

FPGAs “DiSP”lay THEIR PROCESSING PROWESS

DOES THE PERFORMANCE-POWER-PRICE PRODUCT OF YOUR SOFTWARE-CENTRIC APPROACH NO LONGER COMPUTE? DO YOU NEED A NIMBLER PLATFORM THAN A HARDWIRED ASIC CAN PROVIDE? PROGRAMMABLE LOGIC MAY BE YOUR ANSWER, BUT CAREFULLY CALCULATE THE TRADE-OFFS TO CORRECTLY SOLVE YOUR PROBLEM.

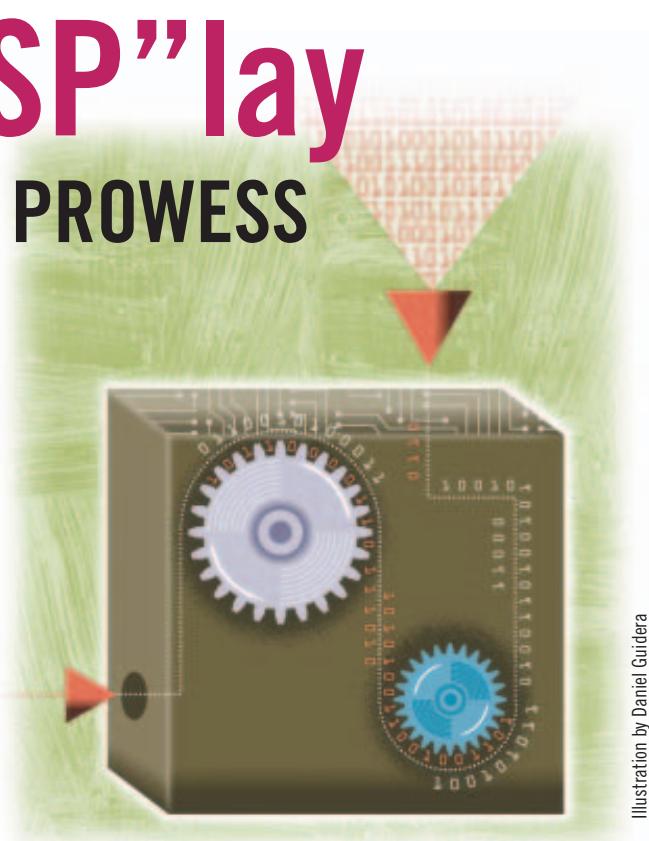


Illustration by Daniel Guidera

MENTION “DSP” IN CONVERSATION around the water cooler, and what vision will pop into other folks’ heads? Probably a picture of a piece of silicon from a company such as Analog Devices, Motorola, or Texas Instruments. This image isn’t necessarily wrong, mind you, but it’s a bit like (as the old saying goes) “putting the cart before the horse.” First and foremost, DSP stands for digital-signal

processing; that is, converting analog signals to the digital domain, arithmetically transforming them in some way and then translating them back to the analog domain for human sensory consumption.

Digital-signal *processors* from the aforementioned companies and many others are only one vehicle for implementing digital-signal-*processing* functions. The earliest DSP chips, after all, were little more than otherwise general-purpose CPUs with Harvard architectures (separate instruction and data buses and separate caches), befitting their algorithms’ data-centric nature. As general-purpose CPUs have grown speedier and particularly as they’ve added on-

board arithmetic coprocessors and signal-processing-optimized instruction-set extensions, they’re increasingly taking over the calculation burden that might have formerly required a separate compute engine. (This article uses “digital-signal processing” to refer to the function and “DSP” to refer to the processor.)

On the other end of the hardware-versus-software-implementation spectrum are ASICs, housing hard-wired arithmetic logic blocks and state machines. Inflexible? Yes. Do you need to design the hardware yourself? Yes, unless you license predesigned IP (intellectual property), which you must still stitch together with the remainder of your chips’ circuits. But

At a glance.....62
Consult with an expert64
For more information68

fast, low-power, and inexpensive (the cost based on the amount of silicon consumed to construct the function)? Yes. The ASIC-based approach (note, again, we're talking about hardwired functions, not a processor core you've integrated into your ASIC) is particularly attractive when your end system will sell in high enough volumes to justify the NRE (nonrecurring-engineering) costs, when time-to-market isn't critical, and when standardization and design experience maximize your first-silicon-functional confidence and preclude the need for post-sale upgrades.

In attempting to simultaneously stay ahead of general-purpose CPUs and prevent you from jumping ship to hardwired ASICs, the DSP vendors have evolved their high-end architectures into multicore VLIW (very-long-instruction-

AT A GLANCE

- ▶ FPGAs' sweet spot straddles software's flexibility and hardware's thriftiness, low power, and performance.
- ▶ Latest generation programmable-logic architectures embed arithmetic blocks for increased acceleration.
- ▶ Silicon strengths strapped by software shortcomings will leave you unsatisfied.
- ▶ Custom processors ease the software-to-hardware transition.

word) "engines" and have incorporated their own hardware-acceleration capabilities in the form of dedicated Viterbi decoders, matrix multipliers, and the like. Although signal-processing functions contain a great deal of parallelism, the incremental performance gain with each added engine is less than 100%, and multiprocessors are challenging to program. Hardware acceleration also tends to make the resultant DSP more application-specific and, therefore, more expensive than a general-purpose alternative.

SILICON FOUNDATIONS

Digital-signal processing, thanks to explosive growth in wired and wireless networks and in multimedia, represents one of the hottest areas in electronics. So it's no surprise that dozens if not hundreds of stand-alone-chip and embedded-core vendors are chasing after the business, representing both the software- and the hardware-centric implementation extremes. But at least one in-between option bears your consideration (see sidebar "Theme and variation" on the Web

version of this article at www.ednmag.com). Like software, a programmable-logic device is almost infinitely customizable, and, as with a processor, the silicon physical-design work is already done for you. FPGAs aren't quite as low-power, fast, or dense as ASICs, but they're superior to processors in those regards (see sidebar "Evaluating performance: FPGAs versus DSPs" on the Web version of this article at www.ednmag.com). You can buy FPGAs, unlike ASICs, in small quantities with no upfront NRE charges, and you need not wait for months' worth of fab, packaging, and test delays after your design's done to obtain a working chip.

