General-purpose µPs for DSP applications: consider the trade-offs

GARRICK BLALOCK, BERKELEY DESIGN TECHNOLOGY INC

Using general-purpose processors instead of dedicated DSPs for DSP-intensive applications has some advantages, as well as some pitfalls. General-purpose µPs are a viable option in some system designs.

As DSP becomes ubiquitous in both PCs and embedded applications, many product designers must decide how to best implement signal-processing functions in their systems. In many cases, designers have to choose between using a dedicated DSP or using a µP or µC already present in the design. For a system designer, choosing whether to implement DSP on a general-purpose µP greatly depends on the application. Cost, power consumption, development tools, software, algorithms, performance, and many other issues affect the choice.

Until recently, this decision was easy—most general-purpose processors simply didn't have sufficient performance to implement most important DSP functions. Furthermore, dedicated DSPs offer several compelling advantages: They typically have strong price/performance ratios for DSP applications, consume relatively little power for DSP tasks, feature architectures that simplify DSP programming, and often have the support of a suite of DSP-oriented application-development tools and software libraries.

Although dedicated DSPs are well-suited to handle a system's signal-processing tasks, most designs also require a µP or µC for other processing tasks. Having two processors contradicts several common design objectives: lowering the system part count, reducing power consumption, minimizing size, and lowering cost. Integrating system functionality into one processor can be the best way to realize these goals. Reducing the processor count from two to one also means you have fewer instruction sets and tool suites to master.

One example of a system in which it can be attractive to use an already-existing general-purpose processor to implement DSP is a desktop PC. Implementing DSP applications, such as audio processing or modems, on an existing µP enables you to add DSP applications with little or no additional cost. Another example is consumer embedded applications, such as cellular telephones and wireless personal digital assistants, which often contain both a DSP and a system µP or µC. In addition to keeping costs down, using the µC or µP for DSP functions reduces product size and may lower power consumption.

In some cases, general-purpose µPs can actually outperform their DSP counterparts. Recent benchmark results reveal the effectiveness of general-purpose processors running DSP functions, such as a 256-point FFT and finite-impulse-response (FIR) filter (see **box** "Benchmark studies demonstrate general-purpose µPs' DSP capabilities").

General-purpose µPs lack some DSP capabilities

Despite the promising potential, obtaining strong DSP performance from general-purpose processors is no easy task. Many general-purpose µC and µP architectures are poorly suited for implementing DSP. Consider, for example, a common DSP algorithm: an FIR filter. The mathematical representation of an FIR filter is

$$y(n) = \sum_{j=0}^{N_{TAPS}} x(n-j)h(j),$$

where N_{TAPS} is the number of taps in the filter. Implementing an N-tap filter using a typical DSP, such as the Motorola (Austin, TX) DSP56002, simply requires executing the last instruction in **Listing 1** one time per tap. Hardware looping

LISTING 1—TYPICAL-DSP INSTRUCTIONS

```
move  #Xaddr,r0,   ; load data address into r0 pointer
move  #Haddr,r4    ; load coefficient address into r4 pointer
rep   #Ntaps       ; repeat the following instruction Ntap times
mac   x0,y0,a      x:(r0)+,x0  y:(r4)+,y0
```

handles the instruction repetition.

In contrast, a typical general-purpose processor requires far more instructions to implement the same filter. To implement one tap of the filter, most general-purpose processors must execute a lengthy series of instructions (**Listing 2**).

Although you can use a few mathematical tricks to slightly simplify this code, general-purpose μPs usually require many more instruction cycles to implement signal-processing algorithms than do DSPs. The high instruction-cycle count results from general-purpose μPs' lack of the many key architectural features of DSPs, such as a single-cycle multiply-accumulate (MAC) instructions, hardware looping, saturation arithmetic, multiple on-chip memory buses, and dedicated address generators that support modulo arithmetic.

