Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

TENSILICA'S SOFTWARE MAKES HARDWARE

New Tool Customizes Processor by Analyzing C/C++ Code

By Tom R. Halfhill {6/23/03-01}

Since the dawn of computing, programmers have been writing software to suit the hardware. What choice did they have? Code is flexible; metal is not. Until now. At **Embedded Processor Forum 2003** last week, Tensilica unveiled an impressive addition to the tool chain for its Xtensa configurable-processor architecture. A new code-analysis and hardware-generation tool—so new it doesn't have a catchy name—automatically creates processor extensions that accelerate critical functions in C/C++ source code. Custom extensions can include new instructions, registers, and function units. In minutes, the tool can evaluate thousands of possible extensions and sort them by performance (clock cycles) and efficiency (gate count). When a developer selects the optimal design for the target application, the tool automatically generates the extension in Tensilica's proprietary hardware design language and integrates it with the Xtensa processor core, ready for logic synthesis.

In addition to creating processor extensions, the new tool links into Tensilica's existing tool chain, which automatically generates a C/C++ compiler, assembler, debugger, cycle-accurate software simulator, and real-time operating system (RTOS)—all tailored for the customized processor. The compiler automatically uses the new custom instructions to accelerate the application program without modifying the original C source code. And when the tool ships next year, it will plug into the new Xtensa Xplorer integrated development environment (IDE) that Tensilica announced on June 16. (See sidebar, "Xtensa Xplorer Unites Tool Chain.")

We are aware of no other configurable microprocessor or tool chain that approaches this level of design automation. The configurable processors from ARC International and MIPS Technologies allow developers to create custom extensions for accelerating software, and code-profiling tools help programmers identify the most critical routines and inner loops. However, someone must still write the VHDL or Verilog code for the custom extensions and integrate it with the processor core. To evaluate alternative implementations of an extension, developers must manually create the extensions and test them, one by one, in a cycle-accurate simulator. In most cases, developers must also modify the software-development tools before using a custom extension, and they must write intrinsic functions to use the new instructions in their application code.

In less time than it takes for a single iteration of that labor-intensive process, a developer using Tensilica's new system will be able to explore thousands of possible extensions, generate the optimal choice, modify all the software-development tools, and recompile the application with optimized instructions.

Tensilica's new tool has the potential to dramatically change SoC development. Hardware/software partitioning—physically dividing an application between functions performed in logic and functions performed by code—has never been easier. It's a significant step toward a unified design methodology that is application-centric, not hardware- or software-centric.

A New Link in the Chain

Tensilica's new auto-generation tool doesn't alter the basic design flow or tool chain the company introduced with its

Xtensa Xplorer Unites Tool Chain

In addition to unveiling the automatic TIE generator at Embedded Processor Forum, Tensilica has announced a new IDE that will give hardware and software developers a common tool platform for designing Xtensa-based SoCs. The IDE, known as Xtensa Xplorer, will host Tensilica's full range of hardware- and software-development tools as plug-in components. The TIE generator will join those components when it ships next year.

Tensilica built Xplorer on the Eclipse IDE, an open-source project backed by a consortium consisting of Tensilica, IBM, Intel, Oracle, QNX, SAP, Wind River Systems, and other companies. Eclipse is a generic IDE written in Java that isn't specific to any particular programming language, processor architecture, operating system, or hardware platform. Plug-in tools adapt it for different purposes. IBM is using Eclipse to host a Java software-development environment; Tensilica is using Eclipse to host its C/C++ compilers, Xtensa assembler, TIE compiler, TIE generator, and other tools.

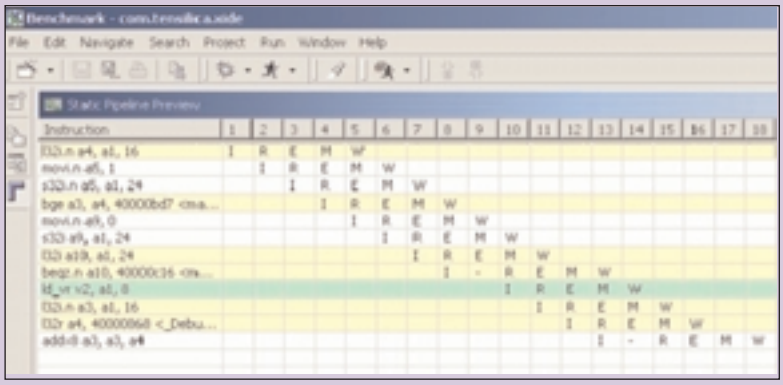
Although Eclipse is an open-source project, it requires a commercial public license (CPL) instead of the more common GNU public license (GPL). Companies like Tensilica say the CPL offers more protection for the proprietary code they add to the open-source code. (For more information about Eclipse and CPL, visit www.eclipse.org.)