FPGA manufacturers have for years now been trumpeting their chips' ability to implement digital-signal processing, even before the emergence of low-latency carry-chain-routing lines that sped addition and subtraction operations spanning multiple logic blocks. The next significant improvement in FPGA arithmetic capability appeared with Atmel's AT40K architecture. An embedded AND gate within each logic block, working in concert with block-to-block diagonal routing lines, boosted performance when the chips were crunching array-multiplication calculations (Figure 1). Ironically, however, AT40K provides no carry chains.

Atmel's FPGAs are partially reprogrammable. (Lattice's ORCA line and Xilinx's Virtex devices are also reprogrammable, but their development tools do not neatly expose the silicon potential.) Theoretically, this capability means that you could dynamically time-swap various logic engines into a common silicon fabric. Pragmatically, the more likely scenario involves your ability to, for example, optimize an imaging filter's coefficients in a digital-still-camera or videocamera application as ambient-light conditions change or tweak processing

(continued on pg 66)

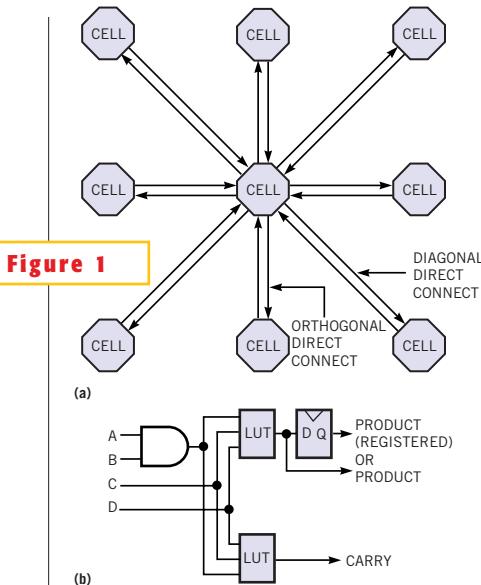


Figure 1

Diagonal routing (a) combines with an AND-enhanced logic block (b) to accelerate array multiplication operations (courtesy Atmel).

TABLE 1—VENDORS, PRODUCTS, AND ARITHMETIC-STRUCTURE SPECIFICATIONS

Company	Structure	Products and structure-count ranges	Features
Altera	DSP block	Stratix EP1S (six to 28)	Dedicated 18×18-bit multiplier, pipeline, and accumulation circuitry; predictable 250-MHz performance, which provides maximum data-throughput performance of 2 GMACS (giga multiply-accumulate operations per second) per DSP block.
QuickLogic	ECU (embedded computational unit)	Eclipse Plus (10 to 18)	Integrated multiply, add, and accumulate functions; 8-bit multiplier, 16-bit accumulator 2.6 billion MACs/sec operations using the ECU, and you can implement additional multiply-accumulate functions in the programmable logic for 2 billion MACs/sec more, when clocked at 100 MHz.
Xilinx	Multiplier	Virtex-II (four to 168) Virtex-II Pro (12 to 566)	18×18 bit multipliers support as many as 18-bit signed or 17-bit unsigned representations, with cascading to support bigger numbers; the multipliers can be fully combinatorial or pipelined, running at more than 300 MHz.

CONSULT WITH AN EXPERT

By Ray Andraka, Andraka Consulting Group

Designing digital-signal-processing hardware is quite different from designing for software-based systems. In addition to the obvious considerations for hardware, such as clocking, timing and so on, a hardware digital-signal-processing designer also has to consider how an algorithm will map to hardware, as well as the availability of design resources.

Before beginning the digital-signal-processing design, a designer should determine the appropriateness of the approach. Generally, if a single DSP or microprocessor provides enough horsepower to accomplish the task at hand, the designer should use it instead of an FPGA-based option. The tools are more mature, talent is cheaper and easier to find, and most of the algorithms currently in use were developed for software platforms.

If, however, the required performance is greater than a single DSP or a microprocessor can achieve, the performance gains that an FPGA presents outweigh the costs. Typical FPGA performance gains over a single DSP are around 100 times, and gains of more than 1000 times are possible under some circumstances. Other considerations can also tip the balance in favor of an FPGA. Power dissipation of a well-executed FPGA design, for example, is typically about 20% of the power consumption of a software-based system operating at the same sample rate.

Programmatic issues, such as the cost of software validation in certain systems, can also tip the scale toward FPGAs. Most systems include both an FPGA and a DSP. In these cases, it is usually best to delegate the "inner loop" to the FPGA and leave the housekeeping and other odds and ends to the microprocessor.

Not all devices are equal for

digital-signal processing. The FPGA needs to have robust arithmetic capabilities, which, until the introduction of Altera's Stratix, heavily favored the Xilinx architectures. Newer devices have fast multipliers that ease the transition to hardware for those of you unfamiliar with hardware digital-signal. A multiplier may not be a panacea, though, because, in most cases, a design lacks sufficient multipliers for you to use them indiscriminately.

The key to designing digital-signal processing in FPGAs is thinking in terms of what the hardware that performs the function must look like. Algorithms for software implementations tend to be heavy on multiplication as well as floating-point operations. Often, small changes in the algorithm or use of alternative approaches can lead to greatly simplified hardware. You can significantly reduce area just by switching from floating-point to fixed-point arithmetic or to a modified floating-point format that uses only minimal bits to represent the signal at given points in the algorithm.

Vendors are offering high-level digital-signal-processing design tools that make the entry into FPGA-based digital-signal processing easier for newcomers. These tools include plug-ins for scientific packages, such as Matlab. These plug-ins include the Xilinx System Generator, as well as high-level-synthesis tools, such as those from Celoxica.

Although these tools sufficiently lower the bar for putting algorithms into hardware for systems designers who are not hardware-savvy, the occupied-area and performance results are bound to be disappointing. A seasoned hardware designer is likely to find the tools frustrating because it is difficult to use

them with intellectual property in the tool's rather limited library.

A designer should simulate the algorithm at the system level before executing the detailed hardware design to verify the selected precision at each point in the design and to iron out any bugs in the algorithm. It is far easier and faster to sort out these issues in a high-level model than at a gate-level simulation. Doing a bit-true system-level simulation in C or The MathWorks' Matlab also has the advantage of providing high-fidelity test vectors that you can later use to verify the proper operation of the digital design, once you complete the detailed design. System-level simulations also allow you to explore alternative algorithms in search of one that better maps into hardware.

