The DMX-1000 Signal Processing Computer

Dean Wallraff
Digital Music Systems, Inc.
Brooklyn, NY

The DMX-1000 is an ultra-fast 16 bit minicomputer designed especially for audio signal processing applications. It produces audio output by means of a self-contained digital to analog converter, and may be controlled by any general purpose computer.

Figure 1 shows the architecture of the DMX-1000. The heart of the machine is the 2901-based 16 bit ALU, which performs arithmetic, logical and switching operations. The ALU contains 16 working registers and has two buses, the input, or D bus, and the output, or Y bus. The Y bus, in most cases, conveys the result of the last ALU operation. The ALU operates upon one or two numbers at a time to produce a result; either of the operands may come from any internal register or one may come from the D bus. The ALU performs the following operations: subtract, add, increment, decrement, two's complement, one's complement, logical and, logical or, exclusive or, exclusive nor, and mask. The result of an operation may be put into one of the operand registers. Normally it is also put onto the Y bus. But it is also possible to put the result of an operation in one of the operand registers and put the contents of the other operand register onto the bus. The 2900 Data Book [1] explains the options in detail.

DMS 1000 Architecture

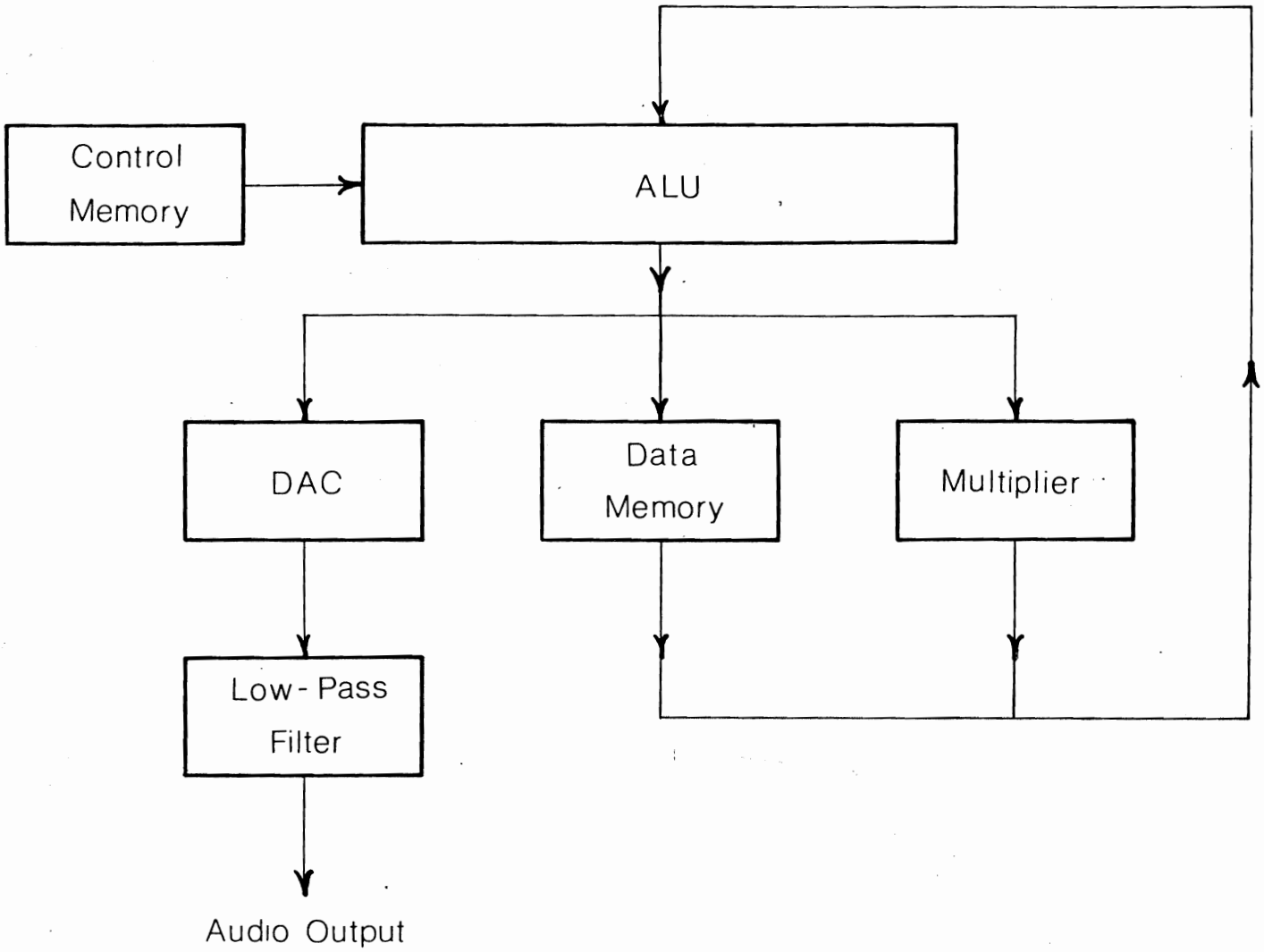

Figure 1

The DMX-1000 is horizontally microprogrammed. The microcode is stored in the bipolar control memory, which is organized as 256 words of 32 bits. The machine is driven by an eight-bit counter, called the microcode program counter (ucode PC), which is incremented every machine cycle, every 200 ns. The ucode PC addresses the control memory, causing it to output a 32-bit microcode word, which is clocked into a microcode pipeline register once per machine cycle. The microcode word is used to control the operations of all the machine's computational elements during the current machine cycle. When a halt microinstruction is executed, or when the ucode PC reaches 255, the machine stops executing momentarily. The sample cycle is over. Normally the sample cycle, which consists of up to 256 machine cycles, is used to compute one sample of sound which is clocked into the DAC sometime near the end of the sample cycle. Executing 256 200-ns microinstructions plus two update cycles (see below) per sample results in a 19.3 KHz sampling rate. The sampling rate may be raised by using a shorter microprogram.

The other, "peripheral" computational and I/O components of the system connect to the Y and D buses through pipeline registers which are controlled by the microcode. Up to eight output registers, which are connected to the Y bus, and eight input registers, which are connected to the D bus, may be accomodated. At the end of each machine cycle, any combination of output registers may be clocked, putting into them the result of that cycle's computation from the Y bus.

