Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

TENSILICA'S AUTOMATON ARRIVES

New Design Tool Creates CPU Extensions From C/C++ Programs

By Tom R. Halfhill {7/12/04-01}

What's even faster and cheaper than outsourcing a design project to India? Answer: outsourcing it to a robot. Or, actually, to a new processor design tool that automatically generates application-specific custom instructions by analyzing software written in plain-Jane C/C++.

Last week, Tensilica announced that its long-anticipated XPRES (Xtensa PRocessor Extension Synthesis) tool will ship in 3Q04. A seat license will cost $100,000 and requires an Xtensa LX processor license, which starts at $550,000. XPRES works only with the new Xtensa LX core (see *MPR 5/31/04-01*, "Tensilica Tackles Bottlenecks"), not with the previous-generation Xtensa V core, which remains available. Instead, Tensilica hints that XPRES may be compatible with a future "economy model" Xtensa processor that lacks all the fancy features of Xtensa LX.

Microprocessor Report first covered XPRES after Tensilica disclosed the then-unnamed technology at Embedded Processor Forum 2003. (See *MPR 6/23/03-01*, "Tensilica's Software Makes Hardware.") After a detailed analysis, we concluded that XPRES was a significant innovation, with the potential to dramatically accelerate SoC projects.

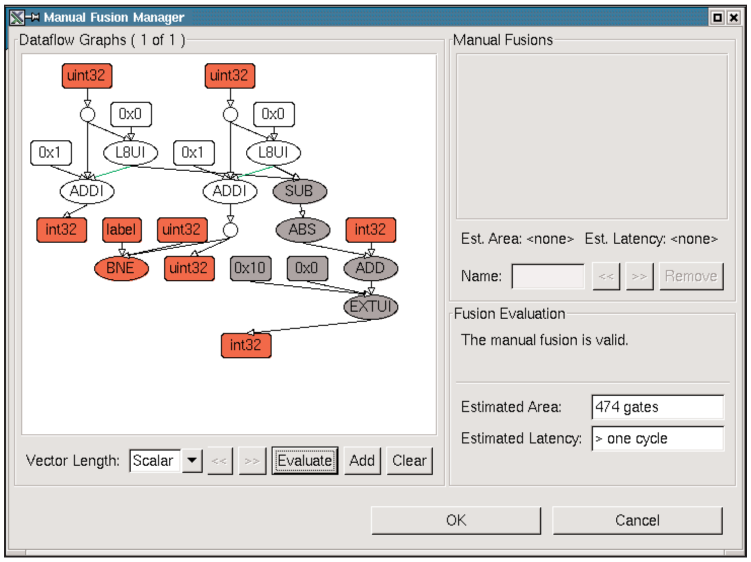
We still feel that way. It's not just that XPRES can automatically generate custom hardware from C/C++ code, a focus of widespread research and development. Rather, it's the whole tool chain and design flow that sets Tensilica's technology apart. Tensilica is closer than any other company to realizing a vision of software-driven automated hardware design that for decades has mesmerized engineers, academic researchers, and entrepreneurs.

Unified Tool Chain Speeds Development

Little about XPRES has changed since our June 2003 report, so we'll summarize the technology here. Basically, XPRES is a static compiler that converts targeted functions in ANSI C/C++ code into Tensilica Instruction Extension (TIE) language, the company's proprietary high-level design language (HDL). TIE is similar to Verilog and VHDL, but it has some special hooks and semantics for Tensilica's Xtensa processors and tool chain. Although TIE is intended primarily for writing user-defined extensions to Xtensa processors, it's a powerful HDL in its own right. In fact, Tensilica's engineers wrote the Xtensa LX core almost entirely in TIE.

Before using XPRES, the first step for customers is to recompile their C/C++ application with Tensilica's Xtensa C Compiler (XCC) and test the program on Tensilica's cycle-accurate Xtensa LX simulator. No other compiler will do, because XCC inserts some special instrumentation for code profiling and analysis. One result of this initial compilation and test run is hints on improving the source code. XCC suggests modifications that will increase performance and make it easier for XPRES to generate application-specific extensions.

