James A. Moorer
Lucasfilm Ltd.
P.O. Box 2009
San Rafael, California 94912

The Lucasfilm Audio Signal Processor

Introduction

The requirements of audio processing for motion pictures present several special problems that make digital processing of audio very desirable and also relatively difficult. The difficulties can be summarized as follows:

1. Large amounts of numerical computation are required, on the order of 2 million integer multiply-adds per second per channel of audio, for some number of channels.
2. The exact processing involved changes in real time but must not interrupt the flow of audio data.
3. Large input/output (I/O) capacity is necessary, simultaneous with numerical calculation and changes to the running program, on the order of 1.6 million bits per second per channel of audio.

To overcome these difficulties, the digital audio group at Lucasfilm is building a number of audio signal processors, the architecture of which reflects the special problems of audio.

Motivation: What Is the Problem?

The sound in motion pictures is usually divided into three and sometimes four categories: dialogue, music, and sound effects. Additionally, sound effects are sometimes divided into two further categories, *Foley effects* and *special effects*. Foley effects (after Jack Foley, who was with Universal Studios during the 1930s) are human nonvocal noises, such as footsteps. Special effects include pistol shots, explosions, and everything else.

Although dialogue is generally recorded at the set, it is most often not of suitable quality and must be recreated in the studio. This entails an actor watching a print of the film and speaking the lines in synchrony with the film. The actor, however, usually hears little or no other sound cues, such as other voices, music, or sound effects while he or she is trying to speak the lines. Needless to say, it requires actors of substantial talent to give convincing performances under these conditions. Some actors are unable to do this, necessitating use of recordings from the set regardless of their quality.

Consider now the problem faced by dialogue editors. They are presented with stacks of recordings, some from a studio (an acoustically dead environment), some from a stage (which can be very reverberant), and some garbled or poorly recorded (with the actor facing away from the microphone, for instance). The editors must make all these recordings sound as if they are coming from the same environment and as if they are coming from the environment that the viewer is seeing on the screen. This consistency and apparent authenticity is usually accomplished with the use of ad hoc combinations of signal processing devices such as filters, reverberators, modulators, and other tools of the analog audio studio. The art of the dialogue editor lies in the skillful use of such equipment.

Sound effects production is a complete art in itself. Take the simple example of a "nose crunch." A real fist hitting a nose sounds a lot like someone hitting a side of beef. Sometime in the 1930s this was deemed to be insufficiently spectacular for movies, and since that time nose crunches (as well as all other sound effects) have been made by combining several sounds. There is a smack, sometimes taken from the sound of a pistol shot, and then a crunch, usually made by dropping a watermelon out of a second-story window. Other sounds, like the whizzing of the fist through the air, are added to produce the final sound. Often, signal processing is added, the most common forms probably being pitch shifting and phasing (feedback delay lines). Each sound must then be synchronized with the picture, which is typically done by splicing bits of

Computer Music Journal, Vol. 6, No. 3, Fall 1982,
0148-9267/82/030022-11 $04.00/00

sound (with silence in between) into reels. As if this were not enough, the sound editors are generally doing this work while the film editors are still making changes in the movie. Needless to say, when a film editor decides to, for instance, delete three frames from a scene, the poor sound editor must go through every reel of sound for that scene and cut out three frames. This is not so bad in most cases, but if a scene that has a background noise (like a factory or a train) is lengthened, generally it means that that sound must be reedited (copied and spliced anew) to fit properly.

Each sound editor (of music, dialogue, sound effects) will then *mix down* (combine) several reels of sound into single, composite reels. Each editor shows up at the final mix session with a stack of reels of sound for each reel of film. This occasion is often the first on which anyone has heard the music, the dialogue, and the sound effects all together. It is usually scheduled for just a few months before the film's release, and much too late to change anything substantial. The final mix usually has several tracks of dialogue, several tracks of music, and several tracks of sound effects. The mixing boards for commercial film work are quite large—a 72-track mixing console is not unusual. A console of this size is operated by a team of two to six people. The most common arrangement is for the "head" mixer to handle the dialogue, while two other mixers handle the music and sound effects. Typically, the mixing consoles have no automation, so level and filter settings must be "rehearsed" in much the same way that musicians rehearse their performances. Balancing the various elements so that important aspects are brought out at the right times without distracting from the action on the screen is a delicate art.

If we gauge the number of reels of sound needed for every reel of film by multiplying the number of premixed reels by the number of reels that have gone into the premix and summing over all reels, we find that a complicated scene can entail as many as 130 separate reels or tracks of audio. Premixing is essential, then, as a way of reducing this to a manageable number, even at the cost of introducing additional noise during the copying process.

This is the way movie sound has been made, virtually as long as there has been movie sound. What we wish to do here at Lucasfilm is to put a computer in the middle of all of this, so that each manipulation is precisely recorded and memorized and there can be complete (and semiautomatic) sharing of information among the different mixers, including the film editor. To this end, we have designed, built, programmed (Abbott 1981), and are currently debugging an audio signal processing station that we hope will be able to perform these tasks with great efficiency and clarity of sound.

This system can only affect the production of sound for the film. We cannot now expect to have a strong influence on how the sound is presented in the theater. (Obviously, to reap the benefits of digital processing fully, theater sound systems have to be thoroughly overhauled also.) Our aim is to reduce the tremendous amount of hand work (splicing and resplicing all those little pieces together) and consequently lower the cost of film production (at least in the sound department). With costs down, we hope funds will become available to bring new talent and creativity into the field.

