Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

# TENSILICA MAKES MUSIC

## *New Extensions and Software Codecs Accelerate Digital Audio*

*By Tom R. Halfhill {9/29/03-01}*

Tensilica has introduced a new package of audio extensions and software codecs for its Xtensa V configurable microprocessor core. Known as the HiFi Audio Engine, the optional extensions include 54 new instructions that accelerate common algorithms for digital-audio encoding and decoding. Tensilica's target applications are portable MP3 players, automotive sound systems, TV set-top boxes, smart phones, and home entertainment systems.

To gain an advantage over audio processors that use other embedded-processor cores—especially the popular ARM-based chips from Cirrus Logic—the HiFi Audio Engine can process audio data with 24-bit precision instead of the more-common 16-bit precision. The difference in fidelity is audible to some listeners. Furthermore, by accelerating the audio algorithms with optimized instructions, Tensilica's extensions let developers run the processor at a lower clock frequency, thereby saving power. Alternatively, the processor could handle additional tasks, perhaps reducing the system's total chip count.

Instead of permanently appending the extensions to the Xtensa V architecture, which would inflate the core for all customers, Tensilica is offering the extensions to Xtensa licensees as an optional package of synthesizable intellectual property (IP). The first publicly announced customer is Solid State System of Taiwan (*www.3system.com.tw*), which specializes in flash-memory products.

### Extensions Designed by Cute Engineers

Tensilica assigned the task of developing the HiFi Audio Engine to CuTe Solutions, an independent design center in Hyderabad, India. The CuTe Solutions engineers have many years of experience in digital audio, so Tensilica left the specifications fairly open. Instead of specifying the instructions in advance and using CuTe Solutions merely to write the code, Tensilica relied on CuTe's experts to define the extensions as well as implement them.

CuTe's design team responded by creating 54 instructions, 23 registers, and 10 software encoders and decoders. The team wrote all the synthesizable logic in Tensilica Instruction Extension (TIE) language—a proprietary hardware-design language that resembles Verilog and VHDL. Customers licensing the HiFi Audio Engine receive the TIE source code, which they can integrate with the Xtensa V processor core and any other custom TIE extensions. (See *MPR 9/16/02-01*, "Tensilica Xtensa V Hits 350MHz.")

Software programmers can use the new instructions in assembly language or by calling intrinsic functions in C/C++, with Tensilica's own XCC tools or GNU tools. Invoking the instructions directly is unnecessary if programmers use the software codecs developed by CuTe Solutions. The codecs support popular audio standards, such as AAC (Advanced Audio Coding), AC-3 (Audio Compression-3), MP3 (MPEG-2 Layer 3), and WMA (Windows Media Audio). They also support the G.723-1 and G.729AB specifications for voice-over-IP (VoIP).

However, to develop applications with some of those codecs, Tensilica's customers must acquire additional licenses from the owners of the underlying software IP—companies such as Dolby Labs, Microsoft, and Thomson Multimedia. The additional licenses are a legal requirement for all resellers of that IP; they are not unique to Tensilica. CuTe Solutions

retains ownership of all IP related to the processor extensions, with Tensilica acting as the exclusive distributor.

As Table 1 shows, the 54 new instructions fall into six broad categories: loads and stores; multiply-accumulate (MAC) operations; shifts; general arithmetic operations; instructions specific to audio-related algorithms; and miscellaneous instructions. There are numerous variations and options not shown in the table, and information for some instructions was not available in time for *MPR*'s deadline. Programmers can choose the various instruction options by using special immediate fields and by writing values into some of the new registers.

One of the 23 new registers is the nine-bit cute_state register, whose individual bits control various options for the extensions. For example, bit 0 is the mac24 selector, which toggles between 16- and 24-bit fixed-point operations for all