Among other things, the XCC profiler identifies which parts of the program execute more frequently, how many instructions of each type execute in each region of code, and which loops would benefit from single-instruction, multiple-data (SIMD) operations. Dataflow graphs for critical loops show the before-and-after effects of SIMD vectorization. (XCC is a vectorizing compiler that can automatically apply SIMD instructions to parallel arithmetic operations.) Other dataflow graphs, as Figure 1 shows, illustrate the effect of fusing multiple instructions into a single operation.



Est. Area: <none> Est. Latency: <none>

The manual fusion is valid.

Estimated Area: 474 gates

Estimated Latency: > one cycle

Vector Length: Scalar

Figure 1. Tensilica's C/C++ compiler and profiler generated this dataflow graph to demonstrate how fusing multiple instructions—in this case, ADD, SUB, and ABS—can create a single custom instruction that calculates a sum of absolute differences. Often, the fused instruction can still execute in a single clock cycle.

XCC's output is the usual executable file—which will run on a base-configuration Xtensa LX processor or simulator—plus some intermediate code that's roughly halfway between source code and object code. The intermediate code contains representations of the dataflow graphs and is the input for XPRES. From this code, XPRES automatically generates application-specific TIE extensions: custom instructions and perhaps some new registers to support the instructions. Indeed, XPRES may generate hundreds or even thousands of possible Xtensa configurations that will accelerate the targeted code regions to different degrees. (Users can fence off regions of code they don't wish to optimize in this fashion.)

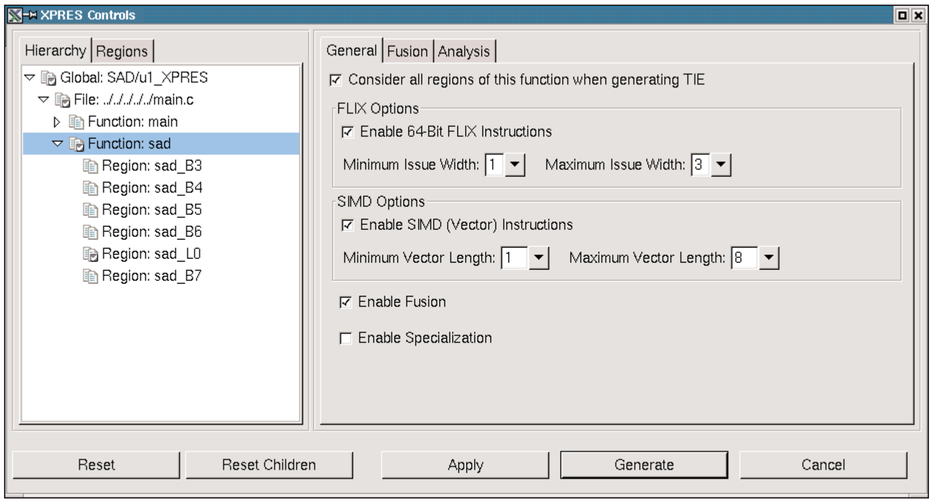
As explained in previous articles, XPRES can generate custom extensions that take advantage of most Xtensa LX features, including SIMD, fused instructions, DSP instructions, and FLIX (Flexible-Length Instruction Xtensions). The XPRES control panel shown in Figure 2 lets users adjust parameters for these different types of extensions—or even disable certain types. For instance, disabling FLIX would save about 2,000 additional gates that the processor requires to decode the VLIW-like FLIX instructions. Frankly, we can't imagine many users disabling these features, considering the huge performance gains they make possible, but it's nice to have the option.

Thousands of Configurations in Minutes

One of the exceptional features of XPRES is its graphical displays that help designers choose the best custom Xtensa configuration from thousands of possibilities. "Best," of course, depends on the designer's goals, balancing higher performance against lower power and cost. XPRES is an impressive tool, but it doesn't possess artificial intelligence. Only a design engineer familiar with the project's specifications can make an intelligent trade-off between the processor's performance and the gate count.

As Figure 3 shows, XPRES aids the decision-making process by presenting an easy-to-read graph that plots the number of gates required for each possible configuration against the number of clock cycles required to execute the optimized function. Picking the optimal Xtensa configuration is as simple as picking the "best" position on the graph, which represents 1,830,796 possible configurations.

