Mark Kahrs
Computer Science Department
University of Rochester
Rochester, New York 14627

Notes on Very-Large-Scale Integration and the Design of Real-Time Digital Sound Processors

Introduction

Since the 1960s, electronic components that make up digital systems have been packaged as tiny integrated circuits (ICs) or chips. There has been steady progress since that time in making the components smaller and thereby packing more components per chip. This has led to several generations of IC devices, from small-scale integration (SSI), with only a couple of components per chip, to medium-scale integration (MSI) and large-scale integration (LSI), with tens of thousands of components per chip (Myers 1980). As a result of improvements in the technology of chip fabrication, achievable circuit densities continue to grow higher. The present level of density extends into very-large-scale integration (VLSI).

The Present State of the Art

New digital sound processor technologies are made possible by the introduction of VLSI technology. In a recent survey of integrated circuits and signal processing, Hoff (1980) points out the increasing complexity that is coming with the introduction of VLSI technology. At the present time, chips like the Motorola MC68000, a 16-bit microprocessor with 68,000 transistors, are at the upper limit of today's mass production (Fig. 1). As the size of circuit features decreases more in the next few years, one can expect to see dramatic increases in chip density and functional capacity. Some estimates place as many as one million logic gates per chip by 1990 (each logic gate may comprise several transistors).

There are five principal limitations in VLSI design: chip size, device density, circuit speed, complexity of device interconnects on the chip, and number of pinouts (the number of pins emanating from a chip's package). When designing a VLSI sound processor, one must consider various tradeoffs. For example, one can conserve on the number of pins used by adding multiplexers to the chip, allowing any one pin to be used for more than one purpose. The demands of real-time computation put an extra burden on digital processors. For such real-time processors, duplication of chip circuitry can increase the speed of signal-processing algorithms at the cost of additional "real estate" on the chip being dedicated to the parallel circuits.

Computer Music Journal, Vol. 5, No. 2, Summer 1981,
0148-9267/81/020020-09 $04.00/0

Current Sound Processors

I began my research into VLSI and digital sound processors by considering existing processors implemented using MSI technology. The 4B Machine at the Institut de Recherche et Coordination Acoustique/Musique (IRCAM) (Alles 1976; 1980) and the Systems Concepts Digital Synthesizer (Samson 1980) were examined for ideas usable in a VLSI implementation.

The 4B is a multiplexed machine, that is, it runs fast enough to compute a number of voices of sound in the time between sample periods. Translating this machine into a single chip would not be a terribly difficult job. The machine does have a problem, however. When generating line segments for functions (e.g., envelope curves) the 4B allows timer interrupts to be set for the endpoints of the

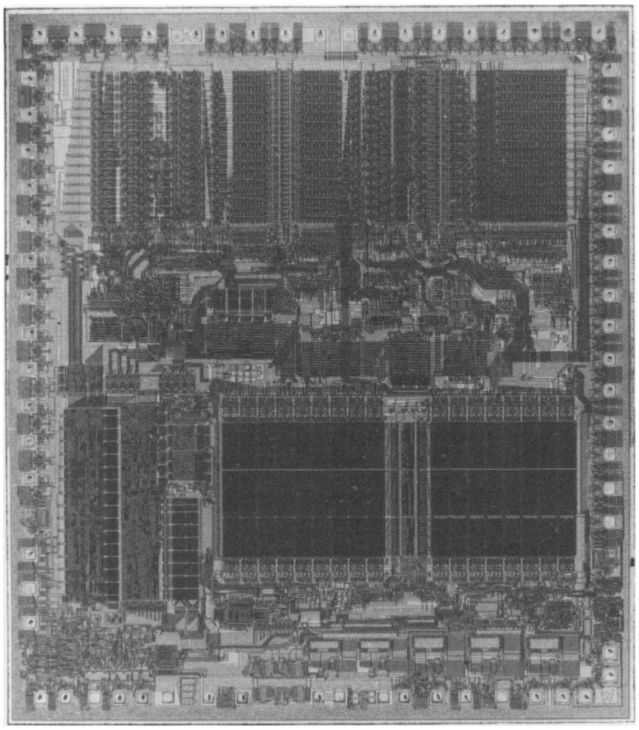


Fig. 1. The MC68000 microprocessor chip. (Photograph supplied by courtesy of Motorola Semiconducter, Austin, Texas.)

line segments. This can put a considerable load on the host processor, which is feeding parameters from the score to the 4B to control the sound synthesis. In the case of the 4B the host computer is a slow LSI-11. The LSI-11 is constantly being interrupted and asked to supply more data; this is known as the *parameter-update problem.*

