The Development of a Software-Based Real-Time Digital Synthesizer: The VERSATRACS™ System

by

Robert C. Maher

A Report Submitted in partial fulfillment

of the Requirements for the Degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

at the

UNIVERSITY OF WISCONSIN

1985

ACKNOWLEDGEMENTS

Sincere appreciation is expressed to Dr. John Scandrett and Dr. David Hudson of First is Fast, Inc. for their assistance with the computer language *FIRST*. Appreciation is also expressed to Robert E. Crawford, Jr. and Clarence Prudhoe of Market Interface, Inc., for their help, ideas, and critical comments regarding the implementation of the software.

The author wishes to acknowledge the help and cooperation of Dr. A. B. Fontaine, and other staff members of the Department of Electrical and Computer Engineering who assisted with this project.

The author is also grateful to his future wife for her understanding and encouragement during these studies.

This project was supported, in part, under a National Science Foundation Graduate Fellowship.

<=< Table of Contents >=>

>> INTRODUCTION <<

This paper describes a music synthesis system using an Apple][series microcomputer, commercially available peripheral electronics, and original software. This hardware/software configuration is currently being marketed under the name **VERSATRACS** by Market Interface, Inc. of Roodhouse, Illinois (Market Interface is a small company partly owned by the author of this paper).

The original goal of this project was to develop a versatile music system consisting of low-cost, commercially available hardware and advanced original software. The computing hardware chosen was the humble Apple][computer. The Apple][had the advantage of available peripheral electronics for music synthesis, reliable and elegant construction, and the accessibility of technical information. The music system was designed using software to implement most of the outwardly visible features. This involved a small performance sacrifice, but the ease of experimentation and modification made software control a big advantage. The experience gained in this evolving development will be of significant importance if a "stand alone" system is ever constructed.

The system philosophy was to include features for both live performance and recorded playback, with an emphasis on versatile manipulation of the musical parameters. This philosophy also included the incorporation of simple "music processor" features so that any note, rhythm, or timbre change entered into the system would always be subject to subsequent modification, if necessary.

This paper is divided into several sections: Hardware, Software, Evaluation, and Conclusion. Appendices are included to provide miscellaneous information, and a **VERSATRACS** manual is also attached.

>> HARDWARE <<

The **VERSATRACS** system requires the following hardware:

* An Apple][,][+, or][e computer with monitor and at least one disk drive.
* The Mountain Hardware, Inc. MusicSystem digital synthesis boards.
* The Legend Industries, Ltd. "S" card , a 256 Kbyte memory expansion board.
* A four- or five-octave electronic switch clavier keyboard and interface card built by Pratt-Read and Co., and supplied by Market Interface, Inc.
* An AM9511 peripheral card for fast floating point calculations using the computer language ***FIRST***.

This section gives a summary of the characteristics and implications of the hardware.

>> THE APPLE

The Apple][system contains a motherboard complete with 6502 microprocessor, 48 Kbytes of dynamic RAM, eight 40-contact edge connector slots for peripherals, and the ROM's, video generator, decoder, and I/O circuitry required by any microcomputer. The motherboard is connected to a built-in keyboard, a small annunciator speaker, and a built-in switching power supply. The Apple generates a 40 column X 24 line text display, a 40 X 48 pixel low-resolution graphics display, a 280 X 192 pixel high-resolution graphics display, or a mixed display of simultaneous text and graphics. In **VERSATRACS** and most other system configurations, one of the 8 expansion slots is occupied by a disk drive controller interface for one or two 5 1/4 inch floppy disks. The current implementation uses 16 disk sectors, with 140 Kbytes stored on one side of the diskette.

(>> THE APPLE cont.)

The 6502 microprocessor is an 8-bit architecture in a 40-pin DIP. In the Apple][, it operates at a 1 MHz clock rate. The 6502 uses a 16-bit address bus, giving 64 Kbytes of directly addressable memory space. The pinout is shown in Appendix A.

The eight peripheral slots (not including slot 0 on the][e) are essentially identical, and provide the address and data bus lines, ± 5 volts, ± 12 volts, ground, and various clock, interrupt, and control signals. The peripheral slot addresses are memory mapped, and each slot occupies 272 bytes in the Apple's address space. Each slot is provided with partial address decoding hardware to simplify interfacing: a signal is available to indicate when the address bus contains an address within the range available to the particular slot. The peripheral connector pinout, signal description, and memory allocation are given in Appendix A. Apple peripherals can use Direct Memory Access (DMA) by providing the appropriate signals to the Apple and any other peripheral cards. The use of DMA will be discussed below in the description of the synthesizer cards' operation.

The Apple specifications are very modest by today's standards. However, the structure of the microcomputer is quite simple, the memory-mapped peripheral slots are easy to use, and the set-up cost is very reasonable. Also, the system encourages the use of tight, fast, and structured assembly code to make up for the lack of processor speed and direct addressing range.

>> MEMORY EXPANSION

To improve the system performance, a 256 Kbyte RAM expansion peripheral card was selected. The Legend Industries "S" card can be placed in any one of the Apple's peripheral slots, and provides sixteen 12 Kbyte main banks and sixteen 4 Kbyte sub-banks. The memory management must be software controlled.

(>> MEMORY EXPANSION cont.)

The Legend "S" card can be expanded to contain up to 1 Mbyte of memory by swapping the 64 K RAMS on the card with 256 K RAMS.

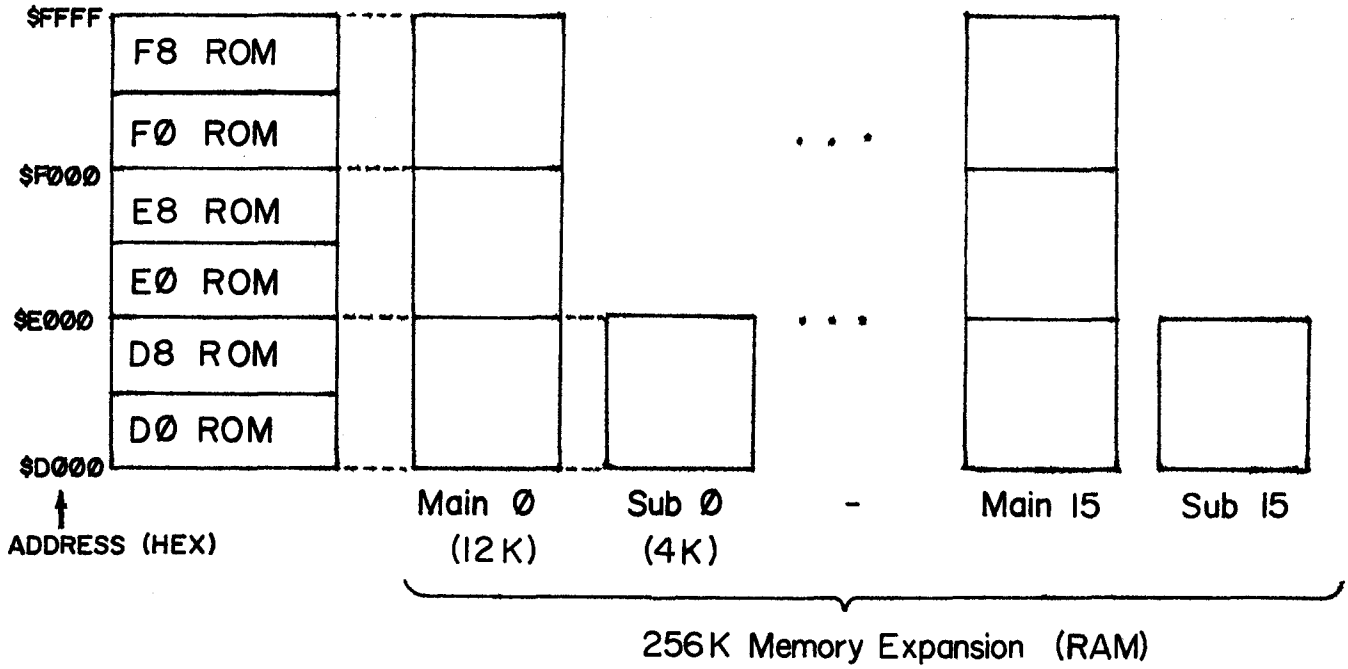
Each of the sixteen main banks occupy the addressing range (hexadecimal) $D000 to $FFFF, and the sub-banks occupy $D000 to $DFFF. The Apple system ROM's also occupy the range $D000 to $FFFF, so the memory must be managed by the software to ensure that the desired bank, sub-bank, or ROM is enabled at the proper time. For example, the memory card must be de-selected whenever a monitor routine from ROM, such as printing a character on the CRT, is to be called. The use of the expansion memory space is detailed in the software section of this paper.

The parallel nature of the memory allocation is given in the diagram below.

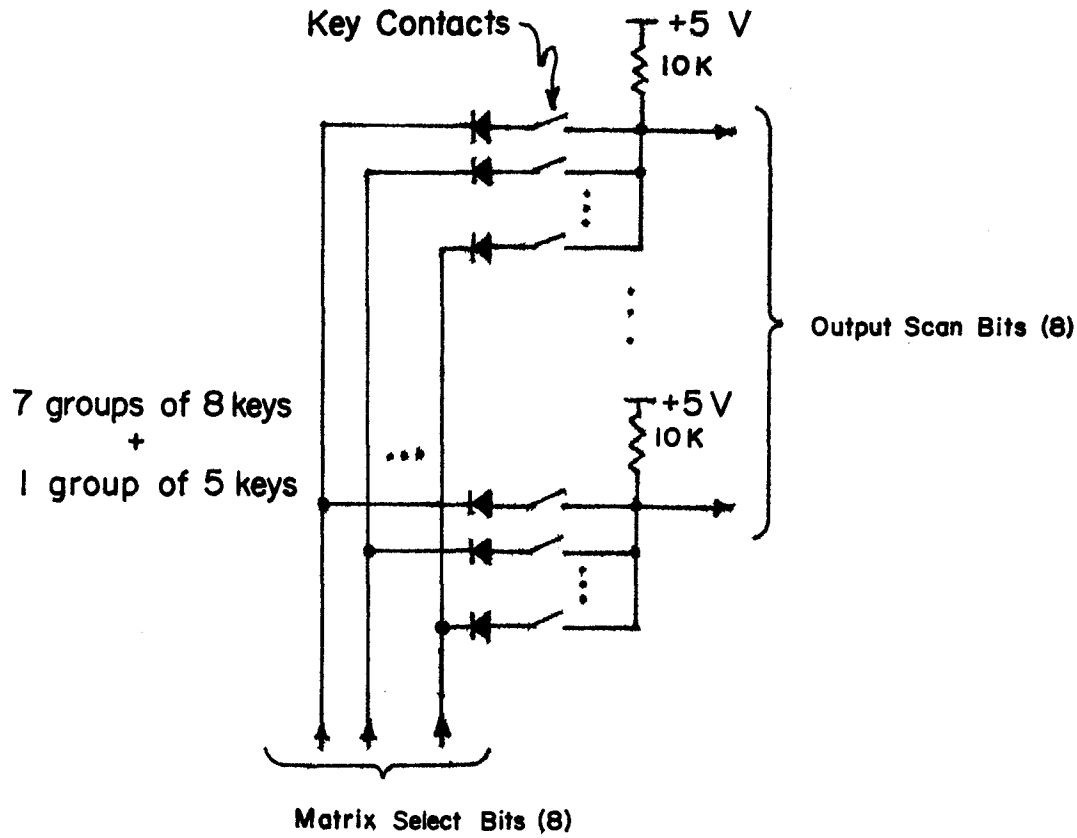
>> CLAVIER KEYBOARD

In a comprehensive music system, a standard piano-style keyboard is very important to allow for conventional composition, experimentation, and performance. Several keyboards manufactured by Pratt-Read and Co. have been successfully adapted for use with the **VERSATRACS** system. The keyboards differ in the type of electrical switches used, the action, or "feel", of the keys, the number of octaves available, and the weight and cost of the mechanical apparatus and case. All use the same diode matrix detection scheme, so a basic software algorithm can be used, with minor modifications, to scan any of the keyboards for key press/release events. The matrix is shown below.