This, then, is the background for our project. Since most of us on the team come from the computer music synthesis world, we have also embedded a great deal of music processing and synthesis capability into the system, with the feeling that when sound editors, particularly those involved with sound effects and music production, learn the true potential of the system, they will be seduced into using it more and more.

And In Signal Processing Terms . . .

The implications of this scenario for the signal processor are numerous. We plan to store the sound itself on standard 300-Mbyte disks. They offer several advantages, such as the fact that they are mature products that are easily available and easily maintained. The packs may be mounted and dismounted, giving us the flexibility we need in face of the startling realization that we cannot afford to keep all the sound for an entire movie on-line all the time. With a 50-KHz sampling rate, an entire 300-Mbyte disk pack, when formatted in a standard way (32 sectors per track at 512 bytes per sector), holds 42 min of monaural sound. Since the 70-mm print of a film uses six magnetic tracks, a single pack only holds between 6 and 7 min of six-track sound, and this is in finished form. Many, many disk packs have been used to produce these tracks. Until storage systems several orders of magnitude more dense (such as optical storage) are readily available, we must accept the need to mount and dismount large numbers of disk packs.

Sounds for movies vary from very short (pistol shots) to very long (background noise or music). During the mix-down process, deliberate steps must be taken to prevent these sounds from being scattered around the disks in unpredictable ways, which would require a great deal of head motion to recover them all. Since the disk rotates at 60 revolutions per second and each track contains 16 Kybtes, a mean transfer rate of about 980,000 bytes per second could theoretically be obtained (despite the fact that the burst rate is about 1.2 Mbytes per second). Since we only need 800,000 bytes per second to provide eight channels of audio at 50 KHz, some margin is provided for head motion, but then the buffer space must be quite large to allow large, contiguous transfers to proceed in an uninterrupted manner.

The processing of the signal can be easily formulated using digital techniques, since it consists mostly of various kinds of filtering. Each sound, however, usually has its own "private" processing that must be applied, which is different from the processing of other sounds that might be going on at the time. For example, we might have two simultaneous pieces of dialogue, one of which was recorded in a studio and the other of which was recorded on the set. When the sound is started, microcode must be loaded to perform the particular processing that is necessary for this sound and must be unloaded when the sound is done (or somewhat after the sound is done if, for instance, the reverberation is to persist after the sound). For the duration of a sound, the microcode for that sound typically does not change, but various parameters (loudness, filter frequencies) will often change slowly with time. This means that we must be able to load a bunch of microcode at or near a particular time without disturbing any other processing (microcode) that is happening at the time. Not all the microcode changes at once; more often, relatively small bits of it flow in and out of existence at various (precise) times.

