC nor Tensilica is likely to lose much business by ceding the speculative market for configurable 64-bit processors to MIPS.

CorExtend Is Simple, Yet Effective

It has always been possible for MIPS licensees to extend the MIPS instruction set, but doing so formerly required an architectural license instead of the more common (and less expensive) core license. In 1996, for example, LSI Logic developed the 16-bit MIPS16 instruction subset to improve code density in embedded applications. (See *MPR 10/28/96-10*, "LSI's TinyRisc Core Shrinks Code Size.") It's also been possible for MIPS licensees to add new instructions and local state registers by attaching function blocks to the coprocessor interface, a feature of some MIPS cores. However, even MIPS now acknowledges that the rarely used coprocessor interface is not the ideal solution. With CorExtend, customers don't need an architectural license to add a few application-specific instructions to a Pro Series core.

Instead, CorExtend allows any MIPS Pro Series licensee to readily add user-defined instructions that enjoy the same functional status as standard MIPS instructions. MIPS has set aside 16 major opcodes for this purpose. This opcode space is separate from the so-called Special2 opcode field, which years ago was reserved for architectural licensees to define new instructions. The six-bit Special2 field occupies bits 26–31 in the 32-bit MIPS instruction format, while the new user-defined instruction (UDI) field occupies bits 0–5.

Within the six-bit UDI opcode field, MIPS has set aside the first four bits for user-defined opcodes, which is enough for 16 custom instructions. Actually, it's possible to add many more custom instructions, because the CorExtend instruction format has 20 bits of user-definable encoding space in each 32-bit instruction word. By using subcoding techniques (which would require additional instruction decoding in the custom logic), designers could add as many as 16,777,216 opcodes!

We suspect most designers will be satisfied with the 16 major opcode slots. For additional instructions, MIPS suggests using a user-definable five-bit field immediately following the UDI opcode field; doing this yields a more reasonable, but still luxurious, 512 custom opcode slots. (Four bits in the UDI opcode field plus five bits in the undefined field equals nine bits, or 512 slots.) Figure 1 shows the CorExtend instruction format.

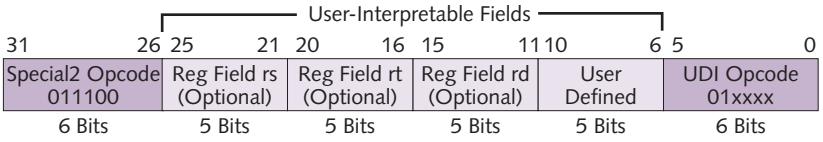

Figure 1. MIPS reserves all or part of the dark-shaded fields for opcodes. Designers can use bits 0–3 in the UDI Opcode field to create opcodes for up to 16 user-defined instructions. The remaining 20 bits in the light-shaded fields are available to designers. These bits can encode source and destination register addresses, immediate values, or subcodes for additional opcodes. For instructions that use registers, MIPS suggests the layout shown above.

User-defined instructions can be single-cycle or multicycle. Multicycle instructions can either stall the main integer pipeline (halting all other instructions until the multicycle instruction executes) or synchronize with the pipeline (allowing other instructions to proceed while the multicycle instruction executes). Pipeline synchronization is automatic, and the processor always checks the custom instruction's source and destination registers for pipeline dependencies. This is a big advantage over function blocks attached to the MIPS coprocessor interface, which require designers to implement their own pipeline synchronization and dependency checking.

By default, custom instructions have a three-operand format: two source registers and one destination register. (In MIPS nomenclature, these are known as *rs*, *rt*, and *rd*.) Therefore, custom instructions can be nondestructive: results need not overwrite operands in the source registers. The register specifiers are five-bit addresses, so instructions can fetch their source operands from any of the 32 general-purpose 32-bit registers in the standard MIPS architecture. Likewise, a custom instruction can return a 32-bit result to any of these registers. (Figure 1 shows how MIPS suggests defining the register fields within an instruction.) Custom instructions cannot fetch operands directly from memory or write results to memory; they are strictly register-to-register operations.