At the beginning of any machine cycle, the tristate outputs of any input pipeline register may be enabled, causing the register's contents to appear on the D bus input to the ALU.

For example, the 16 bit multiplier is connected via 4 pipeline registers. Two 16 bit registers, called X and Y, connect its two inputs to the ALU output Y bus. Two other 16 bit registers, called L and M, connect its output to the ALU input D bus. To multiply two numbers, the microcode first puts one of them on the Y bus and clocks the X register; then it puts the other on the Y bus and clocks the Y register. One machine cycle later, the micricode enables the M register's tristates, causing the high-order half of the 32 bit product to appear on the D bus where it may be used as an ALU operand. Or the microcode could enable the L register's tristates, giving the ALU access to the low-order half of the product. In symbolic form, the microcode looks like this:

```
MOV     R0,X            ;Load contents of register zero
                        ;into multiplier register
MOV     R1,Y            ;Load contents of another register
                        ;into other multiplier register
NOP                     ;Wait a machine cycle.  In almost all
                        ;cases something useful could be done
                        ;here
MOV     L,R2            ;Load lower half of product into R2
```

Note that the result of a given ALU operation may be put into several output pipeline registers at once, and that the ALU operation need not be a simple MOV. For example, this code:

```
ADD     R0,R1,X,Y
```

causes the contents of the ALU's internal register 0 to be added to the contents of register 1, with the result being

put into register 1 and into the X and Y registers. After one more machine cycle, the L and M registers will contain the square of the sum of the contents of R0 and R1.

A 16 bit digital to analog converter (DAC) is used to produce the analog audio output. It is connected to the rest of the system via an output pipeline register. It is usually updated once per sample cycle. It is connected to the analog output through a high speed sample and hold amplifier, which deglitches the signal, and an active low-pass filter, which removes the sampling noise.

The 16 X 1K data memory is used to hold waveform tables, state variables and parameters. It is interfaced to the buses via two input pipeline registers and an output pipeline register. The S register specifies the address at which data is to be read or written. When the ALU clocks a number into the S register and waits a machine cycle, the contents of the memory location addressed by the S register is available in the D register, which may be clocked onto the D bus by the microcode. To write a datum into the memory, the ALU first puts the address in the S register, then the datum in the W register. One machine cycle later the write is complete.