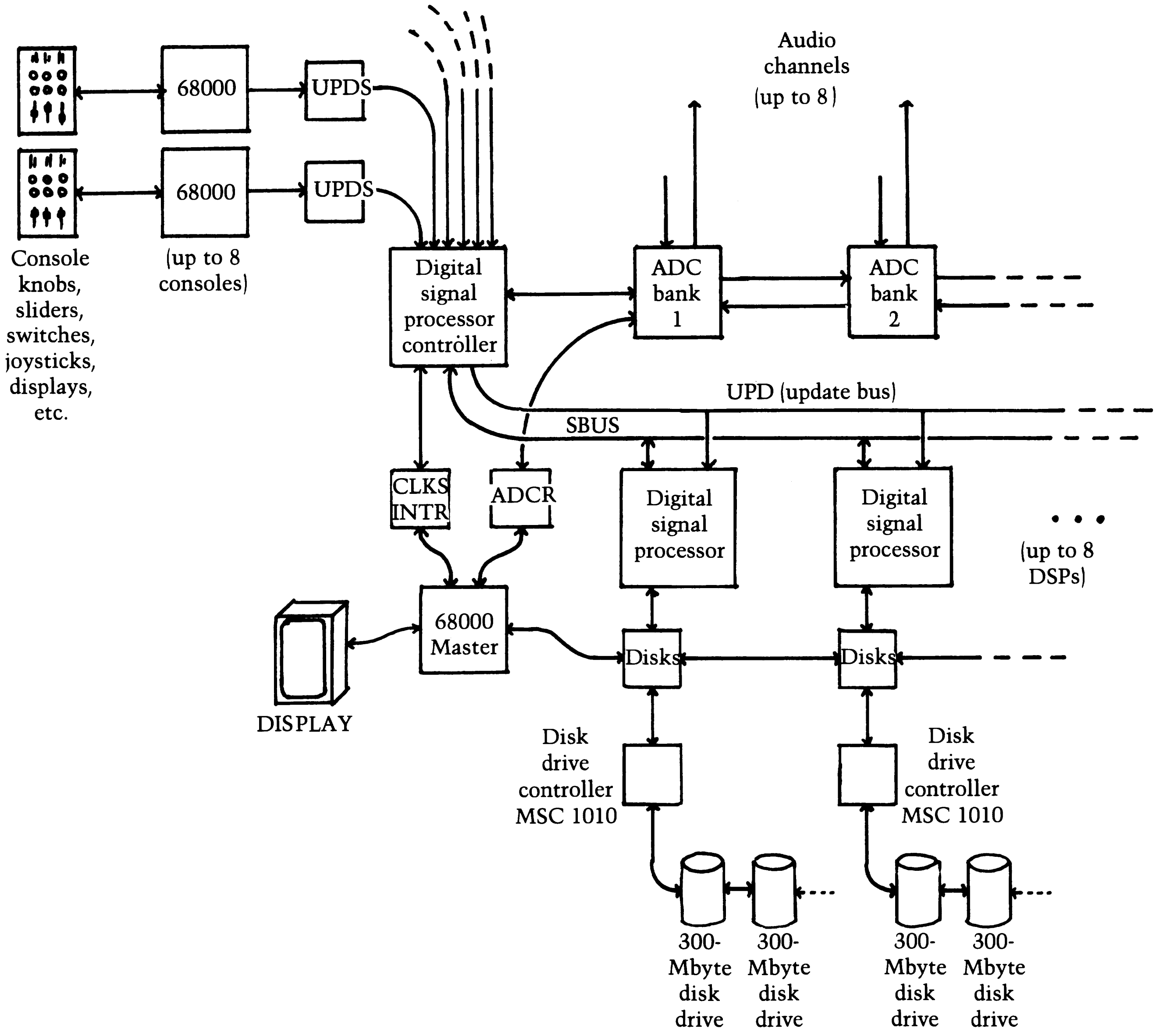
The Audio Signal Processor

The audio signal processing station is a semimodular, self-contained unit composed of several major subassemblies. The control computer is a Motorola 68000 with Winchester disk, 1 Mbyte of main memory, and a high-resolution, bit-map, graphic display screen. The audio signal processor (ASP) is composed of two parts: the controller and up to eight digital signal processors (DSPs). The console is a stand-alone 68000 with a custom-built panel that has various kinds of control devices, such as slide potentiometers and knobs (Snell 1982). We allow up to seven independent control processors to forward updates (changes to microcode and parameter memories) to the ASP simultaneously. There is a priority system for arbitration of simultaneous requests. The remainder of this article will be concerned with the architecture of the ASP itself, since this is where any innovation in audio signal processing is to be found. Figure 1 shows the block diagram of the entire system.