| Instruction | Description | Comment |
|---|---|---|
| **Load/Store Instructions** | | |
| ld_d0_ux | Load from 32-bit data register d0 | 64-bit least significant word |
| ld_d1_ux | Load from 32-bit data register d1 | 64-bit most significant word |
| ld_c0_ux | Load from 32-bit coefficient reg c0 | 64-bit least significant word |
| ld_c1_ux | Load from 32-bit coefficient reg c1 | 64-bit most significant word |
| ld_acc_ux | Load accumulator | Lower 24 or upper 32 bits |
| st16_u | Store 16-bit value | Auto address update |
| st16_x | Store 16-bit value | Index address mode |
| st32_u | Store 32-bit value | Auto address update |
| st32_x | Store 32-bit value | Index address mode |
| st64_u | Store 64-bit value | Auto address update |
| st64_x | Store 64-bit value | Index address mode |
| stacc16_u | Store 16 bits from accumulator | Auto address update |
| stacc32_u | Store 32 bits from accumulator | Auto address update |
| **Multiply-Accumulate Instructions** | | |
| mac_d0c0_hh_ld | 24 x 24-bit MAC, registers d0, c0 | All MAC options [1] |
| mac_d0c0_ll_ld | 16 x 16-bit MAC, registers d0, c0 | All MAC options [1] |
| mac_d1c1_hh_ld | 24 x 24-bit MAC, registers d1, c1 | All MAC options [1] |
| mac_d1c1_ll_ld | 16 x 16-bit MAC, registers d1, c1 | All MAC options [1] |
| mac_d0c0_hl_ld | 24 x 16-bit MAC, registers d0, c0 | All MAC options [1] |
| mac_d0c0_lh_ld | 16 x 24-bit MAC, registers d0, c0 | All MAC options [1] |
| mac_d1c1_hl_ld | 24 x 16-bit MAC, registers d1, c1 | All MAC options [1] |
| mac_d1c1_lh_ld | 16 x 24-bit MAC, registers d1, c1 | All MAC options [1] |
| mac_d0c1_hl_ld | 24 x 16-bit MAC, registers d0, c1 | All MAC options [1] |
| mac_d0c1_lh_ld | 16 x 24-bit MAC, registers d0, c1 | All MAC options [1] |
| mac_d1c0_hl_ld | 24 x 16-bit MAC, registers d1, c0 | All MAC options [1] |
| mac_d1c0_lh_ld | 16 x 24-bit MAC, registers d1, c0 | All MAC options [1] |

| Instruction | Description | Comment |
|---|---|---|
| **Shift Instructions** | | |
| shift_d0l_c0l | Shift low d0 by low c0 | 16/24-bit shift |
| shift_d0h_c0h | Shift high d0 by high c0 | 16/24-bit shift |
| shift c1l_d1l | Shift low c1 by low d1 | 16/24-bit shift |
| shift_c1h_d1h | Shift high c1 by high d1 | 16/24-bit shift |
| shift_d0_c0 | Shift d0 by c0 | 32-bit shift |
| shift_c1_d1 | Shift c1 by d1 | 32-bit shift |
| shift_acc | Shift accumulator | 40/48/56-bit shift |
| **Arithmetic Instructions** | | |
| add_l_d0c0_c1 | Add low d0 + low c0, store in c1 | Optional subtract [2] |
| add_h_d0c0_c1 | Add high d0 + c0, store in c1 | Optional subtract [2] |
| add_l_d1c1_c0 | Add low d1 + low c1, store in c0 | Optional subtract [2] |
| add_h_d1c1_c0 | Add high d1 + high c1, store in c0 | Optional subtract [2] |
| add_d0c0_c1 | Add d0 +c0, store in c1 | Optional subtract [2] |
| add_d1c1_c0 | Add d1 + c1, store in c0 | Optional subtract [2] |
| divs_d0_c0 | Divide d0, c0 | — |
| divq_d0_c0 | Divide d0, c0 | — |
| **Algorithm-Specific Instructions** | | |
| bfly_d0c0 | FFT butterfly on registers d0, c0 | Do butterfly for FFT point |
| get1bit | Extract 1 bit from CMDW register | Huffman / bitstream data |
| ext_load_tr | Extract & load Huffman decode tree | Based on address register |
| huffdecode | Huffman table lookup | Hard-coded Huffman table |
| load_cmdw | Load from CMDW register | Current main data word |
| packstate | n/a | — |
| getstate | n/a | — |
| **Miscellaneous Instructions** | | |
| clracc | Clear accumulator | — |
| absacc | Get absolute value of accumulator | — |
| rndacc | Round accumulator | — |
| readacc | Read unsigned value in accumulator | — |
| writeacc | Write unsigned value in accumulator | — |
| clrovf | Clear overflow flag | MAC & math overflows |
| max_d0c0 | Find maximum among four values | Signed or unsigned |

**Table 1.** Tensilica's HiFi Audio Engine has an impressive library of new instructions with many flexible options. [1]MAC options include 16- and 24-bit fixed-point multiplication, multiplication and addition, and multiplication and subtraction; 40- or 56-bit accumulation; integer and fractional arithmetic; signed and unsigned values; a square mode; overflow checking and saturation; rounding; and single or dual loads. [2]Setting or clearing the add_sub flag (with an immediate operand) determines whether the instruction adds or subtracts the input operands. n/a: information was not available at press time.

