The Evolution of DSP Processors

From Early Architectures to the Latest Developments

Jennifer Eyre and Jeff Bier

Over the last five years, the number and variety of products that include some form of digital signal processing (DSP) has grown dramatically. DSP has become a key component in many consumer, communications, medical, and industrial products, which use a variety of hardware approaches to implement DSP, ranging from the use of off-the-shelf microprocessors to field-programmable gate arrays (FPGAs) to custom integrated circuits (ICs). Programmable "DSP processors," a class of microprocessors optimized for DSP, are a popular solution for several reasons: They can potentially be reprogrammed in the field, allowing product upgrades or fixes. They are often more cost-effective (and less risky) than custom hardware, particularly for low-volume applications, where the development cost of custom ICs may be prohibitive. And in comparison to other types of microprocessors, DSP processors often have an advantage in terms of speed, cost, and energy efficiency.

In this article, we trace the evolution of DSP processors from early architectures to current state-of-the-art devices. We highlight some of the key differences among architectures and compare their strengths and weaknesses. Finally, we discuss the growing class of general-purpose processors that have been enhanced to address the needs of DSP applications.

DSP Algorithms Mold DSP Architectures

From the outset, DSP processor architectures have been molded by DSP algorithms. For nearly every feature found in a DSP processor, there are associated DSP algorithms whose computation is in some way eased by inclusion of this feature. Therefore, perhaps the best way to understand the evolution of DSP architectures is to examine typical DSP algorithms and identify how their computational requirements have influenced the architectures of DSP processors. As a case study, we will consider one of the most common signal processing tasks, the finite impulse response (FIR) filter.

Fast Multipliers

The FIR filter is mathematically expressed as $\Sigma x*h$, where x is a vector of input data and h is a vector of filter coefficients. For each "tap" of the filter, a data sample is multiplied by a filter coefficient, with the result added to a running sum for all of the taps (for an introduction to DSP concepts and filter theory, see [2]). Hence, the main component of the FIR filter algorithm is a dot product: multiply and add, multiply and add. These operations are not unique to the FIR filter algorithm; in fact, multiplication (often combined with accumulation of products) is one of the most common operations performed in signal processing—convolution, infinite impulse response (IIR) filtering, and Fourier transforms also all involve heavy use of multiply-accumulate operations. Originally, microprocessors implemented multiplications by a series of shift and add operations, each of which consumed one or more clock cycles. In 1982, however, Texas Instruments (TI) introduced the first commercially successful "DSP processor," the TMS32010, which incorporated specialized hardware to enable it to compute a multiplication in a single clock cycle. As might be expected, faster multiplication hardware yields faster performance in many DSP algorithms, and for this reason all modern DSP processors include at least one dedicated single-cycle multiplier or combined multiply-accumulate (MAC) unit [1].

Multiple Execution Units

In comparison to other types of computing tasks, DSP applications typically have very high computational requirements, since they often must execute DSP algorithms (such as FIR filtering) in real time on lengthy segments of signals sampled at 10-100 kHz or higher. So DSP processors often include several independent execution units that are capable of operating in parallel—for example, in addition to the MAC unit, they typically contain an arithmetic-logic unit (ALU) and a shifter.

Efficient Memory Accesses

Executing a MAC in every clock cycle requires more than just a single-cycle MAC unit. It also requires the ability to fetch the MAC instruction, a data sample, and a filter coefficient from memory in a single cycle. Hence, good DSP performance requires high memory bandwidth—higher than was supported on the general-purpose microprocessors of the early 1980's, which typically contained a single bus connection to memory and could only make one access per clock cycle. To address the need for increased memory bandwidth, early DSP processors developed different memory architectures that could support multiple memory accesses per cycle. The most common approach (still commonly used) was to use two or more separate banks of memory, each of which was accessed by its own bus and could be read or written during every clock cycle. Often, instructions were stored in one memory bank, while data (or a combination of data and instructions) was stored in another. With this arrangement, the processor could fetch an instruction and a data operand in parallel in every cycle. Figure 1 shows the dif-

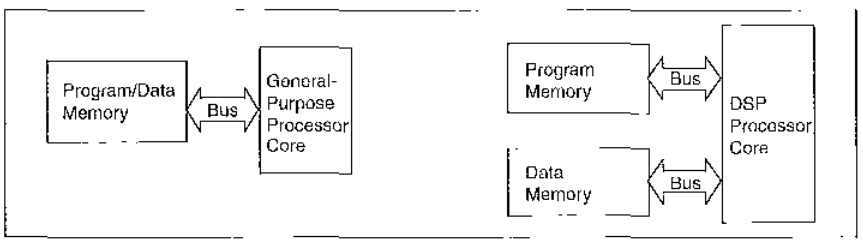


Fig. 1. Differences in memory architectures for early general-purpose processors versus early DSP processors.

ference in memory architectures for early general-purpose processors and DSP processors. Since many DSP algorithms (such as FIR filters) consume two data operands per instruction (e.g., a data sample and a coefficient), a further optimization commonly used is to include a small bank of RAM near the processor core that is used as an instruction cache. When a small group of instructions is executed repeatedly (i.e., in a loop), the cache is loaded with those instructions, freeing the instruction bus to be used for data fetches instead of instruction fetches– thus enabling the processor to execute a MAC in a single cycle.