Even more than Tensilica's previous tools, Xplorer blurs the boundary between hardware and software SoC development. Using a single IDE, developers can write an application program in C, C++, or assembly language, design custom instructions for an Xtensa processor in TIE language, compile the TIE instructions on a desktop workstation, use the custom instructions in the application code, compile or assemble the program, test it in a cycle-accurate simulator, and observe the results. In a few hours, developers can test many variations of custom extensions before settling on their optimal choice.

Only when the extensions are finished is it necessary to submit the design to Tensilica's online Processor Generator to compile the TIE code and Xtensa core into RTL Verilog or VHDL for synthesis with a logic compiler. Postponing logic synthesis is an advantage when developing extensions, because Xplorer's local TIE compiler and cycle-accurate simulator are much faster than a logic compiler and gate-level simulator.

New tools in Xplorer make it easier for developers to evaluate the performance and efficiency of their custom extensions. One tool analyzes TIE code to estimate (within

Continued on Page 3



Xplorer's Pipeline Viewer helps developers tune their custom extensions and application code for Xtensa's five-stage pipeline. The disassembled code is on the left, and the letters on the right represent pipeline stages: instruction fetch (I), register access (R), execute (E), memory access (M), and writeback (W). A dot in a pipeline stage indicates a stall.

first Xtensa configurable processor in 1999. As before, developers begin by configuring the base configuration for the target application by using Tensilica's special design tools. Developers can choose from a variety of prefab options, such as 16- and 32-bit multipliers, a 32-bit floating-point coprocessor, and a 16-bit multiply-accumulate (MAC) extension. For further optimizations, developers can create custom extensions with the Tensilica Instruction Extension (TIE) language, a proprietary hardware description language (HDL).

After configuring and customizing the core, developers send their design over the Internet to Tensilica's web-based Processor Generator, which converts the processor and user-defined TIE extensions into register-transfer-level Verilog or VHDL. The Processor Generator also creates synthesis scripts and modifies the software-development tools to match the processor's customized instruction set. After testing the design in the cycle-accurate software simulator, developers can generate a gate-level netlist with their own synthesis compiler, targeting a fabrication process and foundry of their choice. Meanwhile, programmers can work on their application software by using Tensilica's Xtensa C/C++ Compiler (XCC) or a modified GNU compiler. This basic design flow is common to all Tensilica Xtensa cores, including the current Xtensa V, introduced last year. (See MPR 9/16/02-01, "Tensilica Xtensa V Hits 350MHz.")

What's changing is the crucial step of creating new custom extensions. This is the feature that sets customizable processors apart from other synthesizable processor cores, and it's the reason they can leapfrog the performance of competing processors. (See MPR 4/9/01-01, "Stretching Silicon to the Max.") Until now, Xtensa developers had to manually write all their custom extensions in TIE language. Tensilica's

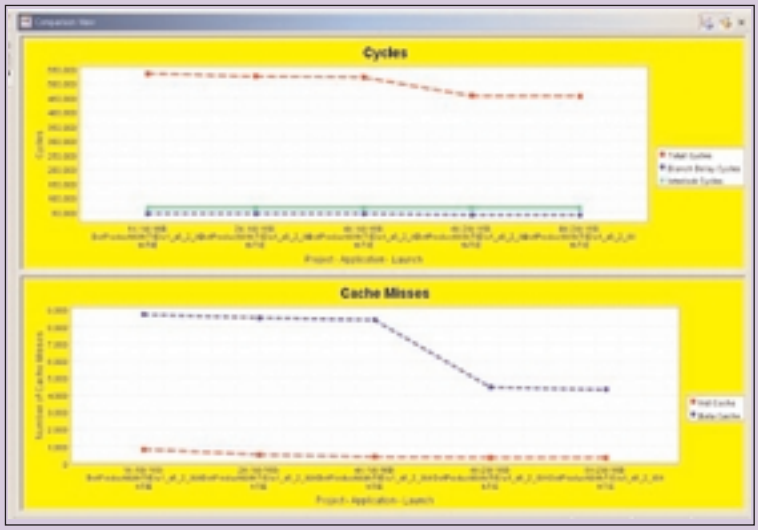
10–20%) how many gates an extension will need—no logic synthesis required. Another tool, shown in the figure below, lets developers step through a section of disassembled application code while observing how the instructions flow through Xtensa's five-stage execution pipeline. This hybrid hardware/software tool allows developers to analyze critical sections of their programs, especially the code that uses custom instructions. By modifying the TIE code for a custom instruction, developers may be able to reduce or eliminate unexpected pipeline bubbles and stalls.