A DSP is designed to handle 8 channels of audio at a sampling rate of 50 KHz. Since movie sound does not normally have a full 20-KHz bandwidth, we may be able to reduce the sampling rate to 35 KHz and thus increase the number of channels to 12. For professional music applications, however, 50 KHz is considered to be standard. Since up to eight DSPs can be connected to a single DSP controller, a maximum of 64 channels for an ASP is achievable. Each DSP is capable of a computation rate of about 18 million 24-bit integer multiply-adds per second, simultaneous with a sustained disk transfer rate of 6.4 million bits per second (800 Kbytes per second) and a sustained analog-to-digital or digital-to-analog converter (ADC or DAC) transfer rate of 6.4 million bits per second.

The DSP is a horizontally microcoded device with 4K 96-bit microcode words. The device is a

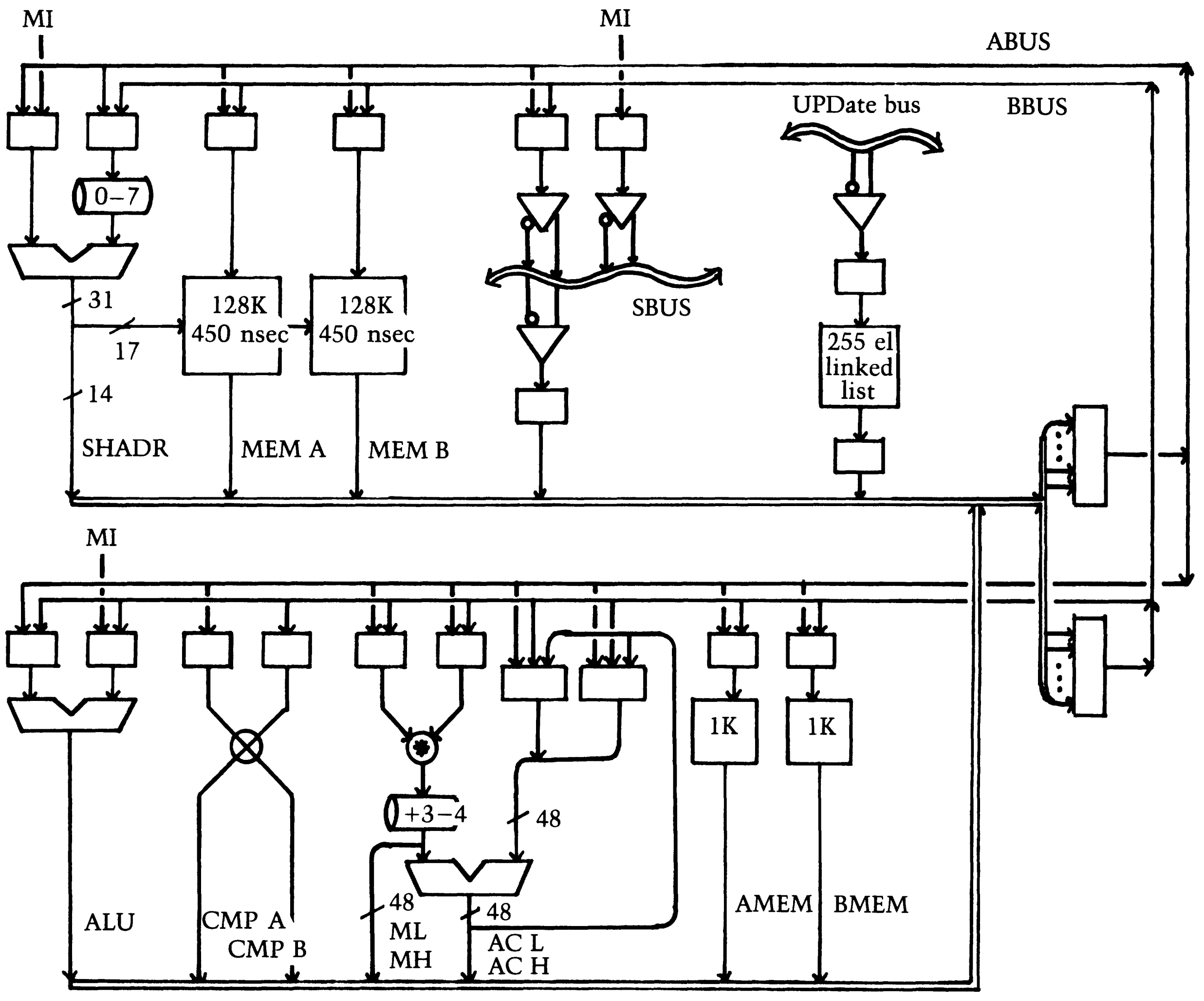
Fig. 1. Block diagram of the Lucasfilm audio signal processor (ASP) system.