The three-operand format for custom instructions appears to be comparatively rigid, but, in practice, it's fairly flexible. If an instruction needs only one input operand (*rs*), it can simply ignore the second input operand (*rt*). Actually, designers can use the 20 bits of user-interpretable fields almost any way they want. For instance, the bits set aside for register specifiers could encode an immediate value.

Results are assumed to be 32 bits long, but there's wiggle room here, too. If a custom instruction returns a result smaller than 32 bits, the custom logic could pad the extra space with zeroes. If a custom instruction returns a result larger than 32 bits, a second custom instruction could fetch the extra bits from a user-defined local register in the custom logic and store them in a second architectural register. (The second custom instruction would be necessary because standard MIPS instructions can't access user-defined registers in custom logic.)

To add a user-defined instruction to a Pro Series processor, designers must write their custom logic in register-transfer-level (RTL) Verilog. This logic is responsible for decoding the instruction, handling the input operands, executing the operation, storing any intermediate results in local state registers, and storing the final result in a core register. The Verilog models for Pro Series cores have special CorExtend interfaces for attaching this logic. At build time, the synthesis compiler tightly couples the custom-logic block to the core's execution unit.

Standard MIPS compilers, simulators, and other development tools would normally be unable to recognize custom instructions: they would either be oblivious to the instructions or balk at what appeared to be undefined opcodes. Designers must therefore write macros and intrinsic functions that allow assemblers and C/C++ compilers to call the instructions, plus dynamically linked libraries (DLL) to describe the operation of the instructions to software simulators. MIPS provides templates to simplify these tasks. At present, two tool chains support CorExtend: Green Hills MULTI and MIPS's own MIPSsde 5.0 (a GNU-based tool suite). MULTI supports multicore debugging, and both packages work with the MIPSsim instruction-set simulator.

Original Assembly Code

```
#r2,r3  var1 and var2 (16-bit inputs)
#r16    var3 (32-bit accumulator)
#r4     a fixed value of 0x4000
mult:   lui r11, 0x4000
        mul r8, r9, r10
        bne r11, r8, sum
        sll r8, r8, 1
        lui r8, 0x7fff
        ori r8, r8, 0xffff
sum:    addu r13, r8, r14
        xor r12, r8, r14
        bltz r12, end
        addu r14, r0, r13
        xor r12, r8, r13
        bgez r12, end
        nop
        bltz r8, end
        lui r14, 0x8000
        lui r14, 0x7fff
        ori r14, r14, 0xffff
end:    sw r14, 0x8(r5)
```

C Code with Custom Instructions

```
short     x[512], y[512];
long      *px, *py;

for (px=x, py=y; px<px+512/2; px++, py++)
          {udi_madd(*px, *py);}
out1 = udi_add(acc_A, r0)+udi_add(acc_C, r0);
```

Figure 2. The inner loop of this algorithm can be replaced by a single custom instruction, `udi.madd`, which executes in one clock cycle. Note, however, that the following line of C code requires two additional cycles (two instances of the user-defined `udi.add` instruction) to fetch and concatenate the 40-bit result from a local accumulator in the custom function block. That's because CorExtend custom instructions can't return a result larger than 32 bits, and standard MIPS instructions can't access local registers in custom logic.

Potential Gains Are Huge

When Tensilica and ARC reported their first certified EEMBC benchmark scores in 2001 and 2002, respectively, they extinguished any remaining doubts about the potential of configurable processors. (See *MPR 2/18/03-06*, "Soft Cores Gain Ground," and *MPR 4/9/01-01*, "Stretching Silicon to the Max.") In most cases, their optimized scores exceed their out-of-the-box scores by an order of magnitude or more. At this writing, Xtensa V still holds the record for the highest certified EEMBC ConsumerMark score of any processor, configurable or not. (See *MPR 9/16/02-01*, "Tensilica Xtensa V Hits 350MHz.")

MIPS hasn't yet published certified EEMBC scores for the Pro Series cores, but there's no reason to doubt the results will be similarly impressive. ARC and Tensilica achieved their high scores by optimizing the benchmark code with a few custom instructions, and MIPS can now do the same. Until then,

MIPS offers a theoretical example of what designers can do with CorExtend. The example demonstrates how a single custom instruction could accelerate a common algorithm in a voice-over-Internet-Protocol (VoIP) application.