Other Xplorer tools are more focused on hardware design, although they also provide important feedback to software developers. One example, shown in the figure below, is the Cache Xplorer, which analyzes the way a program uses the processor's instruction and data caches. It displays a graph that can show at a glance whether, for instance, a 16K two-way set-associative cache is better for the application than an 8K four-way set-associative cache. Dozens of different cache configurations are possible with an Xtensa processor, so the Cache Xplorer makes it easier to find the smallest configuration that yields the best performance.

Another Xplorer plug-in is a tool for designing multicore SoCs. It helps developers manage local and global memory maps and test their chip-level designs in simulation. Tensilica says the average Xtensa-based SoC has at least five processor cores, so this is a valuable feature.

Xplorer works with a new revision of TIE that streamlines the design of custom extensions. It's a higher-level, more-abstract version of the language that automatically handles some low-level details. For instance, developers can create custom instructions without specifying the opcodes, and they can define new register files without manually creating custom load/store instructions to access the registers. The TIE compiler assigns opcodes and creates any necessary load/store instructions automatically. (Without the addition of special load/store instructions, Xtensa's instruction set cannot access user-defined registers beyond the maximum of 64 core registers.) Flexible opcode assignments will make it easier to integrate extensions from different development teams and third-party IP providers without conflicts in the opcode map.

Tensilica will release Xplorer in July and immediately ship it to all current Xtensa V licensees. It will be the standard IDE for future customers, though most of the plug-in tools require an extra-cost seat license.



Visual graphs of simulated cache behavior allow developers to experiment with different configurations of Xtensa's instruction and data caches. In this example, cache misses nosedive when a direct-mapped 4K instruction cache is improved with two-way set associativity.

new TIE generator, scheduled to ship early next year, will eliminate or greatly reduce the manual labor required.

The first phase is to compile an application program with XCC, using a new optimization flag. This flag tells the compiler to suggest ways of tuning the C/C++ source code for higher performance and better TIE generation. (Sloppy source code is more difficult to optimize and leads to inefficient gate-level logic—garbage in, garbage out.) The compiler produces a wealth of information about the program, ranking regions of code by their frequency of execution, identifying loops that lend themselves to vector operations, drawing data-flow graphs for critical routines, and counting the occurrences of different types of opcodes in every code region.

Of course, this code-analysis phase assumes the application software is available before hardware development begins. In many cases, the hardware developers and software programmers work on their respective parts of a project in parallel, posing a potential chicken-and-egg problem. How can the hardware developers create custom extensions to optimize a software application that may not exist yet?

One answer is that the entire software application need not be finished before this kind of code analysis is performed. For instance, high-level user-interface code usually isn't the target of processor optimizations. Instead, custom extensions generally accelerate the critical algorithms or inner loops that are more vital to the program's operation. JPEG compression/decompression routines, encryption algorithms, and network-protocol stacks come to mind, and they may be the first parts of the program that developers write or obtain from a third party.

In other cases, the software may be finished but doesn't run fast enough on a standard-part embedded processor. Or

perhaps the software runs fast enough, but only at a clock frequency that pushes power consumption beyond the project's specification. To achieve the necessary performance within the specified power envelope, the hardware developers must design an SoC around a configurable processor core with custom extensions. They need to analyze the C/C++ source code before partitioning the application into hardware and software components.