lock-step, synchronous machine with no branching. The instruction counter starts at zero, goes to a fixed limit, then returns to zero. It was designed this way for several reasons, not the least of which is that branching is not generally necessary in this kind of sample-at-a-time "stream" processing, as long as logic and decision-making capabilities are provided in some other way. Another reason is that with a number of DSPs in the system, it is very convenient to have them all working on the same sample at the same time. This greatly simplifies interprocessor communication. For computing recurrence relations, such as those involved in digital filtering, it is perfectly clear that the "program" is the same for each sample. (Note: we cannot readily make use of the Fast Fourier Transform [FFT] for realizing these digital filters, since most of these filters have time-varying coefficients, and sometimes the rate of variation approaches the FFT frame rate. We are thus forced to use the time-domain recurrence relation.)

Inside the DSP itself, there are separate functional units for dealing with each of the main problems in this kind of device: (1) transferring data to

Fig. 2. Block diagram of the digital signal processor (DSP) subsystem of the ASP.

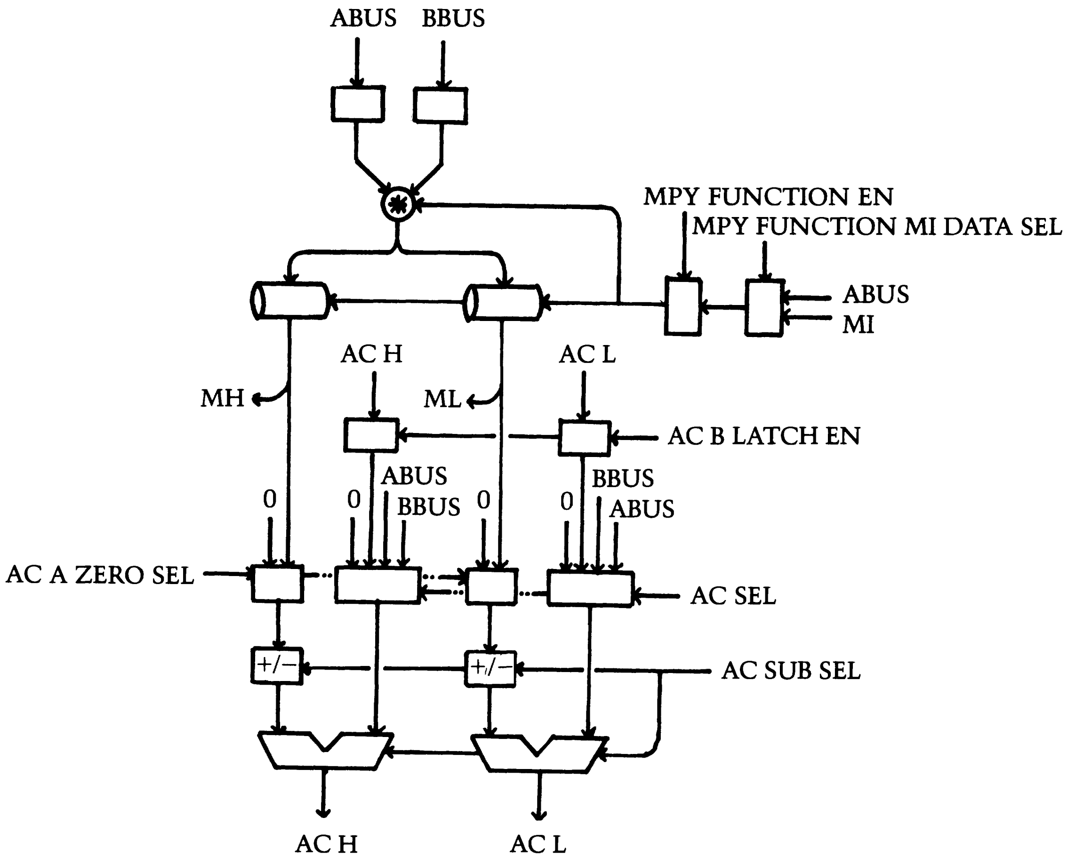


and from the disks continously at near the maximum rate, (2) effecting the required numerical calculation rate, (3) transferring data to and from the ADCs and DACs, and (4) handling the synchronous changes to microcode and parameters. In addition, extensive diagnostic aids are distributed throughout the machine, allowing it to be single-stepped and allowing readback of most of its internal registers.

The Numerical Engine

The heart of the numerical part of a DSP is the multiply-accumulate unit, the scratchpad memories, and the busses. Figure 2 shows a block diagram of the DSP itself. There are two 24-bit busses, called the *ABUS* and the *BBUS*, which supply data to each of the arithmetic units. Each microinstruction has two 4-bit fields that specify the source for each bus, and a number of bits saying which arithmetic unit input latches are to receive the contents of these busses. On each 50-nsec instruction, two 24-bit data are selected and forwarded, via the busses, to the input latches of some functional units. For ease of programming, the two 16-input multiplexers for the busses are identical. This makes the busses largely interchangeable, even though some of the combinations will be seldom

Fig. 3. Data flow through the multiply-accumulate unit.



used. We did it this way because every time in the past that we have attempted to anticipate which paths would be used and which paths would not be used, further developments in signal processing techniques have invariably proved us wrong. After gaining some experience with the machine, we will be able to look back over our programs and ask which combinations have not been used. Then we may reduce the width of the selectors in future versions of the machine.

The multiply-accumulate unit consists of a 24-by-24-bit, signed/unsigned multiplier that develops a full 48-bit product, followed by a shifter that is capable of shifting left up to three places and right up to four places, followed by a 48-bit accumulator. This allows the partial products for a digital filter to be summed in double precision, permitting later truncation to single precision or the use of even higher precision. There are pipeline registers in the multiplier so that a full multiply may be started every 50 nsec (that is, every instruction). If a multiply is started on instruction N, its results may be selected onto one of the busses on instruction $N + 2$, and the output of the accumulator may be selected on instruction $N + 3$. The output of the multiplier may be either added into or subtracted from the accumulator contents.

The Multiply-Accumulate Unit

The multiply-accumulate unit consists of a 24-by-24-bit, signed/unsigned integer multiplier, a 48-bit combinational shifter (three left to four right, signed or unsigned), and a 48-bit accumulator (adder and latch). Figure 3 shows the data flow through this unit. There are pipeline registers so that a new multiply-accumulate can be started every instruction (50 nsec). If the accumulate function is not needed, the direct multiplier output is available (to the bus multiplexers). The programmer may choose either the high-order or the low-order word, or may use the full 48-bit product. There are multiplexers on the accumulator input, so that three different functions may be selected: initialize to zero; accumulate (that is, use previous accumulator contents as input); and initialize to the contents of the ABUS and/or the BBUS. Similarly, the multiplier output can be inverted before it is accumulated to provide for subtraction as well as addition. If a multiply-accumulate is started on instruction N (that is, if instruction N specifies latching of either of the multiplier input latches), then the product may be selected into the bus multiplexers on instruction $N + 2$, and the accumulator output may be selected on instruction $N + 3$.

Although the shift amount is normally specified by a microinstruction, it may also be latched from the ABUS. This allows a limited form of data-dependent shifting, such as is needed for normalization or alignment of floating-point or block-floating-point operations. Since the shift matrix has only a very limited range, large shifts must be synthesized in other ways, such as with multiplication by powers of two. For normalization, we must first determine the position of the high-order bit. In this machine, this is most easily accomplished by the use of table lookups for large shift amounts or of the compare/exchange unit for lesser shift amounts.

The signed/unsigned feature of the multiplier allows simple extension to multiple precision operations. It also simplifies table interpolation, which is used extensively in variable-length delay lines, such as those used in audio "phasers" or reverberators. Multiple precision does not have direct application in garden-variety audio processing, but rather in certain exotic possibilities, such as linear prediction speech modification or deconvolution of room reverberation. In linear prediction, the filter itself can be easily realized in single precision, but the matrix inversion necessary for computing filter coefficients must be done in multiple precision for higher filter orders (such as order 45 or higher).