Digital-signal processing in hardware was around long before the advent of the microprocessor. It dates to the late 1950s, and, for nearly 30 years, you could do digital-signal processing *only* with custom hardware. Because of the high cost of digital logic in those days, developers created algorithms and techniques to handle the processing with a minimum of hardware. These algorithms are just as good today as when they were developed, and they're appropriate for a hardware-digital-signal-processing design because they can offer significant system savings and power-dissipation advantages over designs that you base on an algorithm developed for software. Technical libraries are loaded with these hardware gems, which are buried deep in 20 years' worth of dust, waiting for an astute hardware engineer to resurrect them. Even if you can't find a suitable algorithm, the knowledge that early work

embodies can serve as a useful guide for making your algorithm more hardware-friendly.

After selecting the algorithm and data precisions, the designer should look at the clock cycles available per sample to determine the architecture of the hardware. To get the most bang for the buck, strive for as many clock cycles per sample as possible. Current FPGAs can run at clock rates well beyond 150 MHz, yet most digital-signal-processing applications are still running at sample rates of less than 20 MHz. In these cases, running a faster master clock provides the capability for bit- or digit-serial processing at a considerable savings in logic area and, subsequently, in device cost.

Faster clocks also open the door for implementing certain algorithms with repetitive operations in iterative hardware. A good example of this technique is using a scaling accumulator over several clock cycles in place of a full parallel multiplier. The block diagram that falls out of this step is the blueprint for the rest of the design effort, which is essentially the same as any other digital-hardware-design effort.

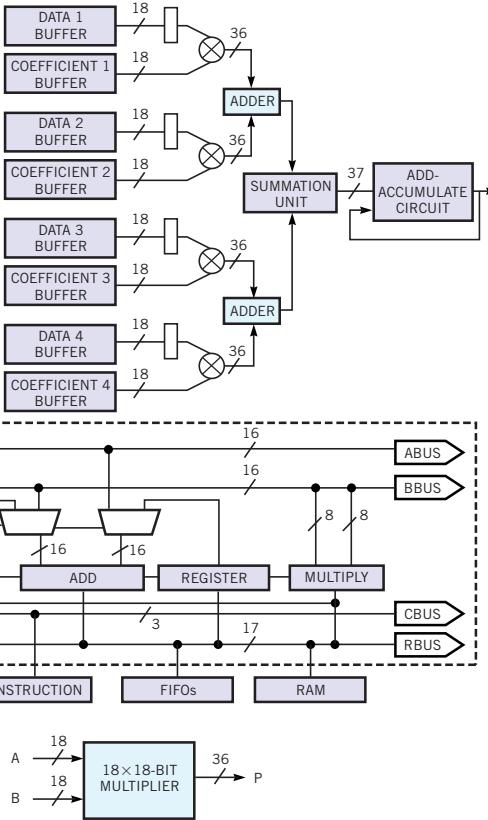
AUTHOR'S BIOGRAPHY
FPGA guru Ray Andraka, founder of Andraka Consulting (North Kingstown, RI) has been designing digital-signal-processing functions into programmable logic since 1988, even before the days when carry chains were considered a state-of-the-art arithmetic innovation. He's an enthusiastic participant on comp.arch.fpga, comp.dsp, and other relevant Internet newsgroups, as well as at industry events such as the ACM (Adaptive Computing Machine) FPGA conference, and his Web site offers lots of free and useful information.

(continued from pg 62)

parameters in response to varying communication-channel SNRs.

The now-dominant LUT (look-up-table)-plus-register-logic-block combination, in combination with fast carry chains, is relatively efficient when implementing addition and subtraction operations. It's not, however, optimal in cost, performance, and power for multiplication and division functions. As a result, Altera (with Stratix), QuickLogic (with QuickDSP, now renamed Eclipse Plus) and Xilinx (with Virtex-II and Virtex-II Pro) have all taken a page from the ASIC book of tricks and embedded dedicated multiplier-function blocks on-chip (Figure 2). Altera and QuickLogic move even further along the integration path, providing full-blown MACs (multiply-accumulators, see Table 1 for specifications and Reference 1 for prices). Altera calls its version the DSP block (Table 2); QuickLogic—the first of the three to take the dedicated-arithmetic-unit-integration plunge—refers to its configurable variant as an embedded computational unit (Table 3).

In examining Altera's and Xilinx's arithmetic structures, you might question why the companies chose nonstan-



Dedicated arithmetic structures from Altera (a), QuickLogic (b) and Xilinx (c) focus on DSP functions.

dard 18-bit data inputs versus the more common 16-, 24-, and 32-bit lengths. One answer is that they wanted to match the bus widths of the FPGA's parity-inclusive embedded-RAM blocks. But a more general answer also exists, and it leads to a subtle but powerful FPGA strength. A sig-

nal-processing function rarely demands exactly 16-, 24-, or 32-bit precision. It requires less if your CPU or DSP is wasting pins, external memory, and bus bandwidth. If more, you have to implement performance-sapping multiple-pass algorithms.

With an FPGA, particularly if you use general-purpose LUT and register structures, you can implement exactly the data precision you need to do the job—at least from a logic standpoint. Memory is the only remaining wrinkle; both the large embedded-memory blocks and the external memories come in predefined bus widths. FPGAs that can alternatively employ LUTs not only for logic functions, but also as small RAM arrays, such as Xilinx's various product families and Lattice's ORCA chips, are helpful here, because they give you density and bus-width flexibility to supplement or replace dedicated RAM arrays. LUTs have another valuable function in calculation-intensive signal processing: They are a more efficient alternative to registers for storing intermediary values.

DESIGN CONTORTIONS

Alas, the semiconductor graveyard is littered with the bones of great FPGA hardware ideas that went that by the way-

TABLE 2—ALTERA DSP-BLOCK-CONFIGURATION OPTIONS AND FEATURES

Operation modes of the DSP block	9×9 bits	18×18 bits	36×36 bits
DSP-block mode			
Simple multiplier	Eight multipliers with eight product outputs	Four multipliers with four product outputs	One Multiplier with one product output
Multiply-accumulator	Two multiply and accumulates (34 bit)	Two 52-bit multiply and accumulates	NA
Two-multipliers adder	Four sums of two multiplier products each	Two sums of two multiplier products each	NA
Four-multipliers adder	Two sums of four multiplier products each	One sums of four multiplier products each	NA
DSP-block features			
Feature	Benefit		
Multiplier	<ul style="list-style-type: none"> ● 9×9-, 18×18-, and 36×36-bit multiplication ● Floating-point arithmetic ● Signed and unsigned operation ● Full precision in all modes ● Optional shift-register chain on inputs 		
Adder/subtractor/accumulator	<ul style="list-style-type: none"> ● Dynamic switching between adder and subtracter ● 9-, 18-, or 36-bit operation for adder and subtracter ● 52-bit accumulator ● Signed and unsigned operation 		
Summation unit	<ul style="list-style-type: none"> ● Summation of as many as four products in one clock cycle 		
Complex shift function	<ul style="list-style-type: none"> ● Barrel shifter, crossbar switch, and FIR filter 		

side because of inadequate design-software support. How can you ensure that your signal-processing algorithms take full advantage of the power, performance, and efficiency of the embedded circuitry within these advanced chips? The answer depends at least to some extent on your design background.