If none of these examples applies and the software simply isn't ready yet, then Tensilica's new TIE generator isn't very useful. The project stalls until the programmers write enough code to begin a meaningful analysis. However, that would happen anyway if the hardware developers were creating the custom extensions manually. Configurable processors are predicated on the assumption that developers know enough about their application to design an intelligent configuration. Rarely are developers so lost in the dark that they can't make an educated guess about which will be the most critical parts of their application.

Making a Silk Purse From C Code

There are three broad techniques for improving the performance of an application with TIE: fusing two or more related operations into a single instruction; using SIMD instructions to vectorize data-intensive operations; and grouping together multiple independent operations using Tensilica's VLIW-like Flexible-Length Instruction Xtensions (FLIX). The automatic TIE generator can use any of those techniques or a combination of them.

Instruction fusion is the most straightforward method. A simple example is an addition followed by a shift, as seen in this fragment of C source code, which adds each element of two arrays and shifts each result two bits to the right:

```
int *a, *b, *c;
for (int i=0; i<n; i++)
   c[i] = (a[i]+b[i]) >> 2;
```

Each iteration of this loop requires two arithmetic instructions (the add and the shift), two loads, one store, and three instructions to increment the address pointers into the arrays—eight machine-code instructions in all. Fusing the add and shift operations into a single instruction will speed up the loop. When the TIE generator analyzes this source code, it creates in TIE language a new instruction called `add_shift`:

```
operation add_shift(out AR c, in AR
a, in AR b)
   {wire t[31:0] = a+b;
   assign c = {2{t[29]}, t[29:0]};}
```

The `add_shift` instruction adds the two input operands (a and b, the array elements in the program), holds the sum as a temporary 32-bit value (t), right-shifts the value by two bits, and returns the result (c).

With Tensilica's existing tools—and with the tools for other configurable processors—a programmer must modify the C source code by creating an intrinsic function that calls the newly defined instruction. The modified source code would substitute the function call (which actually tells the compiler to invoke the new instruction) in the body of the loop:

```
int *a, *b, *c;
for (int i=0; i<n; i++)
   c[i] = add_shift(a[i], b[i]);
```

However, Tensilica's TIE generator and improved tool chain will eliminate that manual step. Merely recompiling the application with XCC will automatically generate machine code that optimizes the loop with the new instruction. The source code doesn't have to change.

The separate add-and-shift operations in the original loop required two clock cycles to execute, whereas the fused `add_shift` instruction executes in a single cycle, thus saving one clock cycle per loop iteration. In addition, the processor must fetch only one instruction instead of two. If the array is large, or if the program frequently executes this loop, the performance improvement could be significant, even with this simple example.

Vectorizing With SIMD

A more sophisticated optimization technique is SIMD, or data vectorization. When Tensilica introduced Xtensa V last year, the company trumpeted XCC's ability to automatically vectorize some data-intensive operations when compiling C/C++ source code for the processor's optional Vectra DSP extension. Now the TIE generator is using the same intelligence to find opportunities for vectorization and to implement the vector operations as custom instructions in TIE language. Consider the following example of C source code, which is similar to the previous one:

```
short *a, *b, *c;
for (int i=0; i<n; i++)
   c[i] = a[i]+b[i];
```

To simplify this example, the array elements are 16-bit integers instead of 32-bit integers, and there's no shift after the add. (The tool can vectorize code that uses 8-, 16-, or 32-bit integer or floating-point values as well as custom data types, such as 24-bit fractional values.) The TIE generator will optimize this loop by creating a new four-way SIMD instruction called `add16x4` and a new SIMD data type called vec, which packs four 16-bit integers into a 64-bit register:

```
regfile vec 64 16 v;
operation add16x4(out vec c, in vec
a, in vec b)
{assign c = {a[63:48]+b[63:48],
a[47:32]+b[47:32],
a [ 3 1 : 1 6 ] + b [ 3 1 : 1 6 ] ,
a[15:0]+b[15:0]};}
```

The first line of TIE code creates a new 64-bit vector register set and associated C data type; the next line defines the new instruction and parameter list; and the third line performs four additions in parallel, using the input operands (a and b), and returns the result (c). Implicitly, this line also directs the logic-synthesis compiler to replicate

enough function units to execute the SIMD instruction in a single clock cycle.