Several types of mechanical switch contacts are used on the Pratt-Read keyboards. One type uses a series of gold-plated bus bars for the output columns, and a small, flexible wire under each key. The wire makes contact with one of the bus bars when that key is depressed.



Apple $D000-$FFFF Address Space



Keyboard Diode Matrix

(>> CLAVIER KEYBOARD cont.)

Another contact type uses a conductive foam material over an interdigitated metal pattern on an epoxy printed circuit structure. A plastic projection or adjustable screw on the underside of each clavier key presses the conductive foam onto the contact pattern, thus closing the switch.

A third type of contact replaces the "foam and fingers" system with soft dome switches similar to those used in hand calculator keypads.

The keys themselves are made of plastic, and a small spring or metal tension band is attached at the back of the key to return it to its resting position when pressure is released.

A feature of some models of the keyboard hardware not presently exploited by the **VERSATRACS** software is a velocity sensing scheme for touch sensitivity. In this arrangement, each key actually contains two switch contacts. One switch closes when the key is depressed a small fraction of its full deflection, and the second switch closes when the key becomes fully depressed. A software routine could measure the elapsed time between the two switch closures, and adjust the sound produced by the synthesizer much the same way the timbre and sound amplitude of a piano may be varied by the way keys are played. The key velocity information, once measured, could also be used to change **any** of the sound characteristics, such as vibrato, pitch tuning, or amplitude history.

>> THE DIGITAL SYNTHESIZER

The actual music-making portion of the **VERSATRACS** system is the MusicSystem sixteen oscillator digital synthesizer made by Mountain Hardware, Inc. The synthesizer consists of two peripheral cards that are placed in adjacent slots inside the Apple. The synthesizer operates using a wavetable synthesis technique, which is described below. The synthesizer output is a pair of standard line-level signals that can drive a pair of headphones, a home stereo amplifier, or a PA system. The two

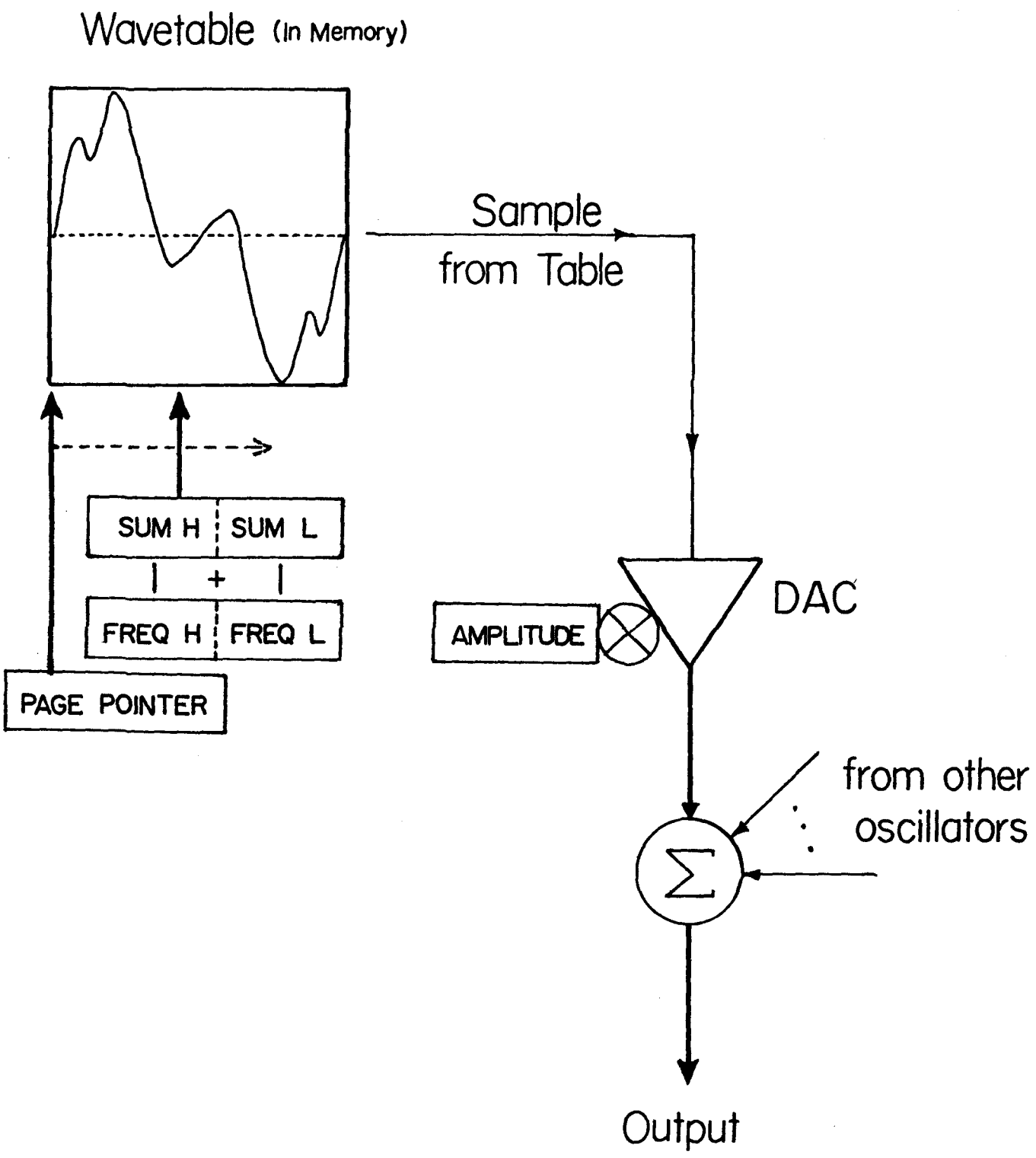
(>> DIGITAL SYNTHESIZER cont.)

signals are actually a stereo pair, with eight of the sixteen oscillators sent to one channel, and the other eight oscillators sent to the other channel.

The synthesizer is conceptually organized into sixteen fully independent oscillators with an overall enable and overall output volume control, all under software direction. Each of the sixteen oscillators consists of a 256 byte wavetable, a 1 byte wavetable page pointer, a 1 byte output amplitude scaler, and a 2 byte frequency specification register. In summary, each oscillator may be independently controlled in output **waveform, frequency, and amplitude.**

The **wavetable** is a seqential list of the samples of the waveform to be produced, or in other words, a digitized representation of the waveshape. In the **VERSATRACS** system, a single cycle of the digitized waveform is usually used. The wavetable is 256 entries long, and each entry is a single byte. Thus, the wavetable is a "drawing" of the wave on a 256 X 256 grid. The wavetables for the oscillators can be stored almost anywhere in the main memory of the Apple, since the setup is controlled by the software. The tables are consulted using DMA to allow fast access to the wavetable entries without relying on slow software read/write schemes and difficult timing loops. When the synthesizer boards are enabled, the DMA controller of the synthesizer "steals" every other 1 microsecond memory cycle to get a new value from a wavetable. Note that this means that the software running on the 6502 processor is executed at one half the normal speed when the DMA system is enabled. Another implication of DMA is that no disk accesses can be made during the time the synthesizer is "on".

For the following description, please refer to the oscillator functional diagram below.



Oscillator Summary

(>>DIGITAL SYNTHESIZER cont.)

The DMA system fetches a wavetable value for each of the sixteen oscillators in turn. Thus, a new sample is fetched every 2 microseconds, and a particular oscillator is updated every 32 microseconds (16 oscillators X 2 microseconds). This is a sampling rate of $1/32 \times 10^{-6}$ = 31,250 Hz. This rate is significant, since the Nyquist sampling theorem states that in order to ensure non-overlapping spectra in the sampled data, the highest frequency allowed is one half the sample rate, or 15,625 Hz. This is the theoretical maximum if an ideal low pass filter were available. The anti-aliasing filter of the Mountain Hardware syntheziser begins to cutoff at a much lower frequency, so the synthesizer output is not really "high fidelity" from a frequency response standpoint.

A wavetable must begin on a page boundary in the Apple's memory, which means a 16 bit address whose least significant 8 bits are all zero. The most significant 8 bits are then sufficient to identify the location of the wavetable, and these bits are placed in the 8 bit page pointer register of the corresponding oscillator. Although **VERSATRACS** uses a separate wavetable for each oscillator, it would be possible to have several oscillators share a wavetable by simply pointing more that one oscillator at the same memory page.

The frequency control of a given oscillator is accomplished by specifying a 2 byte frequency index value in the frequency control register of the oscillator. The most significant byte is the integral part and the least significant byte is the fractional part of the number of points in the wavetable to skip between samples. The hardware adds the index register value to the contents of an accumulator register every time a new wavetable value is to be fetched for that oscillator. The most significant byte of the accumulator register is concatenated to the wavetable page pointer, and the result is the 16 bit address of the next table entry to fetch. As the value in the accumulating register increeses due to the addition of the frequency control register contents, the access address

(>> DIGITAL SYNTHESIZER cont.)

steps across the wavetable. Eventually, the accumulating register reaches its terminal count and "rolls over" to begin scanning the table again.

The required value in the frequency control register to give a particular output frequency can be determined as follows:

1/1.020484 MHz X 32 memory cycles/sample

= 31.3577 microseconds/wavetable sample

Frequency Index Units = "freq" Hz X 31.3577×10^{-6} X 256 samples/cycle

X 256 radix shift

i.e. For a frequency of **440 Hz**, a frequency index of **904** would be required.