Adding other hardware to the DMX-1000 is simple; it is connected through pipeline registers. Extra channels of DACs can be added. An analog to digital converter may be added to provide real time audio input. To the ALU it is an input register that may be read once per sample cycle. A larger slower MOS memory could be added to serve as a tappable delay line. This would allow the DMX-1000 to implement digital

reverberation. The delay memory would interface to the ALU just like the data memory, except that it would be slower. It would take five cycles for a read or write to be done. This wouldn't slow down the machine since it could be doing other things during this waiting time.

The DMX-1000 has no microcode branch instructions. This makes the machine simpler and faster. While one microinstruction is being executed, it is held in the microinstruction pipeline register so that the next one can be fetched from the control memory at the same time. Machine cycles that branch would have to take longer, since, when a branch occurs, the next microcode address isn't known until the very end of the machine cycle. The fetch of the next microinstruction couldn't be done in parallel with the execution of the present one. To allow branches one of two things would have to be done. The fetch and execute portions of the machine cycle could be made sequential instead of parallel. This would add 30 to 50 percent to the cycle time, slowing the machine down considerably. Or only the fetches immediately following a branch could be done sequentially. But this would compound a difficulty already inherent in branching, namely that it makes determining the worst case program execution time difficult. The machine's sampling rate must be set so that the program has time to finish executing each and every sample cycle.

Most computer designers are quite rightly willing to pay these penalties on order to have branching. Branches are the only conditionals in many computers, and they are a

necessary part of the iterative loop, a powerful and basic programming technique. For the DMX-1000, however, iterative loops are not particularly attractive. We have to finish execution of the program within a small and definite number of machine cycles, since the program computes a sample and the samples must be equally spaced and close together. Since we have a 256-word control memory that can't be used for anything else but microcode, and since 256 machine cycles is about the longest sampling period we would ever want, we are just as well off putting a sequence of code 5 times in a row in microcode as we would be executing it five times in a loop. And we can provide conditionals of another sort, to get away from the need for branching.

The DMX-1000 has several bits in each microcode word that specify conditions for execution of the microinstruction. Four condition codes in the ALU are set or reset according to the result of each executed instruction. C (carry), when set, indicates a carry from the MSB. V (overflow) set indicates an overflow into the sign bit. Z (zero) set indicates a zero result. S (sign) indicates a negative result, when set. The execution of any microinstruction may be made conditional upon the results of the last-executed microinstruction. There are eight possible conditionals, two for each condition code described above. One of the two for each code allows execution of the instruction if the code is set; the other allows execution if the code is reset. For example, the IFV code in a microinstruction specifies that the instruction is

to be executed only if the V bit is set. IFNV specifies that the instruction is to be executed if the V bit is clear.

The DMX-1000 is designed to be run by another computer, which will be called the master. The DMX-1000 does the number crunching, the high-speed repetitive computation necessary for real-time digital synthesis. The master computer must provide parameters describing the desired sound and the means to be used to generate it. It does this by writing into the control and data memories through an eight bit parallel interface, using a simple protocol. The lowest-order three bits of each control byte received by the DMX-1000 are interpreted according to the following table:

Value	Meaning
0	Halt
1	Single Step
2	Run
3	Load Data
4	Load Microinstruction

When the DMX-1000 is powered up, it is halted, and its memory contents are indeterminate. The master computer must load its control and data memories, and then let it run. A "2" control byte turns on the DMX-1000 processor. A "0" control byte turns it off. A "1" control byte lets it run a single sample cycle, after which it halts. A control byte of value 3 indicates to the DMX-1000 that the next four bytes clocked through the interface are to be interpreted as a 16 bit address in the data memory and the 16 bit contents to be loaded into that address. a "4" control byte indicates that the next five bytes sent through the interface are to be interpreted as an 8-bit microcode address, and a 32 bit microinstruction

word to be loaded into the specified microcode memory location.

When the DMX-1000 is halted, the microcode and data updates can be done at a rate of one update per 400 ns, about as fast as any master computer could transfer the data. When the DMX-1000 is running, only one update is performed per sample cycle. At the end of each sample cycle is a two-machine-cycle update cycle during which the interface circuitry updates a word in one of the memories if such an update has been requested by the master computer. Only a fairly fast master computer will notice the slowdown in the transfer rate when the machine is running. After the completion of the previous update cycle, the five or six bytes needed to describe the next update may be clocked in through the interface. Then the master computer must wait until the end of the sample cycle, which is at most 52 microseconds long. After the update cycle it may send another 5 or 6 bytes. This gives a maximum average byte transfer rate of about 10 MHz - one byte every 10 microseconds. Most minicomputers won't be able to keep up with that for long.