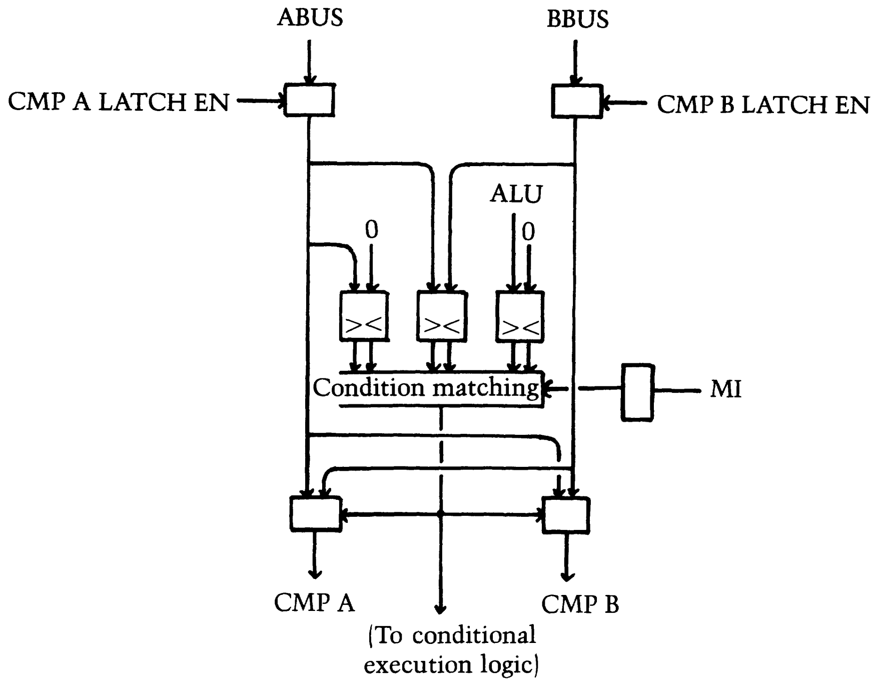
Since the DSP does not have a divide unit as such, divides must be accomplished by other means. There are various schemes for doing this, but probably the most relevant is a reciprocal table for some number of the divisor bits, followed by some number of iterations of Newton's method for the low-order bits. Since the reciprocal is a seldom-used operation even in matrix processing, this is not expected to be a bottleneck, but merely an annoyance. We have found that for a 24-bit number, a 4096-word lookup table followed by one iteration of Newton's method gives us 23-bit accuracy for all but the smallest (i.e., less than 1/4096) numbers. In this range, the dynamic range of the reciprocal exceeds the word length of the machine. If numbers spanning this range are expected, then some more comprehensive method must be used.

The Logical Unit

Even in a stream machine without program branches, decision making is necessary. Decision making is done by (1) the compare/exchange unit or (2) the conditional execution of a microinstruction. A condition code set by a microinstruction specifies the condition, such as arithmetic-logic unit (ALU) output compared to zero, or the relation of the two compare/exchange inputs. All eight logical combinations, including unconditional TRUE and FALSE, are possible. The result of the OR of all the bits showing through the condition mask forms the condition. Based on this condition, the current instruction may be executed or not. Likewise based on this condition, the two inputs to the compare/exchange unit may be swapped. This latter feature is handy for control functions, such as filter frequencies that are being changed in real time. In this manner, we can specify a control function in piecewise-linear form, such that each segment has an increment, a current value, and a final value. When the current value passes the final value, the compare/exchange unit can be employed to substitute the final value. Similarly, other piecewise-linear functions, such as the absolute value of a number, can be simulated. Figure 4 shows the data flow in the compare unit.

The conditional execution of an instruction is

Fig. 4. *Data flow in the compare unit.*



useful in several different ways. It can be used to reset loop variables such as the span, the increment, and the "twiddle" angle in an FFT calculation. It can be used to interrupt conditionally the host processor after a calculation is complete. In general, it is the "escape" from the lock-step of the computing engine.