Two points should be mentioned at this time:

First, a change of one frequency index unit corresponds to an actual output frequency change of **0.4866 Hz**. Thus, the synthesizer must round all frequencies to the nearest multiple of 0.4866 Hz. In the above example, this means the actual output frequency generated with a frequency index setting of 904 would be **439.89 Hz**. This resolution limitation has been of little consequence so far in the development of **VERSATRACS.**

Second, it should be noticed that specifying a frequency index above 32,767 will exceed the sampling limits of the hardware, since that would correspond to an output frequency above the Nyquist limit. No damage is done by specifying a higher frequency index, but little of conventional musical value can be obtained.

Once the sample from the table has been fetched out of the Apple's memory, it is sent to an 8 bit digital to analog converter. The reference voltage provided to the DAC is actually the output of another DAC whose input is the contents of the oscillator amplitude register. This amplitude

(>> DIGITAL SYNTHESIZER cont.)

scaling DAC in turn gets its reference voltage from a third DAC, whose input is the contents of the overall amplitude register. The samples from a wavetable are thus analog multiplied by the oscillator amplitude and overall amplitude settings. The boards do not contain separate DAC chips for every oscillator, because only one sample is being handled at a particular moment. The three DAC chips are time multiplexed to the different oscillators, which saves on hardware cost, size, and power consumption. The results of the conversion and amplitude scaling are filtered and mixed with the outputs of the other oscillators, and routed to one of the two stereo channels (The even-numbered oscillators are sent to one channel, the odd-numbered to the other).

To the programmer, the synthesizer boards appear to be sixteen digital oscillators, with independent control of waveform, frequency, and amplitude. The major implications of this structure will be discussed in the software sections of this paper, but one characteristic should be mentioned here: The controls of the oscillators are directly accessible by the software, and changes can be made whenever desired. This is an important advantage over completely hardware-controlled systems in the ability to create, adjust, and experiment with new ideas and features. However, the dependence on software makes the job of the programmer more complicated, since the parameters to be changed must be timed properly with respect to the many other software tasks which must also be accomplished. The author hopes that the lessons learned in this software development effort will be directly applicable in a high-performance multi-processor implementation for future development.

>> MIDI INTERFACE OPTION

A current trend in the music synthesizer business has been the standardization of signal interconnection and communication methods. At present, the Musical Instrument Digital Interface (MIDI) standard

(>> MIDI INTERFACE OPTION cont.)

has become the most widespread. The MIDI standard covers the electrical and digital protocol aspects of what is essentially a simple local area network for digital musical instruments. The information sent via MIDI includes commands to start and stop notes, to change instrument timbre, and to change other parameters. The command messages can include a channel identifier so that up to sixteen MIDI-equipped synthesizers can be independently controlled. The data is transmitted over a shielded cable in a baseband digital format.

The MIDI standard is rather restrictive, in that no provision is included for avoiding simultaneous use of a single connection line by two MIDI transmitters, nor is there any way for a MIDI device to acknowledge reception of a command. In short, only one source of MIDI commands can be used successfully in a MIDI network, and the source must believe as a matter of faith that its messages are getting through! However, the simple-minded nature of the interface protocol has the advantage of requiring very simple hardware. In most cases, a 6850 ACIA chip is used for transmitting data through an opto-isolator and onto the connecting cable.

The **VERSATRACS** system can be used as a MIDI source if an interface card is plugged into one of the Apple's peripheral slots. When this option is used, the software can issue commands to MIDI-compatible equipment connected to the interface card. MIDI interface cards can be constructed from scratch, or purchased commercially. Many commercial interface cards contain extra hardware, such as timer chips, which can be of use to a programmer if the software is to be dedicated to a particular interface design.

>> GENESYS 1 SYNTHESIZER

One other piece of hardware to mention before turning to the software is an Apple-based synthesizer recently introduced by Mimetics Corp. The Genesys 1 board combines the keyboard interface and synthesizer functions on a single peripheral card. The Genesys 1 thus uses one expansion slot, instead of the three required by the two Mountain Hardware synthesizer cards and separate keyboard interface. The new card also has on-board memory, eliminating the need for DMA. This allows the Apple to run at full speed, and more memory is available in the Apple for program code because no wave tables need to be stored permanently. The Genesys card has several other features not found in the Mountain Hardware cards, but the new card can be operated in a somewhat "crippled" mode and used with a slightly modified version of the **VERSATRACS** software. The Genesys version of the **VERSATRACS** software is now running, and contains all the features of the Mountain Hardware version. Both versions appear the same to the user.

>> SOFTWARE <<

At this point, the basic hardware requirements and operation methods have been described. This section deals with the software requirements of the **VERSATRACS** system, beginning with a description of the programming language used, called ***FIRST***.

>> *FIRST* -- AN INTRODUCTION

The **VERSATRACS** system is implemented using a little-known computer language called *FIRST*. This language is similar in many ways to the language FORTH. *FIRST* was developed several years ago by Dr. John Scandrett and Dr. David Hudson at Washington University in St. Louis, Missouri. Both men have formed a company called First is Fast, Inc. to do consulting in the use of micocomputers in industrial control applications. The language *FIRST* was developed with on-line control and data acquisition tasks in mind, so it is very versatile in the music synthesis system described in this paper.

The language requires an AM9511 floating point co-processor card in slot 5 of the Apple. The language structure is simple: a particular process or sequence of operations is defined as a **word**. The word definition is conceptually just a subroutine or procedure, and is invoked by typing the word on the keyboard or including the word in a subsequent word definition. A complex operation is generated by defining a word as a chain of other previously defined words in an elegant, hierarchical structure. The definition process may be continued until the desired program has been constructed. In this way, the features of the language can be expanded at any time by simply defining a new word. A simple example is given below:

The word : (colon) indicates a new definition is following. The definition is concluded with a ; (semi-colon). Two of the many words supplied with the *FIRST* language are **CLS** (clear screen) and **BELL** (ring annunciator bell). If a function that rings the bell,

then clears the screen is desired, a word can be defined to do this:

: **BLCL BELL CLS ;**

Now whenever the word **BLCL** is typed in from the keyboard or included in another definition, the computer will ring the bell and clear the screen.

FIRST includes the common **FOR-NEXT, WHILE-IF-RPT,** and **IF-THEN-ELSE** structures found in most languages. Additional structures can be formed using the word definition features described above. A few other characteristics of the *FIRST* language are its **stack arithmetic structure, hash table address lookup**, and **easy access to assembler code.**

The **stack organization** means that arithmetic formulas are entered in "reverse Polish" form, and intermediate results may be stored on the stack itself. For example, using the words **+** (addition), ***** (multiplication), **/** (division), the formula

(3 + 2) x 6 + 52 / (9 + 2)

in normal notation is represented by

3 2 + 6 * 52 9 2 + / +

Note that the order of the operands is not changed, but no parentheses are required due to the inherent priority properties of the stack algorithm. Most stack operations are performed on the top two elements of the stack, but *FIRST* includes various swap and rotate instructions for different manipulations, and others can be defined directly by the programmer, if necessary.

The **hash code lookup** technique eliminates the need for linear searches to find jump addresses when compiling the program code. This allows the software to be compiled as it loads. In most cases, the time required to read the program off the disk and compile it is essentially the same as the time required to read the disk only. The hash table is formed

by performing an arithmetic manipulation on the characters of the word defined following a colon definition. Once the jump address is put into the table position corresponding to the result of the arithmetic manipulation, any subsequent occurances of the defined word will directly identify the appropriate jump address from the hash table.

Access to **assembly language programming** is vital for optimum speed and minimum program size. The *FIRST* language contains provision for inclusion of an efficient 6502 assembler. It is possible to define words that are entirely or partially constructed of assembly language constructs. The ease with which the programmer can jump between assembly language routines and standard *FIRST* structures makes real-time programming much less trouble: fundamental programs requiring speed and memory efficiency may be written in 6502 assembly language, while complex structures can be controlled by a driver routine written with the assistance of the *FIRST* language compiler.

Several *FIRST* language examples are included in Appendix D, and in the *FIRST* language manual.

More information on *FIRST*, system disks, manuals, and fast floating point cards for the Apple are available from First is Fast, Inc., 2401 Schuetz Road, Maryland Heights, Missouri 63043 (314) 432-1999.

>> OSCILLATOR AND VOICE CONCEPT

A useful synthesizer must produce interesting sounds. This is essential regardless of whether acoustic instrumental sounds are to be simulated, or just experimental tonal mixtures. The sounds produced by acoustic instruments are generally quite complex spectrally. More significant from a synthesis point of view, the spectrum may change rapidly during the entire duration of the musical note. Certain frequencies may be dominant during the attack, or onset, of the note, but then give way to a completely different sound as the note ages. Effects such as the "chaff" heard at the beginning of a note played on the flute, or the impulsive sound of a piano's hammers striking its strings, are very important to the quality of an acoustic simulation. The same consideration holds true for interesting sounds of all types.

The simple wavetable synthesis method employed by the **VERSATRACS** synthesizer hardware means that an oscillator repeatedly produces the same waveform output. If the sound was simply gated on to start the note and turned off sometime later to conclude the note, the result would be like a car horn: the sound gets monotonous very quickly. Acoustic instruments sound more interesting than car horns because the sound characteristics **change with time.** The manipulation of the data points in the 256 byte wavetables during real-time play was determined to be too cumbersome to be practical, but some method was required to shift the frequency spectrum of the synthesizer over time. One possible solution was to use more than one of the available oscillators for each note. This technique requires some attention to handle all the details.

The oscillator allocation problem involves several compromises. In particular, the hardware contains sixteen independent oscillators which would potentially permit sixteen simultaneous musical lines, called **voices.** Using more than one oscillator per voice allows for more interesting sounds, but fewer simultaneous voices. A flexible solution would be fully dynamic allocation of oscillators, where different voices

(>> OSCILLATOR AND VOICE CONCEPT cont.)

make use of different numbers of oscillators at different times. Initial attempts to implement this idea revealed it to be too demanding of both memory and computing power. It was apparent that a fixed number of oscillators per voice would have to be the method of allocation.