High memory bandwidth requirements are often further supported via dedicated hardware for calculating memory addresses. These address generation units operate in parallel with the DSP processor's main execution units, enabling it to access data at new locations in memory (for example, stepping through a vector of coefficients) without pausing to calculate the new address.

Memory accesses in DSP algorithms tend to exhibit very predictable patterns; for example, for each sample in an FIR filter, the filter coefficients are accessed sequentially from start to finish, then accesses start over from the beginning of the coefficient vector when processing the next input sample. This is in contrast to other types of computing tasks, such as database processing, where accesses to memory are less predictable. DSP processor address generation units take advantage of this predictability by supporting specialized addressing modes that enable the processor to efficiently access data in the patterns commonly found in DSP algorithms. The most common of these modes is register-indirect addressing with post-increment, which is used to automatically increment the address pointer for algorithms where repetitive computations are performed on a series of data stored sequentially in memory. Without this feature, the programmer would need to spend instructions explicitly incrementing the address pointer. Many DSP processors also support "circular addressing," which allows the processor to access a block of data sequentially and then automatically wrap around to the beginning address—exactly the pattern used to access coefficients in FIR filtering. Circular addressing is also very helpful in implementing first-in, first-out buffers, commonly used for I/O and for FIR filter delay lines.

Data Format

Most DSP processors use a fixed-point numeric data type instead of the floating-point format most commonly used in scientific applications. In a fixed-point format, the binary point (analogous to the decimal point in base 10 math) is located at a fixed location in the data word. This is in contrast to floating-point formats, in which numbers are expressed using an exponent and a mantissa and the binary point essentially "floats" based on the value of the exponent. Floating-point formats allow a much wider range of values to be represented and virtually eliminates the hazard of numeric overflow in most applications. DSP applications typically must pay careful attention to numeric fidelity (e.g., avoiding overflow). Since numeric fidelity is far more easily maintained using a floating-point format, it may seem surprising that most DSP processors use a fixed-point format. In many applications, however, DSP processors face additional constraints: they must be inexpensive and provide good energy efficiency. Fixed-point processors tend to be cheaper and less power hungry than floating-point processors at comparable speeds because floating-point formats require more complex hardware to implement. For these reasons, there are few floating-point DSP processors.

Sensitivity to cost and energy consumption also influences the data word width used in DSP processors. DSP processors tend to use the shortest data word that will provide adequate accuracy in their target applications. Most fixed-point DSP processors use 16-bit data words because that data word width is sufficient for many DSP applications. A few fixed-point DSP processors use 20, 24, or even 32 bits to enable better accuracy in applications that are difficult to implement well with 16-bit data, such as high-fidelity audio processing.

To ensure adequate signal quality while using fixed-point data, DSP processors typically include specialized hardware to help programmers maintain numeric fidelity throughout a series of computations. For example, most DSP processors include one or more "accumulator" registers to hold the results of summing several multiplication products. Accumulator registers are typically wider than other registers; they often provide extra bits, called "guard bits," to extend the range of values that can be represented and thus avoid overflow. In addition, DSP processors usually include good support for saturation arithmetic, rounding, and shifting, all of which are useful for maintaining numeric fidelity.

Zero-Overhead Looping

DSP algorithms typically spend the vast majority of their processing time in relatively small sections of software that are executed repeatedly; i.e., in loops. Hence, most DSP processors provide special support for efficient looping. Often, a special loop or repeat instruction is provided which allows the programmer to implement a for-next loop without expending any clock cycles for updating and testing the loop counter or branching back to the top of the loop. This feature is often referred to as "zero-overhead looping."

Streamlined I/O

Finally, to allow low-cost, high-performance input and output, most DSP processors incorporate one or more specialized serial or parallel I/O interfaces and streamlined I/O handling mechanisms such as low-overhead interrupts and direct memory access (DMA) to allow data transfers to proceed with little or no intervention from the processor's computational units.