For illustrative purposes, Tensilica identified the configuration indicated by the arrow in Figure 3 as the optimal choice for this hypothetical project: an accelerated MPEG4 video encoder. This particular configuration includes 183 new instructions that boost performance over the processor's starting configuration by a factor of three. The extensions would add about 170,000 gates to a 50,000-gate configuration of Xtensa LX (not including caches) and encode an MPEG4 test file (three frames of a 240- × 352-pixel reference image) in 27.5 million clock cycles. Excluding other tasks, the processor could execute the encoder at a clock frequency of 42MHz.



Global: SAD/u1_XPRES
File: ../../../../main.c
Function: main
Function: sad
Region: sad_B3
Region: sad_B4
Region: sad_B5
Region: sad_B6
Region: sad_L0
Region: sad_B7

☑ Consider all regions of this function when generating TIE

☑ Enable 64-Bit FLIX Instructions

Minimum Issue Width: 1 Maximum Issue Width: 3

☑ Enable SIMD (Vector) Instructions

Minimum Vector Length: 1 Maximum Vector Length: 8

☑ Enable Fusion

☐ Enable Specialization

Figure 2. Designers can use this XPRES control panel to specify the minimum and maximum vector lengths of SIMD instructions and the minimum and maximum issue widths of FLIX instructions as well as to disable certain types of extensions. In addition, designers can exclude from optimization individual regions of code in the target application.

Tensilica says it took only one person-day to achieve this result—which included compiling the C/C++ source code with XCC, testing the program on a simulator, reviewing the profiles, feeding the dataflow files into XPRES, and generating the 1.8 million possible configurations.

In comparison, manually designing a similar MPEG4 accelerator in Verilog or VHDL would probably take several person-years. More to the point, a regular design team probably wouldn't create 183 new instructions or more than one solution to the problem—much less 1.8 million solutions. Instead, the designers would aim for a particular level of performance within a gate-count budget and try to stay within those specifications. If something went wrong, the performance might fall short, or the gate count might break the budget. Months of work might have to be redone.

It's the ability of XPRES to rapidly generate so many possible configurations that makes it so compelling. Like any automated tool, it probably won't create an extension that's clearly superior to an extension designed by skilled logic engineers, just as a C compiler usually doesn't emit object code that's better than the handiwork of an experienced assembly-language programmer. But compilers have largely replaced assembly-language programmers, because they can do the job well enough for most purposes, and they can do it in minutes. XPRES offers the same trade-off between development time and code efficiency for hardware that compilers do for software. Furthermore, by offering thousands of configurations to choose from—and by presenting the options in easily interpreted form—XPRES allows SoC developers to make intelligent trade-offs more rapidly than would be possible with a lengthy manual-development project.

Recursive Tool Generation

Figure 4 shows the result of an XPRES run: automatically generated TIE extensions, ready to integrate with the Xtensa LX processor. At this point in the development process, designers have an opportunity to manually fine-tune the TIE source code and add any extensions they may have written manually in TIE. For instance, if the designers created TIE extensions for an earlier Xtensa processor, this point is where they can adapt the extensions for Xtensa LX. (Normally, little or no modification is required, because the TIE language is largely unchanged.)

Everything comes together in the Processor Generator, Tensilica's back-end design-automation tool. In about an hour, the Processor Generator integrates all the TIE extensions with the Xtensa LX processor core; converts the whole model into register-transfer-level (RTL) Verilog or VHDL; writes scripts for popular synthesis compilers; creates a cycle-accurate simulator of the customized processor; modifies the software-development tools so they can recognize the new instructions