Two ways to replace a DSP

Given the limits of a typical general-purpose architecture, μPs can achieve reasonable DSP performance by either increasing the instruction-execution rate or incorporating specialized DSP features and instructions. Although they have few DSP-oriented features, high-end PC processors, such as the original Pentium (Intel, Santa Clara, CA) and PowerPC 604e (Motorola/IBM, Fishkill, NY), can achieve strong DSP performance using their floating-point datapaths. Despite the high number of instructions necessary to implement DSP algorithms, advances in instruction-execution rates using techniques such as high clock speeds and superscalar architectures have bolstered these processors' DSP performance.

The design and advanced fabrication techniques of the Pentium and 604e allow them to run at instruction-cycle rates of 200 MHz and higher. In contrast, many dedicated DSPs have only recently achieved instruction-cycle rates of around 100 MHz. (The TMS320C62xx from Texas Instruments (Dallas), which runs at 200 MHz, is the lone exception). Of course, these high clock speeds contribute to the high power consumption of most high-end, general-purpose μPs and make them unsuitable for many portable DSP applications.

Multiple-issue architectures speed execution

The Pentium and the 604e feature two- and four-issue dynamic superscalar architectures, respectively. A dynamic superscalar architecture automatically executes nearby instructions in parallel whenever possible. Although data dependencies within programs and restrictions on which types of instructions can execute in parallel often prevent programs from taking maximum advantage of the potential instruction throughput, parallel execution significantly increases the average rate of instruction execution. Combined with high clock speeds, multiple-issue architectures can yield high instruction-execution rates that compensate for a general-purpose μP's poor instruction-set efficiency in DSP applications.

LISTING 2—GENERAL-PURPOSE-PROCESSOR INSTRUCTIONS

```
loop:   mov     *r0,r3      ; load data into r3 pointer
        mov     *r1,r4      ; load coefficient into r4 pointer
        mpy     r3,r4,r5    ; multiply into r5
        add     r5,r6       ; add r5 into accumulator r6
        inc     r0          ; increment pointer to read delay line
        inc     r1          ; increment pointer to coefficients
        dec     ctr         ; decrement loop counter
        jnz     loop        ; jump back to top if more taps remain
```

Unfortunately, dynamic superscalar architectures pose a problem for DSP programmers: Because instruction scheduling is dynamic, code-execution time is difficult to predict and can vary widely depending on many factors. Poor execution-time predictability is a serious concern, because many DSP applications are subject to real-time constraints. Furthermore, other dynamic characteristics of high-end, general-purpose processors—such as caches, data-dependent instruction-execution times, and branch prediction—make the problem worse. Although all these features can increase a processor's instruction-execution rate, they complicate the prediction of program-execution time.

The difficulty of predicting execution time can also hinder the optimization of performance-critical DSP inner loops. In many DSP applications, a small number of inner loops consumes a large portion of the execution time. To achieve maximum performance, DSP programmers typically optimize these critical inner loops in assembly language. Without the ability to predict the execution time of the instruction sequences in these inner loops, optimizing for efficient DSP code is difficult.

Fortunately, using good tools mitigates the problem of execution-time predictability. For example, you can use a cycle-accurate instruction-set simulator to calculate code-execution time and forecast worst-case scenarios to avoid violating real-time constraints. Unfortunately, cycle-accurate simulators, which are standard tools for dedicated DSPs, are sorely missing for most high-end general-purpose processors. Although not fully cycle-accurate, Intel's Vtune, a tool for profiling and optimizing 32-bit Pentium code, is perhaps the closest tool to a DSP-oriented instruction-set simulator among Pentium and PowerPC 604e tools. Vtune first collects a trace of a program's execution by running the program on a physical sample of the processor. The tool then uses an approximate, timing-only model of the processor to predict the performance of the traced program, to identify places that incur performance penalties, and to suggest possible optimizations. To demonstrate how difficult predicting execution time can be in high-end superscalar architectures, consider the section of simple PowerPC 604e assembly code (**Listing 3**).