MAC, arithmetic, load/store, and shift instructions. Bit 1 is the mmode selector, which toggles between integer and fractional arithmetic.

Two of the new registers are 16 bits wide; they store index values for Huffman bitstream decoding. One new register is 24 bits wide; it stores the lower 24 bits of the accumulator. (The upper accumulator register is 32 bits wide, which allows 40- or 56-bit accumulation.) All other registers are 32 bits wide, including four data registers and four coefficient registers. The data and coefficient registers are paired, so two of them can hold the lower and upper words of extended values beyond 32 bits. Table 2 summarizes the new registers and their functions.

## MAC the Knife

All the new instructions are useful, but the MAC instructions are the most versatile. There are 12 in all, each with a different combination of input and output registers. Programmers can use hundreds of possible variations by toggling bits in the cute_state register to select the options listed in the Table 1 footnotes. The two-cycle MAC instructions are intelligently matched with dual load/store instructions that can read or write two 32-bit values or two signed 24-bit values in a single clock cycle. By pipelining the loads, MACs, and stores, an Xtensa V processor can execute a 24-bit MAC on every other cycle.

The option to use 24-bit precision with the MACs and other instructions gives Tensilica a selling point over competing processors having less precision. It's not just marketing hype. Many listeners insist they can hear the difference with high-quality 320Kb/s MP3 files and CD-quality music, even when those files were recorded with 16-bit precision. The most popular competitors—such as Cirrus Logic's ARM-based audio processors—are limited to 16-bit precision. SigmaTel's D-Major MP3 decoders have 18-bit precision. In 2001, Cirrus Logic announced the EP7409 audio chip, which was supposed to have a 24-bit DSP core in addition to a 32-bit ARM720T processor, but it never shipped. (See *MPR 9/10/01-02*, "Ripping Good Audio Optimization.") For non-portable applications, Cirrus Logic does make higher-end audio processors that offer better precision, including some chips with 32-bit DSP cores.

Most of Tensilica's new instructions are fairly self-explanatory, but the numerous options are less obvious. For instance, the load/store instructions can read or write single or dual 32-bit operands while automatically incrementing the memory address or using indexed addressing. The shift instructions can operate on 16-, 24-, 32-, 40-, 48-, or 56-bit signed or unsigned operands, with or without saturation, while checking for overflows. Programmers can specify the shift amount by referencing a register or using an immediate value.

One oddity is that all instructions whose mnemonics begin with add can also *subtract* if the programmer changes the newly defined add_sub flag. A one-bit immediate field toggles the flag, effectively reversing the function of the instruction. It seems that an add_sub mnemonic would make the source code more understandable—but then, programmers accustomed to inverting the meaning of an if statement in C by adding an exclamation point (shorthand for NOT) probably won't find anything strange about using an add instruction to subtract.

## Butterflies and Huffman Trees

Some of the algorithm-specific and miscellaneous instructions require more description. For instance, the max_d0c0 instruction can find the maximum value among four values in the low-and-high data and coefficient registers. The values in the registers can be signed or unsigned. When used iteratively, this instruction might be useful for finding the maximum amplitude of a waveform before compressing the data.

The huffdecode instruction accelerates data decompression by returning the Huffman code for the input operand. Huffman compression is based on a binary tree, and decompressing the data requires numerous table lookups. If the lookup table is stored in off-chip memory, the processor spends a good deal of time fetching data over the I/O bus. The huffdecode instruction eliminates that penalty because the lookup table is, in effect, hard-wired into the function unit. A related instruction, ext_load_tr, extracts and loads values from the Huffman tree while performing some address computations.

Another interesting instruction is bfly_d0c0, which performs a sequence of arithmetic operations known as a butterfly for each point in a fast Fourier transform (FFT) algorithm. The number of butterflies varies with the number of points (butterflies = points/2 * $\log^2$(points)), so a 1,024-point FFT requires 5,120 butterflies, or about 20,000 add, subtract, and multiply operations. The bfly_d0c0 instruction performs