The Systems Concepts Digital Synthesizer is a large, pipelined processor built using MSI technology. It has 256 generators, 128 modifiers, and a mixing memory called *sum memory.* Unfortunately, the machine appears to be designed mainly for frequency modulation (FM) and additive synthesis, rather than as a truly general-purpose synthesizer. Attempts to use the machine for synthesis techniques other than additive and FM synthesis are sometimes successful, but implementation is somewhat contrived. As a stream processor, the Systems Concepts synthesizer has another problem. Commands to the synthesizer are read from memory and executed until a command is given to stop temporarily and allow processing of sound to take place. The problem arises when one must integrate a command stream that comes from diverse input sources such as knobs and keyboards being played by musicians. This *real-time input problem* must be solved if such processors are to be used in live performance. Designers of digital sound processors will face other problems that will come up as the designs become more refined.

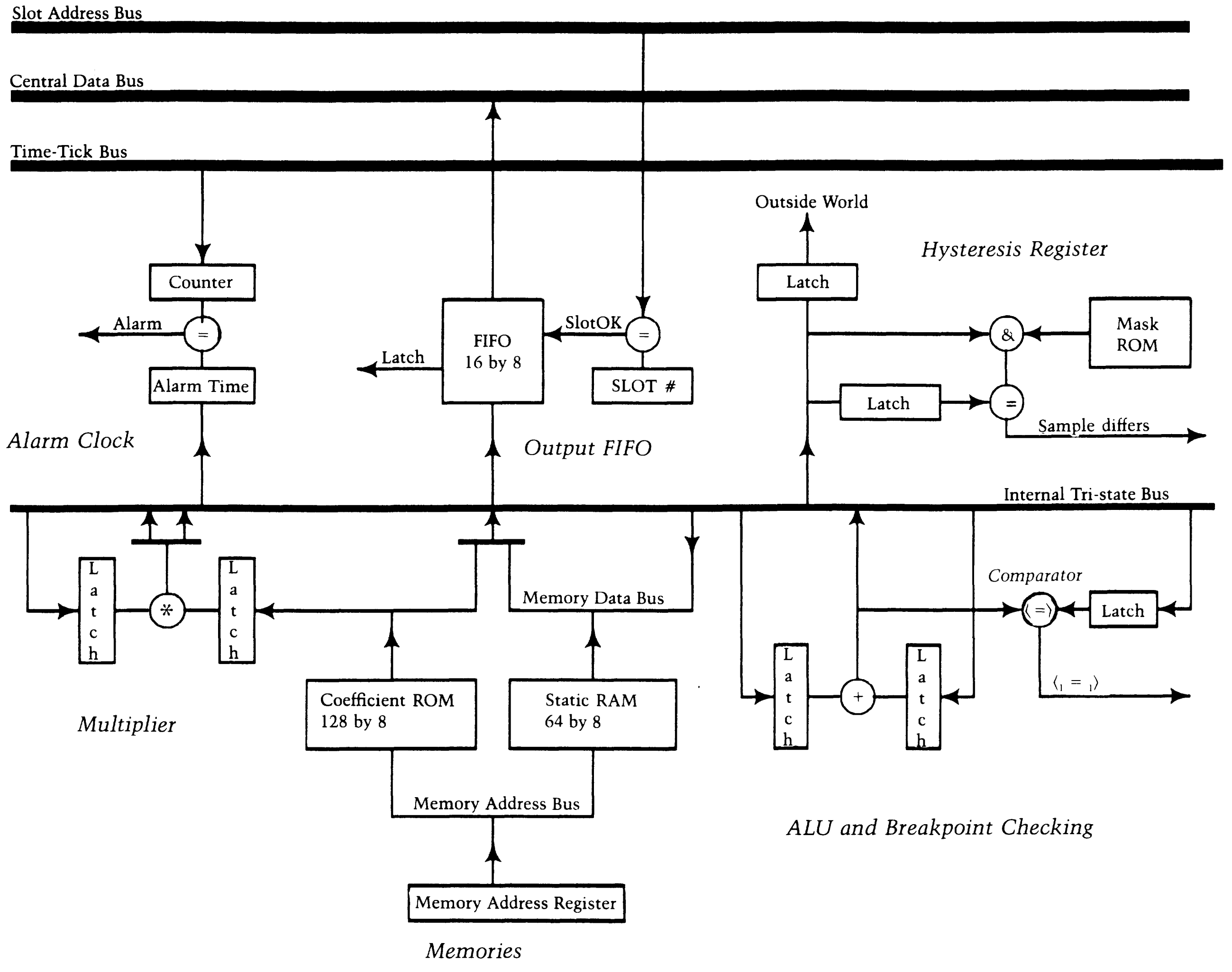
Current Algorithms

As more processing power becomes available on each chip and as the complexity of what musicians want to accomplish with such chips increases, it becomes necessary to think about processors differently. Commonly used signal-processing algorithms can be committed to firmware, thus simplifying the software environment considerably. Rather than consider, What algorithms can I implement using this beast? the chip designer must consider, What algorithms should I implement in silicon? Therefore, the designer of any new digital sound processor must consider what algorithms the user wants to implement. These might include lattice filtering, convolution, waveshaping, additive synthesis, and other algorithms useful for sound generation. In the following sections, two VLSI architectures for music processing will be discussed that address the issues pointed out so far.

