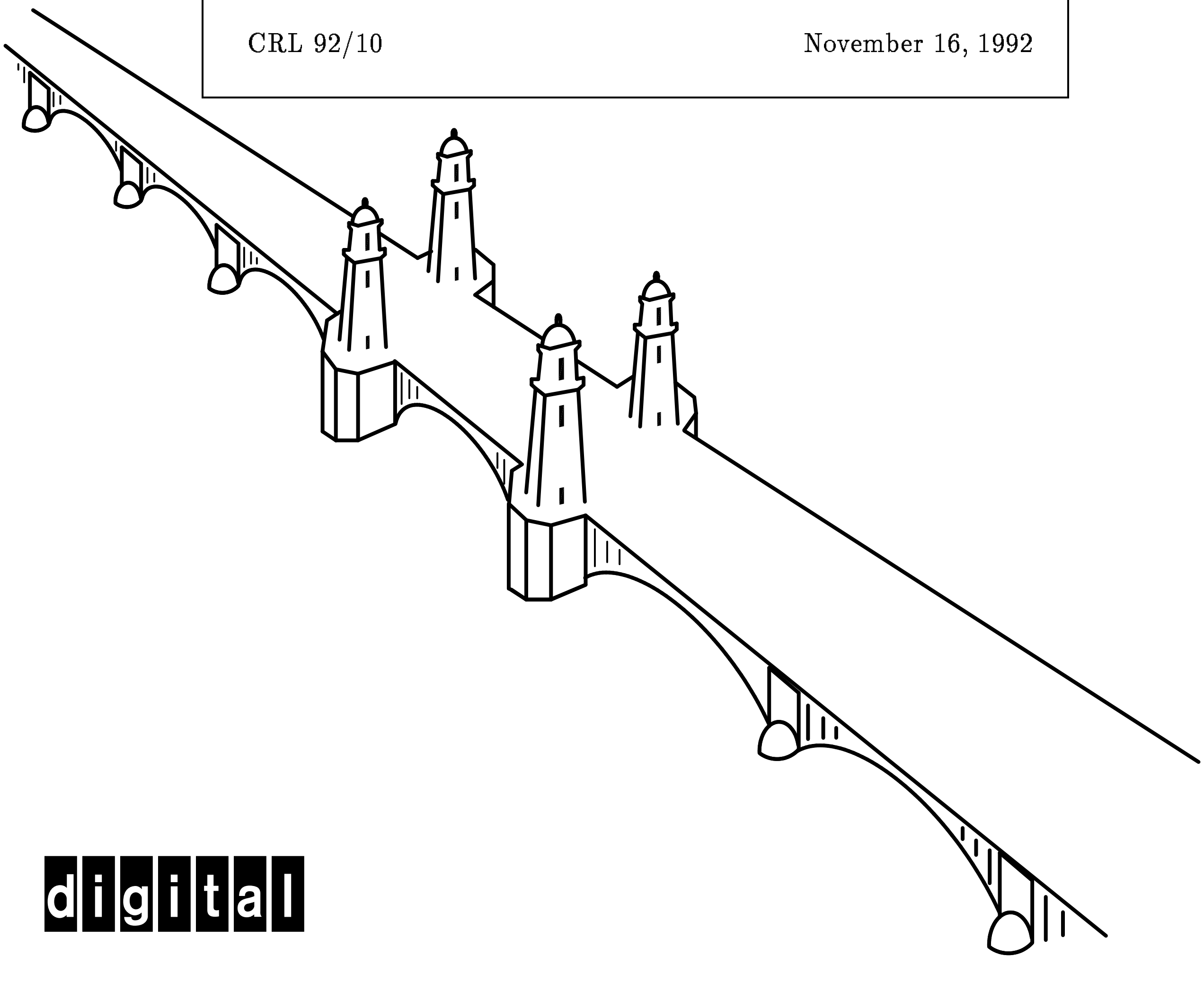
Are DSP Chips Obsolete?