Specialized Instruction Sets

DSP processor instruction sets traditionally have been designed with two goals in mind: 1) to make maximum use of the processor's underlying hardware, thus increasing efficiency and 2) to minimize the amount of memory space required to store DSP programs, since DSP applications are often quite cost-sensitive and the cost of memory contributes substantially to overall chip and/or system cost. To accomplish the first goal, conventional DSP processor instruction sets generally allow the programmer to specify several parallel operations in a single instruction, typically including one or two data fetches from memory (along with address pointer updates) in parallel with the main arithmetic operation. With the second goal in mind, instructions are kept short (thus using less program memory) by restricting which registers can be used with which operations and which operations can be combined in an instruction. To further reduce the number of bits required to encode instructions, DSP processors often offer fewer registers than other types of processors and may use mode bits to control some features of processor operation (for example, rounding or saturation) rather than encoding this information as part of the instructions. The overall result of these features is that conventional DSP processors tend to have highly specialized, complicated, and irregular instruction sets, which is a significant drawback because it complicates the task of creating efficient assembly language software—whether by a programmer or by a compiler. Why is this important?

Programmers who write software for PC processors, such as Pentiums or PowerPCs, typically don't have to worry much about the ease of use of the processor's instruction set because they generally develop programs in a high-level language, such as C or C++. Life isn't quite so simple for the DSP processor programmer because high-volume DSP applications, unlike other types of applications, are often written (or at least have portions optimized) in assembly language.

There are two main reasons why DSPs aren't usually programmed in high-level languages. The first is that most widely used high-level languages, such as C, are not well suited for describing typical DSP algorithms. The second reason is that conventional DSP architectures, with their multiple memory spaces, multiple buses, irregular instruction sets, and highly specialized hardware, are difficult for compilers to use effectively. It is certainly true that a compiler can take C source code and generate assembly code for a DSP, but to get efficient code, the programmer often must hand-optimize the critical sections of the program in assembly language. DSP applications typically have very high computational demands coupled with strict cost constraints, making program optimization essential. For these reasons, programmers often consider the palatability (or lack thereof) of the instruction set of a DSP processor as a key aspect of its overall desirability.

The Current DSP Landscape

Conventional DSP Processors

The performance and price range among DSP processors is very wide. In the low-cost, low-performance range are the industry workhorses, which are based on conventional DSP architectures. These processors are quite similar architecturally to the original DSP processors of the early 1980s. They issue and execute one instruction per clock cycle and use the complex, multi-operation type of instructions described earlier. These processors typically include a single multiplier or MAC unit and an ALU, but few additional execution units, if any. Included in this group are Analog Devices' ADSP-21xx family, Texas Instruments' TMS320C2xx family, and Motorola's DSP560xx family. These processors generally operate at around 20-50 MHz and provide good DSP performance while maintaining very modest power consumption and memory usage. They are typically used in consumer and telecommunications products that have modest DSP performance requirements and stringent cost and/or energy consumption constraints, like disk drives and digital telephone answering machines.

Mid-range DSP processors achieve higher performance than the low-cost DSPs described above through a combination of increased clock speeds and somewhat more sophisticated architectures. DSP processors like the Motorola DSP563xx and Texas Instruments TMS320C54x operate at 100-150 MHz and often include a modest amount of additional hardware, such as a barrel shifter or instruction cache, to improve performance in common DSP algorithms. Processors in this class also tend to have deeper pipelines than their lower-performance cousins. (Pipelining is a hardware technique for overlapping the execution of portions of several instructions to improve instruction throughput [6].) These differences notwithstanding, however, mid-range DSP processors are more like their predecessors than they are different; architectural improvements are mostly incremental rather than dramatic. Hence, this group of DSP processors can still be classified as having conventional architectures. By using this approach, mid-range DSP processors are able to achieve noticeably better performance while keeping energy and power consumption low. Processors in this performance range are typically used in wireless telecommunications applications and high-speed modems, which have relatively high computational demands but often require low power consumption.

Enhanced-Conventional DSPs

DSP processor architects who want to improve performance beyond the gains afforded by faster clock speeds and modest hardware improvements must find a way to get significantly more useful DSP work out of every clock cycle. One approach is to extend conventional DSP architectures by adding parallel execution units, typically a sec-

ond multiplier and adder. These hardware enhancements are combined with an extended instruction set that takes advantage of the additional hardware by allowing more operations to be encoded in a single instruction and executed in parallel. We refer to this type of processor as an "enhanced-conventional DSP processor" because it is based on the conventional DSP processor architectural style rather than an entirely new approach. With this increased parallelism, enhanced-conventional DSP processors can execute significantly more work per clock cycle—for example, two MACs per cycle instead of one. Figure 2(a) and (b) compares the execution units and buses of a conventional DSP (the Lucent Technologies DSP16xx) to an enhanced-conventional DSP that extends the DSP16xx architecture (the Lucent Technologies DSP16xxx).

Enhanced-conventional DSP processors typically have wider data buses to allow them to retrieve more data words per clock cycle to keep the additional execution units fed with data. They may also use wider instruction words to accommodate specification of additional parallel operations within a single instruction. Increases in cost and power consumption due to the additional hardware and architectural complexity are largely offset by increased performance (and, in some cases, by the use of more advanced fabrication processes), allowing these processors to maintain cost-performance and energy consumption similar to those of previous generations of DSPs.