If you're used to creating hard-wired circuits in ASICs, you'll be treading the easier of the two paths to FPGA nirvana. In a perfect world, the combination of a third-party synthesis compiler and the FPGA-vendor-provided place-and-route software *should* be able to automatically infer from your HDL code where to use specialized multipliers and MACs, much as they *should* do with embedded memory arrays and other on-chip function blocks (references 2 and 3). The vendors' documentation often makes recommendations on coding styles, which help guide the design software to the ideal end result (Reference 4).

In the real world, you still sometimes need to explicitly instantiate references to these and other device primitives in your HDL, a task that leaves your code less architecture- and device-generic, thereby contradicting a key motivation for using an HDL in the first place. The vendors have all developed pushbutton utilities that greatly simplify your creation of higher level functions, such as filters and transforms. Enter a few parameters, click "OK," and out comes a netlist "black box," which your HDL code references. At minimum, you have the option of creating a fully serial (to minimize pin count and logic usage) or fully parallel (to maximize performance) circuit. Depending on the size of your design and device and on your performance and power bud-

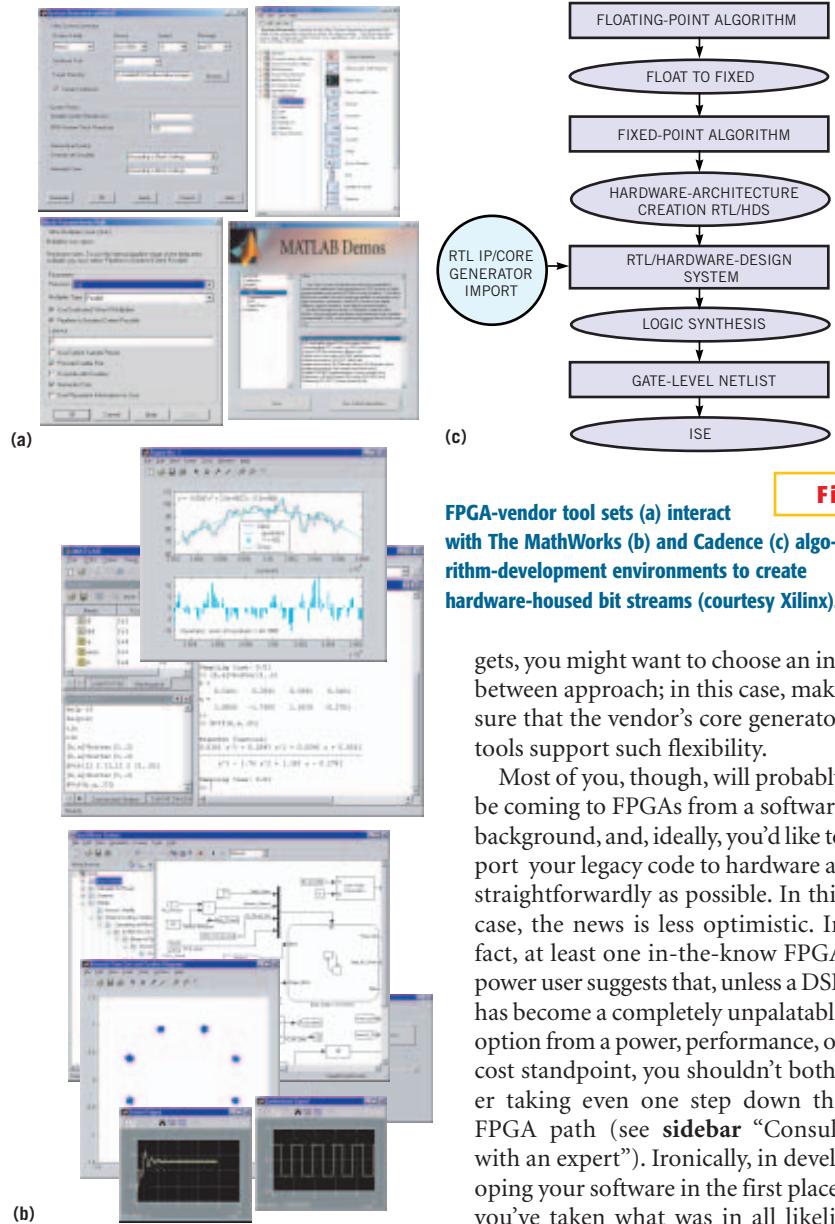


Figure 3

FPGA-vendor tool sets (a) interact with The MathWorks (b) and Cadence (c) algorithm-development environments to create hardware-housed bit streams (courtesy Xilinx).

gets, you might want to choose an in-between approach; in this case, make sure that the vendor's core generator tools support such flexibility.

Most of you, though, will probably be coming to FPGAs from a software background, and, ideally, you'd like to port your legacy code to hardware as straightforwardly as possible. In this case, the news is less optimistic. In fact, at least one in-the-know FPGA power user suggests that, unless a DSP has become a completely unpalatable option from a power, performance, or cost standpoint, you shouldn't bother taking even one step down the FPGA path (see sidebar "Consult with an expert"). Ironically, in developing your software in the first place, you've taken what was in all likeli-

FOR MORE INFORMATION...

For more information on products such as those discussed in this article, go to www.edn.com/info and enter the reader service number. When you contact any of the following manufacturers directly, please let them know you read about their products in *EDN*.

Altera
1-408-544-7000
www.altera.com
Enter No. 301

Atmel
1-408-441-0311
www.atmel.com
Enter No. 302

QuickLogic
1-408-990-4000
www.quicklogic.com
Enter No. 303

Triscend
1-650-968-8668
www.triscend.com
Enter No. 304

Xilinx
1-408-559-7778
www.xilinx.com
Enter No. 305

OTHER COMPANIES MENTIONED IN THIS ARTICLE
3DSP
www.3dsp.com

Actel
www.actel.com
Adelante Technologies
www.adelantetechnologies.com

Analog Devices
www.analog.com
Andraka Consulting Group
www.andraka.com

Berkeley Design Technology Inc
www.bdti.com

Cadence
www.cadence.com

Celoxica
www.celoxica.com
Improv Systems
www.improvsys.com

Leopard Logic
www.leopardlogic.com
The MathWorks
www.mathworks.com

Motorola
www.motorola.com

QuickSilver Technology
www.qstech.com

Synopsys
www.synopsys.com
Tensilica
www.tensilica.com

Texas Instruments
www.ti.com

SUPER INFO NUMBER

For more information on the products available from all of the vendors listed in this box, enter no. 306 at www.edn.com/info.