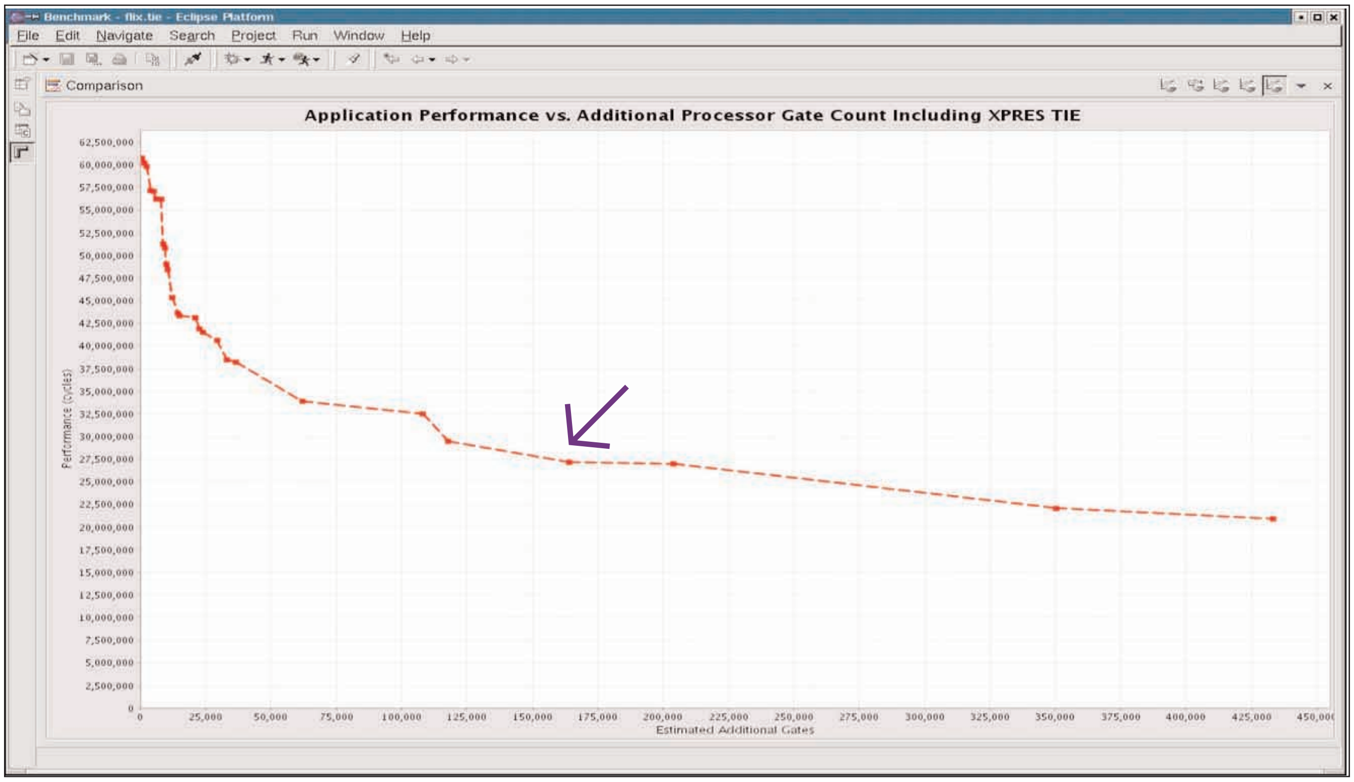
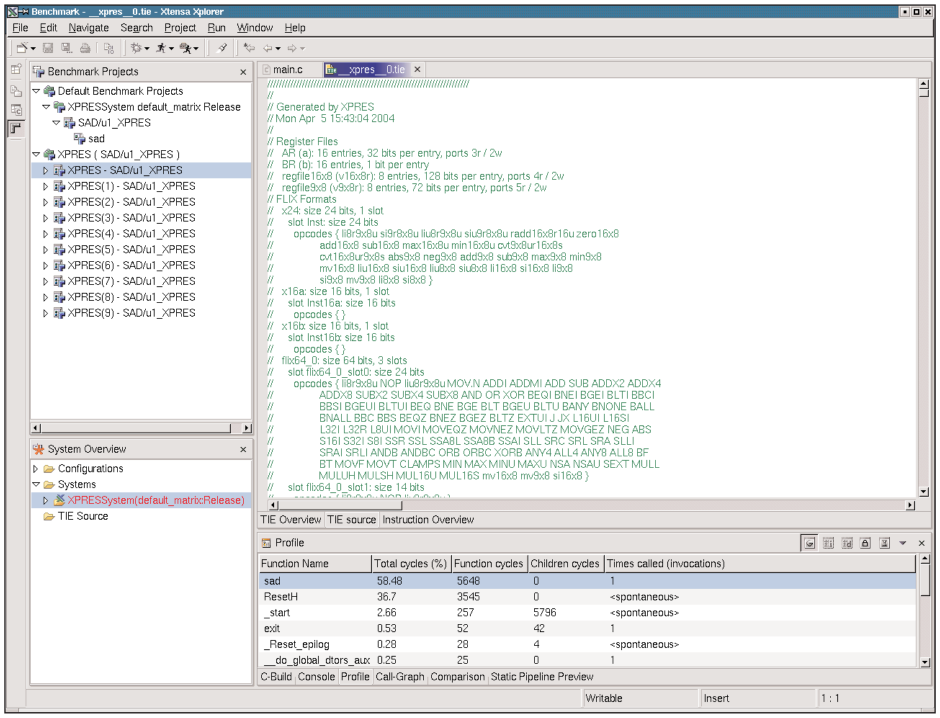