Lawrence C. Stewart
Andrew C. Payne
Thomas M. Levergood

Digital Equipment Corporation
Cambridge Research Lab

CRL 92/10 November 16, 1992



digital

CAMBRIDGE RESEARCH LABORATORY
Technical Report Series

Are DSP Chips Obsolete? [1]

Lawrence C. Stewart
Andrew C. Payne
Thomas M. Levergood

Digital Equipment Corporation
Cambridge Research Lab

CRL 92/10 November 16, 1992

Abstract

Today's fastest general purpose microprocessors have performance equaling or exceeding that of the fastest available DSP chips. This paper argues that signal processing applications will migrate from specialized DSP coprocessors into the host processor of high performance personal computers and workstations. DSP chips will have a place in synchronous, low latency data handling, but increasingly, the computation of signal processing algorithms will be handled by general purpose processors.

Keywords: digital signal processing, RISC

[1] A version of this paper appeared in the *International Conference on Signal Processing Applications and Technology* on November 2-5, 1992.

1 Introduction

The price/performance of computer hardware has been improving at a rate of about 50 percent per year. Each year, the same amount of money will buy a machine 50 percent faster than the year before. This trend is driven by semiconductor technology and shows no sign of changing well into the future. In the past 10 years, personal computers have improved in price/performance by about a factor of 60. In the next 10 years, we will see another sizeable factor - and many industry analysts think that 50 percent per year is a conservative estimate. Regardless of the exact figure, a revolution is coming.

Today, personal computers and workstations rely on external devices (such as modems) or DSP coprocessor boards to handle the processing requirements of signal processing algorithms. We argue that these devices will be replaced by simple - and less expensive - I/O devices, and that the signal processing algorithms will be executed by the host processor.

Today's fastest implementations of general purpose processor architectures, such as the Digital Alpha AXP[1], the HP PA-RISC, and others, have performance equaling or exceeding the fastest available DSP chips. Next generation personal computers based on the forthcoming Intel P5, for example, will also have enough computational horsepower for many of today's DSP applications.

This paper argues that many signal processing applications, from modems to speech synthesis and recognition, will be handled by time-shared general purpose processors, while the DSP, if present at all, will be relegated to handling the interrupts and data buffering requirements of the I/O system.

Our thesis is predicated on two points. First, the performance of general purpose CPUs on signal processing tasks easily matches the performance of more specialized DSPs. Second, many applications which appear to require dedicated signal processing can be implemented in a way which does not require the the low-latency computation provided by a dedicated DSP. (We also note that real-time capabilities are turning up in general purpose operating systems such as OSF/1 - so perhaps low-latency applications are viable as well.)

[1] The AXP mark is a trademark of Digital Equipment Corporation.

2 Performance

The present state of the art in floating point DSP chips may be represented by the Texas Instruments 320C40. The 320C40 can execute at a peak rate of 150 MIPS [2] [TI91]. In contrast, the current state of the art in general purpose CPUs may be represented by the Digital Equipment Alpha AXP 21064 processor [DEC92], which can execute at a peak rate of 400 MIPS.

At somewhat lower levels of absolute performance and substantially lower prices, general purpose chips such as the MIPS architecture integrated R3500A from IDT [IDT91] have enough horsepower for such tasks as speech synthesis. In our laboratory, we have demonstrated the DECtalk speech synthesizer as a purely software algorithm running in real-time while consuming about 37% of a 25 MHz R3000 processor. This work is discussed in more detail later in this section.

Many of the special architectural features that allow DSP processors to deliver high performance also exist in modern general purpose processors. However, general purpose processors are missing most of the awkward features that make DSPs hard to use. The net result is that modern general purpose processors offer performance equal to or greater than today's DSPs, while remaining substantially easier to use.