A Command-Stream Processor

The purpose of the command-stream processor is to solve the real-time input problem. One needs to be able to mix many external sources of commands and data into a single command stream. One immediate difficulty is the instability of analog components often used in input devices such as potentiometers. In particular, the reference levels of analog-to-digital converts (ADCs) tend to wander. Use of a *hysteresis register* is one way to solve this problem. The contents of this register are compared against the current input (which is masked) and will match if the masked bits equal the last known

Fig. 2. The internal bus is a tri-state precharged bus. The multiplier and adder are latched, allowing operations to proceed while computations are being performed. There is one multiplexer that connects the coefficient ROM and the register bank to the internal bus. The output of the adder is connected to a "breakpoint" detection circuit that detects the endpoints of line segments. The time-tick bus is a serial input connected to a global clock. The slot-address bus contains the address broadcast by the slot clock. When the slot of the chip is recognized by the slot recognition logic, one byte from the FIFO is put onto the central data bus. The external world can communicate by using the latched "world" input. This circuit has a "hysteresis" circuit that allows for instability in ADCs and other inputs.

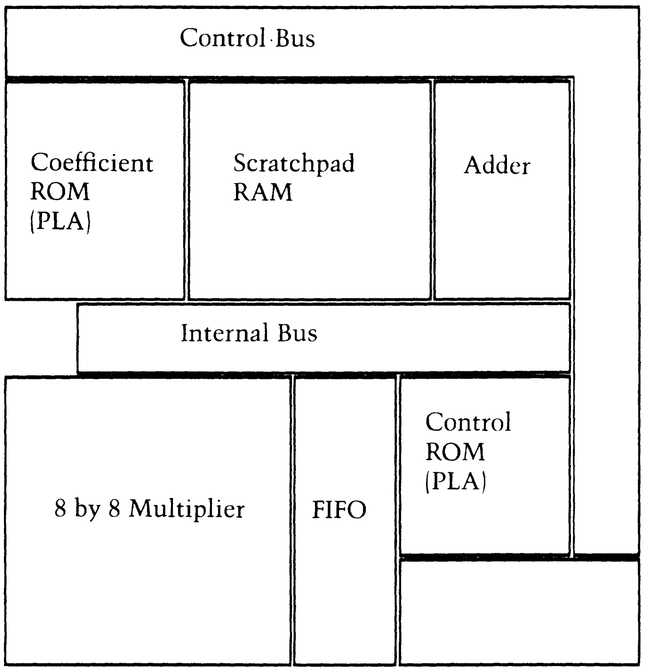


input. Such a sample can then be ignored. Thus the main processor doesn't have to be interrupted for a "new" value so often.

Samples gathered from these input devices (knobs, joysticks, etc.) usually must be scaled at some point in the processing. The command-stream processor can do this for the sound processor if it has its own on-chip multiplier. Then, in order to provide linear interpolation between endpoints of envelope line segments, an adder is also available on the command-stream-processor chip. The whole processor is controlled by a horizontal microprogram store. In such a scheme, a very wide (hence horizontal) control word directly controls the flow of internal signals within the processor. The control words are stored in a microprogram read-only-memory (ROM) implemented as a programmable logic array (PLA). PLAs are ubiquitous in VLSI design (Mead and Conway 1980). Most commonly, they are implemented as an array of AND/OR gates which the designer can selectively connect by changes in the metalization layer. The data paths for the command-stream processor are shown in Fig. 2.

In the implementation of the command-stream processor, note that the data paths are parallel

Fig. 3. This is a layout of the command-stream-processor chip without the input or output pads. The area taken up is approximately 3000 by 4000 microns.



rather than serial. Although the space consumed by the parallel circuits is larger than serial (Freeny [1975] states that parallel multipliers consume four times the space of serial multipliers), the payoff is speed. Serial circuits can be used effectively in signal processors (Jackson, Kaiser, and McDonald 1967; Lyon 1980), but I felt that the speed to be afforded by parallel design was worth the cost in chip space.

The implementation of this processor will be in n-channel metal-oxide semiconductors (NMOS) (Mead and Conway 1980). A layout of the chip is shown in Fig. 3.