As with the previous example, XCC will automatically use the new instruction to optimize the loop after recompiling the program. With the existing tool chain, a programmer would have to modify the C source code by calling the new instruction as an intrinsic function and changing the loop increment from a single-step walk through the arrays to a four-step hop.

Although the `add16x4` instruction doesn't execute any faster than an ordinary add instruction—both require one clock cycle—it's a four-way SIMD operation, so it performs four adds during that cycle. (Xtensa V supports two-, four-, and eight-way SIMD operations, and wider operations are possible by further customizing the processor.) As with the fusion example, SIMD can significantly boost performance if the arrays are large or frequently summed.

It's unreasonable to expect the TIE-generation tool to identify every opportunity for vectorization, but it will have a better chance if programmers optimize the program in ways suggested by the XCC profiler.

ROSS MEHAN

Tensilica's Dror Maydan, director of software development, presenting at EPF2003.

Quick, Henry, the FLIX!

Another technique for accelerating software with TIE language is to combine multiple operations into a long instruction word, using FLIX. This unusual capability allows developers to define their own 32- or 64-bit VLIW bundles containing one or more instructions. The instruction lengths are extremely flexible. In an example presented at last year's Microprocessor Forum, Tensilica showed how to pack nine instructions into a 64-bit bundle, using odd instruction lengths, including some five-bit custom instructions. FLIX can greatly improve code density, already a strong point of the Xtensa architecture. (See *MPR 11/25/02-06*, "FLIX: The New Xtensa ISA Mix.")

With Tensilica's existing tool chain, developers must manually create their FLIX extensions in TIE language. The new TIE generator can create FLIX bundles automatically. Here's the same C source code seen earlier in the fusion example:

```
int *a, *b, *c;
for (int i=0; i<n; i++)
   c[i] = (a[i]+b[i]) >> 2;
```

This time, instead of fusing the add and shift operations into a single instruction, the TIE generator optimizes the loop by creating three-instruction FLIX bundles. To do this, it must define the FLIX format and specify which kinds of instructions the compiler can schedule in each instruction slot. Here is the TIE code automatically generated from the C code:

```
length l 64 {InstBuf[3:0] == l4}
format f l
   slot slot0 f[*] l32i, s32i, nop
   slot slot1 f[*] add, srai, nop
   slot slot2 f[*] addi, nop
```

The first two lines of TIE create the new 64-bit-wide, three-slot FLIX format. The next three lines list the instruction types allowed in each slot. Note that `l32i` and `s32i` are 32-bit load and store instructions, respectively; *srai* is a shift-right instruction; and each slot must be able to hold a null operation if dependencies prevent the compiler from scheduling a useful instruction in that slot. Implicitly, the TIE code also directs the logic-synthesis compiler to replicate function units or add ports to the register file as necessary, so the processor can execute each VLIW bundle in a single clock cycle. (In other cases, the TIE generator has the capability to define VLIW bundles that execute in multiple cycles.)

As with the previous optimization examples, programmers don't have to modify the application's C source code. XCC automatically uses the newly defined VLIW bundles to optimize the loop. Here is the disassembled code:

```
   loop:
   {addi    a9,a9,4;    add
a12,a10,a8; l32i a8,a9,0}
   {addi   a11,a11,4;   srai
a12,a12,2; l32i a10,a11,0}
   {addi   a13,a13,4;   nop;
s32i a12,a13,0}
```

Because the customized Xtensa processor can execute each of these VLIW bundles as a single-cycle pipelined instruction, each loop iteration now requires only three clock cycles. That's more than twice as fast as the unoptimized processor, which must execute eight instructions to grind its way through each pass of the loop.

Evaluating Options for Extensions

As shown by the examples, Tensilica's TIE generator can use three different techniques to accelerate substantially the same code in a program. It can also combine techniques, usually by pairing instruction fusion with SIMD or FLIX. But which technique or combination is best? The quandary grows when modifying a real application of any appreciable size, because the TIE generator could find thousands of code fragments to optimize, and it can explore thousands of different extensions.

Without human intervention, the TIE generator will use XCC's profiling information to automatically optimize those sections of the program it deems most important. Developers can exercise control over this process by choosing which parts of their programs to optimize and which optimization techniques to apply. Figure 1 shows a screen from the TIE generator in which developers can pick individual functions in a C program and specify options for the fusion, SIMD, and FLIX techniques.