2.1 Architecture

Digital Signal Processors traditionally have a few unique architectural features that have set them apart from general purpose processors:

- Single cycle multiplier
- Multiple operations per cycle
- Bit reversed addressing
- Circular buffers
- Zero overhead looping

These features are designed to perform well for signal processing algorithms like convolutions and Fast Fourier Transforms. With the exception

[2] At 25 Mhz, the 320C40 can perform six operations per cycle: 2 data access, 2 address register updates, 1 FP multiply and 1 FP ALU operation.

of bit reversed addressing, all of these architectural features are matched by mechanisms that appear in recent general purpose implementations.

Recent general purpose implementations offer very good floating point performance. A fully pipelined floating point unit can issue one floating point operation (multiply or accumulate) per cycle [DWYF92, MHR90]. Some architectures have a multiply/accumulate instruction that can effectively perform two FP operations per cycle (IBM RS/6000 and HP PA-RISC 1.1.) There is typically a several cycle latency until the result of a operation is available, but these latencies can be hidden with *software pipelining* - which is possible due to the large number of registers in general purpose architectures [HP90, Smi92].

Most DSPs can perform several operations in a single cycle. The ability to fetch two operands and perform a multiply/accumulate gives good performance for most signal processing algorithms. Many recent general purpose processor implementations are *superscalar*: they can issue multiple instructions per processor cycle [HP90]. For example, Digital's 21064 can issue two instructions per cycle. IBM's RS/6000 and Texas Instruments' SuperSPARC [BK92] can issue three instructions each cycle.

Superscalar general purpose processors are also more powerful than DSPs in the combinations of operations that can be performed in a single cycle. Most DSPs have very structured parallel operation instructions that can perform a limited number of parallel combinations. In contrast, most superscalar general purpose implementations contain instruction issue units which fetch ahead in the instruction stream and aggressively issue multiple instructions as long as the necessary resources (registers and functional units) are available. For example, Digital's 21064 has four functional units: load/store, branch, integer, and floating point, and in general can issue two instructions to different functional units. IBM's RS/6000 can issue three instructions to branch, fixed point, and floating point functional units.

Several general purpose architectures have been designed to facilitate multiple issue implementations, but multiple issue is not a necessary part of the architecture. Consequently, future implementations may issue more instructions per cycle as implementation technology improves. In contrast, most DSPs use special instruction encodings to specify parallel operations. Therefore, changing the form of the parallel operations requires creating a new, incompatible architecture. For general purpose architectures, only the *implementation* needs to be changed, the architecture can remain the same (and backwards compatible.) For example, the HP-PA architecture has binary compatible single and multiple issue implementations, while TI's four generations of DSP processors all have incompatible instruction sets.

Another popular feature for DSPs is zero overhead looping, that allows a block of instructions to be repeatedly executed with no looping overhead. One popular implementation uses two registers: a count and the end address of the block to repeat. No branch instruction is required to iterate the loop. General purpose implementations can achieve zero loop overhead through other mechanisms. IBM's RS/6000 has a special count register with a decrement-and-branch-if-non-zero instruction. Since the RS/6000 can issue a branch together with one integer and one floating point instruction, the looping branch instruction is effectively free. Other general purpose architectures may require an instruction to update the loop index. The overhead is minor: in multiple issue implementations the update instruction can be scheduled in an unused issue slot. The overhead can be reduced even further by unrolling loops and by software pipelining (again, possible because of the large number of registers available on general purpose machines.) Modern general purpose processors use branch prediction logic to reduce looping overhead. This mechanism allows the instruction fetch unit to feed the correct sequence of instructions to the processor pipeline with no delays.