A Sound Processor

So far, I have discussed ways to overcome the limitations in the bandwidth brought about by multiple, real-time command lists. In this section, I will discuss some considerations in the design of the signal processor that does the work of musical sound generation.

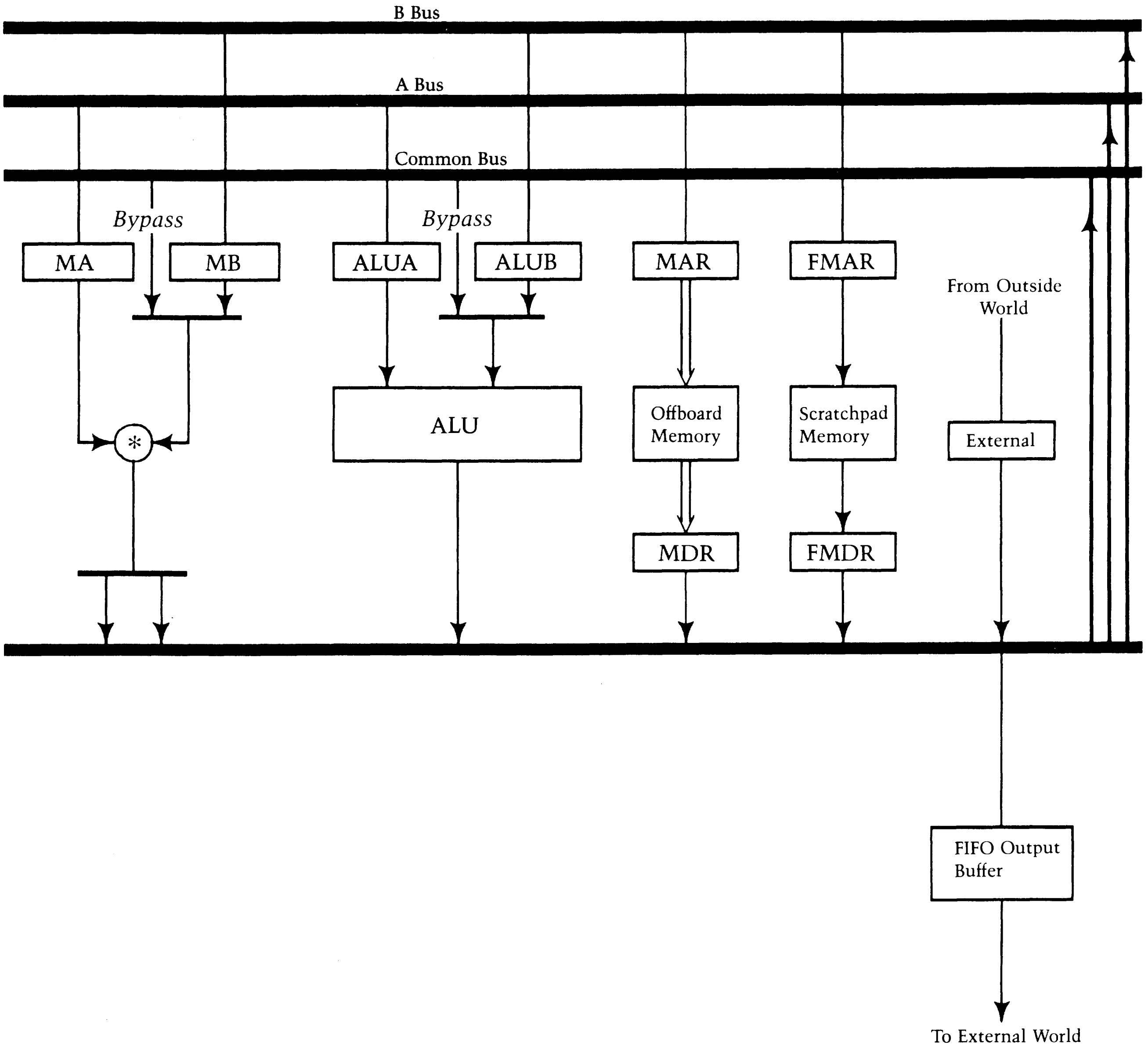
One can begin by considering exactly what algorithms will be implemented using the sound processor. This is a synthesis machine, not to be used for sound analysis. By and large, the computational demands of analysis and synthesis machines differ. Analysis algorithms are mostly block algorithms. These require an entire array (block) of samples before processing can take place. Although such algorithms are useful, in this article I will not discuss processors that implement them. What kinds of synthesis algorithms would be implemented? There are at least three major techniques:

1. Additive synthesis. This involves combining many relatively simple signals (such as sine waves), each with its own frequency and amplitude envelopes, to form complex, evolving timbres (Moorer 1977).
2. Subtractive synthesis. This takes a spectrally rich waveform and filters the sound down into a less complex timbre (Markel and Gray 1976).
3. Nonlinear synthesis. This involves modulations of signals with each other (FM, waveshaping, etc.) to produce rich, evolving timbres (Chowning 1973; LeBrun 1979; Arfib 1979).