Figure 3. In 30 minutes, XPRES generated 1,830,796 possible Xtensa LX configurations for accelerating an XviD MPEG4 video encoder, assuming Quarter Common Intermediate Format (QCIF) at 15 frames per second. Each configuration has one or more custom extensions, which may include new instructions and registers. The x-axis shows the estimated number of additional gates required for the extensions (not including the processor core), and the y-axis shows the number of clock cycles required to encode an MPEG4 test file.



```
//////////////////////////////////////////////////////////////////////
//
// Generated by XPRES
// Mon Apr  5 15:43:04 2004
//
// Register Files
//   AR (a): 16 entries, 32 bits per entry, ports 3r / 2w
//   BR (b): 16 entries, 1 bit per entry
//   regfile16x8 (v16x8r): 8 entries, 128 bits per entry, ports 4r / 2w
//   regfile9x8 (v9x8r): 8 entries, 72 bits per entry, ports 5r / 2w
// FLIX Formats
//   x24: size 24 bits, 1 slot
//     slot Inst: size 24 bits
//       opcodes { li8r9x8u si9r8x8u liu8r9x8u siu9r8x8u radd16x8r16u zero16x8
//                 add16x8 sub16x8 max16x8u min16x8u cvt9x8ur16x8s
//                 cvt16x8ur9x8s abs9x8 neg9x8 add9x8 sub9x8 max9x8 min9x8
//                 mv16x8 liu16x8 siu16x8 liu8x8 siu8x8 li16x8 si16x8 li9x8
//                 si9x8 mv9x8 li8x8 si8x8 }
//   x16a: size 16 bits, 1 slot
//     slot Inst16a: size 16 bits
//       opcodes { }
//   x16b: size 16 bits, 1 slot
//     slot Inst16b: size 16 bits
//       opcodes { }
//   flix64_0: size 64 bits, 3 slots
//     slot flix64_0_slot0: size 24 bits
//       opcodes { li8r9x8u NOP liu8r9x8u MOV.N ADDI ADDMI ADD SUB ADDX2 ADDX4
//                 ADDX8 SUBX2 SUBX4 SUBX8 AND OR XOR BEQI BNEI BGEI BLTI BBCI
//                 BBSI BGEUI BLTUI BEQ BNE BGE BLT BGEU BLTU BANY BNONE BALL
//                 BNALL BBC BBS BEQZ BNEZ BGEZ BLTZ EXTUI J JX L16UI L16SI
//                 L32I L32R L8UI MOVI MOVEQZ MOVNEZ MOVLTZ MOVGEZ NEG ABS
//                 S16I S32I S8I SSR SSL SSA8L SSA8B SSAI SLL SRC SRL SRA SLLI
//                 SRAI SRLI ANDB ANDBC ORB ORBC XORB ANY4 ALL4 ANY8 ALL8 BF
//                 BT MOVF MOVT CLAMPS MIN MAX MINU MAXU NSA NSAU SEXT MULL
//                 MULUH MULSH MUL16U MUL16S mv16x8 mv9x8 si16x8 }
//     slot flix64_0_slot1: size 14 bits
```

Function Name	Total cycles (%)	Function cycles	Children cycles	Times called (invocations)
sad	58.48	5648	0	1
ResetH	36.7	3545	0	<spontaneous>
_start	2.66	257	5796	<spontaneous>
exit	0.53	52	42	1
_Reset_epilog	0.28	28	4	<spontaneous>
__do_global_dtors_aux	0.25	25	0	1

Figure 4. In this screen shot from Tensilica's Xtensa Xplorer development tool, TIE source code automatically generated by XPRES appears in the large window at the upper right. The smaller window at the bottom contains profiling information, such as the number of clock cycles required to execute a particular C/C++ function. The window at the upper left allows users to navigate the TIE files XPRES generates.

and registers; and makes popcorn for the design team. (OK, the last step is an exaggeration, but everything else is true.)