A final architectural feature unique to DSPs is circular buffering. This feature increments or decrements a buffer index and automatically wraps the index based on the length of the buffer, and is convenient for implementing delays and convolutions. This feature can be easily implemented on general purpose machines using, for example, the conditional move instruction available on Sun's SPARC and Digital's Alpha AXP architectures. A decrement-modulo-N requires only two instructions (and no branches):

```
sub     r0, #1, r0   ;r0 = r0 - 1
cmovelt r0, #127, r0 ;if (r0 < 0) then r0 = 127
```

General purpose processors are missing most of the DSP's awkward features, such as a Harvard architecture with separate program and data memories. Some DSPs such as Motorola's 96K [Mot89] use this memory architecture to achieve performance, but place the burden of managing the memory on the programmer. The size of program and data memory are fixed and complicated memory models or qualifiers are required for memory access [Mot92].

General purpose machines can achieve the same memory performance with split instruction and data caches. Unlike DSPs, this split is an implementation detail: there is no distinction made at the architectural level (instruction set) between program and data memory. All memory is alike, relieving the programmer of the management burden.

Another common DSP feature is fast on-chip memory. While this memory provides single cycle access, it is the programmer's responsibility to allocate code and data between this memory and the slower off-chip memory. While some general purpose implementations have on-chip memory, most use caches to improve performance. Like the on-chip memory, the cache provides single cycle access, but is automatically managed by the hardware. If a datum is available in the cache, it is read or written in a single cycle. Otherwise, a cache miss occurs and the processor automatically goes off chip to retrieve the datum.

Cacheing helps average performance, but may perform badly on some portions of a program due to cache conflicts, which occur when two or more sections of code or data displace each other from the cache. However, the same programming style that allocates data to X and Y memories (such as for Motorola's DSP 56000) can be used to arrange cached data so that it does not collide and near optimal performance can be achieved. Similar techniques can be used to manage instruction stream cacheing. It is therefore possible to get predictable performance from a cached system. The programming environment for the MIPS architecture even has an automatic tool (CORD) for reorganizing code to minimize cache conflicts. CORD uses profile information gathered at runtime to guide its modifications of the program.

Modern DSPs and general purpose processors have about the same amount of on chip memory, but they put the memory to different uses. DEC's 21064 has 16K bytes of on-chip cache (split between instruction and data caches.) TI's 320C40 has 8K of on-chip memory and a 512 byte cache. But the 320C40's on-chip memory must be managed by the programmer, so the software must be changed for different model chips. In contrast, because the general purpose cache is automatically managed, different implementations can offer different size caches without requiring code changes.

One area in which it might be argued that general purpose processors are at an architectural disadvantage compared to DSPs is that general purpose processor instructions tend to do less work than DSP instructions. This effect is countered by the ability of superscalar general purpose machines to issue multiple instructions per cycle, but it does lead to somewhat larger programs. The requirement for compilers to unroll loops and to use software pipelining to achieve peak performance also tends to result in a larger number of instructions for the general purpose architecture than for the DSP.

However, modern general purpose implementations have sufficiently large instruction caches coupled with instruction prefetch units that can anticipate cache misses, so that the somewhat larger programs do not result in reduced performance.

2.2 Performance

We have argued that modern general purpose processors can approach DSPs on cycle-per-cycle performance. However, most general purpose processor implementations have very short cycle times that put them ahead of DSPs in throughput. Table 1 compares the instruction cycle times for several general purpose and DSP chips.

Device	Cycle Period (ns)
DEC 21064	5
HP 7100	10
Motorola 96K	30
TI 320C40	40
TI 320C30	60

Table 1: Cycle Times

To verify our theory that general purpose processors have good performance for signal processing algorithms, we implemented a number of popular algorithms on general purpose processors and compared them to DSP performance reported in the literature. It is important to remember that our implementations may not be the best possible for the general purpose processor, but instead represent lower bounds on achievable performance.

Fast Fourier Transform

Our first benchmark was a 1024 point complex FFT. Table 2 compares the execution time on the Digital 21064 Alpha AXP with the best reported times for other DSP chips [MS90].