Additive synthesis, with its individual envelopes for each partial of a complex sound, can be computationally expensive. Subtractive synthesis techniques also make substantial demands. This arises, for example, in realizing the linear-predictive-coding (LPC) synthesis technique, often used for synthesizing speech sounds (Markel and Gray 1976). In the lattice implementation of LPC, a multiplier must be provided. In waveshaping, the principal demand is for a large lookup table and another multiply. (This table could also be used for artificial reverberation.) Clearly, a large lookup table is a necessary requirement for any synthesis processor. Note that the Systems Concepts synthesizer has an external memory of 128 Kbytes accessible through the modifiers. Although a large memory is desirable, its place is not on the processor chip. It is easier to buy the memory chips and put only the interface to the memory on the processor chip.

Many existing signal processors do not have branch instructions. This is often due to the com-

Fig. 4. In this rather conventional sound processor, notice that all the inputs from the common bus are latched. This one level of pipelining allows the computation to proceed in parallel throughout the chip. Notice also that the offboard memory could be quite large, which is the reason for placing it off the chip. External input is buffered in the external register. Note the bypasses on the inputs to the ALU and multiplier. This allows results from the common bus to be used immediately in the next step of computation.



plexity of the machine architecture. This lack of branch instructions can prove to be a handicap in the implementation of algorithms such as matrix inversion or pitch extraction.

If the signal-processing lessons of the Digital Voice Terminal are of any help (Gold 1974), then the processor should have as much raw speed as possible. Pipelining within the processor is one way to achieve this. Multiple computation units that are spread across the task (such as found in the IBM Model 360/91) are another way to achieve higher execution speeds (Tomasulo 1967).

In Fig. 4, the data path of a rather ordinary sound processor that can be implemented in NMOS is given. Notice that the processor has an "outboard" memory and an internal multiplier. Once again, the processor is microprogrammed. This is more than a consideration for user programming of the control store. As processors get more complicated, it gets more difficult to design the processor to be logically correct. Using random logic only exacerbates this problem. The MC68000 is an example in which microprogramming was used to avoid bugs in the control section of the processor (Stritter and Tredennick 1979).

Each unit of the processor has latches on the input and the output so that processing may proceed without a wait for the results of a given operation (i.e., overlapped execution). The machine is synchronous, and results are latched when the execution unit finishes its action. Notice that the result bus can be connected directly to the execution units. This allows the current results on the bus to "bypass" the latches on the input side of the execution units.