There are three main factors to consider when evaluating custom extensions: performance, power consumption, and die area. Higher performance comes at the expense of more gates and power. Fewer gates save silicon and reduce power at the expense of performance. To make it easier for developers to balance those trade-offs, Tensilica provides a graphical tool that plots a range of automatically generated extensions according to their degree of acceleration and the additional gates they will add to the processor core. Developers can adjust these ranges to span the performance requirements and gate-count budgets of their projects.

Figure 2 is a sample screen display using this tool. The vertical scale plots the degree of acceleration from 1× to 9×, and the horizontal scale plots the number of gates from 0 to 75,000. Note that the graphed line shows only the tip of the iceberg—each data point represents the best performance for one automatically generated extension with a particular number of gates. Below the line are thousands of invisible data points representing other possible extensions that are worse by one measure or the other.

As Figure 2 shows, the number of additional logic gates required for a custom extension can vary over an extraordinary range—in this case, from 300 to 74,000. The reason is that some acceleration techniques are more logic-intensive than others, depending on the task being optimized. The instruction-fusion technique usually has the smallest effect on the gate count. Indeed, it's so efficient that every variation of the extension plotted in Figure 2 uses fusion in combination with other techniques. Lavish use of SIMD and FLIX rapidly inflates the gate count, because they must replicate arithmetic function units and load/store units to support parallel operations. Sometimes those techniques add new registers or new ports to existing registers, as well. Table 1 shows the numbers behind the data points plotted in Figure 2.

The fastest extension in this group is three times larger than the Xtensa V processor core it extends, illustrating how dramatically the gate count (and therefore the die area and power consumption) can rise if the developer's goal is maximum performance. It's tempting to blame the gate inflation on the TIE generator, which one might assume is no better at generating tight HDL code than some software compilers are at generating tight object code. Although that's probably a factor, huge custom extensions are not uncommon with configurable processors, even when they are designed by expert engineers. When Tensilica optimized Xtensa V for EEMBC's consumer benchmark suite last year, the beefed-up processor tipped the scales at 263,000 gates—more than 10 times the size of the base configuration.

In addition to consuming more power and silicon, larger configurations will have difficulty reaching higher clock frequencies, owing to the extra gate delays and longer (and more convoluted) signal paths. Remember, Xtensa V is synthesizable as well as configurable, so the automatically generated TIE code eventually passes through a logic compiler and a place-and-route tool before materializing as actual transistors and wires on a chip. From start to finish, there are multiple levels of design automation that introduce their own inefficiencies compared with a manually optimized processor and hand-packed circuit layout. The 25,000-gate Xtensa V base configuration could hit 400MHz when fabricated in a 0.13-micron CMOS process, whereas the 263,000-gate heavyweight that Tensilica designed for the EEMBC consumer suite was huffing and puffing to reach 260MHz.

Nevertheless, developers may be happy to trade gates—even tens of thousands of gates—for the benefits of a processor with application-specific extensions. At 260HMz, the 263,000-gate Xtensa V configuration racked up the highest EEMBC ConsumerMark score ever seen, surpassing even a 1GHz PowerPC processor. Higher performance at a lower clock frequency saves power, despite the extra gates. Moreover, in a 0.18- or 0.13-micron fabrication

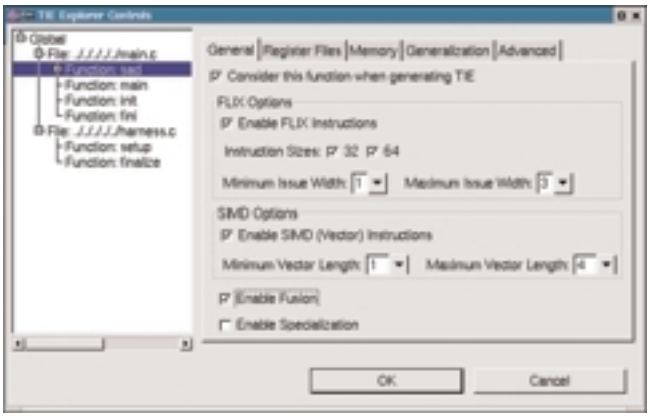


Figure 1. Tensilica's IDE allows developers to focus on critical functions of C/C++ programs and direct the TIE generator to apply various optimizations.