The new instruction, `udi.madd` (user-defined instruction/multiply-add), performs a 16 × 16-bit multiply-accumulate (MAC) operation. It stores the extended-precision 40-bit result in a user-defined local accumulator, with saturation. A barrel shifter (also part of the custom logic) scales the results. MIPS estimates this function block would require 15,000 to 25,000 gates, depending on the synthesis compiler. (Without the function block, the minimum configuration of an M4K Pro Series core is about 32,000 gates.) For even higher performance, a designer could create a dual-MAC instruction.

Either way, MIPS says the custom function block typically requires fewer gates than a DSP coprocessor with similar MAC instructions. In most cases, that's true. However, the total number of gates required for an M4K Pro core with the custom function block is similar to that of an ARCtangent-A5 with MAC extensions (about 55,000 gates).

Figure 2 shows the MIPS assembly-language code for the VoIP algorithm's inner loop, using standard MIPS instructions. The loop consists of 18 instructions (including a NOP in a branch-delay slot) that execute in 10–20 clock cycles, depending on the outcome of three conditional branches controlled by input variables. To the right of the assembly-language code, Figure 2 shows how the whole loop collapses into a few lines of C code by calling a predefined intrinsic function, `udi.madd`. That function in turn calls the custom `udi.madd` instruction, which executes in a single clock cycle—on average, 15 times faster than the original code.

Although the MIPS example is relatively simple, it's relevant to real-world applications, and the huge performance improvement is not an isolated case. ARC and Tensilica used similar techniques to achieve their stellar EEMBC benchmark scores, which often exceed unoptimized scores by a comparable magnitude. Indeed, ARC recently introduced telephony extensions for ARCtangent-A5 that accelerate the same kind of algorithms in the MIPS example, and Tensilica has announced 64-bit VLIW extensions that a customer (Conexant) is using for similar applications. (See *MPR 11/25/02-06*, "FLIX: The New Xtensa ISA Mix.")

What MIPS Left Out

For now, the most significant new feature of CorExtend is the ability to add custom instructions without a MIPS architectural license. Pro Series cores have other configurable options, but, in general, those options have been available since MIPS introduced its first synthesizable processors in 1999. (See *MPR 5/31/99-04*, "Jade Enriches MIPS Embedded Family.")

Depending on the core, designers can configure the sizes and set associativity of the instruction and data caches; choose between a fast or small multiply-divide unit; select an MMU with or without a TLB; and configure an interface for on-chip scratchpad RAM. The M4K is cacheless, but it has a configurable SRAM interface with separate or unified instruction and data memory. All Pro Series cores also support up to four duplicate register files for hardware-supported multitasking. (See *MPR 5/20/02-01*, "MIPS' Latest Core Goes Multiprocessor.")

In total, MIPS offers about 40 build-time configuration options. Most options preceded CorExtend and haven't been widely publicized. For instance, designers can implement extensive clock gating to reduce power consumption, and they can configure the trace-and-debug interfaces in several ways. Figure 3 shows a screen from the MIPS graphical configuration tool, which allows designers to rapidly configure a MIPS core by clicking on various checkboxes.

Still, ARC and Tensilica offer more. With the ARCtangent and Xtensa cores, nearly every feature of the microarchitecture is customizable by the user. They allow more ways of balancing performance, design complexity, die area, cost, and power consumption, so they are generally adaptable to a wider range of problems.