Despite the simplicity of the code—it merely adds two vectors—even engineers familiar with the 604e have difficulty predicting how many instruction cycles it takes to execute one iteration of the loop in steady-state operation. The PowerPC architecture has two load/store units, a floating-point multiplier, and a branch execution unit, all of which can execute in parallel. Thus, some experienced programmers might conclude that this assembly code executes in one cycle per loop iteration. However, the PowerPC archi-

tecture also imposes complicated rules on what instructions can execute in parallel, which suggests that the code executes in five cycles. If engineers cannot easily predict the number of instruction cycles necessary for such a simple operation, optimizing code in critical DSP inner loops can be nearly impossible. (In fact, the code executes in four instruction cycles per iteration.)

In addition to boosting a processor's instruction-execution rate, designers can strengthen DSP performance by increasing the amount of DSP work the processor accom-

BENCHMARK STUDIES DEMONSTRATE GENERAL-PURPOSE µPs' DSP CAPABILITIES

Independent benchmark studies by Berkeley Design Technology Inc (BDTI) reveal that general-purpose processors possess strong DSP capabilities. In fact, the execution times of several general-purpose processors on BDTI's FFT benchmark are less than that of many DSPs (**Figure A**).

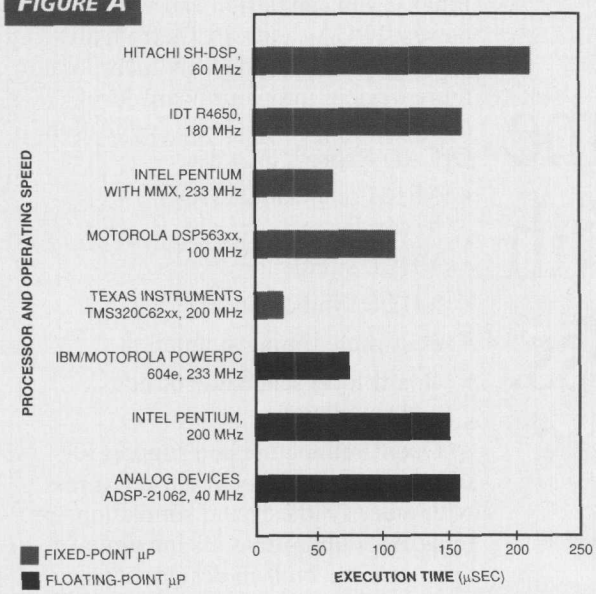
For example, in floating-point calculations, both the PowerPC 604e (IBM, Fishkill, NY/Motorola, Austin, TX) and the Pentium (Intel, Santa Clara, CA) complete the 256-point FFT in less time than the ADSP-21062 (Analog Devices, Norwood, MA), also known as "SHARC," a popular floating-point DSP. In fixed-point calculations, the Intel Pentium with multimedia extensions (MMX) outperforms the Motorola DSP563xx, a common fixed-point DSP. However, the Pentium with MMX is no match for the TMS320C62xx, the latest fixed-point DSP from Texas Instruments (Dallas).

A cost/performance ratio of the same processors on BDTI's complex finite-impulse-response (FIR) filter benchmark reveals the advantages of dedicated DSPs when you also consider cost (**Figure B**). Because the fastest versions of many µPs, especially PC processors, command a price premium, the study uses the most cost-effective speeds for all of the processors. In floating-point calculations, a representative DSP, the ADSP-21062, scores far better than either of the general-purpose processors.

Similarly, in fixed-point calculations, dedicated DSPs, such as the DSP563xx and the TMS320C62xx, score much better than the general-purpose processors. Of course, if the system specifications require a general-purpose processor for other reasons, the DSP functionality that it implements comes at little or no additional cost.

More information on DSP benchmark results and brief analyses of many processors' DSP capabilities are available on BDTI's Web site, www.bdti.com.