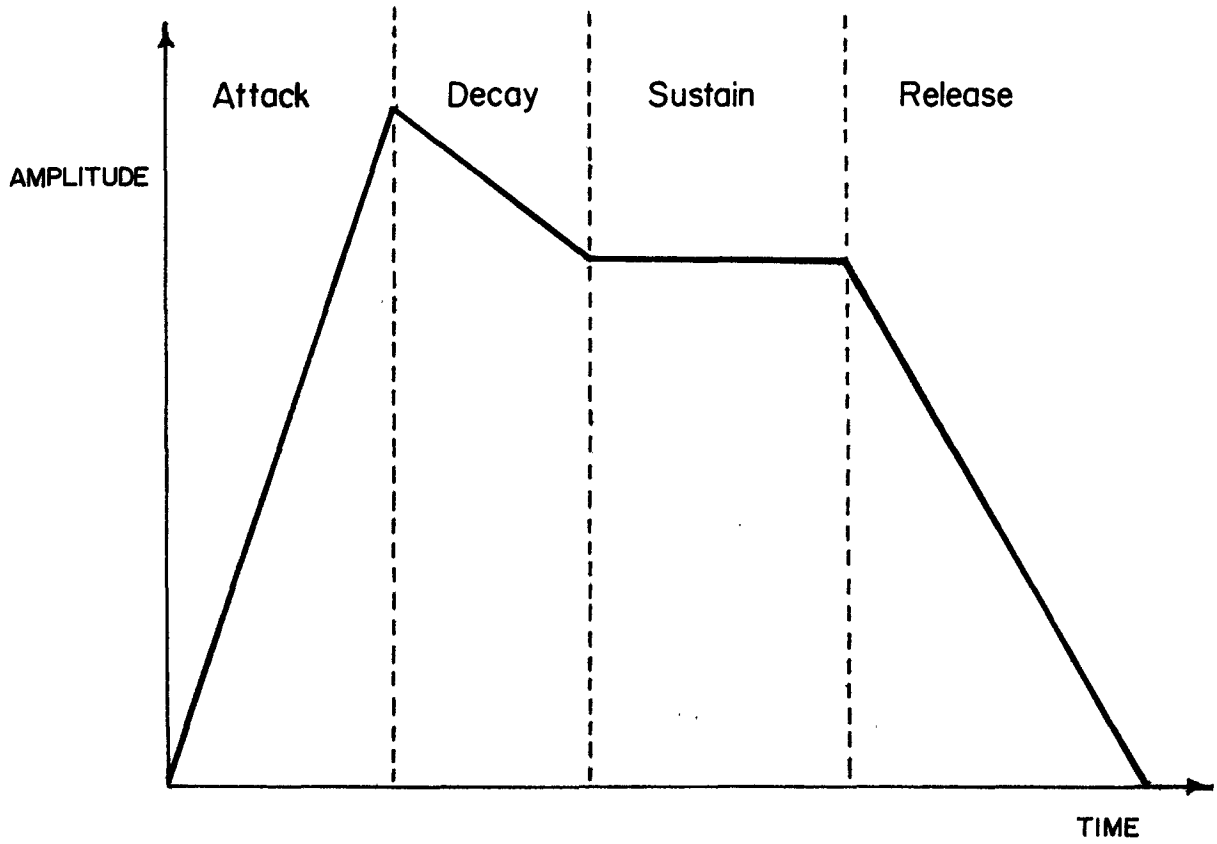
An informal check of several examples of piano music, various orchestral works, and a few popular vocal pieces showed that eight voices (2 oscillators/voice) would be more useful that five or less voices (3 or more oscillators/voice). Thus, the solution adopted was to assign a musical note to **two of the sixteen** available oscillators. In this way, one oscillator could contain the **attack** waveform found early in the evolution of the note, while the second oscillator could contain the **steady-state** waveform. By ramping down the volume of the first oscillator while simultaneously ramping up the volume of the second, a central "blurred" region with both oscillators on could successfully mask the shift from initial timbre to final timbre.

In summary, **VERSATRACS** can produce **up to eight simultaneous voices**, where each voice makes use of **two of the sixteen oscillators** provided by the hardware. For example, if each part of an eight part song requires no more than one note sounding at a given instant, **VERSATRACS** would be able to handle the eight parts simultaneously and independently.

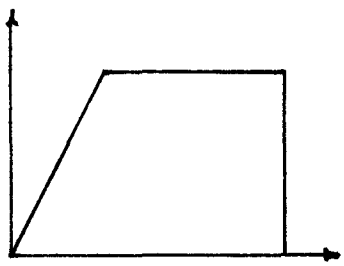
>> AMPLITUDE ENVELOPE

The amplitude variation of a note as it ages is referred to as the **amplitude envelope** of the note. Many electronic synthesizers control the amplitude vs. time characteristic of notes using an **A**ttack - **D**ecay - **S**ustain - **R**elease (**ADSR**) envelope representation. This scheme treats the amplitude evolution of a note as four successive stages, each with a rate or level value (see diagram).

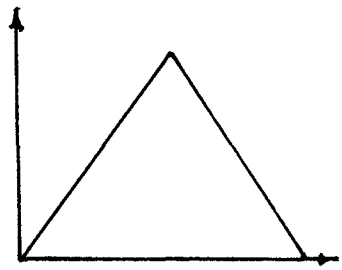
The ADSR representation is quite flexible if all the conceivable rates, limit values, and breakpoints are controllable. ADSR has the advantage



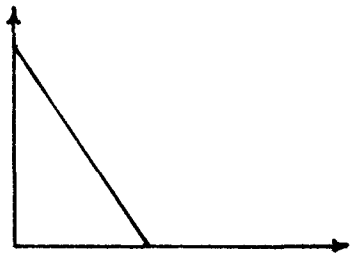
A few other ADSR examples...



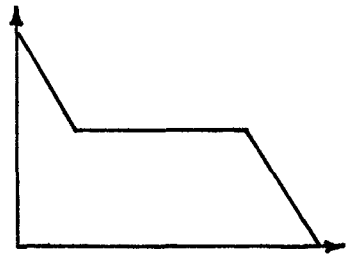
(D=R=Ø)



(D=S=Ø)



(A=D=S=Ø)



(A=Ø)

ADSR Representation

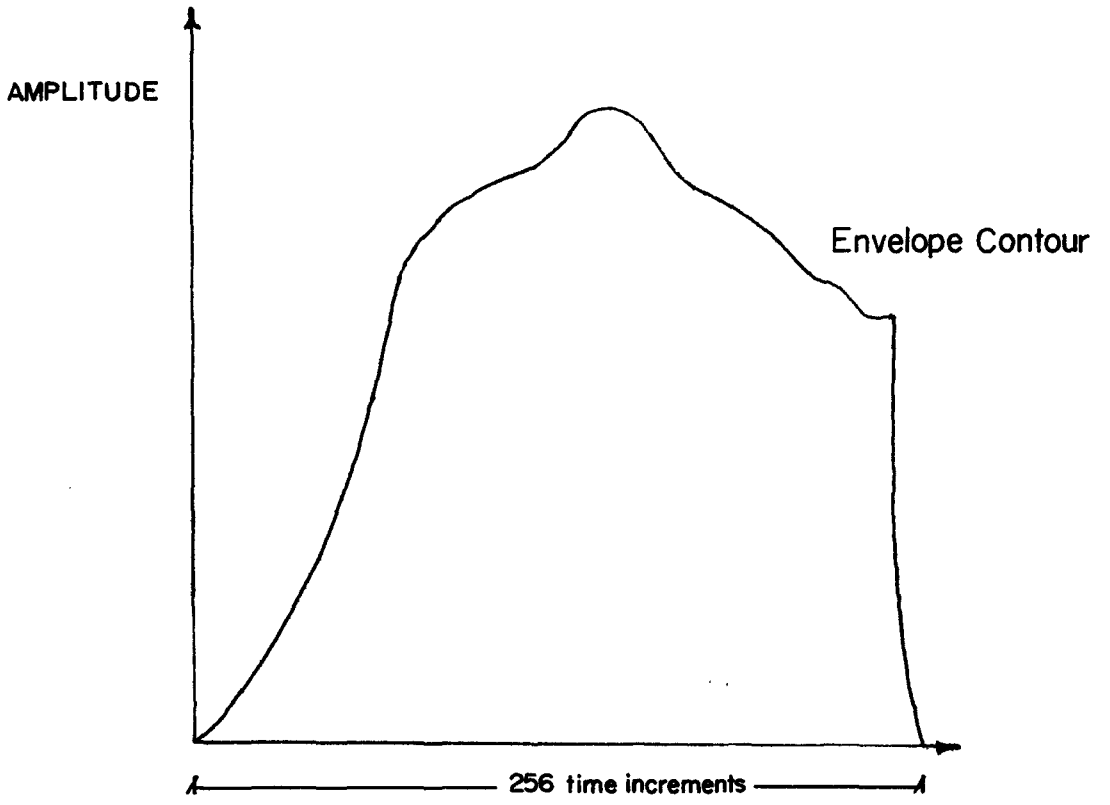
(>> AMPLITUDE ENVELOPE cont.)

that a small number of parameters can produce a convincing result. However, an ADSR system must adapt to the rate at which notes occur, and the duration of those notes. For instance, if a note's duration is shorter than the time required to fully execute the ADSR envelope, some means must be included to properly truncate one or all of the four stages. Another problem is that the four segment envelope cannot produce all the subtle effects required to fully exploit the oscillator pair concept described above. The **VERSATRACS** system provides for subtleties by using a separate 256 byte envelope table to describe the amplitude characteristic for each oscillator. This obviously requires more memory than that for storing the ADSR parameters, but any envelope that can be drawn on a 256 x 256 grid format is acceptable (see diagram).

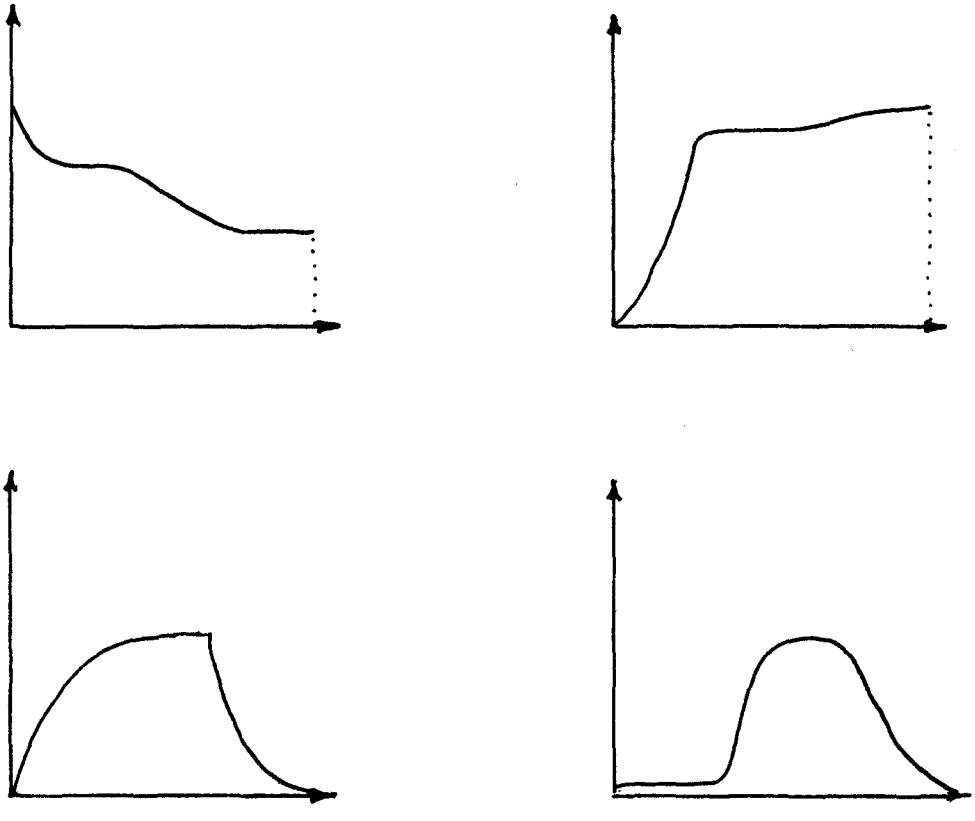
For acoustic instruments, the attack portion of a note generally provides the most important amplitude cues, while the steady-state and release portions are less critical for an adequate simulation. For this reason, the 256 bytes of the envelope tables are scanned sequentially until the 256th point is reached. The amplitude then sustains at that level until the note is terminated. The release is accomplished by ramping the amplitude from its final sustain value down to zero. The ramp-to-zero scheme is also used when a note is terminated before the entire envelope table has been scanned: whatever the amplitude was at the time of termination, the volume is ramped from there down to zero. The envelope for a note may be defined entirely within the 256 bytes of the table by ending at zero volume, or the envelope table may be a complex attack pattern, with the steady-state amplitude and release starting point set by the last value specified in the envelope table.