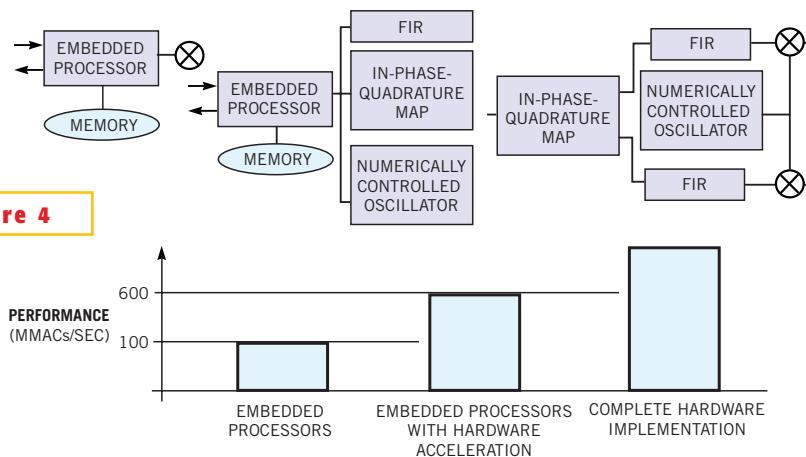
hood a highly parallelizable function and gone through a lot of work to convert it to a time-sequential serial algorithm in C or assembly code. Now, to port the algorithm to an FPGA, you need to reparallelize it and, sometimes, convert it from floating-point back to sufficiently-precise fixed-point arithmetic.

Companies such as Adelante Technologies (formerly, Frontier Design, with A|RT), Celoxica (with Handel-C), and Synopsys (the original developer of SystemC) all claim the ability to generate hardware designs from C code. And, to a degree, they all deliver on their claims. None of them, though, fully supports generic ANSI C code. (Pointers are particularly problematic.) Their languages are tailored to the task of hardware creation. So, although they might be acceptable for brand-new designs, they will be less help with your years' worth of existing software.

The designs created by C-to-hardware tools are less efficient than those you develop from the beginning in an HDL (an analogy to programming in C versus assembly language), but, thanks to the near-guaranteed performance boost in software-to-hardware conversion, lost efficiency may be a small concern. If you've developed your software algorithms in The MathWorks' Matlab, you *might* be in better luck. Both Altera and Xilinx offer tool sets that interface to Matlab and Simulink and output vendor-optimized hardware code and IP blocks. Xilinx also supports Cadence's SPW (Signal Processing Worksystem, **Figure 3**).

INTERMEDIATE APPROACHES

Although we're talking about digital-signal processing, you shouldn't assume that you've got at your disposal a "binary" set of implementation options—either fully software or fully hardware. The spectrum of choices is, in reality, far more "analog," reflecting a possible partitioning of tasks among both hardware for optimum speed, power consumption, or per-unit cost and software for ease of development and legacy compatibility (**Figure 4**). Both Atmel and Xilinx, for example, have, with their respective FPSLIC and Virtex-II Pro product lines, combined arithmetic-tuned programmable logic and "hard" CPU cores on a single device. If you develop your signal-processing algorithm in Matlab and Simulink, you can iteratively direct portions of



Soft CPU customization enables partitioning between hardware and software (courtesy Altera).

it to C code and the remainder to FPGA hardware. Otherwise, don't waste too much time on tedious hardware-versus-software partitioning and repartitioning; focus your efforts on obvious software bottlenecks that benefit from FPGA acceleration.

Altera's ARM-based Excalibur chips and QuickLogic's QuickMIPS line do not include the hardware MACs in the vendors' respective Stratix and Eclipse Plus products. However, years' worth of successful design examples suggest that, although MACs might speed signal processing, they aren't an absolute requirement. Embedded memory blocks, for example, can also find use as multipliers. For the same reason, don't forget about Triscend's A7 and E5 chips. If you do need dedicated arithmetic circuits to hit your hardware-design targets, a "soft" CPU core might conversely provide you with sufficient software capabilities. Altera's Nios fits inside Stratix, and Xilinx's MicroBlaze works with Virtex-II and Virtex-II Pro. Supplementing Virtex-II's "hard" PowerPC cores with one or multiple "soft" MicroBlaze cores results in some interesting single-chip, multi-processor-architecture possibilities.

Look, for example, at Altera's recently announced DSP-development kit, an expanded version of last year's DSP Builder tool. Working hand in hand with Matlab and Simulink, it enables you to develop custom instruction-set and hardware-accelerated variants of the Nios processor that tap into the MACs and other resources in the programmable-logic fabric. Altera claims to have more than 60

DSP cores in its IP library, spanning specific functions, such as encryption, error correction, image processing, and modulation, as well as general-purpose functions, such as filters and transforms. □

REFERENCES

1. Dipert, Brian, "EDN's third annual programmable-logic directory," *EDN*, Sept 5, 2002, pg 44.
2. Dipert, Brian, "Lies, damn lies, and benchmarks: The race for the truth is on," *EDN*, May 27, 1999, pg 54.
3. Dipert, Brian, "Synthesis shoot-out at the EDN corral," *EDN*, Sept 11, 1998, pg 95.
4. Dipert, Brian, "Getting a handle on HDLs," *EDN*, May 7, 1998, pg 71.

ACKNOWLEDGMENTS

Kudos to Jeff Bier from Berkeley Design Technology and to Ray Andraka from the Andraka Consulting Group for their editorial contributions.

AUTHOR'S BIOGRAPHY

Technical Editor Brian Dipert wonders how long it will be before it's possible to squeeze all of the electronics in a personal video recorder or a digital-cell-phone base station into a single hybrid FPGA—including the mixed-signal front- and back-end stuff. Seriously. You can reach Brian the fantasist at 1-916-454-5242, fax 1-916-454-5101, bdipert@edn.com, and www.bdipert.com.