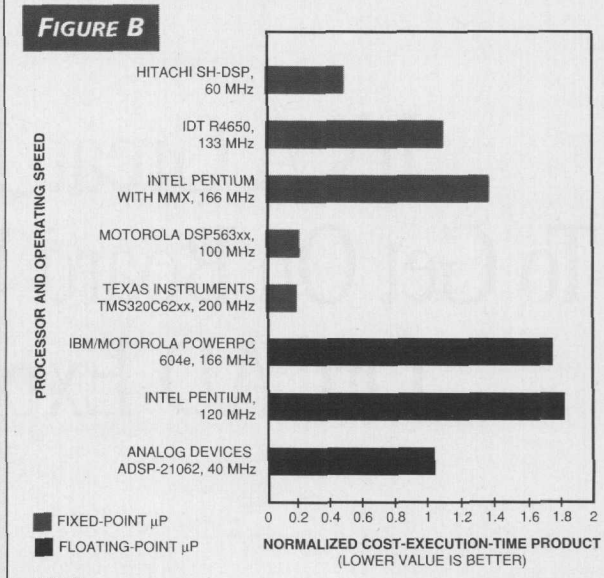
FIGURE A



NOTES:
THE FFT IS A 256-POINT, COMPLEX, RADIX-2 IMPLEMENTATION.
ALL RESULTS ASSUME THAT PROGRAM AND DATA ARE PRELOADED INTO PROCESSORS' ON-CHIP MEMORY.
RESULTS FOR THE IDT R4650 ARE ESTIMATED.

A study of processor execution time on BDTI's FFT benchmark reveals some surprises: Some general-purpose µPs outperform dedicated DSPs.

FIGURE B



NOTES:
PROCESSORS' COST/PERFORMANCE RATIO IS INDICATED BY THE NORMALIZED COST-EXECUTION-TIME PRODUCT (PROCESSOR'S COST MULTIPLIED BY EXECUTION TIME) FOR THE BDTI COMPLEX-BLOCK FIR-FILTER BENCHMARK.
PROCESSOR'S COSTS ARE FOR QUANTITY-1000 ORDERS, AS REPORTED BY THE MANUFACTURER, APRIL 1997.
RESULTS FOR THE IDT R4650 ARE ESTIMATED.
ALL RESULTS ASSUME THAT PROGRAM AND DATA ARE PRELOADED INTO PROCESSORS' ON-CHIP MEMORY.

SOURCE FOR FIGURES A AND B:
DSP ON GENERAL-PURPOSE PROCESSORS AND *BUYER'S GUIDE TO DSP PROCESSORS (THIRD EDITION)* (BERKELEY DESIGN TECHNOLOGY INC).

Cost/performance results of a complex-block FIR-filter benchmark show the advantages of DSPs when you also consider cost.

DSP ON GENERAL-PURPOSE µPs

plishes per instruction. Several vendors of high-end, general-purpose processors have added single-instruction, multiple-data (SIMD) instruction-set extensions to their processors. SIMD instructions partition registers and ALUs so that multiple items of data are present in one register or memory location and so that one instruction can process the data in parallel. For example, an SIMD processor might contain 64-bit registers that you can partition into eight 8-bit data elements, four 16-bit data elements, two 32-bit data elements, or one 64-bit data element. Typically, an SIMD processor performs an operation, such as addition or multiplication, on multiple pairs of data elements using just one instruction. Processor vendors commonly use SIMD instructions to add DSP capabilities to 32- or 64-bit RISC/CISC architectures, because these architectures often already contain the necessary wide buses and registers.

One of the attractions of SIMD instructions is the ability to select an appropriate data-word length. If low precision is acceptable, programmers can use 16-bit data elements and operate on four elements in parallel, for example. Alternatively, if higher precision is necessary, programmers can choose 32-bit data elements at the price of performing fewer operations in parallel.

If SIMD instructions use fixed-point arithmetic, processor designers can sometimes accomplish parallel processing by simply partitioning an existing datapath. For example, if a processor contains a 32×32- to 64-bit multiplier, designers can dissect the multiplier into four 8×8- to 16-bit multipliers that operate in parallel. Unfortunately, realizing the performance potential of SIMD instructions often requires restructuring algorithms to process elements simultaneously. This requirement can make optimizing code for SIMD instructions difficult. Furthermore, some applications may see little improvement over non-SIMD instructions. For example, applications with sequential data dependencies, such as adaptive filtering, may be limited in the number of calculations that can run in parallel.