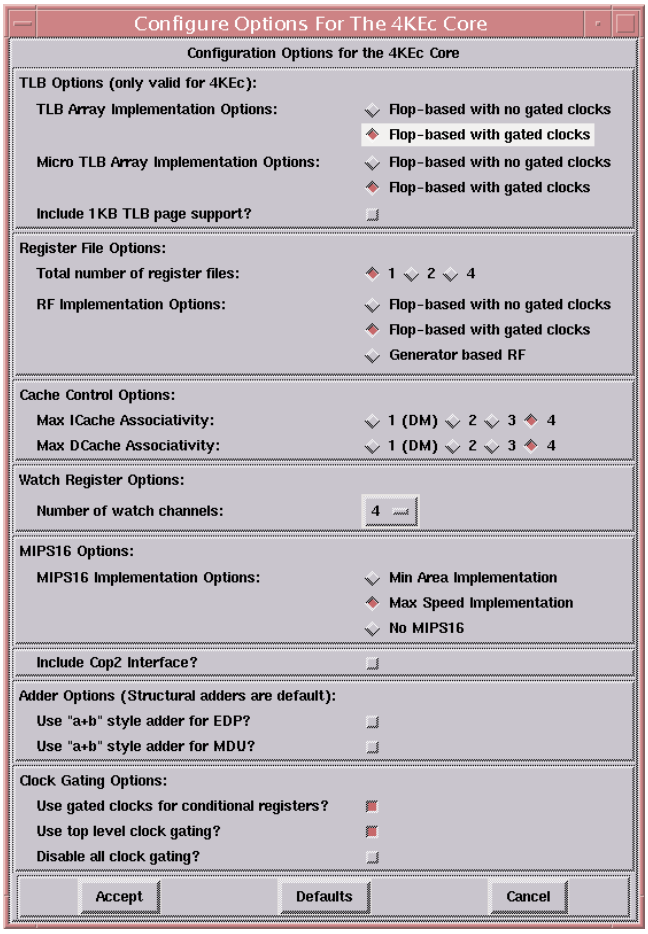

Figure 3. With its synthesizable processor cores, MIPS supplies a graphical configuration tool that is similar to the tools from ARC and Tensilica.

Consider the I/O options. ARC allows designers to implement as many as four 32-bit I/O buses, with separate or unified memory for instructions and data. Tensilica allows designers to configure the processor I/O bus to 32-, 64-, or 128-bit widths, supplemented by an optional local-memory interface for tightly coupled scratchpad RAM and application-specific logic. Pro Series MIPS cores have a fixed 32-bit data bus and the rarely used 32-bit coprocessor interface.

ARCtangent-A5 and Xtensa V each have 32 core registers, extendable to 64—twice as many as MIPS has. Except for four reserved registers in the ARCtangent architecture, all these 32-bit registers can be used by standard and custom instructions, unlike the user-defined registers in a MIPS extension. Xtensa V's optional FPU has 16 registers, 32 bits wide, and the optional Vectra DSP extension has 16 registers, 160 bits wide. With ARCtangent-A5, special load/store instructions in the standard instruction set can access optional 32-bit auxiliary registers, which have a 32-bit addressing range, allowing the addition of more than four billion registers. Xtensa V extensions can also have almost any number of registers —although, as with the MIPS cores, they are not visible to standard instructions.

MIPS Pro Series cores do support up to four duplicate register files, allowing fast context switching among different tasks. However, that's not quite the same as having more registers for a single task. Theoretically, a single task could access multiple register files in a Pro Series core, but some extra housekeeping would be needed, because the MIPS architecture expects a program to see only 32 registers at a time. ARC and Tensilica allow designers to add multiple register files, too, although such an addition is not an easy configuration option: writing HDL code is required to implement the additional register files.

User-defined interrupts with configurable priority levels are another important feature of ARCtangent-A5 and Xtensa V. So are their optional DSP extensions and flexible instruction formats. Most ARCtangent-A5 instructions are conditional, and users can define new condition codes that standard and custom instructions can recognize. With the new ARCompact ISA, designers can create 16- or 32-bit custom instructions, whereas CorExtend limits designers to 32-bit instructions—an important consideration for memory-challenged embedded applications.

Both ARC and Tensilica offer libraries of predefined extension instructions, ranging from simple bit manipulations to DES cryptography acceleration. ARC goes a step further and sells configurable peripheral IP, such as USB cores. Of course, considering how much longer ARC and Tensilica have been in the configurable-processor business, it's no surprise that their products are more mature.