Device	Cycles	Cycle Period (ns)	Transform Time (μs)
DEC 21064	96,000	5	480
TI 320C40	38,629	40	1545[3]
TI 320C30	40,457	60	2430[4]

Table 2: 1024 Point Complex FFT Performance

FIR Convolution

As another example, we tested an FIR convolution. On the 21064, an N point convolution requires 23+2N cycles. Table 3 compares the convolution performance of the 21064 with other DSP chips:

Device	Cycles	Cycle Period (ns)	Convolution Time (ns)
21064	$23 + 2N$	5	$115 + 10N$
TI 320C40	$7 + N$	40	$280 + 40N$
TI 320C30	$10 + N$	60	$600 + 60N$

Table 3: N-Point Convolution Performance

In fact, for FIR convolution, the 21064 achieves peak performance of 400 MIPS and 200 MFLOPS when the filter coefficients fit in the on-chip cache.

Speech Synthesis

The previous sections reported on DSP algorithm performance running on a new state-of-the-art general purpose microprocessor. We have also explored the performance of an important signal processing application running on a low cost commodity micro- processor - speech synthesis on a MIPS R3000 processor.

Developed by Dennis Klatt and Digital Equipment Corporation, DECtalk is an algorithm that accepts arbitrary text as input and produces sampled synthetic speech using as many as eight internal voices (speakers) [Kla87]. DECtalk comprises three major algorithms; the **letter-to-sound system**, the **phonemic synthesizer**, and the **vocal tract model**.

The letter-to-sound system accepts arbitrary ASCII text as input and produces a phonemic transcription as output. This component uses a pro-

[3] From TI's databook [TI91].
[4] From [MS90].

nunciation lexicon and a collection of letter-to-sound rules for English to convert text to phonemes with lexical stress markings.

The phonemic synthesizer accepts the phonemic transcription output from the letter-to-sound system and produces parameter control records for the vocal tract model. Each parameter control record produced results in a 6.4 ms frame of synthesized speech.

The vocal tract model accepts the control records from the phonemic synthesizer and updates its internal state in order to produce the next frame of synthesized samples. The vocal tract model is a formant synthesizer based on the model described by Klatt [Kla80]. The vocal tract model consists of voiced and unvoiced waveform generators, cascade and parallel resonators, and a summing stage. Frequency, bandwidth, and gain parameters in the control record are used to compute the filter coefficients for the resonators. The vocal tract model also receives speaker definition parameter records which are used to update the internal state specific to a particular speaker. The vocal tract model operates at an 8 KHz sampling rate.

DECtalk is an algorithm that has many implementations. In this system, DECtalk is implemented in 'C' source code and is compiled using the native Ultrix/RISC compilers using default optimizations. The tests were performed on a system running V4.2a ULTRIX and V2.10 of the compiler.

The performance data were collected on a DECstation 5000 Model 200 workstation. This workstation uses a 25 MHz R3000 MIPS CPU with 128 KB (I+D) of external cache memory. All components of the DECtalk algorithm execute on the workstation; no DSP coprocessor is needed to achieve real-time synthesis.

Performance measurements were made by synthesizing samples from an ASCII text file and measuring the execution times for the three components. To synthesize 43.9 seconds of speech, the vocal tract model, phonemic synthesizer, and letter-to-sound system require 15.4, 0.98, and 0.032 seconds of CPU time. On average, this algorithm requires 370 milliseconds of CPU time to produce 1 second of synthesized speech. In other words, an R3000 is nearly three times faster than necessary to synthesize speech. In fact, all computations were done in fixed-point, and there is reason to believe that converting the algorithm to floating point would result in still greater performance, since the R3000 floating point multiply is substantially faster than the integer multiply instruction.

2.3 Development Environments

The separation of architecture from implementation in most general purpose processors results in improved software longevity. Because the architecture is stable across several generations of implementations with increasing performance, software for general purpose processors can remain useful for many years. Historically, general purpose architectures have enjoyed much longer lifetimes than their DSP counterparts.