The One-Voice One-Processor Doctrine

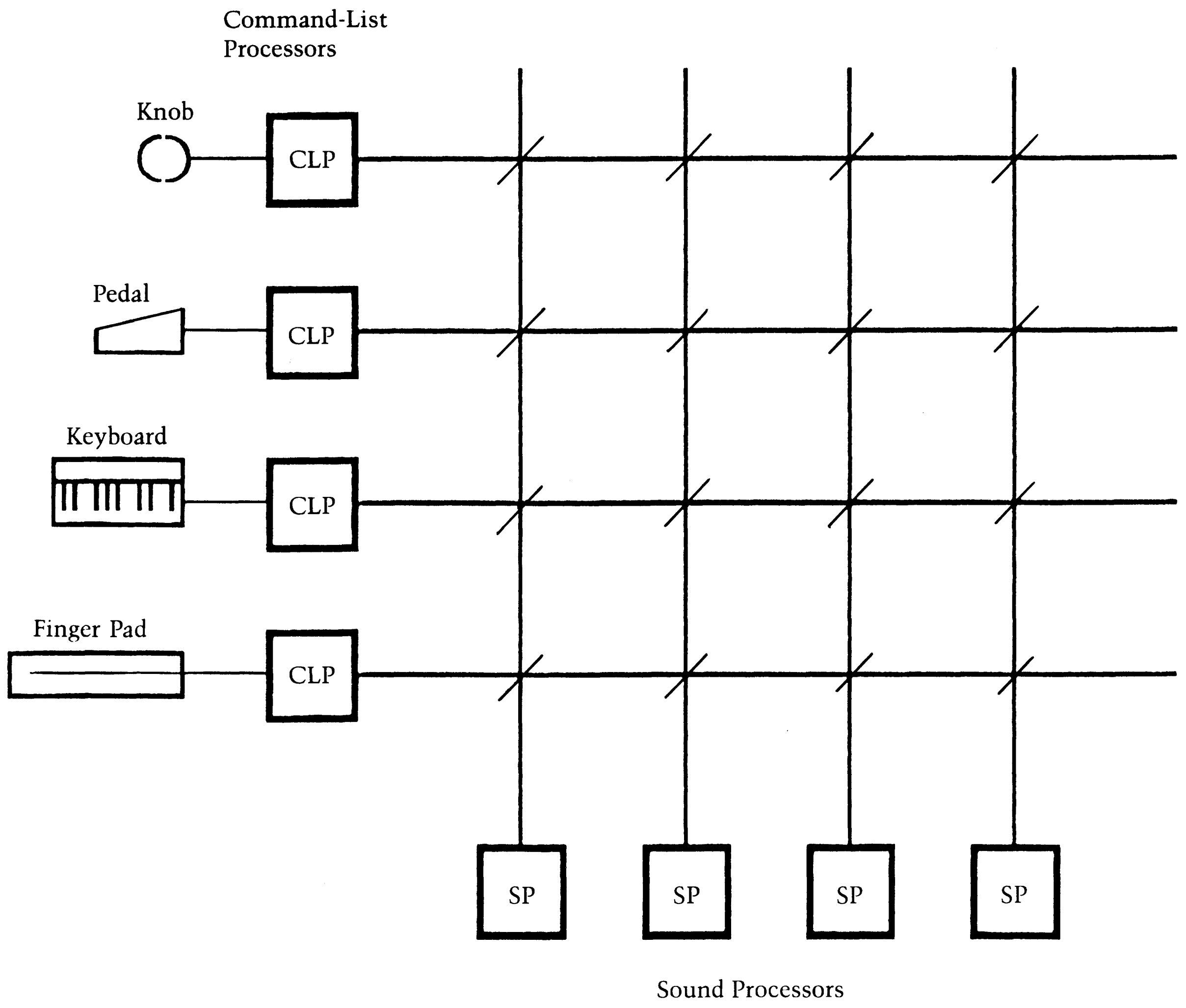
In the processor organizations discussed so far, it has been assumed that only one high-speed processor is available. With the advent of low-priced processors, lack of processor cycles should not be permitted to be a problem. Unfortunately, some of the problems of computer networking are then introduced. If we modify the existing architecture of the processor shown in Fig. 1 to have multiple processors, the interconnection of input modules and the sound processor modules becomes a problem. It's not sufficient just to restrict each sound-processor module to have one input module because one might want a single input device to affect many processors. A crossbar switch such as is shown in Fig. 5 would be ideal. Such a switch was used in C.mmp (Wulf and Bell 1972). C.mmp was composed of slightly modified DEC PDP-11/40 processors augmented with a writable control store. Up to 16 of these processors could be connected to up to 16 shared memory modules through a 16 by 16 crosspoint switch (Jones and Schwartz 1980). Of course, C.mmp's problem is well known: the number of switches increases as the square of the number of inputs and the area increases likewise.

Another solution to the problem is to use a bus and grant the bus (arbitrate it). Suppose each of the processors is connected in a chain. Then each processor could pass the control token to the next processor when it was finished putting data on the bus. But of course then the sound processor must know which real-time input placed the data on the bus. Therefore, a new bus is needed that buses the processor ID of the input on the data bus. All processors can sample the bus to find out which input is there. The problem with this is that each processor must filter out the real-time requests on the data bus. This again introduces the parameter-update problem!

Other possible bus organizations include a time-division multiplexed (TDM) bus or an Ethernet-like network where each real-time input has a time slot (an address in the Ethernet scheme) and they communicate by broadcasting to the other processor (Metcalfe and Boggs 1976). Of course this too has its problems—the TDM bus requires that processors wait for the proper time slot. Ethernet was designed to be "unreliable" in the sense that it keeps retransmitting a message over the network until it receives an acknowledgment; it does not assume the message got through on the first transmission. Unfortunately, there is no upper bound on the time it can take a packet of information to be transmitted and received. Because of the basically slow rate of change (with respect to the processor) involved in parameter update, a TDM-bus scheme was used in the command-stream processor.

Notice that in Fig. 6 a new mixing processor has been added to the output of the three sound processors. There must be one mixing processor per channel of output sound. With a fast mixing processor, this could be simulated through the use of multiplexing, since even at a 50-KHz sampling rate for audio, the samples would have to be added at a rate of 20 μsec each, well within the range of the processor's speed.

Fig. 5. A musical crossbar processor (à la C.mmp) that allows the command stream to be directed to any sound processor. The problem with the network is the second-order nature of the connections (N by M).