>> ENVELOPE AND DECAY RATES

Up to this point, no provision for adjusting the envelope table and the rate of final decay has been described. In **VERSATRACS**, the rate at



A few examples...



VERSATRACS Envelope Representation

(>> ENVELOPE AND DECAY RATES cont.)

which new values are fetched from the table is fixed using an 8 millisecond hardware interrupt as a time base. The interrupts can be generated by the synthesizer hardware, if desired, and they make a convenient way to have a relatively slow system clock for various timing and polling tasks. The interrupt handling routine is used to increment several clock variables and set several flags, one of which is the envelope fetch indicator.

If a new value was fetched from the table on every interrupt, the table would provide 8 msec X 256 = 2.048 seconds. This time would be too short for certain sounds. More importantly, the computer would be hard-pressed to keep up with the sixteen envelope tables at that rate, along with all the other control tasks. Instead, the oscillator amplitudes are read from the tables every fourth interrupt, giving a maximum table scan time of more than eight seconds.

To accomodate music with notes of shorter duration, a rate number is used for each pair of oscillators. The rate number, called the **envelope rate**, is simply the number of points added to the last envelope table pointer value to calculate the next pointer value. for example, an envelope rate of one means that every point in the 256 byte table is used, while an envelope rate of two indicates that only the first, third, fifth, etc. points are used. Thus, when a number greater than one is used for the envelope rate, the duration of the envelope pattern is **compressed** by that amount.

The envelope rate control allows quick adjustment of the amplitude characteristics of a voice. Envelope rates from 1 through 20 have been used with good results, but this simple skipping technique results in an understandable loss of resolution of the details of the envelope table.

The rate at which the amplitude of an oscillator is ramped down to zero when a note is released is controlled by a number called the **decay rate**. The decay rate is subtracted from the current amplitude of the oscillator on every other interrupt (every 16 msec), until the amplitude

(>> ENVELOPE AND DECAY RATES cont.)

reaches zero. This allows louder notes to decay for a longer time than softer notes when released. Decay rates from 1 through 20 are used.

The envelope and decay methods used in **VERSATRACS** are simple to implement in software, using arrays to keep track of pointers and amplitude values. Although the amplitude techniques are not ideal in all ways, many informal experiments have shown them to be a reasonable compromise between complicated computing requirements and speed of execution.

At this point, the discussion has covered the wavetable, the envelope table, the envelope rate, and the decay rate. Two other oscillator and voice controls are used.

>> FREQUENCY OFFSETS

Since two oscillators are assigned to each voice, various interesting effects can be obtained by exploiting the independence of the oscillators. Already mentioned above, is the ability to provide a timbre shift over time by choosing appropriate wave and envelope tables. Another method of adjustment would be to have both oscillators **on** simultaneously, but with **different** frequency indices. If this **frequency offset** is small, the effect produced is an amplitude beat pattern at a frequency equal to the oscillators' frequency difference. Small offsets give a "fat" and "resonant" result that can help produce a pleasing sonority. It is also possible to offset the frequencies by large amounts to obtain different effects. A one octave offset, for instance, gives the opportunity to create more harmonically rich timbres than could be obtained by other means.

VERSATRACS uses a multiplicative offset scheme to allow offsets in one **cent** increments over a two octave range (**1 cent** = $^{1}/_{100}$ of a semitone, or a frequency change by a factor of [**freq** / 100] * [$2^{1/12}$ - 1]). Due to the frequency resolution limitations of the hardware mentioned previously, the actual frequency produced is rounded to the nearest 0.4866

(>> FREQUENCY OFFSETS cont.)

Hz. The offset setting is tied to a particular voice, so each of the eight available voices may have its own offset value.

The frequency offset serves another important function: prevention of uncontrollable interference between oscillators. The hardware has no provision for control of the relative phase of the oscillators, so any two oscillators set to exactly the same frequency will either constructively or destructively interfere depending on the random startup phase of each. The frequency offset feature overcomes this problem by ensuring that the relative phase between two oscillators making up a voice is constantly progressing with time.

>> VIBRATO

The final oscillator and voice control is **vibrato.** Vibrato is a variation in the frequency of a note with time. The variation is often a deterministic function of time. Vibrato adds interesting richness to the harmonic content of the sound, due to spectral spreading caused by the frequency modulation. Vibrato also helps simulate real acoustic instruments, such as the members of the violin family, the oboe, or the flute.

As a frequency modulation process, vibrato follows a general form

$$\textbf{frequency} = f(g(w_0,t) \times t),$$

where w_0 = angular frequency, and t = time. In the case of **no** vibrato, the function $g(w_0,t)$ is simply w_0, with no time dependence. With vibrato, $g(w_0,t)$ is some time varying function, such as

$$w_0 \cdot [1 + d \cdot \cos(w_v \cdot t)],$$

where d = vibrato depth index, usually $\ll 1$, and w_v = vibrato pattern repetition angular frequency. The **vibrato repetition rate** controls how fast the vibrato pattern repeats. The **vibrato depth index** sets the

(>> VIBRATO cont.)

maximum amplitude of the frequency deviation. Musically useful depths depend upon the center frequency, the periodic waveform used, the vibrato rate, etc.

Due to the tight timing constraints of the **VERSATRACS** system, it was not feasible to calculate, continuously in real time, the vibrato function exactly as indicated above. The function above would require several floating point calculations and conversions, and, even with the fast floating point co-processor, that task would occupy most of the processor capability if all sixteen oscillators were to have active vibrato simultaneously. Instead, the cosine function is calculated using a 256 byte lookup table, and the required multiplication is carried out using a special-purpose assembly language routine optimized for the vibrato calculations only. (Note that the cosine function could be replaced by any other periodic function).

The vibrato rate is timed using the hardware interrupts, and uses a software algorithm similar to the hardware frequency index technique. The vibrato rate index is a two byte value repeatedly added to a two byte accumulating value. The most significant byte of the accumulated value is used as a pointer into the vibrato lookup table. The rate at which the vibrato calculation is performed must be fast enough to avoid audible frequency "jumps", yet slow enough to keep from tying up all the available processing capability.

The **VERSATRACS** plan originally called for individual vibrato rate and depth controls for each of the eight voices. This resulted in a very out-of-tune sound, since the fixed rates and depths caused notes played as a chord to go in and out of tune with each other as the vibrato patterns changed their relative phase. Various means of getting around this problem were proposed, but all were too cumbersome to include in the later versions of the program. Until the vibrato interactions can be further examined, the vibrato rate has been defined as a single **global**

(>> VIBRATO cont.)

parameter that controls all eight voices in parallel. Each voice has an independent depth control.

Some trial and error revealed that vibrato rates vary considerably in conventional music, and experimental music may use an even wider range. For this reason, **VERSATRACS** accomodates vibrato rates from about five seconds per cycle to around eight cycles per second.

>> INSTRUMENT CONCEPT

In **VERSATRACS**, 48 wavetables and 48 envelope tables are stored on-line in the extended memory space, as well as on floppy disk. The number 48 was chosen because 48 256 byte tables occupy exactly one of the 12 Kbyte main memory banks, and all 48 titles can be displayed, 2 per line, on the Apple's 24 line text display. The waves and envelopes can be mixed and matched into groups of wave and envelope pairs, thus defining a particular timbre and amplitude characteristic. For ease of operation, a means of saving configurations of waves, envelopes, and various other control parameters was a must. These configurations are often called "presets" or "patches" in the electronic music vernacular, since the first electronic synthesizers required knobs to be preset and spaghetti-like patch cords to be routed. In **VERSATRACS**, the "patches" are called **instruments**. Each instrument definition is stored on disk and in memory, and up to 40 instruments are accessible at any time. Further, each of the 40 instruments is actually made up of four separate timbreal definitions, called **levels**, or **sub-instruments**. Each level contains a pair of waves and envelopes, and the required parameter data. This method of organization was adopted after it was found that a group of pre-determined modifications for each instrument would be very useful. Only one of the sub-instruments is active at a particular moment, but the others are available quickly. For example, an instrument called STRINGS could have four sub-instruments that range from a bowed sound, to

(>> INSTRUMENT CONCEPT cont.)

a pizzicato sound. Alternatively, the levels may be entirely different sounds, such as an instrument called COMBO that has a trumpet, clarinet, electric guitar, and string bass as sub-instruments.