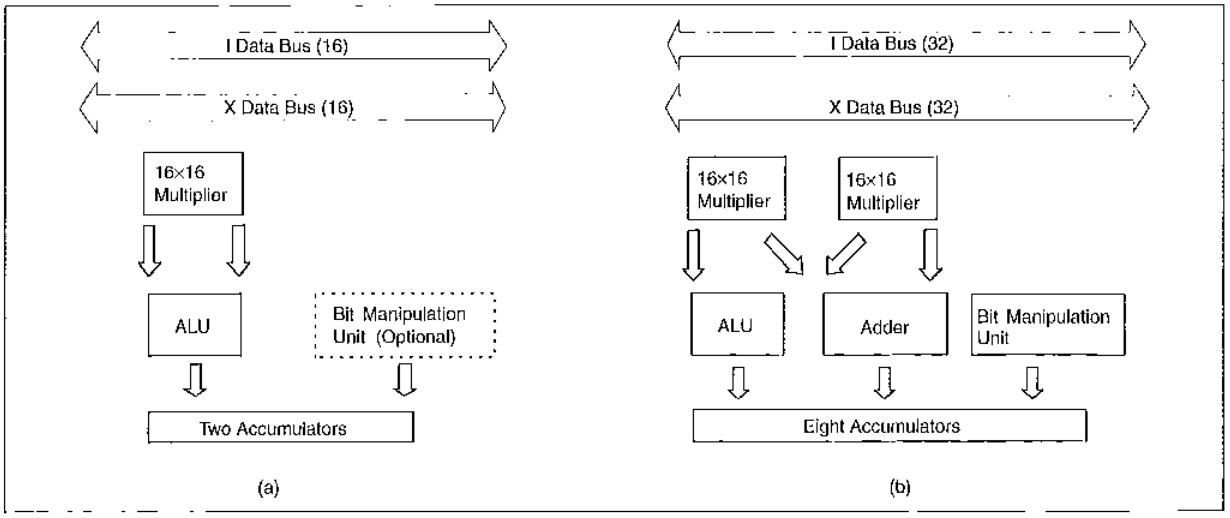
Multi-Issue Architectures

Enhanced-conventional DSP processors provide improved performance by allowing more operations to be encoded in every instruction. But because they follow the trend of using specialized hardware and complex, compound instructions, they suffer from some of the same problems as conventional DSPs: they are difficult to program in assembly language and they are unfriendly compiler targets. With the goals of achieving high performance and creating an architecture that lends itself to the use of compilers, some newer DSP processors use a "multi-issue" approach. In contrast to conventional and enhanced-conventional processors, multi-issue processors use very simple instructions that typically encode a single operation. These processors achieve a high level of parallelism by issuing and executing instructions in parallel groups rather than one at a time. Using simple instructions simplifies instruction decoding and execution, allowing multi-issue processors to execute at higher clock rates than conventional or enhanced conventional DSP processors.

TI was the first vendor to use this approach in a commercial DSP processor. TI's multi-issue TMS320C62xx, introduced in 1996, was dramatically faster than any other DSP processor available at the time. Other vendors have since followed suit, and now all four of the major DSP processor vendors (TI, Analog Devices, Motorola, and Lucent Technologies) are employing multi-issue architectures for their latest high-performance processors. The two classes of architectures that execute multiple instructions in parallel are referred to as VLIW (very long instruction word) and superscalar. These architectures are quite similar, differing mainly in how instructions are grouped for parallel execution. With one exception, all current multi-issue DSP processors use the VLIW approach.

VLIW and superscalar architectures provide many execution units (many more than are found on conventional or even enhanced conventional DSPs), each of which executes its own instruction. Figure 3 shows the execution units and buses of the TMS320C62xx, which contains eight independent execution units. VLIW DSP processors typically issue a maximum of between four and eight instructions per clock cycle, which are fetched and issued as part of one long super-instruction—hence the name "very long instruction word." Superscalar processors typically issue and execute fewer instructions per cycle, usually between two and four.

In a VLIW architecture, the assembly language programmer (or code-generation tool) specifies which in-



▲ 2. *(a) Lucent DSP16xx (conventional DSP processor) and (b) Lucent DSP16xxx (enhanced-conventional DSP processor).*

structions will be executed in parallel. Hence, instructions are grouped at the time the program is assembled, and the grouping does not change during program execution. Superscalar processors, in contrast, contain specialized hardware that determines which instructions will be executed in parallel based on data dependencies and resource contention, shifting the burden of scheduling parallel instructions from the programmer to the processor. The processor may group the same set of instructions differently at different times in the program's execution; for example, it may group instructions one way the first time it executes a loop and then group them differently for subsequent iterations. The difference in the way these two types of architectures schedule instructions for parallel execution is important in the context of using them in real-time DSP applications.