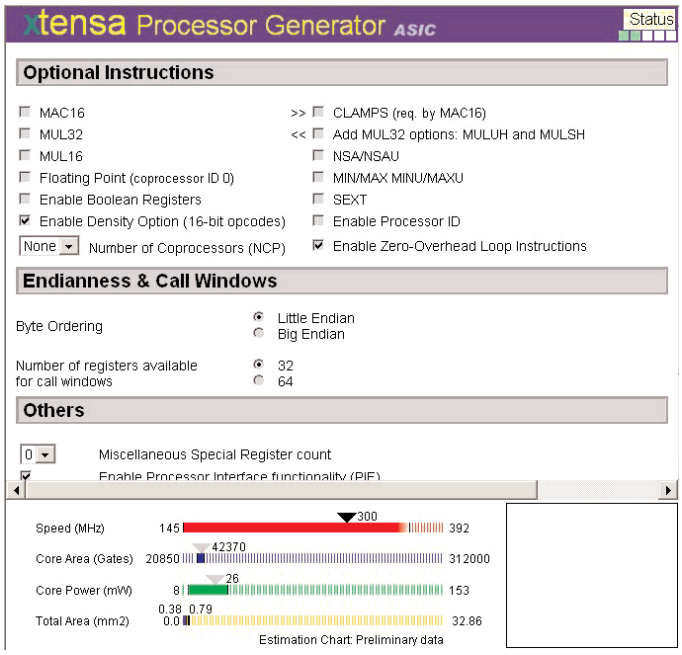

Figure 4. Tensilica's Processor Generator is a Web-based graphical configuration tool for the Xtensa V processor core. Note the continuously updated status display at the bottom.

MIPS's Tools Are Manual

Another shortcoming of CorExtend is the relative lack of tool-chain automation. In this contest, MIPS is slightly behind ARC, and both companies are behind Tensilica. On the hardware side, all three companies have graphical configuration tools that let designers configure the core with a few mouse clicks. On the software side, the differences are starker: MIPS programmers must manually write intrinsic functions and macros to use custom instructions with the development tools; ARC requires a similar level of effort but has some automation for so-called "standard extensions"; and Tensilica has significantly better automation.

Tensilica's advantage was in starting with a blank slate. ARC developed its configurable technology in a piecemeal fashion by modifying a conventional design flow, and MIPS is basically following the same course. In contrast, Tensilica was founded as a configurable-processor company. Instead of retrofitting an existing architecture, Tensilica created a whole front-to-back system that automatically generates software-development tools that match the user's hardware configuration.

An Xtensa user starts by logging onto Tensilica's Web site with a password and accessing the Processor Generator, shown in Figure 4. This is a Web-based tool that presents the configurable options for the Xtensa processor in a series of pages with checkboxes, radio buttons, and menus. With a few mouse clicks, users can configure the caches, buses, interrupts, and other microarchitectural features of the core. The Processor Generator also offers extension options, such as predefined instructions and the Vectra DSP unit. With each selection, a real-time status display estimates the gate count, clock frequency, die area, and power consumption of the current processor configuration.

After selecting all their options, users click a button that sets the back-end part of the Processor Generator into motion. At Tensilica's site, the Processor Generator builds a model of the user's configuration in RTL Verilog or VHDL (another user option) and sends it back across the Internet to the user's site. In addition to the processor core, users also receive preconfigured synthesis scripts, test benches, and software-development tools. The software tools include an assembler, C/C++ compiler, linker, debugger, and instruction-set simulator already modified to match the hardware configuration.

To create custom instructions, Xtensa users write a functional description of the custom logic in the Tensilica Instruction Extension (TIE) language, a proprietary HDL that resembles Verilog. TIE has special semantics that allow the back-end Processor Generator to automatically modify the software-development tools so they can recognize and use the new instructions. The Processor Generator also converts the TIE code into RTL Verilog or VHDL and integrates it with the RTL model of the Xtensa core. Users can compile the RTL files with an industry-standard synthesis tool and target any fabrication process of their choice.

ARC's design flow is more conventional and less automated. As Figure 5 shows, the ARChitect graphical configuration tool resembles Tensilica's Processor Generator and offers similar options for the ARCtangent-A5 processor. One important difference is that ARChitect allows users to configure and integrate ARC's peripheral IP, such as a USB controller core, while they're configuring the processor. When a user clicks the "Go" button, ARChitect builds an RTL model of the processor to the user's specifications by assembling prewritten Verilog or VHDL files, and it generates preconfigured synthesis scripts and test benches.