An important new part of this automated process is modification of the software-development tools. In the past, Xtensa programmers could invoke custom instructions by calling automatically generated intrinsic functions from their C/C++ programs, but not until they manually inserted the function calls into the source code. With Xtensa LX, the improved Processor Generator modifies XCC so it can automatically use the custom instructions when compiling the targeted C/C++ functions. The program's source code doesn't change. In fact, XCC is smart enough to use the custom instructions when compiling *any* similar C/C++ code—in the target program or any other program—even if the customer didn't initially identify that code for optimization.

It was this new feature—compiler-directed optimization with- out source-code modifications—that allowed Tensilica to shatter all records in the EEMBC out-of-the-box consumer benchmark suite. (See the sidebar, "How Tensilica Busted the Benchmarks," in *MPR 5/31/04-01*, "Tensilica Tackles Bottlenecks.") Although Freescale's MPC7447A recently seized the crown from Tensilica with an out-of-the-box ConsumerMark score of 197.2, the PowerPC processor had to gallop at 1.4GHz to beat the Xtensa LX score of 171.6 at a simulated clock speed of only 330MHz.

To sum up: Tensilica's design flow is a giant feedback loop. At the front end, SoC developers use XCC to compile, test, and profile their C/C++ programs. Using this feedback, developers can write their own custom extensions in TIE or choose from the zillions of extensions XPRES churns out. At the back end, the Processor Generator modifies XCC so it can recompile the C/C++ program, using the custom extensions it helped create in the first place. With each iteration of this feedback loop, the tools automatically generate new application-specific versions of themselves, in addition to new configurations of the Xtensa LX processor. And developers can repeat this process as often as necessary to optimize their design. (See Figure 5.)

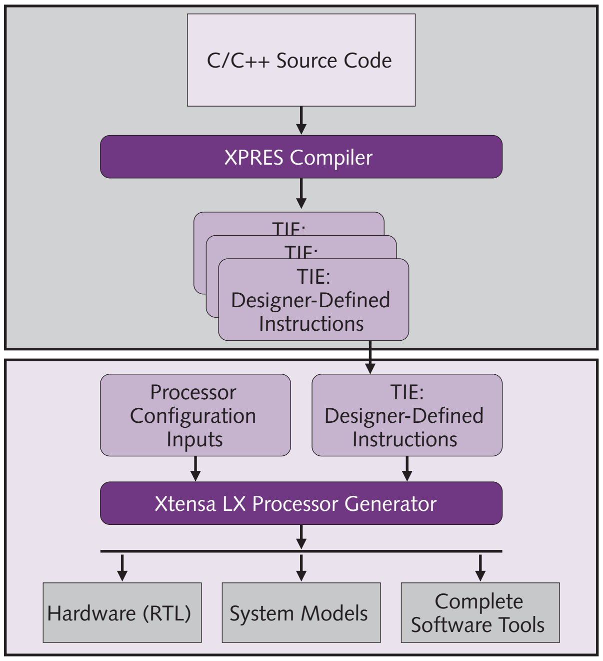

C/C++ Source Code
XPRES Compiler
TIE:
TIE:
TIE: Designer-Defined Instructions
Processor Configuration Inputs
TIE: Designer-Defined Instructions
Xtensa LX Processor Generator
Hardware (RTL)
System Models
Complete Software Tools

Figure 5. Tensilica's XPRES design flow uses feedback-directed optimization to enhance both the configurable Xtensa LX processor and the software-development tools that drive the front end of the process. Every customer can end up with a processor configuration and tool chain tuned for the application and truly unique.

Design-Time Automation Speeds Up Projects

No conventional ASIC/SoC design flow or tool chain can match the speed and interactivity of Tensilica's XPRES system. A complete iteration of the whole XPRES feedback loop might take as little as one day, compared with months for a conventional design project using ordinary tools. ARC International and MIPS Technologies offer customizable processor cores with some slick automated tools, but they fall short of Tensilica's highly automated front-to-back system.