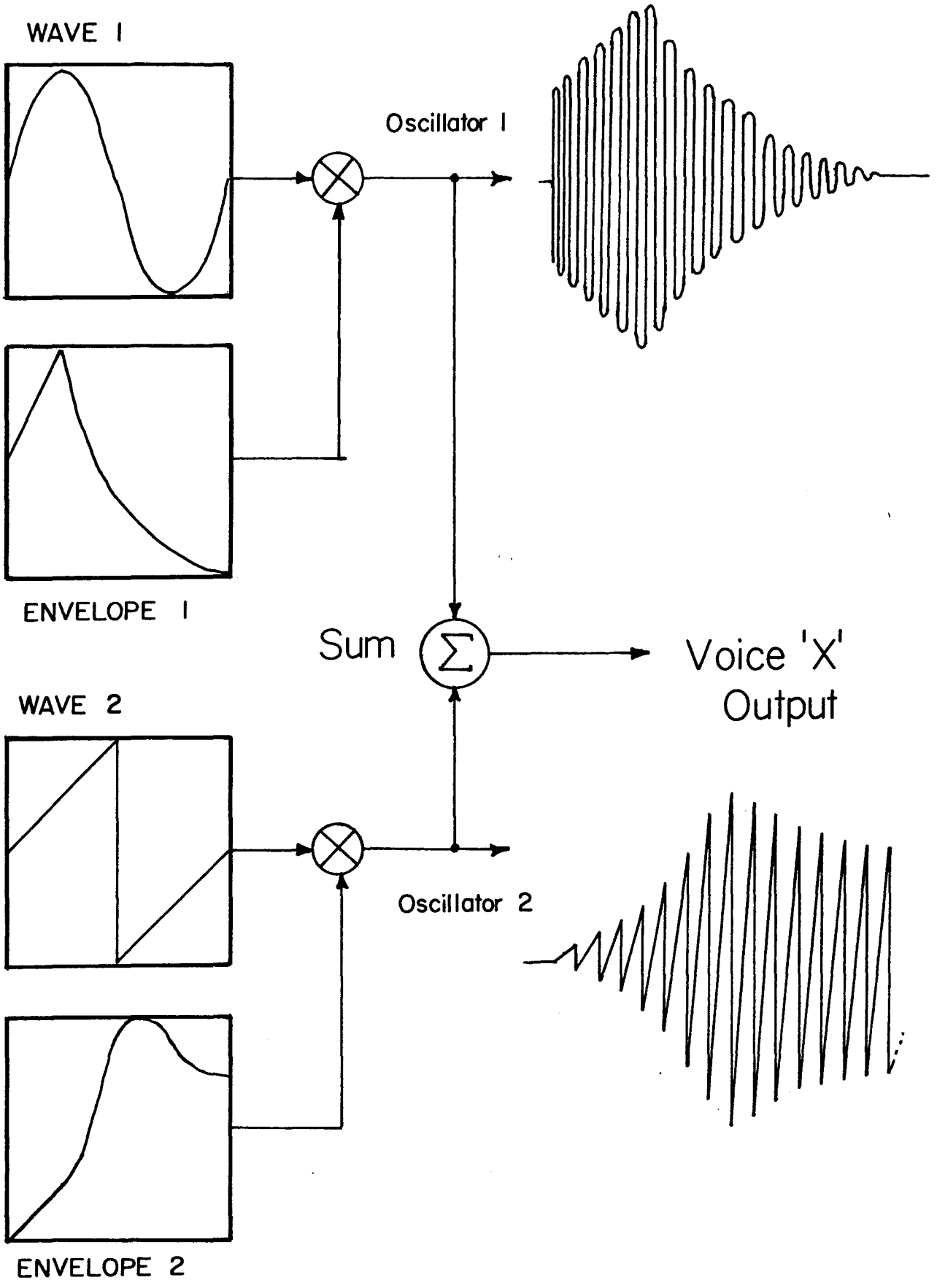
Each instrument definition consists of:

* the envelope rate
* the decay rate
* the vibrato depth index
* the frequency offset
* the two wave tables
* the two envelope tables
* the voice amplitude setting
* the octave specification

The **voice amplitude** setting allows different voices to be scaled for different loudness levels. This will be described further in subsequent sections. The **octave** specification is important, because it allows each instrument to be played in the proper register, i.e., a tuba plays in a lower range of pitch than a piccolo, but both may be played simultaneously in different voices by specifying the appropriate octave number. The octave setting also allows the range of the keyboard to be extended up or down, despite the presence of only five octaves of keys on the keyboard.

VERSATRACS allows the instrument definitions to be modified at will, and any changes may be saved to the memory and disk for later use, if desired.

This concludes the introduction to the oscillator and voice controls. A functional block diagram of the two oscillator voice is shown below.



Versatracs Voice

>> KEYBOARD REAL-TIME INPUT

Once a voice has been defined with waves, envelopes, offsets, etc., some means of starting and stopping notes in sequence is required to make music. In the **VERSATRACS** system, one of the required features was to produce music in real-time in response to key presses on a piano-style keyboard. The keyboard mechanism, described above in the hardware section of this paper, can be scanned by the software. The keyboard currently in use occupies two bytes of the Apple's peripheral I/O space. The software scans the 61-key keyboard in eight key blocks by writing a bit-mask to the **matrix select** address, and then reading the current key pattern from the other address, the **output scan buffer**. The bit mask is simply a byte value with one of the eight bits low (0), and the other seven bits high (1). This connects the corresponding eight key switches to the output scan buffer. The bit pattern read from the buffer contains zeros in the positions of keys being pressed, and ones in the positions corresponding to inactive keys. The software can then compare the current pattern to the pattern observed when the last scan was performed. In this way, the key presses and releases can be detected and recorded for use by the note start and stop routines. The scan process is repeated frequently to keep up with the fingers of the person using the keyboard. A scan repetition rate of 32 milliseconds is used. This has provided adequate time resolution without picking up switch bounces.

In its most basic form, the **VERSATRACS** software can be used to start and stop notes according to the keypress information provided by the performer. This is called Live Keyboard mode, and with it, the system operates like any electronic keyboard instrument. In Live Keyboard mode, the instrument active on the keyboard can be changed at any time by pressing a few keys on the Apple's keyboard. Other sound parameters, such as the overall volume and vibrato rate, can be changed the same way. The

(>> KEYBOARD REAL-TIME INPUT cont.)

system can display various types of information on the CRT, such as a list of the 40 instrument titles, the current instrument and parameter settings, a flashing metronome indicator, etc. Some of these features will be described later, and all are explained for the user in the manual found at the end of this paper. System "HELP" screens are available on-line to answer simple questions or solve common problems.

If one could play the piano adequately, and live performance was the only time the synthesizer would be used, the Live Keyboard mode would be sufficient for all needs. However, most users of the **VERSATRACS** system -- including the author -- are (1) interested in recording what is played, (2) interested in trying out various instrumental combinations and orchestrations, and (3) not virtuoso pianists. For support of these additional needs, additional modes -- such as recording -- were required.

>> THE **EVENT** CONCEPT

The **VERSATRACS** system was designed with recording and editing features in mind. This meant that the recorded data format would have to accomodate insertions, deletions, and modifications without undue complexity.

Recording is accomplished by writing eight byte records to a reserved region in the extended memory card. The eight byte records are called **events**, and are stored in the order in which they occured. In the case of a key press, the event data includes the time at which the key press was noticed, the key number of the key pressed, and the elapsed time between the press and release (the note's duration). This event recording method allows notes to be modified by changing the data stored in the eight bytes of the event. Insertion and deletion of notes is accomplished by simply inserting or deleting an eight byte event. Four of the main memory banks in the extended memory are set aside for event records, which is enough room for 6144 events.

(>> THE **EVENT** CONCEPT cont.)

The event recording method also allows **track-on-track recording**. This means means that two or more passes can be made through a piece of music, in order to record new notes, while any previously entered notes play back in the proper time sequence. The reference to "tracks" comes from the common recording studio technique of recording different parts of a song on different physical tracks on a multiple track tape recorder. In the studio, this method allows the same musician to sing, play piano, play drums, etc., all in one composition, by recording each part in a seperate track on the tape. The complete composition sounds like all the instruments were played simultaneously, when, in fact, they were not. **VERSATRACS** allows entry of up to eight tracks of recorded notes. The events from the keyboard are all stored sequentially in the same region of memory, but each event is given a track number indicator so that the individual tracks are still distinguishable to the software. Each of the eight tracks may have an entirely **different** instrument definition assigned to it, so the music may have up to **eight** timbres playing simultaneously (The problem of how to map the eight tracks and instruments onto the eight available oscillator pairs will be discussed later).

In the memory, the note event record appears like this:

BYTE #

0 - Note event start time, least significant byte
1 - Note event start time, most significant byte
2 - Track number (0-7)
3 - Key number (0-60)
4 - Note duration time, least significant byte
5 - Note duration time, most significant byte
6 - (Spare)
7 - (Spare)

The two spare bytes are for future uses, such as the key press velocity data described in the hardware section of the paper.

>> SYSTEM TIMING AND SEQUENCING

As mentioned, the system time base is fixed via a hardware generated interrupt occurring every 8 msec. The interrupts simplify timing, because the clock "ticks" are not affected by any temporary delays in the main software loop.

Early in the development of the system, the interrupt handling routine simply incremented a "clock" variable by one every time an interrupt occurred. At 8 msec per clock tick, and with two bytes allotted for the time indicator, this gave about nine minutes of recording time before the clock "rolled over". Nine minutes was considered satisfactory, but the single-tick-per-interrupt scheme did not allow for varying the clock rate.

Variable clock rate capability was considered important, because many useful recording techniques require entry of notes at a comfortably slow pace, with the final result somehow sped up to the desired tempo. Clock rate changes over time could be used for *accelerando* (gradual increase in speed) or *ritard* (gradual decrease in speed). The method of recording the music as a sequence of note **events** enables the speed changes to occur without getting the "chipmunk" effect heard when playing a tape recorder at the wrong speed: the only data used are the starting time and note duration, not the actual analog waveforms involved.

The playback speed can be changed in two ways: A) change all the starting times and durations by changing the actual data values stored with each event record, or B) vary the "tick rate" of the clock used for timing the events. For real-time changes, it is not desirable to stop the playback and re-calculate all the event timing data whenever a speed change is called for. Instead, an adjustable clock rate is used by **varying the number of interrupts per clock tick**. In **VERSATRACS**, it was determined that a useful range of rate changes would be from one half as fast to four times as fast as the original rate used. Choosing a base rate of four 8 millisecond interrupts per clock increment, the system usually runs with a 32 millisecond event time resolution. Thus, the number of

(>> SYSTEM TIMING AND SEQUENCING cont.)

interrupts per clock increment ranges from one (four times faster than normal) to eight (half as fast as normal). This has proved to be quite adequate. Fractional increases and decreases in increment rate are achieved using the same two byte technique used by the frequency control hardware and the vibrato rate software. In this case, the high-order byte of the clock rate index is the **integral part**, and the low-order byte the **fractional part**, of the number of interrupts to count before incrementing the clock. In this way, the space between clock increments is always an integral number of 8 msec interrupts, but that number varies from clock increment to clock increment to give the proper **average** increment rate.

Whenever a completed composition is to be saved permanently in a disk file for later retrieval and use, the actual data values in each recorded event are re-calculated, and the system rate index is returned to the normal setting of four interrupts per clock increment. This is done to simplify any editing or other data manipulations attempted if the composition was ever recovered from the disk. The re-calculation time is bearable prior to saving a composition, because of the relatively large delay required to write the file to disk.

>> RECORD AND PLAYBACK TECHNIQUES

Recording of note events is accomplished using two software modules. The first module assigns new notes to available oscillator pairs and handles the necessary bookkeeping information. This module must adjust the memory pointers to accomodate the next event data, and remember the address of the duration bytes so that the elapsed time can be filled in later when the note is released. The second module terminates any notes whose controlling key has been released, and completes the event cycle by writing the elapsed time into the duration bytes of the proper event record. The termination action also begins the amplitude

decay ramp of the oscillators assigned to the note.