The advantages of a single architecture are obvious: there is no learning curve for next year's model, no re-writing of assembly language code, and pin compatible upgrades. This is especially important since software development is frequently the dominant cost of a system.

Software development costs are also reduced by the use of higher level languages in preference to assembler. Programmers tend to be more productive when using a high level language, and the resulting software is easier to debug and maintain. Historically, DSP software has been written primarily in assembler, although in recent years the main program is frequently written in a high level language with the inner loops written in assembler. In contrast, almost all code for general purpose processors today is written in high level languages. This trend is supported by the availability of very good optimizing compilers for general purpose processors [CHKW86].

Optimizing compiler technology for general purpose architectures has progressed farther and faster than that for DSP architectures. This has happened for two reasons. First, it is easier to generate code for the very regular instruction sets of general purpose architectures than for the sometimes peculiar instructions of DSPs. Second, competitive pressures of the general purpose processor market force vendors and third party compiler writers to concentrate on performance. In addition, while DSP parts ship in large volumes, the number of DSP programs written is small compared to the number of programs written for general purpose processors. This adds to the market demand for high quality compilers and software development tools.

Of course, ease of software development is not the whole story. The design of a system is usually based on minimizing the sum of development, manufacturing, and maintenance costs over the life of the system. Developers are willing to code in obscure minimalist assembler in order to reduce the manufacturing cost of a high volume product - at the expense of development and maintenance costs. However, as signal processing applications become more common, the industry must do what it can to make signal processing software easier to develop.

3 Is Real Time Necessary?

Section 2 of this paper presented the case that general purpose processor architectures are competitive with DSP architectures, and that the extremely short cycle times of modern general purpose processor implementations actually gives them higher throughput than a modern DSP. This section presents the case that signal processing in the time shared host processor of a workstation or PC is as effective as the dedicated processing of a DSP.

Signal processing is generally done in lockstep with I/O with a fixed (and short) delay from input to output. In contrast, the operations of general purpose processors running multiple tasks is bursty. Each task runs for a while and is then suspended in favor of another higher priority task. This results is a variable (and longer) delay from input to output.

Our argument in favor of general purpose processors is based on the idea that "real-time" is application dependent, and that for many applications of interest, the larger latency of a signal processing application does not cause a problem.

3.1 Why is latency higher on a general purpose processor?

General purpose processors will handle signal processing tasks in larger chunks for two reasons. First, they must devote some resources to other tasks, and second, getting good performance from a modern general purpose processor requires allocating each task a sufficiently large run interval to amortize the cost of switching processes.

It will be the exception rather than the rule for a signal processing application to be the only task competing for the attention of a PC or workstation. It is not at all uncommon for a workstation to have thirty or forty processes running at any given time and even on a PC, device interrupts and screen updates will use some processing power.

When the processor must be shared among several activities, it is evident that larger time slices will use the machine more effectively than small ones, simply because there will be less overhead in switching.

Even if the overhead of switching is small, a given algorithm will use less total run time if it is run for large blocks of data than for small ones. This effect happens because of caches. A DSP is designed so that most or all memory references will be single-cycle. In contrast, modern general purpose machines use a small automatically managed cache to hold frequently used code and data. The first time a datum is referenced, it is fetched from main memory, sometimes with a substantial delay. Later references to the datum, or to nearby data that is fetched at the same time, are handled very quickly from the cache. However, when a task begins to run, it experiences a "cold start" effect until its working set is loaded into the cache. The cold start can be effectively amortized by handling larger blocks of data per invocation of a task.

3.2 Low latencies are usually not a concern

There are really two aspects to "real-time". The first is whether the processor has adequate performance to handle one time unit's worth of input data in less than one time unit. A certain level of performance is necessary or the work cannot be done on time. The second aspect is latency - what is the delay between input and output?

In many applications, latency is not a concern. For example, a fax modem could be implemented with arbitrary latency, since the transmit waveform can be precomputed, and the received signal can be recorded as a waveform and processed off-line. Speech synthesis, for example in an application reading electronic mail, could operate satisfactorily with latency of one second. Depending on the application, a speech recognition application might also operate with substantial delay.