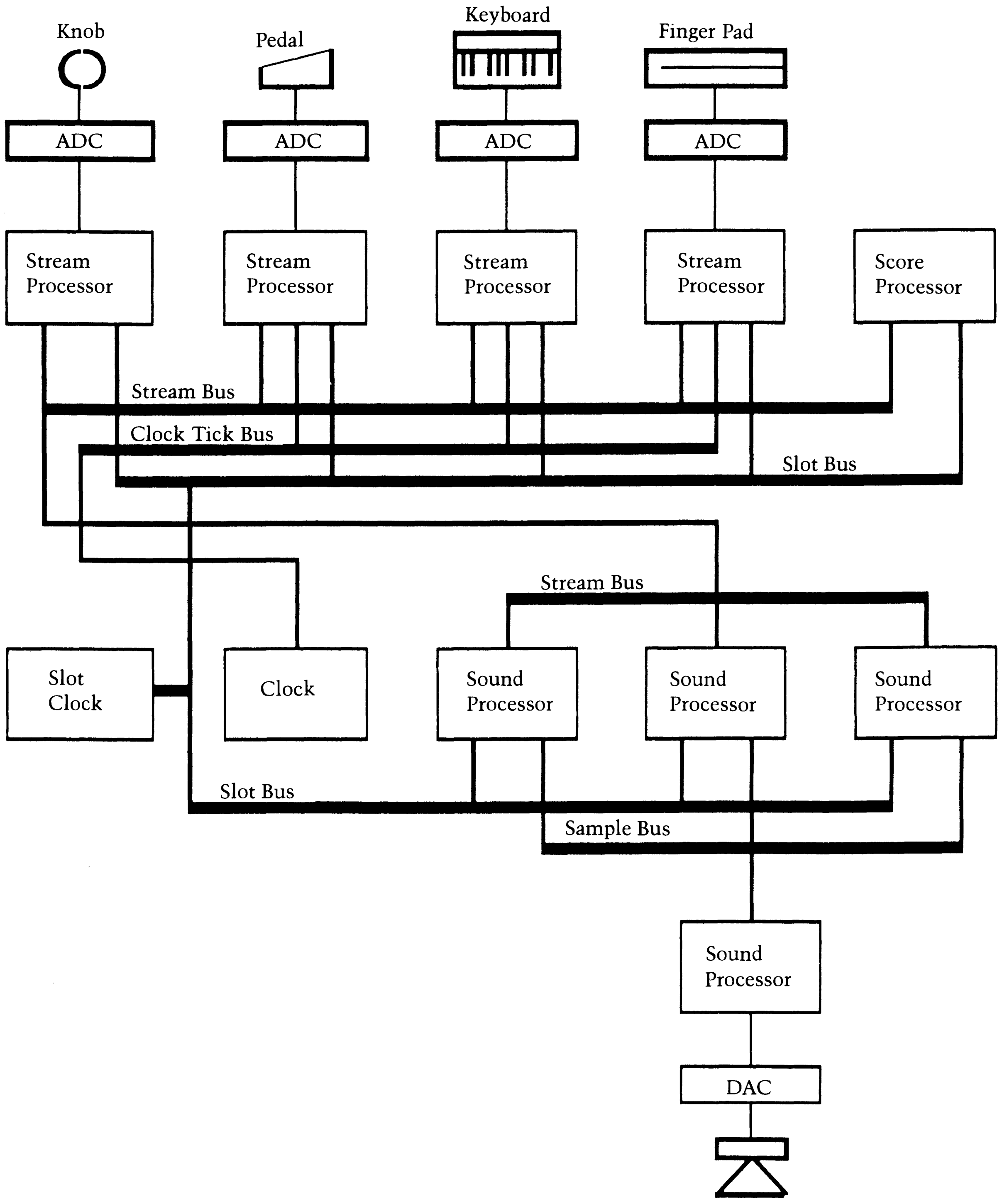
Designing the Processor for the Algorithm Instead of Vice Versa

In the design of VLSI processors, a prominent concern is to design the processor for the algorithm. This is based on the belief that processors will decrease in cost, and that by customizing the processor for the algorithm speed can be obtained. An example of customizing the processor can be found in the work of Kung (1980). *Systolic algorithms* are formed by arrays of processors that communicate with their neighbors and form rectilinear arrays. For example, Kung has designed second-order filters, a convolution box, and a discrete-Fourier-transform (DFT) box. A special-purpose reverberation box could use a convolution box to implement reverberation using the impulse response of a hall. As more research is done into sound-generation algorithms, perhaps more of them can be placed into systolic form.