Playback is accomplished by starting the system clock and repeatedly comparing the starting time of the next event in sequence to the current value in a clock variable. When the clock reaches a value equal to or greater than the current event starting time, the event data is decoded and the synthesizer hardware is directed to start playing the note. The duration value obtained from the event record is placed in a countdown array. The two-byte elements of the countdown array are decremented on every clock increment. When the software detects that an element has decremented all the way to zero, the corresponding note has been played for the proper duration, and it is terminated. Even as the duration countdown process is proceeding, the memory pointers have been updated to point at the next event starting time, and the comparison process continues without change.

For track-on-track recording, the software must simultaneously record the new key press events from the keyboard, **and** playback the previously recorded notes in accurate time sequence -- and all in real-time. This is the most demanding task for the **VERSATRACS** software. The modularity and functionality of the software will be described further in the following sections.

>> THE PARAMETER CHANGE EVENTS

One feature of particular importance must be mentioned here before proceeding to a description of the software sequence: **parameter change events.** The **VERSATRACS** parameters include the tempo, overall volume, instrument assigned to each track, envelope and decay rates, and so on. For maximum versatility, **VERSATRACS** can record the parameter changes along with the note events, and then playback everything in the proper time sequence. Alternatively, the parameter changes may be recorded in a separate step following the entry of all the notes. In any

(>> THE PARAMETER CHANGE EVENTS cont.)

case, the parameter changes are recorded in timed-event format similar to the note events described before. The parameter events are interspersed with the note events in sequential order, and the software detects the difference by decoding the sign bit of the third byte in the event: bit 7 = 1 means that the event data should be interpreted as a parameter change.

Within the parameter event classification, three sub-types exist: **track, global, and system markers**. As the name implies, **track** parameters control only the values for a specified track, like the assigned instrument or track volume. **Global** changes affect the parameters controlling global values, such as tempo and overall volume. The **system markers** are used to control the playback sequence of events and note editing features. The format of a parameter event is given below.

BYTE #

0 - Parameter event start time, least significant byte
1 - Parameter event start time, most significant byte
2 - 128+track# (track), or 192 (global), or 255 (system)
3 - Parameter-to-be-changed identifier
4 - Parameter data, if any
5 - Parameter data, if any
6 - (Spare)
7 - (Spare)

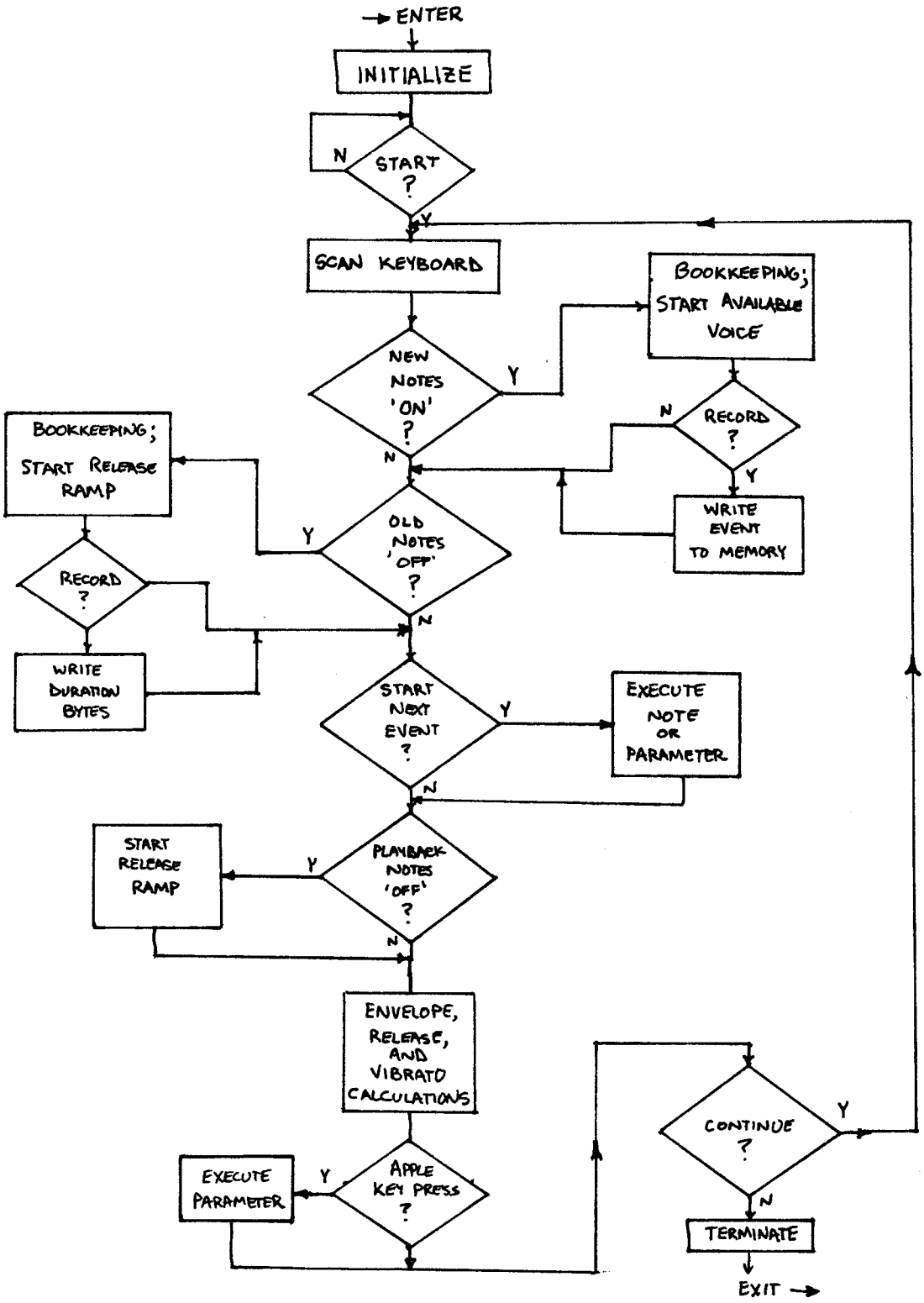
The two spare bytes are for future uses. A list of the parameter change functions and identifier codes can be found in Appendix C of this paper.

>> THE SOFTWARE SEQUENCE

Since one of the goals of **VERSATRACS** was to accomodate direct performance and recording, a great deal of effort has been directed toward increasing the ease with which a performer or researcher makes music with the system. Most features chosen for inclusion required the now-familiar trade-offs between execution speed, versatility, and memory requirements.

(>> THE SOFTWARE SEQUENCE cont.)

A simplified flow diagram for the track-on-track overdub function is given below.



>> AVAILABLE VOICE **HUNT**

With a finite number of voices available, some logical decision must be made about how to handle the situation when more than eight notes need to be played at the same time. This can occur when several tracks are played back at once, when more than eight keys on the keyboard are pressed at once, or even when an eight note chord is stopped, then started again before the amplitudes of the original notes have been ramped down to zero. How ever the situation occurs, several methods can be imagined to cope with the problem:

A) Once eight notes are playing, no new notes can be started until one of the eight voices becomes available again.

B) Replace one of the existing eight notes at random with the new note.

C) Replace the existing voice that has been "on" for the longest time with the new note.

D) Retain the existing voice with the lowest pitch, but replace at random from among the remaining seven voices.

E) Some other method, or some combination of methods.

The trick is to disguise the eight-voice limitation as well as possible within the execution speed and program size restrictions.

The most significant consideration for the track-on-track playback system is that new notes convey the most information in musical terms, and must have priority. This eliminates method (A) as a possible solution. Method (B) is probably more haphazard than necessary, since no consideration of relative note priority is made. Method (C) considers the history of each note, but the "oldest" note may be a misguided choice for replacement, particularly when notes are changing rapidly. Method (D) makes use of the observation that the ear pays more attention to the lowest note than to most of the others, because we have all been trained

(>> AVAILABLE VOICE **HUNT** cont.)

to hear the chord structure of conventional music this way.

VERSATRACS uses method (D), coupled with a priority check, to determine into which voice a new note should be placed.

The "voice hunt" algorithm looks through the eight available voices and picks the one with the lowest priority according to the following order of preferred replacement.

1 - A currently INACTIVE voice.

2 - A voice in the decay ramp to zero state (note terminated).

3 - A voice holding at the end of the envelope table (full scan done).

4 - A randomly selected voice other than the one of lowest pitch.

The HUNT software finds the voice with the lowest priority number, and starts the new note in that voice. The algorithm was not difficult to implement, and it works very effectively -- except when deliberate attempts are made to thwart its actions.

Once a voice has been selected, the various parameters associated with the note's track need to be assigned to the oscillator pair comprising the voice. The hardware wavetable page pointer register of each oscillator must be set to the proper page in memory, the correct freqency indices must be calculated using the offset value, and the initial oscillator amplitude must be provided. In addition to the hardware initialization, the track parameters -- such as vibrato depth, envelope and release rate, and track volume -- must be specified so that the control routines connect the correct parameter values with the right oscillators throughout the duration of the note.

>> LOAD SHEDDING

For slow to moderate musical tempos, the software sequence can proceed directly as indicated in the flow diagram given previously. However, when many tracks must be serviced, or when large chords must

(>> LOAD SHEDDING cont.)

be played back quickly, the software loop simply cannot keep up. This overload problem originally manifested itself as "rolled" chords, where a noticeable delay was heard between the start of each note in a chord, instead of a simultaneous attack. Another symptom was an occasional problem with delay between key presses on the keyboard and the corresponding sound out of the system. The solution to the overload problem is continuing, but the current version of **VERSATRACS** has overcome the major delay difficulties. An adaptive scheme is employed to shed some loads of lesser importance temporarily when many notes need to be handled in a short period of time. This method makes use of the observation that when many notes are changing rapidly, the need for proper **attack timing** and **pitch** is more important to the performer and listener than vibrato, exact release characteristics, and frequent checks for input from the Apple keyboard. The goal, of course, is to make any changes in the software sequence undetectable by system users. The load shedding technique can be stated in words as:

A) Whenever a recorded event from memory is executed, the sequence should adjust to check the next event in memory as rapidly as possible. This means that all envelope, release, vibrato, and Apple keyboard scan modules are skipped temporarily. This adaptation is used to process all the notes in a chord as quickly as possible. If no event is forthcoming immediately, a fast chord is unlikely and the software sequence can return to its normal state.

B) Whenever a live or recorded keyboard event occurs, the vibrato calculations are skipped on the following three main loop cycles to guard against a potential overload.

Note that the time required to execute a main loop cycle varies depending on how many events need to be executed, whether a vibrato calculation occurs, and so on.