Figure 2. Tensilica's graphical evaluation tool shows how different variations of the same custom extension can achieve different degrees of acceleration, depending on the gate count. Developers can choose the extension that makes the most sense for their project.

process, a square millimeter of silicon is enough for 100,000 to 200,000 gates. On-chip memory and peripherals account for much more silicon in a typical SoC than the microprocessor core does—even when it's a configurable processor with large custom extensions.

To wring the last drop of performance and efficiency from a design, developers can manually tweak the automatically generated TIE extensions before synthesizing the processor. There's nothing hidden or special about the TIE code that the automatic tool emits. Likewise, there's nothing to stop developers from continuing to write their own extensions in TIE language, either instead of or in addition to the extensions created by the tool.

In any event, manual labor is still necessary when optimizing very complex tasks. The TIE generator can accelerate code fragments within a complex algorithm, but taking a holistic approach to large algorithms is as challenging for an HDL generator as it is for a software compiler. Except for that understandable limitation, it's difficult to find fault with Tensilica's new tools. Even if they work only half as well as Tensilica claims, they're still remarkable.

Rapid Evaluation Saves Time

It's easy to overlook the most important part of Tensilica's achievement. There's no doubt that custom extensions to a configurable processor can dramatically boost application performance; certified EEMBC benchmarks of manually optimized processors already prove that. And custom extensions will be more popular than ever before with a tool that can automatically generate them from high-level software code. What's more important, however, is the tool's ability to speculatively generate thousands of possible extensions, compare their relative performance, estimate their gate counts, and present the information to developers in a manner that's easy to understand.

Table 2 shows the effect of this technology on some common tasks. The TIE generator found enough ways to optimize a radix-4 fast Fourier transform (FFT) to improve performance by an order of magnitude. It generated more than 175,000 different configurations in only three minutes. Given half an hour, it discovered 1.8 million ways to accelerate an MPEG4 encoder. Although many of those configurations might be too trivial or inefficient for a skilled engineer to consider, any tool working that fast can afford to take an almost random approach. The tool's ability to sort the configurations by their vital parameters allows the best solutions to rise to the top.

In minutes, developers can now explore many more options than would be possible with months of manual design work and simulation. It's like making the jump from programming software with toggle switches to using assemblers, or from assemblers to high-level languages—except now it's the hardware (albeit "soft") making the leap.

This escalation of design automation will help developers accelerate their projects, not just their applications. As time-to-market pressures continue to build steam, the opportunity to cut a project's development cycle by weeks or months could be even more attractive than saving a few clock cycles in a program loop.

Configuration	Speedup	Gates Added	SIMD Width	FLIX Slots	Load/Store Units	Fusion
1	8.7x	74KG	8	3	2	Yes
2	8.1x	57KG	8	2	2	Yes
3	7.6x	46KG	4	3	2	Yes
4	7.6x	37KG	8	2	1	Yes
5	6.8x	33KG	4	2	2	Yes
6	6.8x	26KG	8	1	1	Yes
7	6.1x	18KG	4	2	1	Yes
8	5.1x	12KG	4	1	1	Yes
9	4.3x	8KG	2	2	1	Yes
10	3.4x	5KG	2	1	1	Yes
11	1.4x	0.3KG	1	1	1	Yes

Table 1. These 11 variations of an automatically generated custom extension span a performance range from 1.4× to 8.7× (40% to 770% acceleration over an unmodifed Xtensa processor) and add 300 to 74,000 gates. The Xtensa V base configuration is about 25,000 gates.

Tensilica's Strategy: Total Automation

The new TIE generator is another sign that Tensilica is striving to differentiate itself from other soft-IP providers by offering superior design automation. Since the very beginning of the company, the TIE language has been central to this strategy.