To supplement ARChitect, ARC recently introduced a new Extension Instruction Automation (EIA) tool. It lets developers create packages of custom extensions, which may include new instructions, core registers, auxiliary registers, and condition codes. These packages are then available from ARChitect. By using the EIA tool to build a library of packages, developers can manage their custom extensions separately from individual projects and more easily reuse the extensions in multiple projects.

Like Tensilica's Processor Generator, ARChitect also offers some predefined extension options, such as additional instructions and DSP extensions. ARC refers to the predefined instructions as "standard extensions," and programmers can use them without writing the assembler macros or C/C++ intrinsic functions normally required for extension instructions. ARC's software-development tools include the MetaWare assembler, High C/C++ compiler, linker, debugger, and instruction-set simulator. ARC also provides a cycle-accurate simulator, signal-visualization tools, a configurable real-time operating system (RTOS), communication protocol stacks, and other system software designed to work with its processor.

For user-defined instructions, ARC's design flow has more in common with MIPS's flow than with Tensilica's. ARC users write their custom logic in Verilog or VHDL, not a proprietary language like TIE, and integrate their RTL code with the ARCtangent-A5 core by hooking into predefined interfaces provided for that purpose. ARC's new EIA tool includes an HDL editor, and custom extensions created with this tool become available in ARChitect, which simplifies the task of extending the core. To use the new instructions with the software-development tools, programmers must write macros for the assembler, intrinsic functions for the C/C++ compiler, and DLLs for the software simulators. Like MIPS, ARC provides templates to make these tasks easier.

The final design step is much the same for ARC, MIPS, or Tensilica: users compile the RTL model with an industry-standard synthesis tool that targets the chip-fabrication process of their choice. All three companies trumpet process portability as a strength of their products.

Don't Sweat the Software Tools

As the preceding analysis shows, Tensilica offers the highest degree of design automation, although one trade-off is that users must learn Tensilica's proprietary TIE language. ARC comes in second, providing a graphical configuration tool like Tensilica's, an extension-management tool, some toolchain automation for standard extensions, and (uniquely) the option to integrate some sophisticated peripheral IP

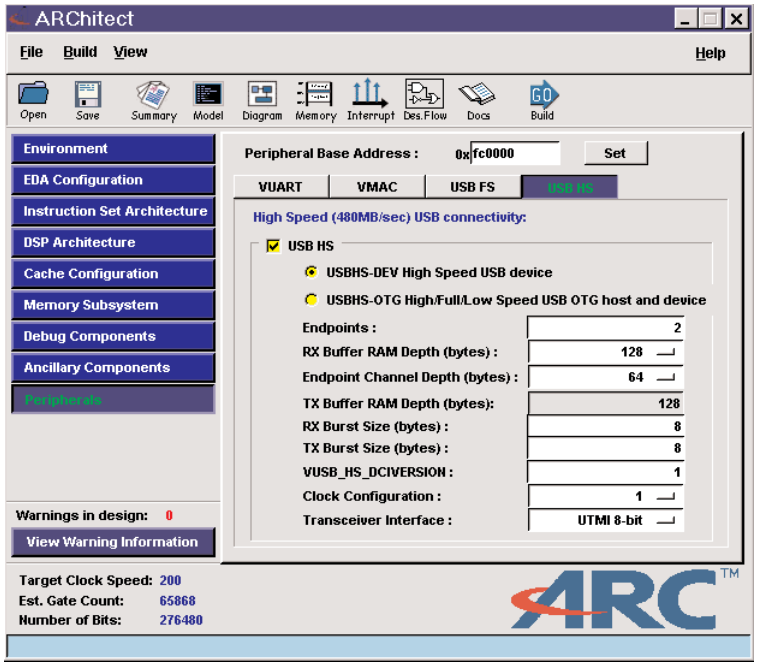

Figure 5. ARC's graphical ARChitect tool lets users configure the ARCtangent-A5 processor core and integrate some peripheral IP. Notice the continuously updated status display at the lower left.

while configuring the processor. MIPS, the newbie in this club, finishes third: it has a graphical configuration tool for the processor but no software tool-chain automation and no options for integrating peripheral IP.