EVALUATING PERFORMANCE: FPGAs VERSUS DSPs

By Jeff Bier, Berkeley Design Technology Inc

The latest digital-signal-processing-enhanced FPGAs boast huge gate counts, ample amounts of hard-wired SRAM, and an abundance of hard-wired multipliers. These attributes hint at the potential for phenomenal performance in digital-signal-processing applications. But experienced engineers know that the difference between potential performance and actual performance can be huge.

SIMPLE METRICS DON'T CUT IT

Experienced engineers also know to view the performance claims of chip vendors with skepticism. This cynicism definitely holds true when evaluating the digital-signal-processing performance of FPGAs. For example, FPGA vendors sometimes quote the digital-signal-processing performance of their chips in terms of MACs (multiply-accumulates) per second. On the surface, this approach seems reasonable; after all, many digital-signal-processing algorithms

performance metrics, such as MIPS, the MACs-per-second measurement suffers from several serious flaws, not the least of which is that no universal definition exists for exactly what a MAC operation comprises. When FPGA vendors quote MACs-per-second numbers for their devices, they often base their figures on distributed-arithmetic implementations of FIR (nonrecursive) filters. Distributed arithmetic is a natural way to implement a FIR filter on an FPGA. The problem is that in a distributed-arithmetic FIR, the MAC operators rely on the fact that the coefficients of the filter are constants. If the MAC operations in your algorithm don't use constant coefficients (for example, if you're building an adaptive filter or a correlator), the FPGA will deliver lower MACs-per-second performance.

How, then, should you go about determining the true digital-signal-processing application performance of these new

processors using the BDTI Benchmarks—a suite of digital-signal-processing algorithm kernels (for example, an FFT and a Viterbi decoder)—coupled with a methodology designed to ensure fair comparisons. The company has evaluated more than 50 processors with the BDTI Benchmarks, so using the same benchmarks for FPGAs seemed to be an attractive approach that would allow the company to quickly compare FPGAs with those and other processors. Unfortunately, BDTI found that this approach was the wrong way to go.

HOLISTIC OPTIMIZATION

Although a handful of algorithms dominates the computation requirements of a digital-signal-processing application, individual algorithm kernels are unsuitable as benchmarks for high-capacity FPGAs, for several reasons. With a processor, digital-signal-processing-application

developers aggressively optimize each of the key performance-hungry algorithms for speed. When an algorithm is running, it has exclusive use of all of the processor's execution units. But with an FPGA, designers have the flexibility to trade off parallelism (and hence performance) against resources used (logic blocks and multipliers, for example). Unlike with a processor,

with an FPGA, it makes no sense to use all of the available resources for a single algorithm, because doing so would leave no resources for the rest of the application. Instead, a designer must optimize the application as a whole, allocating the available hardware amid each of the constituent algorithms (Table A). This observation led the company to the conclusion that, to evaluate the digital-signal-processing performance of FPGAs, it needed a benchmark that looked more like a complete application and less like a single-algorithm kernel. The next question facing BDTI was what kind of application to use as the basis of the benchmark. Informal market research indicated that the new digital-signal-processing-enhanced FPGAs were of strong interest to developers of communications systems, leading us to develop a benchmark based on a communications receiver.

COST VERSUS/PERFORMANCE

Although they are powerful, the latest digital-signal-processing-enhanced FPGAs are also expensive. The lowest priced members of Xilinx's Virtex-II and Altera's Stratix devices cost hundreds of dollars, and the most expensive family members cost thousands of dollars per chip. Such prices render these chips unsuitable for highly cost-constrained products, such as cable-TV set-top boxes or DSL modems. But in communications-infrastructure equipment, a several-hundred-dollar price tag does not automatically disqualify a chip—especially if that chip can handle the processing for many

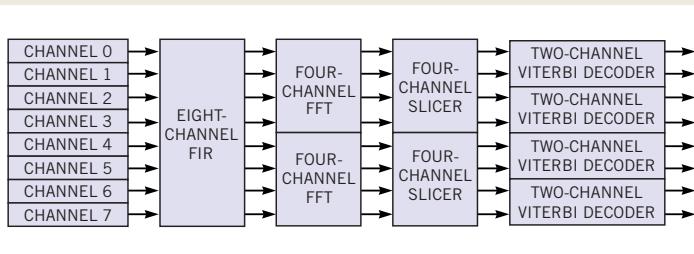


Figure A Altera implemented the BDTI Communications Benchmark using this eight-channel cluster, replicated until the benchmark iterations exhausted the Stratix FPGA's resources (courtesy BDTI).

make heavy use of MACs, and DSP vendors are also fond of quoting processor performance in terms of MACs per second.

But, like other oversimplified

FPGAs? This question is the one that BDTI (Berkeley Design Technology) recently began considering. For years, the company has been benchmarking the sig-

nals of the processor's execution units. But with an FPGA, designers have the flexibility to trade off parallelism (and hence performance) against resources used (logic blocks and multipliers, for example). Unlike with a processor,

communications channels. Therefore, for the initial benchmarking of digital-signal-processing-enhanced FPGAs, the company evaluated how many channels of a communications receiver each of the benchmarked chips could support.

The benchmark comprises a simplified OFDM (orthogonal-frequency-division-multiplexing) receiver (Figure A). OFDM is a complex technique that is finding increasing use in a variety of high-speed data-communications applications, such as fixed wireless systems. A detailed benchmark specification spells out all of the key parameters of the benchmark receiver, such as sample rates, filter lengths, and channel-code constraint lengths. Benchmark coders implement as many channels of the receiver as they can cram into a single chip.

THE BENCHMARK

BDTI invited Altera and Xilinx to implement the BDTI Communications Benchmark on their digital-signal-processing-enhanced FPGAs. Xilinx initially agreed but later withdrew from the project. If you'd like to see Xilinx results for the benchmark, e-mail BDTI at xilinx-benchmark@BDTI.com; BDTI will send Xilinx a summary of your requests. BDTI also invited Motorola and Texas Instruments to implement the benchmarks on their high-end DSPs, which target communications-infrastructure equipment. Altera and Motorola took our challenge, and each delivered a highly optimized implementation of the benchmark.

Because Altera was not yet

shipping its recently announced Stratix family of digital-signal-processing-enhanced FPGAs at the time of the benchmarking effort, BDTI projected Altera's results based on simulations. In contrast, Motorola has been shipping its MSC8101 DSP, which it based on the StarCore SC140 core, in volume for a while. BDTI scrutinized both Altera's and Motorola's benchmark implementations for correct operation and conformance to specifications.