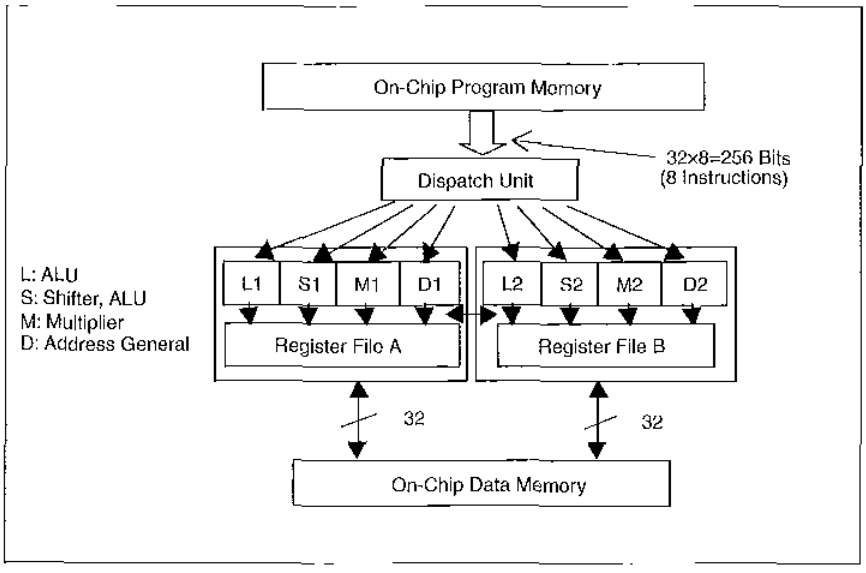
Because superscalar processors dynamically schedule parallel operations, it may be difficult for the programmer to predict exactly how long a given segment of software will take to execute. The execution time may vary based on the particular data accessed, whether the processor is executing a loop for the first time or the third, or whether it has just finished processing an interrupt, for example. This uncertainty in execution times can pose a problem for DSP software developers who need to guarantee that real-time application constraints will be met in every case. Measuring the execution time on hardware doesn't solve the problem, since the execution time is often variable. Determining the worst-case timing requirements and using them to ensure that real-time deadlines are met is another approach, but this tends to leave much of the processor's speed untapped. Dynamic features also complicate software optimization. As a rule, DSP processors have traditionally avoided dynamic features (such as superscalar execution and dynamically loaded caches) for just these reasons; this may be why there is currently only one example of a commercially available superscalar DSP processor.

Although their instructions are very simple and typically encode only one operation, most current VLIW processors use wider instruction words than conventional DSP processors—for example, 32 bits instead of 16. (Because there is only one current example of a commercial superscalar DSP, it is difficult to generalize about this class of architecture.) There are many reasons for using a wider instruction word. In VLIW architectures, a wide instruction word may be required to specify information about which functional unit will execute the instruction. Wider instructions allow the use of larger, more uniform register sets (rather than the small sets of specialized registers common among conventional DSP processors), which in turn enables higher performance. Relatedly, the use of wide instructions allows a higher degree of consistency and regularity in the instruction set. These instructions have few restrictions on register usage and addressing modes, making VLIW processors better compiler targets (and easier to program in assembly language). There are disadvantages, however, to using wide, simple instructions. Since each VLIW instruction is simpler than a conventional DSP processor instruction, VLIW processors tend to require many more instructions to perform a given task. Combined with the fact that the instruction words are typically wider than those found on conventional DSP processors, this characteristic results in relatively high program memory usage. High program memory usage, in turn, may result in higher chip or system cost because of the need for additional ROM or RAM.

When a processor issues multiple instructions per cycle, it must be able to determine which execution unit will process each instruction. Traditionally, VLIW processors have used the position of each instruction within the super-instruction to determine to where the instruction will be routed. Some recent VLIW architectures do not use positional super-instructions and instead include routing information within each subinstruction.

To support execution of multiple parallel instructions, VLIW and superscalar processors must have sufficient instruction decoders, buses, registers, and memory bandwidth. VLIW processors typically use either wide buses or a large number of buses to access data memory and keep the multiple execution units fed with data.

The architectures of VLIW DSP processors are in some ways more like those of general-purpose processors than like those of the highly specialized conventional DSP architectures. VLIW DSP processors often omit some of the features that were, until recently, considered virtually part



3. TMS320C62xx execution units and memory architecture.

of the definition of a "DSP processor." For example, the TMS320C62xx does not include zero-overhead looping instructions; it requires the processor to explicitly perform the operations associated with maintaining a loop. This does not necessarily result in a loss of performance, however, since VLIW-based processors are able to execute many instructions in parallel. The operations needed to maintain a loop, for example, can be executed in parallel with several arithmetic computations, achieving the same effect as if the processor had dedicated looping hardware operating in the background. The advantage of the multi-issue approach is a quantum increase in speed with a simpler, more regular architecture and instruction set that lends itself to efficient compiler code generation.