However, in the real world of ASIC and SoC development projects, the differences in software tool-chain automation among ARC, MIPS, and Tensilica are relatively minor. With any customizable processor, the real work isn't in configuring the software-development tools to recognize custom instructions—it's in creating the custom instructions to begin with.

First, developers must determine whether custom logic will provide a large enough increase in performance to justify the extra design and verification effort. Usually, this takes many hours of code profiling and analysis. It's really a process of hardware/software partitioning, because there may be several ways to implement critical algorithms or functions in various combinations of logic gates and software.

Next, developers must create the custom logic in Verilog, VHDL, or TIE and integrate it with the processor core. They must then test and verify the whole design. The final step of configuring the software-development tools—even if it requires manually writing an intrinsic function and DLL for each new instruction—is almost trivial compared with these other tasks.

In fact, by following the project leader's design specifications, a programmer can configure the software tools in a few hours, even before the hardware engineers have written their first line of HDL code. Programmers can begin writing the application software and testing it on simulators in parallel with the hardware flow of the project. Given the turnaround time for spinning silicon, they will probably finish the software before the chip is ready.

If an unusually large project requires several dozen custom instructions, the task of manually configuring the software tools could become more significant. Larger projects multiply the chances for errors. For instance, if a DLL that describes the operation of a new instruction to the simulator varies from the HDL implementation of the instruction, then the simulator may not produce correct results. Today, few developers add more than a dozen instructions to a customizable processor, so the projects are manageable. In the future, as embedded applications grow more complex, that could change.

Still, the larger development risk will probably always be the silicon, not the software. Therefore, the relative degree of tool-chain automation shouldn't drive the decision about which of these configurable processors is the best choice for a particular project.

Read the Fine Print

One potentially important difference among the configurable-processor technologies from ARC, MIPS, and Tensilica is legal, not technical: the ability of developers to protect the extensions they create.

ARC and Tensilica let their customers fully protect independently developed IP, even to the point of not disclosing it. Indeed, at least one company using the ARC processor (Cisco) has patented its custom extensions. MIPS says it will likewise respect the independent ownership of third-party IP. But MIPS adds that it reserves the right to defend the integrity of the MIPS architecture by preventing other companies from using CorExtend to create their own ad hoc versions of the ISA. According to MIPS, that defense could take the form of absorbing third-party instructions into the standard ISA. A clause in the license contract reserves this right for MIPS, even if the licensee patents the extensions.

That's a stark difference from the policies of ARC and Tensilica, and it could cause discomfort for some CorExtend licensees. If MIPS decides to absorb a customer's extensions into the standard ISA, those extensions would become available to all MIPS licensees—including competitors of the company that developed the IP. This policy would seem to discourage third parties from developing IP that contains proprietary algorithms or other unique technology.

MIPS says its architectural licenses have long included a similar clause, and it's never been a problem. The real issue, says MIPS, is preventing a renegade licensee from "hijacking" the MIPS architecture by creating a nonstandard ISA, which could hurt the whole MIPS community. It's difficult to see how this could happen, though, because the CorExtend license also forbids customers from sublicensing their extensions without going through MIPS. That restriction should be enough to stop a licensee from spreading a nonstandard ISA beyond the pins of its own chips.

There is a remote possibility that chips with a nonstandard ISA (that is, chips based on a MIPS core with custom extensions) could become so popular in the marketplace that software vendors might overwhelmingly support it, to the detriment of chips based on the standard MIPS ISA. To our knowledge, hijacking the ARCtangent or Xtensa architectures in this or any other way has never been tried, but perhaps it's because they are less popular than the MIPS architecture. Although Lexra once posed a threat to MIPS by licensing a workalike architecture, that situation was different, because Lexra wasn't originally a MIPS licensee.

A more relevant historical example is the legal entanglement that once encumbered the MIPS16 instruction subset. When MIPS introduced its first synthesizable embedded-processor cores in 1999, the processors conspicuously omitted MIPS16, even though code compression is valued by many embedded-system developers. There was no technical reason for the omission, because it was equally conspicuous that some of Lexra's unsanctioned MIPS-like cores supported MIPS16—which, in effect, threatened to splinter the MIPS ISA.