Conclusion

In this article, I have explored a tip of the VLSI iceberg. Computer music's high computational requirements are an ideal problem for VLSI technol-

Fig. 6. This figure demonstrates the connections of the various chips in a time-division multiplexed system. Notice the addition of a score processor, which generates commands that are not dependent on real-time inputs. There is a global clock and slot clock, which broadcasts the current slot on the slot bus. Data is gated to the stream bus, which then feeds the sound processors. The sound processors in turn feed another sound processor that acts as a mixer. A DAC connects the signal back to analog levels.



ogy to solve. Research is required in many areas. More research is needed into the structure of algorithms used by computer musicians, particularly the implementation of algorithms that are computationally expensive. Multiprocessor implementation of complex algorithms would be of considerable interest to VLSI designers. If computer musicians engage in a dialogue with VLSI designers, then profitable results for both sides will surely follow. I hope this article will start some of that dialogue.

Acknowledgments

Many of these ideas have been "in the air." I hope I haven't broken any feet by presenting them here. I appreciate the comments of John Snell, Peter Eastty, and Curtis Abbott, but I am solely responsible for this article's content. This clandestine research has been sponsored by the Sloan Foundation under grant 78-4-15 and by the Defense Advanced Research Projects Agency under contract number N00014-78-C-0164.

References

Alles, H. G. 1976. "A Portable Digital Sound Synthesis System." *Computer Music Journal* 1(4): 5–9.

Alles, H. G. 1980. "Music Synthesis Using Real-time Digital Techniques." *Proceedings of the IEEE* 68(4): 436–449.

Arfib, D. 1979. "Digital Synthesis of Complex Spectra by Means of Multiplication of Nonlinear Distorted Sine Waves." *Journal of the Audio Engineering Society* 27(10): 757–768.

Chowning, J. M. 1973. "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation." *Journal of the Audio Engineering Society* 21(7): 526–534. Reprinted in *Computer Music Journal* 1(2): 46–54.

Freeny, S. L. 1975. "Special-purpose Hardware for Digital Filtering." *Proceedings of the IEEE* 63(4): 633–648.

Gold, B. 1974. "Parallel and Sequential Trade-Offs in Signal Processing Computers." In *National Telecommunications Conference of 1974*, pp. 491–495.

Hoff, M. E., Jr. 1980. "IC Technology: Trends and Impact on Digital Signal Processing." *Proceedings of the ICASSP*, pp. 1–6.

Jackson, L. B., J. F. Kaiser, and H. S. McDonald. 1967. "An Approach to the Implementation of Digital Filters." *IEEE Transactions on Audio and Electroacoustics* AU-16(3): 413–421.

Jones, A., and P. Schwartz. 1980. "Experience Using Multiprocessor Systems—A Status Report." *Computing Surveys* 12(2): 121–165.

Kung, H. T. 1980. "Special-purpose Devices for Signal and Image Processing: An Opportunity for VLSI." CMU Technical Report. Pittsburgh, Pennsylvania: Department of Electrical Engineering and Computer Science, Carnegie-Mellon University.

LeBrun, M. 1979. "Digital Waveshaping Synthesis." *Journal of the Audio Engineering Society* 27(4): 250–265.

Lyon, R. F. 1980. "Signal Processing with VLSI." Unpublished ms.

Markel, J. D., and A. H. Gray, Jr. 1976. *Linear Prediction of Speech*. New York: Springer-Verlag.

Mead, C., and L. Conway. 1980. *Introduction to VLSI Systems*. Reading, Massachusetts: Addison-Wesley.

Metcalfe, R. M., and D. R. Boggs. 1976. "Ethernet: Distributed Packet-switching for Local Networks." *Communications of the ACM* 19(7): 395–404.

Moorer, J. A. 1976. "The Synthesis of Complex Audio Spectra by Means of Discrete Summation Formulas." *Journal of the Audio Engineering Society* 24(9): 717–727.

Moorer, J. A. 1977. "Signal Processing Aspects of Computer Music." *Computer Music Journal* 1(1): 4–37.

Moorer, J. A. 1979. "About This Reverberation Business." *Computer Music Journal* 3(2): 13–27.

Moorer, J. A. 1981. "Synthesizers I Have Known and Loved." *Computer Music Journal* 5(1): 4–12.

Myers, G. 1980. *System Design with LSI Bit-slice Logic*. New York: Wiley Interscience.

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

Stritter, S., and N. Tredennick. 1979. "Microprogrammed Implementation of a Single Chip Microprocessor." In *11th Micro Proceedings*, pp. 8–16.

Tomasulo, R. M. 1967. "An Efficient Algorithm for Exploiting Multiple Arithmetic Units." *IBM Journal of Research and Development* January: 25–33.

Wulf, W. A., and C. G. Bell. 1972. "C.mmp—A Multi Mini Processor." *Proceedings of the Fall Joint Computer Conference* 41: 765–777.