VLIW and superscalar processors often suffer from high energy consumption relative to conventional DSP processors. In general, multi-issue processors are designed with an emphasis on increased speed rather than energy efficiency. These processors often have more execution units active in parallel than conventional DSP processors, and they require wide on-chip buses and memory banks to accommodate multiple parallel instructions and to keep the multiple execution units supplied with data.

Because they often have high memory usage and energy consumption, VLIW and superscalar processors have mainly targeted applications which have very demanding computational requirements but are not very sensitive to cost or energy efficiency. For example, a VLIW processor might be used in a cellular base station, but not in a portable cellular phone. One notable exception is the recently announced VLIW architecture from Lucent's and Motorola's StarCore partnership, the SC140, which is expected to have sufficiently low energy consumption to enable its use in portable products.

SIMD

SIMD, or single-instruction, multiple-data, is not a class of architecture itself, but is instead an architectural technique that can be used within any of the classes of architectures we have described so far. SIMD improves performance on some algorithms by allowing the processor to execute multiple instances of the same operation in parallel using different data. For example, an SIMD multiplication instruction could perform two or more multiplications on different sets of input operands in parallel in a single clock cycle. This technique can greatly increase the rate of computation for some vector operations that are heavily used in multimedia and signal processing applications.

On DSP processors with SIMD capabilities, the underlying hardware that supports SIMD operations varies widely. Analog Devices, for example, modified their basic conventional floating-point DSP architecture, the ADSP-2106x, by adding a second set of execution units that exactly duplicate the original set. The new architecture is called the ADSP-2116x. Each set of execution units in the ADSP-2116x includes a MAC unit, ALU, and shifter, and each has its own set of operand registers. The augmented architecture can issue a single instruction and execute it in parallel in both data paths using different data—effectively doubling its performance in some algorithms.

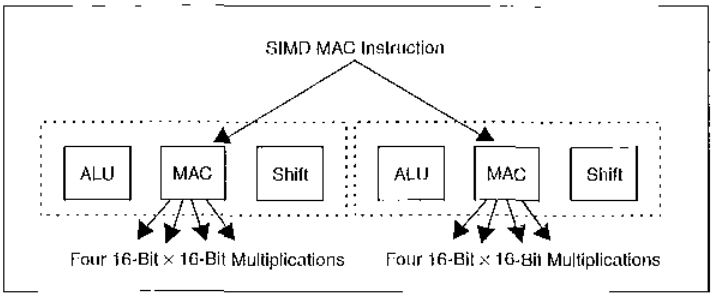
In contrast, instead of having multiple sets of the same execution units, some DSP processors can logically split their execution units (e.g., ALUs or MAC units) into multiple subunits that process narrower operands. These processors treat operands in long (e.g., 32-bit) registers as multiple short operands (e.g., as two 16-bit operands or four 8-bit operands). Perhaps the most extensive SIMD capabilities we have seen in a DSP processor to date are found in Analog Devices' TigerSHARC processor. TigerSHARC is a VLIW architecture and combines the two types of SIMD: one instruction can control execution of the processor's two sets of execution units, and this instruction can specify a split-execution-unit (e.g., split-ALU or split-MAC) operation that will be executed in each set. Using this hierarchical SIMD capability, TigerSHARC can execute eight 16-bit multiplications per cycle, for example. Figure 4 shows TigerSHARC's SIMD capabilities.

Making effective use of processors' SIMD capabilities can require significant effort on the part of the programmer. Programmers often must arrange data in memory so that SIMD processing can proceed at full speed (e.g., arranging data so that it can be retrieved in groups of four operands at a time) and they may also have to re-organize algorithms to make maximum use of the processor's resources. SIMD is only effective in algorithms that can process data in parallel; for algorithms that are inherently serial (for example, algorithms that use the result of one operation as an input to the next operation), SIMD is generally not of use.

Alternatives to DSP Processors

High-Performance CPUs

Many high-end CPUs, such as Pentiums and PowerPCs, have been enhanced to increase the speed of computations associated with signal processing tasks. The most common modification is the addition of SIMD-based instruction-set extensions, such as MMX and SSE for the Pentium, and AltiVec for the PowerPC. This approach is a good one for CPUs, which typically have wide resources (buses, registers, ALUs) which can be treated as



4. TigerSHARC's SIMD features.

multiple smaller resources to increase performance. For example, a CPU with a 64-bit data bus, 64-bit registers, and a 64-bit ALU can be treated as having four times as many 16-bit data buses, registers, and ALUs—resulting in up to four times the performance on 16-bit data (the data size most often used in DSP). Image processing, which tends to be based on 8-bit data, can be sped up even further. Using this approach, general-purpose processors are often able to achieve performance on DSP algorithms that is better than that of even the fastest DSP processors. This surprising result is partly due to the effectiveness of SIMD, but also because many CPUs operate at extremely high clock speeds in comparison to DSP processors; high-performance CPUs typically operate at upwards of 500 MHz, while the fastest DSP processors are in the 200-250 MHz range. Given this speed advantage, the question naturally arises, "Why use a DSP processor at all?"