SURPRISING RESULTS

It's clear that the new digital-signal-processing-enhanced FPGAs, with dozens of hard-wired multipliers and RAMs and tens of thousands of configurable-logic blocks, are capable of vast parallelism in applications that are amenable to parallelization. As with a processor, though, an FPGA's throughput in an application depends not only on how much work it can perform in one cycle but also on the clock speed it can attain. For processors, the maximum clock speed is easy to ascertain. But for a given FPGA, the clock speed attainable depends in part on what you're doing with it. Altera used simulations to project the clock speed attainable on its Stratix FPGAs with its implementation of BDTI's Communications Benchmark. BDTI will verify these projections when silicon becomes available.

Although BDTI expected the digital-signal-processing-enhanced FPGAs to perform well, the company was surprised

TABLE A—ALLOCATION OF ALTERA STRATIX FPGA-HARDWARE RESOURCES

	Logic elements (%)	DSP units (%)
FIR	15	0
FFT	35	100
Viterbi	50	0

by just *how* well they did. For example, the Altera Stratix 1S20-6 chip (scheduled to begin to be available for sampling this month, according to Altera) is projected to support more than a dozen channels of BDTI's Communications Benchmark. In contrast, the 300-MHz Motorola MSC8101 can support only about one-fifth of one channel.

The 1S20-6 has a projected price of \$325 (1000). The MSC8101 current sells for \$140 in similar quantities, which is fairly typical of high-end DSPs. As a result, on a cost/performance basis, the advantage of the 1S20-6 over the MSC8101 is somewhat smaller when you compare it with its throughput advantage, but FPGAs retain a significant lead.

Full details on the results of the FPGA-versus-DSP-benchmarking work appear in BDTI's just-published report, *FPGAs for DSP*.

OTHER FACTORS

BDTI's initial benchmarking work suggests that the new digital-signal-processing-enhanced FPGAs can indeed achieve impressive performance in certain types of DSP applications. But experience with these new devices and discussions with users indicate that factors other than performance often greatly influence decisions regarding

FPGA use. For example, one key challenge facing digital-signal-processing developers using FPGAs is the relative complexity of the design process and the lack of digital-signal-pro-

cessing-specific features in the development tools, compared with the tools available for the best supported DSPs.

Clearly, as with most technology-selection choices, deciding whether to use an FPGA for a digital-signal-processing application requires a sophisticated, multidimensional evaluation—one that depends on many specifics of the target application. In researching BDTI's recently completed report, the company developed a framework to guide developers in this challenging process. BDTI plans to continue its benchmarking and analysis, tracking the new generations of digital-signal-processing-enhanced FPGAs, processors, and other emerging alternatives.

AUTHOR'S BIOGRAPHY

Jeff Bier is general manager and co-founder of BDTI (Berkeley, CA), a well-known digital-signal-processing technology-analysis and software-development company. BDTI's Web site contains a wealth of free information of interest to developers of digital-signal-processing systems. The company recently expanded its focus beyond traditional DSPs and microprocessors to include FPGAs.

THEME AND VARIATION

General-purpose processors and DSPs...ASICs...FPGAs... hardware/software blends... Enough options to make your head spin? Well, hold onto your hat, because a few more alternatives are coming your way.

One criticism of conventional DSPs is their lack of instruction-set flexibility. Rarely do you find, with apologies to Goldilocks, an architecture that's not "too generic" or "too specific" but, instead, "just right" for your application. As a result, you have to either buy a faster, more expensive, and more power-hungry processor to compensate for missing functions or pay for functions you'll never use. User-customizable DSPs from companies such as 3DSP and Improv Systems and custom CPUs from folks such as Tensilica, aim to address your needs with tailored chips. Be careful, though; you're taking a long-term availability risk by buying a sole-sourced device from a small supplier, and you won't be able to benefit from the high-volume economies of scale of generic DSPs.

Are you looking for hardware that's flexible like a FPGA but doesn't require you to tediously stitch together fine-grained logic primitives, such as LUTs (look-up tables) and registers? Well, instead consider QuickSilver Technology's ACM (Adaptive Computing Machine), which interconnects higher level memory and processing primitives in

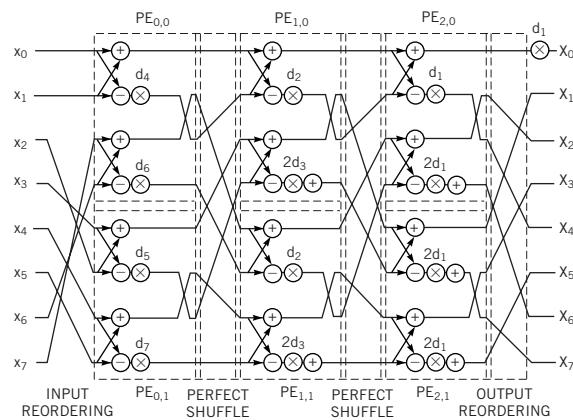


Figure A

The 8×8-bit DCT algorithm presents many opportunities for parallelism (courtesy Leopard Logic).

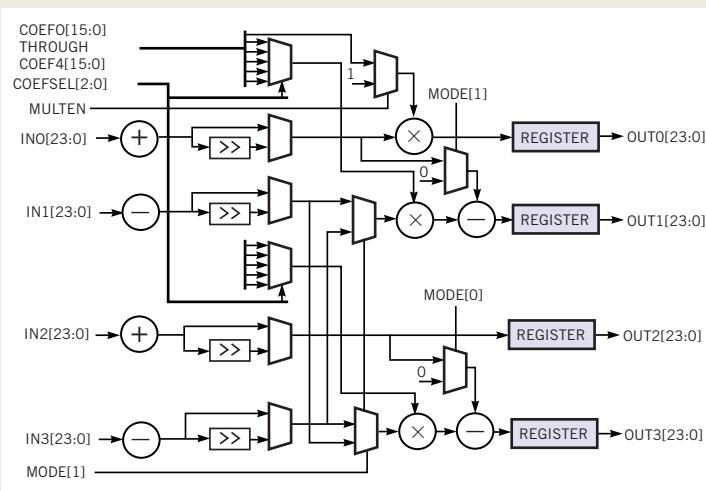


Figure B

A single, flexible Radix-4 Butterfly design, driven by external mode inputs, can comprehend all required operations (courtesy Leopard Logic).

a user-programmable fashion tailored for high-speed dynamic reconfiguration. QuickSilver has a test chip in hand, and the company plans both to develop its own application-tailored ACM variants through subsidiaries and spin-offs and to license the technology to other companies as an embedded core.