The Arithmetic-Logic Unit

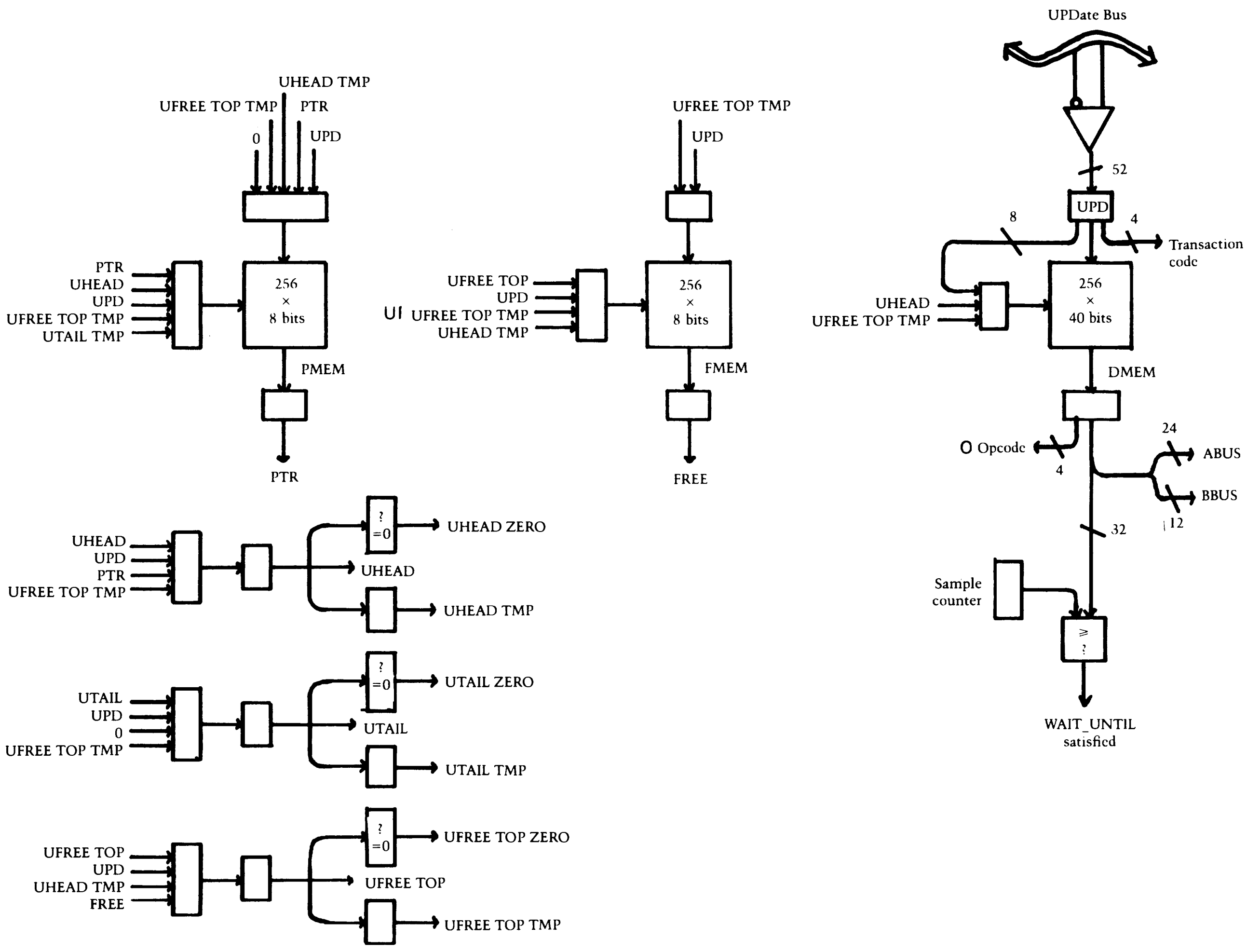
For performing all the other operations that are needed, such as Boolean functions, a general-purpose ALU is included. This provides AND, OR, and EXCLUSIVE-OR, as well as addition and subtraction operations. Furthermore, a second register is included for accumulation of high-order bits in multiple-precision operations (i.e., the carry bit is accessible), with optional sign-extension of either or both operands.

Main Memory

The main, bulk memory of the system is arranged in boards of two banks of 128K 24-bit words each. Up to eight boards may be connected to each DSP. Each bank can be cycled simultaneously such that two 24-bit transfers may be accomplished each 450-nsec interval. Error detection and correction are pipelined with the memory cycle.

There are two addressing schemes for the main memory. The first is the DSP's address-calculation engine, consisting of a shift matrix and an adder, which provides two-dimensional addressing capabilities. This is most useful for table lookup with tables of power-of-two lengths. The shift matrix scales down the address such that the entire 24-bit range is reduced to correspond to the length of the table, then the origin of the table is added in to produce the final address. The high-order 17 bits of this combined number are used as the memory address, and the low-order 14 bits (24 bits plus a possible seven-position shift) are available to the multiplexers for interpolation. This makes operations such as delays quite simple. Likewise, special functions such as computing square root or arctangent can be accomplished with limited precision by table lookup and interpolation. For interpolation, we might store the function values in one memory bank and the differences between adjacent values in the other bank, so that the difference and the low-

Fig. 5. Update queue mechanism.



order address bits may be forwarded directly to the multiplier for the interpolation calculation.

The second addressing scheme is asynchronous and is not under the control of the DSP microengine. This is direct memory access (DMA) from the disk. There is a separate word count and memory-address counter for this data path. The DMA is operated on a *cycle-stealing* basis in that it uses cycles when the DSP is not referencing the memory.

In the normal mode of operation, we plan to use four memory boards for a total of 1 million 24-bit words of storage. This gives enough room for adequate disk buffering with some space left for delay lines and table lookups. With the 450-nsec cycle time, somewhat more than 120 memory references could be made in the 20-μsec sampling interval if we started a memory cycle at every available opportunity.

The Update System

Certainly the most unconventional part of the machine is the update queue (Fig. 5). This is a linked list, implemented in hardware, of triples (opcode, address, and data). The opcodes specify for the most part which memory is to be written, but one of the opcodes is critically important for timing updates: the WAIT_UNTIL code. In this operation, the address and data are taken as a single 32-bit number and compared to a sample number counter that is "incremented" automatically each time the DSP program counter is reset to zero (i.e., at the beginning of each sample calculation). The operation of the queue is as follows. Up to eight computers may give update transactions. A priority arrangement selects one transaction and forwards it to the specified DSP. The transaction code (different from the opcode) tells the disposition of this transaction. Some of the options are (1) insert at beginning of queue, (2) insert at end of queue, and (3) insert at specified address in queue (i.e., after a specified element of the queue). Normally, the master processor will insert at the end of the queue. At the end of the processing for a sample, all the DSPs will go into update mode. In this mode, triples are read from the head of the update queue (if it is nonempty) and the appropriate actions are taken at the rate of 50 nsec per update. If the operation code specified WAIT_UNTIL, then the datum is compared to the sample number. If the datum is greater than the sample number, then updates for that DSP will be halted, and the WAIT_UNTIL will not be removed from the queue. If the datum is less than or equal to the sample number, then the WAIT_UNTIL is removed from the queue and updates proceed as described above. When all the DSPs have either emptied their queues or have hit an unsatisfied WAIT_UNTIL, they will all simultaneously exit update mode and start again at instruction zero. If there are no updates to do, all DSPs will spend a total of three instruction times (150 nsec) in update mode. The point of this complexity is that the 68000 can forward changes to be made at a specific time to the DSP as fast as it can. When that time comes, the DSP will be held in update mode until all of those changes are made. The net result is that large amounts of changes can be made (up to 255 per DSP) that appear to happen entirely between two samples. This occurs quite often in the audio processing case, since each time a new sound is begun, it typically requires an amount of processing (filtering, reverberation, etc.) that must be commenced at the same time. This involves, as often as not, the "splicing" of new processing elements into the audio stream. As we might expect, there is a critical section problem here: if the splicing is done in the wrong order, discontinuities in the signal can result. We solve this problem by making a group of updates into an indivisible unit so that updates are all effected at the same "time." Likewise, in order not to slow down the ASP, the changes are accumulated and effected at the natural rate of the ASP rather than at the somewhat slower rate of the controlling computer.