There are a number of reasons why DSP processors are still the solution of choice for many applications. Although other types of processors may provide similar (or better) speed, DSP processors often provide the best mixture of performance, power consumption, and price. Another key advantage is the availability of DSP-specific development tools and off-the-shelf DSP software components. And for real-time applications, the superscalar architectures and dynamic features common among high-performance CPUs can be problematic.

DSP/Microcontroller Hybrids

There are many lower-cost general-purpose processors, referred to as "microcontrollers," that are designed to execute control-oriented (decision-making) tasks efficiently. These processors are often used in control applications where the computational requirements are modest but where factors that influence product cost and time-to-market, such as low program memory use and the availability of efficient compilers, are important.

Many applications require a mixture of control-oriented software and DSP software. An example is the digital cellular phone, which must implement both supervisory tasks and voice-processing tasks. In general, microcontrollers provide good performance in controller tasks and poor performance in DSP tasks; DSP processors have the opposite characteristics. Hence, until recently, combination control/signal processing applications were typically implemented using two separate processors: a microcontroller and a DSP processor. In recent years, however, a number of microcontroller vendors have begun to offer DSP-enhanced versions of their microcontrollers as an alternative to the dual-processor solution. Using a single processor to implement both types of software is attractive, because it can potentially:

- Simplify the design task,
- Save circuit board space,
- Reduce total power consumption, and
- Reduce overall system cost.

Microcontroller vendors such as Hitachi, ARM, and Lexra have taken many different approaches to adding DSP functionality to existing microprocessor designs, borrowing and adapting the architectural features common among DSP processors. Many of these hybrid processors achieve signal processing performance that is comparable to that of low-cost or mid-range DSP processors while allowing re-use of software written for the original microcontroller architecture.

Conclusions

DSP processor architectures are evolving to meet the changing needs of DSP applications. The architectural homogeneity that prevailed during the first decade of commercial DSP processors has given way to rich diversity. Some of the forces driving the evolution of DSP processors today include the perennial push for increased speed, decreased energy consumption, decreased memory usage, and decreased cost, all of which enable DSP processors to better meet the needs of new and existing applications. Of increasing influence is the need for architectures that facilitate development of more efficient compilers, allowing DSP applications to be written primarily in high-level languages. This has become a focal point in the design of new DSP processors because DSP applications are growing too large to comfortably implement (and maintain) in assembly language. As the needs of DSP applications continue to change, we expect to see a continuing evolution in DSP processor architectures.

Jennifer Eyre is a DSP Analyst at Berkeley Design Technology, Inc. (BDTI), a DSP technology analysis and software development firm. She received her BSEE and MSEE degrees from University of California at Los Angeles. She is a member of Eta Kappa Nu.

Jeff Bier is a Cofounder and General Manager of BDTI. He received engineering degrees from Princeton University and University of California at Berkeley. He is a member of the IEEE Design and Implementation of Signal Processing Systems (DISPS) technical committee.

References

[1] Lapsley et al., *DSP Processor Fundamentals: Architectures and Features*. New York: IEEE Press, 1996.

[2] R.G. Lyons, *Understanding Digital Signal Processing*. Reading, MA: Addison Wesley, 1996.

[3] E.A. Lee, "Programmable DSP architectures, part I," *IEEE Acoust., Speech, Signal Processing Mag.*, vol. 5, Oct. 1988.

[4] E.A. Lee, "Programmable DSP architectures, part II," *IEEE Acoust., Speech, Signal Processing Mag.*, vol. 6, pp. 4-14, Jan. 1989.

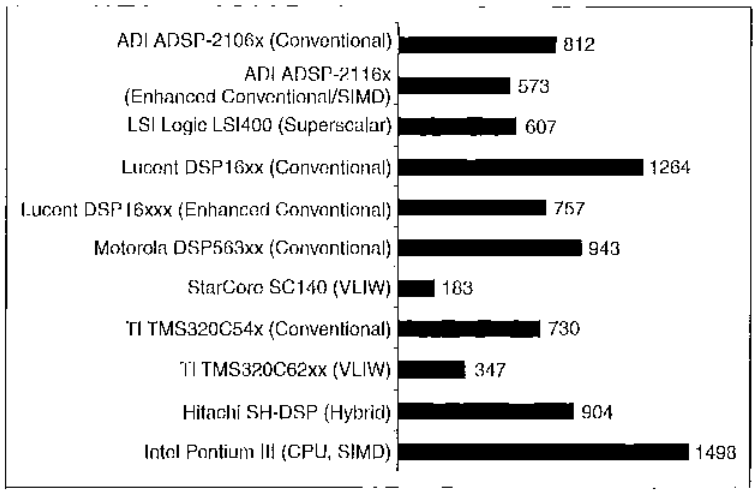
[5] W. Strauss, "DSP strategies 2000," Forward Concepts, Tempe, AZ, 1999.