Moreover, Tensilica automates one of the most time-consuming parts of a project: grinding out line after line of RTL for application-specific logic. XPRES can outstrip all that effort in minutes. The other tedious phase of a project—verifying the new logic—also contracts, because TIE is a correct-by-construction language that's supposed to make a faulty design impossible. (If XPRES does spit out a faulty design, the onus is on Tensilica, not the customer, to diagnose and fix the problem.) XPRES frees people for the real brainwork, such as specifying the design, writing better software, and making the design trade-offs that will distinguish the product in the marketplace.

As far as is publicly known, the only alternatives faster than XPRES are those that altogether eliminate the need to spin a custom chip. For instance, startup Stretch Inc. will soon begin selling standard parts that integrate some reprogrammable logic with an Xtensa V processor core. Stretch's C/C++ compiler can generate custom instructions to accelerate critical parts of a program, and the company's proprietary tools implement the instructions in the chip's programmable logic. (See *MPR 4/26/04-01*, "Stretching Performance.") Other companies (such as Adaptive Silicon, M2000, Proceler, and STMicroelectronics) have offered or promised similar technology, generally without much success in the marketplace.

Tensilica's technology differs from the programmable-logic solutions because it's designed to speed up a conventional ASIC/SoC project. In high-volume applications that justify the nonrecurring engineering costs of a custom design, Tensilica's system has a cost advantage over more expensive standard parts having programmable logic. Tensilica would probably have a performance advantage, too, because standard logic is faster than programmable logic. In addition, XPRES can automatically generate SIMD, DSP, and FLIX instructions that may not be available with other processor architectures or microarchitectures. (For example, the Xtensa V core in Stretch's chips doesn't support FLIX.) All things considered, XPRES achieves an important milestone in design automation for programmable ASICs and SoCs.

From C to TIE to RTL

There's an intriguing aspect of the XPRES system that Tensilica isn't strongly promoting but that nevertheless stirs some thought. XPRES translates C/C++ source code into TIE, and the Processor Generator translates TIE into RTL. The main assumption is that the C/C++ code comes from a software application that needs optimizing. But it doesn't have to be. Someone could use Tensilica's system to design custom logic in C/C++ instead of in Verilog, VHDL, or even TIE.

Note that XPRES would allow a designer to write the functional specification for the custom logic in plain old ANSI C or C++, not a newfangled variant like System C that has special constructs for hardware design. Even without those constructs, XPRES can generate surprisingly sophisticated logic that uses parallel SIMD and FLIX operations. Of course, the finished logic works only with an Xtensa LX processor, so Tensilica's system isn't a universal substitute for System C and similar technologies.

Anything as good as XPRES must have other drawbacks. One, already mentioned, is the limitations of an automated tool's output compared with the handiwork of a skilled engineer. Sometimes, XPRES will lack the human insight to thoroughly understand a problem and create an elegant solution. At other times, XPRES might generate relatively sloppy TIE code that requires more gates than handcrafted TIE code does. Because XPRES can generate the extensions in minutes instead of months, most customers probably won't care. If they do care, they can manually tweak the generated TIE code. And with time, XPRES will only get better. At this stage, it's like an early Cobol compiler from the 1950s.

Another limitation of XPRES is that it's a processor design tool, not an SoC design tool. At present, it cannot create the user-defined TIE ports described in our previous Xtensa LX article—very wide parallel I/O ports that connect the Xtensa processor core to external hardware blocks. Nor can XPRES partition and distribute a large task across multiple cores in a multiprocessor SoC. At least, not yet.

In short, XPRES doesn't make engineers obsolete, any more than compilers made programmers obsolete. Just as there remains a place for skilled assembly-language programmers, there will continue to be a role for skilled logic designers who understand a complex algorithm better than XPRES can. Meat still rules. However, XPRES allows SoC developers to approach a project at a higher decision-making level, with the opportunity to delegate the routine logic design to a robot. The engineers can think globally while XPRES acts locally. ◇

Price & Availability

Tensilica's XPRES extension-development tool is an extra-cost option for Xtensa LX customers. A floating seat license (one user at a time) costs $100,000, not including the Xtensa LX license, which starts at $550,000 for a single-processor design. For more information, see *www.tensilica.com*.