Why did Tensilica bother to create TIE from scratch instead of adopting industry-standard HDLs such as Verilog and VHDL for custom extensions, as competitors ARC and MIPS have done? One reason is that TIE is a "correct by design" language that keeps developers from writing extensions that won't work with the Xtensa processor core or might break it. Another reason is that TIE has the explicit semantics required to support other links in the tool

Application	Speedup	Configurations Evaluated	Time to Evaluate Configurations	Original Code Size Before Acceleration	Code Size After Acceleration	Code Size MIPS32
Radix-4 FFT	10.6x	175,796	3 minutes	1.5K	3.6K	4.4K
GSM Encoder	3.9x	576,722	15 minutes	17K	20K	38K
GSM Encoder (TIE FFT)	1.8x	n/a	n/a	17K	19K	38K
MPEG4 Encoder	3.0x	1,830,796	30 minutes	111K	136K	356K

Table 2. A salmon can lay 25,000 eggs to spawn a few offspring, but that's nothing compared to the number of configurations that Tensilica's TIE generator can create to speed up a program. By sorting the results with the evaluation tool shown in Figure 2, developers can ensure that only the fittest configurations survive.

Price & Availability

The automatic TIE generator will be available early next year as an extra-cost plug-in for Tensilica's Xtensa Xplorer IDE. Licensing fees for the Xtensa V processor with GNU software-development tools start at $350,000, plus royalties based on the volume of chips manufactured. The Xtensa C/C++ compiler, Xtensa instruction-set simulator, TIE generator, and TIE compiler are priced separately.

chain—not just the original links, such as the Processor Generator and adaptive software-development tools, but also new links, such as the automatic TIE generator.

It's no coincidence that Tensilica is filing most of its patents for the whole start-to-finish design system, especially the automated tools, instead of for the Xtensa microprocessor core. (See *MPR 12/9/02-01*, "Tensilica Patents Raise Eyebrows.") The Xtensa processor itself is competent but not breathtaking. It's an unconventional 32-bit architecture with a 16- and 24-bit instruction set that trades off some performance and programming convenience for code density. The 24-bit instructions don't align well on 32-bit memory boundaries, and they lack the encoding space for the large, flat register files typical of other RISC processors. Instead, Xtensa has a smaller set of revolving register windows, reminiscent of the SPARC architecture. But it matters little, because Xtensa is still as fast as the synthesizable competition, or even faster, and Tensilica's real product is the impressive design-automation system wrapped around the processor.

As time goes by, Tensilica begins looking more like an EDA (electronic design automation) company that also sells a configurable microprocessor core and less like a soft-IP company that provides some development tools. Perhaps we're not the only observers to notice this evolution. When an unknown party recently asked the U.S. Patent and Trademark Office to open one of Tensilica's patents for reexamination, the most likely suspects seemed to be direct competitors ARC and MIPS. However, it's possible that an EDA company is growing uncomfortable with Tensilica's encroachment on its turf and is seeking to narrow the scope of the patent's claims. (See *MPR 6/2/03-03*, "Tensilica Patent Challenged.")

Certainly, Tensilica is pushing into territory that has attracted a great deal of exploration by researchers at companies and universities all over the world. (See the sidebar with our Tensilica patents article at *MPR 12/9/02-01*, "Earlier Configurable Processors: Close, But No Cigar.") For decades, visionaries have pursued something like a unified field theory for computer design—a complementary unification of the separate design flows for hardware and software. Most research concentrates on software-directed hardware design, in which a program written in a high-level software language modifies or creates a processing machine optimized for the software. Of course, that's exactly what Tensilica's new tool does.

Hewlett-Packard is another company working on new approaches to hardware/software design automation. In 1999, HP Labs and STMicroelectronics announced their Lx system, which can automatically generate application-specific VLIW processors with compatible development tools, simulators, and real-time operating-system kernels. (See *MPR 1/24/00-03*, "HP and ST Collaborate on VLIW.") Only a trickle of information about Lx has been released, and the system isn't as broadly licensed as Tensilica's system.

Another HP Labs project in this field is PICO (Program In, Chip Out). But PICO isn't shipping, although HP recently licensed the technology to a new company, Synfora, which will continue developing PICO and eventually bring it to market. The PICO project suffered a setback last year when its primary inventor, HP engineer Bob Rau, passed away. (See *MPR 12/30/02-04*, "VLIW Pioneer Bob Rau Dies.")

Someday, an engineer will write a solution to a computing problem in a high-level language, and automated design tools will implement the solution with the best combination of hardware and software—"best" as defined by the engineer for that project. Instead of hardware engineers and software engineers, there will only be engineers. This may not happen for a long while, but Tensilica is taking an important step toward realizing that vision. ◇