Finally, what if you're con-

vinced that a hardware-centric signal-processing approach is the way to go, but a pure FPGA won't meet your performance, power, or per-unit-price goals, and a pure ASIC is too inflexible? In such cases, embed one or multiple FPGA cores within your ASIC to provide the best of both worlds. Today's embedded FPGA suppliers include Actel (Vari-

Core), Atmel (AT40K), and Leopard Logic (HyperBlox), and IBM and partner Xilinx are scheduled to begin shipping their first hybrid ASICs by mid-decade. The following design example from Leopard Logic illustrates the technique:

"The DCT (discrete-cosine-transform) algorithm is a common application kernel used in the image-processing graphics domain. This example implements an 8×8 DCT, typically used in video encoders with 8-bit data inputs and 16-bit data outputs. **Figure A** defines the 8×8 DCT algorithm.

DSP MAPPING

"This algorithm can run in software on a processor, even on a generic 16-bit DSP, but it will incur a large latency and require many DSP clock cycles for execution. The relative inefficiency of logic activity per clock cycle versus a direct-parallel-hardware implementation also incurs a significant power penalty. In general, it is advisable in digital-signal-processing computation to calculate at the highest possible throughput for the least amount of time.

TRADITIONAL MAPPING TO DISCRETE FPGAs

"Another design alternative is to map the 8×8 DCT into hardware using a conventional FPGA technology. The design challenge in this case is to maximally use the preselected generic tiled

resources. This challenge manifests itself as an issue when tables need to match available memory-segment sizing, or when arithmetic processing needs to match to available multiplier units and their associated bit widths. Independent of the mapping process, the final design realization in a commercial FPGA will likely consume a substantial amount of power and have a significant cost premium compared with a full custom, standard-cell, or gate-array ASIC version.

"This example maps the 8×8 DCT into the Xilinx Virtex-II XC2V250-5-FG456 FPGA. The design uses the hardwired multipliers distributed through the FPGA logic array. The resultant design runs at a maximum operating frequency of 103 MHz under worst-case commercial conditions.

MAPPING TO HYPERBLOX FP

"To present the closest comparison of the HyperBlox FP to the Virtex-II architecture, this mapping uses hardwired multipliers implemented in standard-cell ASICs, and the remainder of the logic maps to the Leopard Logic HyperBlox FP embedded FPGA.

"The FPGA portion of the design requires 975 HyperBlox Core Cells, representing more than 95% of the 1024 available Core Cells. Whereas speed degradation is common in highly used discrete FPGAs, the patented hierarchical interconnect of the HyperBlox array allows this design to operate at a maximum operating frequency of 288 MHz under worst-case

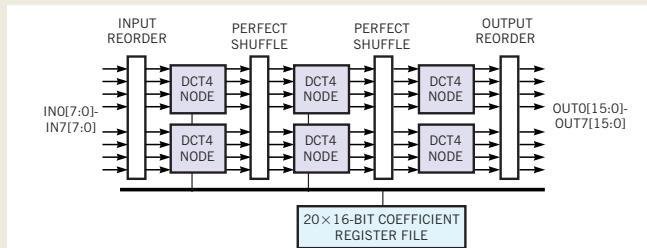


Figure C

Combining six Butterfly nodes creates the complete structure (courtesy Leopard Logic).

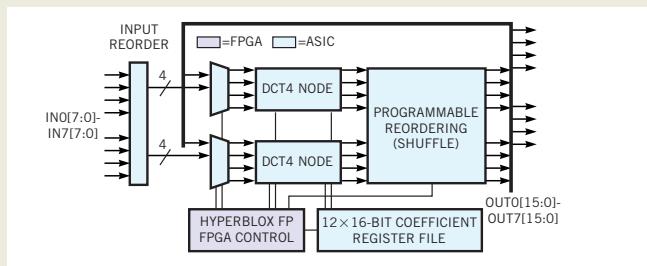


Figure D

Further expanding the ASIC-and-FPGA partitioned design boosts its performance (courtesy Leopard Logic).

commercial conditions on a 0.13-micron CMOS process.

"Although the traditional mapping shows impressive performance gains and is useful for comparison purposes, it is not the best method to implement designs with the HyperBlox FP.

"BEST-OF-BREED" MAPPING TO THE HYPERBLOX FP

"The final implementation of the 8×8 DCT illustrates the Leopard Logic design approach that encompasses the best technology from the ASIC and FPGA worlds. ASIC designers are no longer forced to accept a specific configured platform option.

"You can decompose the DCT algorithm into three columns, each of which contains two Radix-4 DCT Butterfly operations. The six Butterfly components each take on a slightly different form, but you can design

a single Butterfly that encompasses all of the required modes (Figure B).

"Note that Figure B shows shifters added to the basic structure to control bit growth. Six of the Butterfly nodes combine into the structure required for the 8×8 DCT as shown in Figure C. Place the Radix-4 Butterfly of Figure B in low-power, area-efficient standard-cell ASIC circuitry, because the Butterfly contains well-defined arithmetic blocks that have fixed functions.

"Note, too, the control lines in Figure B. Although the ASIC components are fixed in function, the control lines allow external selection of the desired mode of operation, in this case, by the FPGA circuitry.

"The HyperBlox FP embedded FPGA provides flexible, reprogrammable control to the ASIC datapath components. Two of

the Radix-4 Butterfly nodes combine to form one of the three columns of the 8×8 DCT. If you allow three cycles for data processing, you can use a single column and circulate the data through the pair of Radix-4 Butterfly nodes three times. On each pass, the controller implemented in the HyperBlox FP embedded FPGA selects the proper mode of operation for each of the Butterfly nodes.

"The resultant architecture, shown in Figure D, reveals the partitioning between ASIC and FPGA circuitry. The circuit has a maximum operating frequency of 900 MHz under worst-case commercial conditions

on a 0.13-micron CMOS process. Because this architecture requires three passes through the circuit, the net processing rate of this circuit is 300 MHz. If the design requires the full 900 MHz, you must instantiate all three DCT columns.

"Note that this partitioning of the 8×8 DCT, with a simple reprogramming of the HyperBlox FP embedded FPGA, can also implement a 32-point DCT used in audio compression."

Clearly, Xilinx, not Leopard Logic, is the expert at implementing designs in Xilinx architectures. Leopard Logic might also have, intentionally or not, chosen a signal-processing function that is simultaneously optimum for its HyperBlox technology and not optimum for Xilinx FPGAs. If Xilinx responds to Leopard Logic's claims, you'll find the company's response here.