In many DSP applications, however, SIMD instructions are effective. For example, Intel uses SIMD instructions in its multimedia extensions (MMX), which greatly improve the DSP performance of its Pentium processor. However, these extensions have complications. To implement the extensions and maintain operating-system compatibility, Intel designed the MMX instructions to share registers with the processor's floating-point unit. Thus, programs incur a penalty of many cycles when switching from floating-point to MMX modes. Fortunately, the cost of this switch is unlikely to significantly affect many DSP applications, because the MMX datapath is fixed-point, and few DSP applications require frequent mixing of fixed-point and floating-point arithmetic. Thus, the slow switch from floating-point mode to MMX mode should occur infrequently in most DSP applications.

Processors for cost-sensitive embedded applications typically run at much lower clock speeds than do processors in high-end desktop PCs. Thus, it's not surprising that embedded-processor vendors add coprocessors and other hardware enhancements to boost DSP performance. Although many processor vendors attempt to boost DSP performance by simply adding MAC units to their existing architectures—the R4650 from Integrated Device Technology (Santa Clara, CA) is a good example—other vendors make more extensive modifications.

LISTING 3—POWERPC 604E ASSEMBLY CODE

```
@vec_add_loop:

lfsu fpTemp1,4(rAAddr)          # Load A data, ptr. update
lfsu fpTemp2,4(rBAddr)          # Load B data, ptr. update
fadds fpSum,fpTemp1,fpTemp2     # Perform add operation
stfsu fpSum,4(rCAddr)           # Store sum, ptr. update
bdnz @vec_add_loop              # loop
```

For example, the ARM7TDMI processor core (Advanced RISC Machines, Cambridge, England) is a simple, general-purpose processor core that targets embedded consumer applications in which low cost and low power consumption are paramount. Unmodified, the DSP performance of the ARM7TDMI suffers from poor memory bandwidth and a slow MAC instruction. To improve performance in DSP applications, the company now offers the Piccolo coprocessor. Piccolo accepts operands and instructions from the main processor and then executes them in parallel with normal ARM instructions executing on the main processor. The DSP-oriented Piccolo instruction set allows single-instruction-cycle throughput of important DSP instructions, such as MAC. Because Piccolo executes independently, the main ARM processor is free to execute other instructions or load more data from external memory, which reduces the memory-bandwidth bottleneck.

Hitachi (Brisbane, CA) has adopted a contrasting strategy in its SH-DSP. The SH-DSP adds a complete fixed-point DSP datapath and instruction set to the company's successful SH-2 µC architecture. This unusual hybrid approach allows programmers to add DSP functionality and protects their investment in SH-2 code, which runs unaltered on the SH-DSP. Programmers can access the SH-DSP's DSP datapath by adding DSP instructions to an SH-2 program. The SH-DSP sequentially fetches instructions and issues DSP and µC instructions to the appropriate execution unit.

The DSP capabilities of the SH-DSP are similar to those of many 16-bit DSPs and enable strong fixed-point DSP performance at clock speeds much lower than those of the Pentium and the 604e. The SH-DSP's compatibility with the SH-2 provides a natural migration path for SH-2 customers who contemplate DSP-intensive designs. Of course, this compatibility has a price. Although the SH-DSP is a single processor, it has two personalities: two instruction sets, two datapaths, two sets of registers, and so on. This duality complicates the programming model and hinders performance in some instances.

Although high-end processors, such as the Pentium, Pentium with MMX, and PowerPC 604e, offer excellent DSP performance, their high cost and power needs make them pro-

DSP ON GENERAL-PURPOSE μPs

hibitive for small, battery-powered, or low-cost consumer applications. For desktop-PC applications, however, power consumption is of little consequence, and the cost of the host processor is unavoidable. Any DSP functionality that the host processor can provide comes with little marginal cost.