| Register | Width | Description |
|---|---|---|
| cute_state | 9 bits | Toggles nine instruction options |
| cute_d0_l | 32 bits | Data register 0, low word |
| cute_d0_h | 32 bits | Data register 0, high word |
| cute_d1_l | 32 bits | Data register 1, low word |
| cute_d1_h | 32 bits | Data register 1, high word |
| cute_c0_l | 32 bits | Coefficient register 0, low word |
| cute_c0_h | 32 bits | Coefficient register 0, high word |
| cute_c1_l | 32 bits | Coefficient register 1, low word |
| cute_c1_h | 32 bits | Coefficient register 1, high word |
| cute_acclo | 24 bits | Accumulator, lower 24 bits |
| cute_acchi | 32 bits | Accumulator, upper 32 bits |
| cute_bidx | 16 bits | Bit index for Huffman/bitstream |
| cute_widx | 16 bits | Word index for Huffman/bitstream |
| cute_cmdw | 32 bits | Current main data word |
| cute_tr_data | 32 bits | Huffman tree data |

**Table 2.** A major advantage of configurable microprocessor cores is that developers can add new instructions and registers to those available in the standard architecture. Tensilica's Xtensa processors use revolving register windows—much like those in Sun's SPARC architecture—to expand the register map beyond that allowed by the number of address bits in the 16- and 24-bit instructions.

### Price & Availability

Tensilica's Xtensa HiFi Audio Engine is available now. The up-front licensing fee starts at $50,000, not including chip royalties or an Xtensa V microprocessor license. The engine comes with software codecs for popular audio standards, but licensees must acquire separate licenses from the owners of that IP. For more information about Tensilica, visit www.tensilica.com. For more information about CuTe Solutions, visit www.cutesolinc.com.

the same operations with only two instructions per point. It's useful for audio encoders and other signal-processing tasks.

Tensilica is an active member of EEMBC and has dutifully released benchmark scores for its processors in the past. We hope Tensilica will test its new audio extensions with the soon-to-be-released EEMBC Version 2 consumer suite, which includes MP3 decoding.

#### Better Late Than Never

Ideally, Tensilica would have introduced the HiFi Audio Engine four years ago, when MP3 players were just starting to take off and early design wins were as plentiful as dot-coms. That was when Cirrus Logic introduced its innovative EP7209 and EP7212 Maverick audio processors, which anticipated one of the few consumer markets that has grown at a *prestissimo* tempo since 1999. (See MPR 11/15/99-03, "Cirrus Logic Makes Music With ARM.")

At that time, In-Stat/MDR analyst Mike Paxton predicted that sales of portable Internet-audio products would grow from 850,000 units in 1999 to 9.6 million units in 2003. Despite dismal economic conditions that seemed unimaginable in 1999, sales will almost certainly surpass his optimistic forecast. According to analyst Cindy Wolf—who now tracks the digital-audio market for In-Stat/MDR—consumers bought 6.8 million MP3-capable players in 2002 and will buy another 12.5 million units this year. Those sales include all players capable of handling MP3 files on any media, such as Apple's popular iPod (which stores files on an internal 1.8-inch hard-disk drive) and CD players that support MP3-format discs. Sales of players that store MP3 files in solid-state memory (such as flash) surpassed two million units in 2002 and are expected to exceed three million units this year. Although average selling prices are dropping, some MP3 players (such as Apple's high-end iPod) still cost as much as $400, relatively high for portable audio products.

All this market data adds up to a healthy environment for audio processors, even before considering other applications that use digital audio. Certainly, the unit volumes and revenues are large enough to justify the development of an SoC. Tensilica's HiFi Audio Engine adds only about 50,000 gates to the Xtensa V core—relatively little silicon for such a useful supplement to the standard instruction set.

The most threatening competition will not necessarily come from other licensable-IP cores but from digital-audio ASSPs. Cirrus Logic's latest chips for this market are the EP7312 (designed for solid-state MP3 players) and the CS7410 (designed for CD-based MP3/WMA players). Both are available off the shelf for about $11 and $5, respectively, so product developers don't have to spend many months and millions of dollars on an SoC project.

More competition is coming soon from Philips Semiconductor, which has announced the SAA7752 digital-audio processor for MP3-capable CD players. MP3 CD players—the fastest-growing MP3 category—already cost well under $100. Rapidly falling prices that squeeze profits add more uncertainty to a long-term SoC project, making standard parts seem more attractive.

In Tensilica's favor, an SoC built around the Xtensa V configurable processor can be more optimized than an ASSP designed for a broad market and could provide better audio processing (24-bit precision vs. 16-bit) than comparable chips. In large enough volumes, an Xtensa-based audio processor could cost less, too. All things considered, Tensilica's HiFi Audio Engine offers developers a valuable alternative to existing solutions.