[6] J.L. Hennessy and D.A. Patterson, *Computer Architecture a Quantitative Approach*. San Mateo, CA: Morgan Kaufmann, 1996.

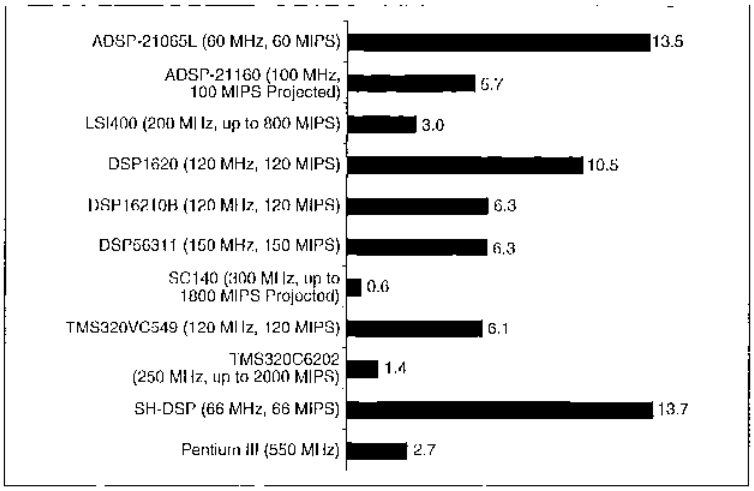
[7] Berkeley Design Technology, Inc., *Buyer's Guide to DSP Processors*. Berkeley Design Technology, Inc., 1994, 1995, 1997, 1999.

Comparing Performance

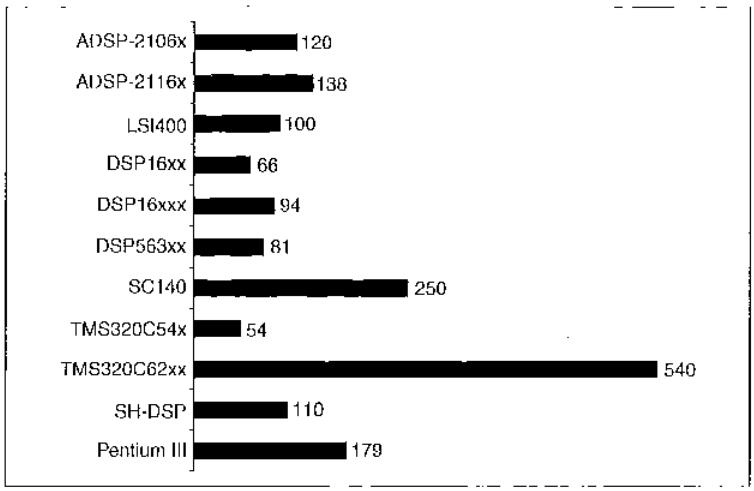
One of the key characteristics of a processor is its performance. Performance can be evaluated in several dimensions, including the number of cycles required to execute a given task (which indicates architectural efficiency), speed, energy consumption, and memory usage. In an effort to provide meaningful comparisons of processor performance, BDTI has developed its own suite of DSP benchmarks, which we have implemented and hand-optimized in assembly language on virtually every commercial DSP processor currently available. In Figs. 5 through 7, we present sample DSP benchmark results for a selection of processors, illustrating some of the concepts we have discussed earlier. Note, for example, that the VLIW DSP processors execute BDTI's block FIR filter benchmark using far fewer cycles than the conventional DSP processors and require much less time to execute the benchmark, but require more program memory. Note also that when comparing processor's speeds, MIPS ratings (that is, millions of instructions per second, not Dhrystone MIPS) are poor indicators of relative performance, because they do not measure how much work is accomplished in each cycle [7]. For example, VLIW processors tend to have high MIPS ratings relative to conventional DSP processors because they issue and execute multiple instructions per cycle, but their performance is generally lower than what might be expected based on their relative MIPS ratings because they use simple instructions and require many more of them to execute a given task. As DSP architectures diversify, comparing performance across architectures becomes more difficult—but it is a crucial part of choosing a processor for an application.



5. Cycle counts for BDTI's block FIR filter benchmark.



6. Execution times for BDTI's block FIR filter benchmark (microseconds).



7. Program memory usage for BDTI's block FIR filter benchmark (bytes).