In general, the latency required by a signal processing application is controlled by the ultimate consumer of the data. While an interactive application might require delays of a hundred milliseconds or less, a batch application, such as the reception of documents via fax, can operate with delays of minutes. Still further along the spectrum of latency, an application using speech recognition to establish an index for voice mail might give perfectly satisfactory operation overnight.

Some applications may even benefit from the relaxation of latency constraints. Consider the example of a V.32 modem. A DSP implementation typically operates with a processing delay of a few milliseconds. This means that only a limited amount of processing can be applied to each sample, and that ancillary processes such as echo canceller training and line equalization must be performed "off line" before data can be transmitted.

If a longer, or variable processing delay is acceptable, parameterized soft decision decoding algorithms such as the M,L algorithm can be used, which benefit from more processing during periods of lower signal to noise ratio that during periods of high SNR [JA71]. In addition, if the received waveform is recorded, then echo canceller training and line equalization may proceed jointly and in parallel with data transmission, allowing better line utilization and lower communications costs.

Delay budgeting

It is customary in the design, for example, of optical datalinks to engineer the link with a certain transmitted optical power and a certain required received power to achieve a low error rate. This power ratio is then carefully budgeted to different components, such as transmitter aging, fiber transmission losses, connector losses, and so forth, leaving adequate margin for uncertainly.

The same kind of process is followed in the design of real-time systems, where the total allowable input-output delay is carefully allocated to different components. One classic example of this process is in the telephone industry, where years of human factors work has established the public requirement for maximum round-trip delay of a telephone call as somewhere around 300 milliseconds. This delay is then allocated to different parts of the telephone system. One consequence is that frequently only one direction of a phone call will be carried by satellite, because two satellite hops will exceed the delay budget. The other half of the call will be carried on terrestrial facilities.

This same style of engineering can be used to first identify the actual input to output delay requirements of signal processing applications and then to assure that the requirements are met. In many cases, this analysis reveals that there is no need for sample by sample processing in a dedicated DSP, but instead "batch" processing by the host computer will be perfectly satisfactory.

Examples

One interesting line of development is to consider the connection of telephony and the personal computer. Today, most PC users of telecommunications have specialized fax and data modems, and occasionally an interface for voice communications as well. These devices permit the personal computer to transmit and receive data (via modem), documents (via fax) and perhaps voice mail (via waveform recording.) It is worth considering the possibility of replacing all three of these devices by a single device containing only an A/D converter, a D/A converter, and a telephone line data access arrangement

(DAA.) This device, plus software to emulate the fax and data modems, could accomplish all three tasks at a lower cost than any single task today.

Use of the telephone interface for voice mail is particularly simple, as there really is no signal processing required. The only functions required are the recording and playback of digital voice waveforms. If desired for reasons of storage economy, speech compression algorithms may be used, and these may introduce additional timeliness requirements for decompression, since the user may request playback of a recorded message at any time. The compression side has no particular timeliness requirement, since the message may be stored uncompressed until sufficient time is available to do the compression - even overnight.

The fax modem presents a slightly more difficult case. The operation of a fax session has two phases. In the first phase, the two modems follow a protocol to establish the degree of compatibility between them. This part of the process is "interactive" in the sense that the protocol requires a particular pattern of signalling to occur within certain time limits. Once this protocol negotiation phase is over, fax transmission becomes a half-duplex transmission from sender to receiver. During this phase of operation, there really are no "interactive" requirements. The sender is free to precompute the transmit waveform as far ahead as desired, and the receiver is free to simply record the received waveform and to process it at leisure.

In many ways, the operation of a data modem comes the closest to traditional "real-time" requirements - not because low delay operation is inherent in the data modem protocols, but because the ultimate user of the data may be an interactive computer user whose requirements include small round-trip delays for keyboard echoing. Even in this case, the actual delay requirements for a data modem are related to human perceptible time scale, rather than requiring sample by sample processing.