As mentioned above, the master computer is connected through an 8 bit parallel interface. This interface may be connected fairly directly through commercially available interfaces to the buses of most computers. It is also possible to connect it to a UART, resulting in a standard EIA RS-232 interface, so that the DMX-1000 can be connected to a serial line like a terminal. Such an interface is available from the manufacturer. It is cheaper, more flexible and easier to install

than a bus interface, but it is much slower. At 9600 baud, data memory can be updated only 175 times a second. This may be fast enough for some applications but certainly will not allow the full capacity of the DMX-1000 to be used.

The DMX-1000 has enough capacity to implement:

20 simple oscillators with amplitude control and frequency control, or

14 oscillators with envelope control, or

6 voices of frequency modulation, or

15 first order filter sections, or

8 second order filter sections, or

30 linear congruential white noise generators,

or various combinations of the above, such as 10 oscillators and 3 voices of frequency modulation. The DMX-1000 is a small computer to use for digital synthesis. Nevertheless, it is at least as powerful as a large analog synthesizer. And it has the advantage over most other types of digital synthesizers that it is completely programmable; it provides a certain amount of computational resource that can be programmed to do what the user needs done. Most other digital synthesizer designs contain digital analogs of the standard analog synthesizer modules - digital oscillators, digital filters, and so on. This means that most of the time, much of the hardware isn't being used, since few synthesis algorithms will fit perfectly into the machine and use all its hardware. And complete programmability means that the DMX-1000 is virtually guaranteed to be able to implement any synthesis algorithm developped in the future; this can't be said for

other digital synthesizers.

Three major pieces of software are needed in the master computer to run the DMX-1000 from a user-defined score. An assembler is needed to translate symbolic DMX-1000 assembly language into the 32-bit microcode, allowing the user to easily assemble his "orchestra." A score processor is needed to translate the user score format in accordance with a symbolic orchestra description into a processed score. The user-format score contains data ina a form that's easy for the user to understand and modify. The orchestra description provides rules for mapping from the user's input format to parameters in DMX-1000 data memory. A typical note will require many data memory updates, some at the beginning, some in the middle of, and some at the end of the note. The processed score is very simple in format, containing one entry per data memory update required during the piece. Each entry tells the time of the update, the address to be updated, and the value to be put in the data memory at the specified address. Finally, a player is required. It reads the processed score and plays the music in real time by putting the right value in the right data memory location at the right time.

To run the DMX-1000 as a musical instrument from real time human input requires slightly different software. The assembler needed is exactly the same. The user must provide exactly the same orchestra and orchestra descriptor. But the processor and player are combined. For example, if

the user is providing input via a computer-readable organ keyboard, when the user depresses a key the processor/player uses the orchestra descriptor to translate the information conveyed in the keystroke into processed form -- a table of entries in the form described above. This table is merged into an event queue that drives the player.

As of this writing (Nov. 1978) a prototype version of the DMX 1000 has been built and is being tested. The production version is scheduled to become available during the first quarter of 1979. It is expected to be priced at $7995 in unit quantities. The DMX-1000 signal processing computer is an OEM product; it must be built into a larger computer system to be of use. Doing this involves obtaining a suitable master computer, interfacing it to the DMX-1000, and writing a certain amount of software. The DMX-1010 Digital Synthesizer, a stand-alone computer music system, is being developed for users who don't want to solve these problems or who don't already own a computer. It includes the DMX-1000, and an LSI-11 based computer system with floppy disk, CRT terminal and the software described above. Additional software will allow the user to write signal-processing microcode in terms of pre-defined unit generators -- by specifying the number of scollators, filters, etc. to be implemented and the way they are to be connected. A computer-readable 61-note or organ klavier is optional. The DMX-1010 has all the capability of a large analog studio synthesizer, with the following additional capabilities: it is polyphonic; it is software-patchable; and it can play user-defined scores. The DMX-1010

will become commercially available during the summer of 1979.

REFERENCES

[1] Advanced Micro Devices, Inc. The AM2900 Family Data Book. Sunnyvale, CA, 1976