Why, then, are desktop-PC host processors rarely used for DSP? Problems with tools, operating systems, and hardware definition are all part of the answer. As stated earlier, few tools are available to develop and fully optimize DSP applications on general-purpose μPs. In addition, most programmers of high-end, general-purpose μPs use a high-level language, such as C or C++. Predicting the execution time of compiled code is even more difficult than predicting that of assembly-language code, and performance is often many times worse than the prediction. Moreover, compiler-generated code is typically difficult to optimize. In the case of the Intel Pentium with MMX, a C compiler that generates MMX instructions is not even publicly available.

Furthermore, even with good application software, the operating systems of most popular PCs don't have the real-time support necessary to guarantee that DSP applications get the sufficient system resources to meet real-time constraints. For example, many general-purpose μPs and the operating systems on which they run offer no practical way to lock a cache. Without cache locking, execution times vary from PC to PC depending on the size of the cache and the speed of external-memory accesses. Although some applications can compensate for variable execution times—a video-conferencing system, for example, could drop a frame—many real-time applications, such as modems, cannot endure a shortage of processor time.

Embedded applications have more options

Consumer embedded applications usually place priority on cost. Replacing both a DSP and a general-purpose μP with one general-purpose μP saves money and simplifies manufacturing. In mobile applications, reducing the processor count reduces product size and possibly power consumption.

The challenges of coding DSP algorithms on embedded general-purpose μPs are less daunting than those of PCs. Because most low-cost, general-purpose processors don't employ dynamic superscalar execution and branch prediction, predicting execution time and optimizing DSP code are easier. And because embedded processors tend to implement fixed functions, code can often execute from on-chip ROM. This feature reduces the processor's dependency on instruction and data caches, which further increases execution predictability.

In addition, unlike desktop-PC designers, embedded-system designers are not locked into a choice of only one or two operating systems; they are free to choose among real-time operating systems. Embedded-processor vendors have also been more aggressive in providing DSP-oriented tools. Hitachi and Advanced RISC Machines, for example, offer cycle-accurate simulators for the SH-DSP and the Piccolo coprocessor, respectively. Of course, overcoming the momentum of dedicated DSPs won't be easy. Large selections of DSP-oriented software, development tools, and third-party support are available for DSPs from vendors such as Texas Instruments and Analog Devices (Norwood, MA).

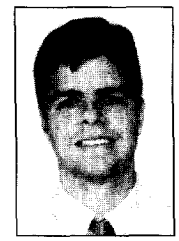
To continue to leverage the advantages of DSPs but still achieve more system integration, some designers may try to implement general control and computing on a DSP. Motorola's DSP568xx family targets this market by adding many μC features to a 16-bit DSP core. Other designers may try to get the best of both worlds by choosing a chip with both μC and DSP cores. Texas Instruments has supported this approach by introducing a chip for cellular handsets that combines a TMS320C54x DSP core with an ARM7TDMI μC core. The ARM core handles supervisory control functions and the user interface, and the C54x implements voice compression and baseband signal processing.

What happens next?

The incentive to integrate functionality will undoubtedly drive further attempts to add DSP capabilities to general-purpose μPs. In the PC arena, advances in tools, operating systems, and standards will eventually make host-based signal processing a reality. Architecture extensions, such as Intel's MMX, will accelerate this process. In embedded markets, the choices will proliferate to meet the varied requirements of applications. Undoubtedly, general-purpose processors will increasingly become viable and attractive choices for DSP-algorithm implementation.

However, it's unlikely that general-purpose processors will replace dedicated DSPs in all applications. For DSP-intensive applications that require a demanding mix of performance, price, power consumption, software, and development tools, DSPs will remain the first choice of designers. EDN

Author's biography



Garrick Blalock is an engineer at Berkeley Design Technology Inc (Berkeley, CA), a firm specializing in the analysis of DSP technology and DSP-firmware development. Blalock is a coauthor of BDTI's recent technical reports DSP on General-Purpose Processors *and* Buyer's Guide to DSP Processors. *You can reach him at 1-510-665-1600, info@bdti.com, or www.bdti.com.*