4 Conclusion

What then are DSPs good for?

Aside from their performance related internal architectural features, which we discussed in the first section of this paper, modern DSPs are distinguished by the ease which they may be applied to data handling. One common feature of the DSP is an extremely capable serial port, often coupled with low-latency interrupt facilities which permit, but do not require, program intervention on every sample. Other new DSP designs have very flexible interfaces for permitting multiprocessor systems to be built.

DSPs are often used in applications which exploit their data handling features rather than computation. For example, DSPs are often used for data reduction, where simple signal processing is performed to reduce the output data rate. We expect this trend to continue, with DSP like devices used as data handling front end processors - even just as smart DMA engines - while fast general purpose processors handle the computation.

Summary

General purpose processors are faster, and easier to program than DSPs. In applications where very low latency is not required, we think that signal processing will be done in the central processor of a PC or workstation, rather than in DSP coprocessors. The remaining market for DSP chips will be in truly dedicated applications such as sample-rate changing in real-time digital audio, and in video compression. Even in these low-latency application areas, the use of general purpose processors will increase as real-time capabilities are added to their operating systems. We urge the DSP industry to concentrate on the design of flexible and efficient data handling devices rather than on signal processing. In particular, there is a need for components with the communications and data handling capabilities of today's DSPs, but without the expensive computational core. Another intriguing possibility is to combine the data handling capabilities of a DSP with a general purpose computational core. This would bring the performance, architectural, and software development advantages of general purpose processors to bear on many new problems.

References

[BK92] G. Blanck and S. Krueger. The SuperSPARC microprocessor. In *COMPCON*, pages 136–141, 1992.

[CHKW86] F. Chow, M. Himelstein, E. Killian, and L. Weber. Engineering a RISC compiler system. In *COMPCON*, pages 132–137, 1986.

[DEC92] Digital Equipment Corporation. *DECchip 21064-AA RISC Microprocessor Data Sheet*, Apr., 1992. Available for anonymous ftp from `crl.dec.com`.

[DWYF92] E. DeLano, W. Walker, J. Yetter, and M. Forsyth. A high speed superscalar PA-RISC processor. In *COMPCON*, pages 116–121, 1992.

[HP90] J. L. Hennessy and D. A. Patterson. *Computer Architecture, A Quantitative Approach*. Morgan Kaufman, San Mateo, California, 1990.

[IDT91] Integrated Devices Technology. *RISC 1991 Databook*, 1991.

[JA71] F. Jelinek and J. B. Anderson. Instrumentable tree encoding of information sources. *IEEE Transactions on Information Theory*, 17(1):118–119, January 1971.

[Kla80] Dennis H. Klatt. Software for a cascade/parallel formant synthesizer. *J. Acoust. Soc. Am.*, 67(3):971–995, 1980.

[Kla87] Dennis H. Klatt. Review of text-to-speech conversion for English. *J. Acoust. Soc. Am.*, 82(3):737–793, 1987.

[MHR90] R. K. Montoye, E. Hokenek, and S. L. Runyon. Design of the IBM RISC System/6000 floating-point execution unit. *IBM Journal of Research and Development*, 34(1):59–70, January 1990.

[Mot89] Motorola. *DSP96002 IEEE Floating-Point Dual-Port Processor User's Manual*, 1989.

[Mot92] Motorola. *GCC56K User's Manual*, 1992.

[MS90] R. Meyer and K. Schwarz. FFT implementation on DSP-chips – theory and practice. In *Proceedings ICASSP*, pages 1503–1506, 1990.

[Smi92] M. R. Smith. To DSP or not to DSP: Will a RISC chip do it better? *Circuit Cellar Ink*, 28:14–25, August 1992.

[TI91] Texas Instruments. *TMS320C4x User's Guide*, May 1991.