At the time, MIPS vaguely alluded to legal issues that blocked it from using MIPS16. Apparently MIPS and LSI Logic resolved those issues, because some of the more recent MIPS processor cores do support MIPS16 or its enhancement, MIPS16e. Under the terms of the CorExtend license,

MIPS could head off any future splintering by simply absorbing third-party extensions into the standard ISA.

MIPS may be correct in indicating that CorExtend licensees will tolerate this provision, just as architectural licensees apparently do. But even if the provision doesn't discourage licensees from embedding their proprietary IP in custom instructions, it could impede the flowering of a free market in third-party extensions. That's because the CorExtend license doesn't allow MIPS customers to sublicense their extensions directly to other companies without the approval of MIPS. Although this arrangement eliminates another way a licensee might fracture the architecture, it undermines a key advantage MIPS has over ARC and Tensilica: a more popular CPU architecture that's capable of attracting more third-party support.

A few years ago, ARC tried to encourage third parties to develop and sell proprietary IP for the ARCtangent processor, following the model of plug-in software modules for Web browsers and other PC software. The bait was a special low-cost design license that included documentation and tech support. (See *MPR 6/19/00-03*, "ARC Cores Encourages 'Plug-Ins.'") Unfortunately for ARC, few (if any) third parties snapped up the bait, probably because the relatively small market for ARCtangent-specific IP didn't justify the cost of developing it. Tensilica faces the same obstacle.

Because the MIPS architecture is more widespread than ARC's or Tensilica's, it's easy to envision a lively market of third-party extensions for MIPS processors. However, that market won't be as free to flourish if third parties can't fully protect their IP and license it without involving MIPS as a middleman.

The legal fences surrounding CorExtend indicate that MIPS's highest priority is maintaining full control over the MIPS architecture—even at the cost of deterring some third-party development. It's a tough business decision that can be argued either way. In any case, MIPS has elected to follow a different path in this regard than have ARC and Tensilica.

Pondering CorExtend's Future

Nobody could reasonably expect MIPS to duplicate overnight the years of effort by ARC and Tensilica. Delivering a complete system for customizing a microprocessor requires much more than simply allowing users to modify an RTL model, which is why the open-source or freely licensed processor cores appearing in recent years haven't seriously threatened ARC and Tensilica. But even if CorExtend doesn't duplicate every feature of the competing technology, it does achieve MIPS's initial goal: to deliver the powerful flexibility of an extendable instruction set on a more-popular CPU architecture.

What's next? The most obvious improvement would be more processor-configuration options, all available in the graphical configuration tool. Customizable I/O buses and interrupts would be nice to have. So would a library of predefined instructions for accelerating the kinds of tasks MIPS developers frequently encounter. We would also like to see configurable DSP extensions, like those offered by ARC and Tensilica, surely something MIPS could salvage from the wreckage of Lexra. (See *MPR 8/23/99-05*, "Lexra Adds DSP Extensions.")

Better tool-chain automation would help counter Tensilica's effective marketing of this feature, but MIPS could run into some legal obstacles here. Tensilica recently won two U.S. patents for its system of automatically generating a customized processor with matching software-development tools. In those patents, Tensilica makes broad method claims that could limit MIPS's future options. (See *MPR 12/9/02-01*, "Tensilica Patents Raise Eyebrows.")

ARC's patent claims could also prove troublesome for MIPS, as well as for other newcomers. ARC holds at least three international patents and has 39 patent applications pending in various regions around the world. Those applications should start issuing within a year. In its present state of development, though, CorExtend doesn't appear to trample on any patented IP from ARC or Tensilica, nor on any pending patents we know about.

CorExtend is further proof that configurable-processor technology is coming of age. ARC and Tensilica have blazed the trail, and now this technology is attracting the attention of more-established companies like MIPS. In time, ARM will almost certainly follow. The rewards for making hardware as malleable as software are becoming too great to ignore.

Price & Availability

The MIPS M4K and other Pro Series processor cores with CorExtend are available now. MIPS does not publicly disclose license fees.