This update scheme has one problem, and that is with real-time manual intervention. What we have described above works quite well if the changes are known beforehand. If the changes are not known, such as when the operator is manipulating a potentiometer or some other input device, then the queue must be "short-circuited" so that the parameter changes may be introduced ahead of any timed updates that are already in the queue. This is why we allow updates to be entered at the beginning of the queue, thus going ahead of any WAIT_UNTIL that might delay its effect. Of the eight computers that can access the update queue, only one can be the *queue master* and insert WAIT_UNTIL instructions. All the others must insert untimed changes at the beginning of the queue. Otherwise, two computers that are not perfectly synchronized could insert WAIT_UNTILS that were not in order.

There is no substantial hardware limit to the rate at which updates can be forwarded to the DSP. The only limit is how fast the 68000 processors can deliver them, which is a maximum of about 100,000 updates per second, per 68000 processor. The DSP can handle more than 1,200,000 updates per second as a sustained maximum rate.

The SBUS

The SBUS is the catch-all for data transfer. It is used to communicate among DSPs and to send data to the DACs and read data from the ADCs. To communicate with other DSPs, one DSP merely writes a 24-bit word into its SBUS output port on instruction N. All other DSPs can then read this word on instruction $N + 2$. To communicate with a DAC, a DSP places the converter number (a 6-bit quantity) and an opcode into an SBUS function register, then provides a 24-bit datum to its SBUS output port. Automatically, the SBUS controller (a global resource) polls the first-in first-out (FIFO) for the named DAC. If there is room in its FIFO, the 16 bits in the middle of the 24-bit word are placed in that FIFO and the ASP proceeds. If the FIFO for the named DAC is full, indicating that the ASP is running ahead of the DAC, then the clock of the ASP will be held until the FIFO is no longer full. The situation with the ADCs is analogous. If the named ADC FIFO is empty, the ASP clock will be held. These FIFOs are 64 samples long, giving a maximum delay of 1.28 msec to the audio path.

Another SBUS feature, which we call the *bulletin board*, is a 256-element memory that may be written by an SBUS operation and may be read by any computer with an update bus (UPDS) interface. This provides a channel for certain kinds of feedback, such as the root-mean-squared (RMS) level of an audio signal or an overload condition of some kind. The controlling computers may also read back the current value of the sample counter and thus keep track of time.

Conclusions

The ASP, through its multiple data paths and unique architecture, is capable of the very high numerical computation rate required for processing many channels of high-quality digitized audio. Both synchronous and asynchronous data exchange proceeds at very high sustained rates without interfering with computation. Large blocks of program memory may also be changed without interference with the sample data stream. The device is ideally suited to large-scale, real-time audio processing applications, such as film sound mixing, special effects processing, and music synthesis.

Status of the Project

The prototype machine became operational in April 1982 and is functioning reliably at the full clock rate. All of the complex features, such as the DAC FIFO mechanism, the direct DMA path to the bulk memory system, and the 24-by-24-bit multiply followed by the 48-bit accumulate, function reliably with a substantial amount of timing margin. The software, under development for more than a year, is now operational. We hope to turn the machine over to users very soon. As soon as this is done, we will begin building two more devices for our own in-house use.

Acknowledgments

The hardware was designed by James A. Moorer with the exception of the main memory board, which was designed by John M. Snell. The technicians who did the board preparation and final assembly were Charlie Keagle, Kris Handwerk, and Sharon McCormick. The project was started in April 1980 by George Lucas and it is entirely his foresight and vision that have enabled us to proceed.

References

Abbott, C. 1981. "Microprogramming a Generalized Signal Processor Architecture." Paper presented at the 1981 International Computer Music Conference, 5–8 November 1981, in Denton, Texas.

Samson, P. 1980. "A General-Purpose Digital Synthesizer." *Journal of the Audio Engineering Society* 28(3): 106–113.

Samson, P. Forthcoming. "Architectural Issues in the Design of the Systems Concepts Digital Synthesizer." In *Computer Music*, ed. C. Roads and J. Strawn. Cambridge, Massachusetts: MIT Press.

Snell, J. 1982. "The Lucasfilm Real-Time Console for Recording Studios and Performance of Computer Music." *Computer Music Journal* 6(3):33–45.