>> PARAMETER EXECUTION METHOD

As mentioned, the parameter changes and other controls are activated by pressing keys on the Apple keyboard. During Live Keyboard mode, recording, and playback, approximetely 30 keys on the Apple are used to convey information. Most use the key with the first letter of the parameter to be changed, such as R for **RECORD**, D for **DECAY RATE**, and I for **INSTRUMENT**. Each parameter or control change is handled by its own subroutine in the software. A linear search case structure, where the key code received from the Apple keyboard is compared sequentially to test-values in an if-then-else form until a match is found, would require a considerable amount of time and software code. In the worst case, a key code might have to be compared to every other key test code, if the matching code was the last to be checked. In **VERSATRACS**, the linear search would not be efficient enough for real-time response. Instead, the parameter driver module uses a look-up table of jump addresses formed during the program's load and compile steps. The jump address is the starting address of the subroutine for the given parameter, and its position in the table is determined directly from the Apple ASCII value out of the keyboard using a simple mathematical calculation. After the appropriate jump address is fetched and the subroutine executed, control is returned to the parameter driver module and the main loop continues. This method is facilitated by the *FIRST* language environment, since each of the parameter handling subroutines can be defined as a "word", and addressed with ease.

>> **VERSATRACS** MODULE ORGANIZATION

VERSATRACS contains software modules other than the event play/record and Live Keyboard systems. Movement from place to place within the modules is controlled using a short sequence of menus. The modular organization is shown in a diagram below.

Each of the modules requires several Kbytes of memory space for its

(>> **VERSATRACS** MODULE ORGANIZATION cont.)

own program code, so it is not possible to have all of the modules resident in the directly addressable space of the Apple at any one time. Instead, a large portion of the extended memory card is allocated for use as a "virtual disk drive". The virtual disk space contains the program code for each module, and a software memory management routine is used to shuffle the required code segments from the extended memory to the Apple memory as the need arises. The virtual disk accesses are faster than reading a physical disk, and fewer floppy disk changes are required once the system has been loaded at the start of a session. A utilization description of the extended memory is given in Appendix B.

The following is a brief description of the main program modules. A description of their use can be found in the **VERSATRACS** manual attached to this paper.

The Auto-Sequencer module contains the Live Keyboard system, and the play/record systems described above. The name Auto-Sequencer is used because the software automatically executes the recorded sequence of events in real-time. The Auto-Sequencer requires the user to play a particular track with accurate pitch and rhythm, although editing features are available for subsequent modifications. The Auto-Sequencer module has the most comprehensive display of the parameters, so it is the module used for assembling new instrumental sounds from the library of waves and envelopes.

The Pitch-Rhythm module has not yet been described. It allows entry of notes without the necessity of playing both the pitch and the rhythm information at the same time. First, a non-real-time pass through a track is made. The user finds and presses in sequence the desired notes on the clavier keyboard without concern for the correct rhythm. Once the pitches are entered in sequence, a second pass is made so that the rhythm can be entered by tapping **any** key on the clavier

keyboard. With each key press, the software plays the next pitch in the recorded sequence and fills in the appropriate timing information in the event record in memory. After the rhythm entry, the track is in the standard format for playback or editing. Pitch-Rhythm allows a non-keyboard musician access to the music capabilities of **VERSATRACS** by simplifying the task of getting both the pitch and the rhythm correct at the same time. Tracks can be entered using the Pitch-Rhythm method alone, or they may be inter-mixed with tracks entered using the Auto-Sequencer system.

Chords up to four notes are handled by the Pitch-Rhythm system during the pitch entry step. The notes comprising the chord are played in sequence starting with the lowest note, then either the **2**, **3**, or **4** key on the Apple keyboard is pressed to indicate how many of the previously entered notes belong together. One of the "spare" bytes in the event record (see the EVENT CONCEPT section above) of the lowest note in the chord is used to provide the grouping information to the rhythm entry routine. When the rhythm is tapped, the software checks the spare byte to detect the presence of a chord, and fills in the timing information accordingly.

The Editing module is used to delete or double tracks, remove parameter change events, and eventually to support graphic display of notes for visual editing. In addition, an active editing function can be called directly from the Auto-Sequencer. The feature allows mistakes recorded to be fixed immediately after they occur. The function is invoked by pressing the **F** (for **Fix**) key on the Apple during a recording, which causes the software to reset the timing clock to the time corresponding to the start of the fourth preceeding measure. The system then plays back the previously recorded tracks and the notes just entered in the four measures prior to the **F** key press, and watches the keyboard. Before the keyboard mistake plays back, the user simply starts playing

along with the music. The software begins to replace the previous key press events with the new ones, and the recording process continues as if nothing had happened. This process can be referred to as **seamless real-time editing**.

The MIDI Setup module is used to assign particular tracks within the **VERSATRACS** system to the desired channels of the external MIDI network. The module also enables or disables the output of MIDI information from any of the tracks.

The Compositions module handles the transfer of event data files between the on-line memory and floppy disk. The results of any recording step using the Auto-Sequencer or the Pitch-Rhythm system may be saved permanently on disk using this module. The data saved with each composition includes all the note and parameter change events, a user-entered title, and the initial parameter settings, such as which instrument is assigned to which track. The Compositions module also supports a few data manipulation functions, including retrieval, deletion, and linking of compositions on the file disk.

The following modules are called the "Fundamental Elements", because they involve the building blocks of the synthesizer system.

The Wave Creator module is used to generate new wave tables or modify existing tables. The waveforms are specified by the relative weights of component harmonics, and the software handles the calculations to generate the required 256 byte table format. The wave tables can be saved on floppy disk for later use. Up to 48 tables can be present in the on-line memory at any given time.

The Envelope Creator module is similar to the Wave Creator, except that this module uses game paddles to "draw" the desired amplitude envelope on the CRT display. Like the waves, up to 48 envelope tables

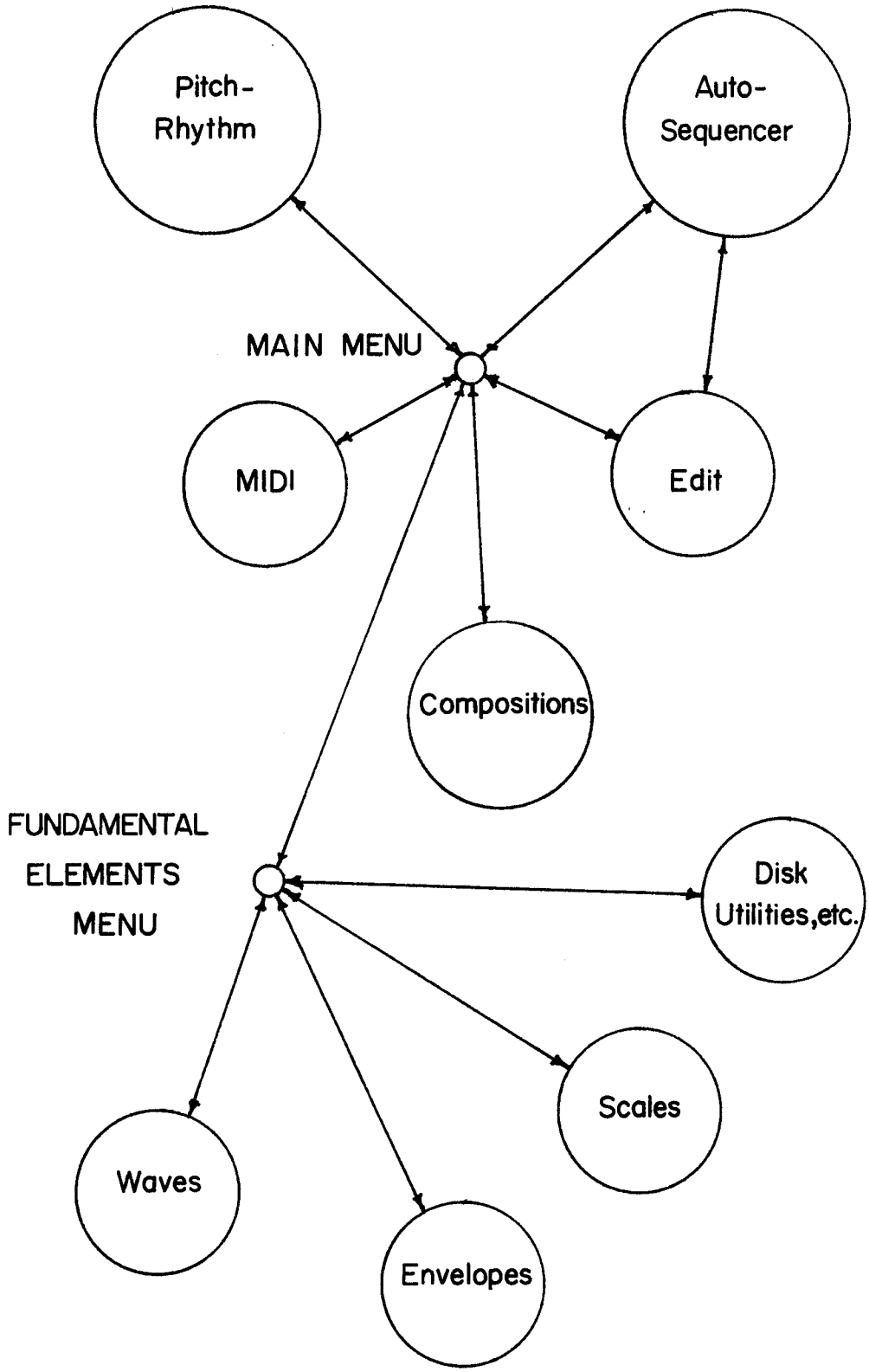
can be held in memory at once.

The Scale Creator module is unique among keyboard synthesizers. It allows the user to specify the particular tuning of each note on the keyboard. Since the **VERSATRACS** software has direct control over the frequency of each voice, the frequency index assigned to a key on the keyboard can be chosen at will, and stored in a scale array. The way in which the whole and half steps on the keyboard are tuned relative to each other can be varied to produce either a subtle or distinct effect. Most keyboard instruments are tuned with equal temperment, meaning the half steps are spaced by the same multiplicative factor, $2^{1/12} \approx 1.05946$. Other scales are based on particular frequency ratios or formulas. Adjustable scales can be used for slight tuning adjustments, or even to have the notes on the keyboard go from high to low as they move from left to right. **VERSATRACS** can have up to sixteen different scale patterns available on-line.

The Scale Creator can generate the scale pattern after the user enters one octave directly, specifies the desired formula or temperment, or enters the entire pattern note by note.

The Disk Utilities module is used for disk initialization and copying.

This concludes the SOFTWARE section of this paper. Please refer to the **VERSATRACS** manual and the Appendices of this paper for additional information.



Module Organization

>> EVALUATION

The **VERSATRACS** system has been under development for approximately one year. Prior to this development project, the author spent time with several musicians and serious amateurs, discussing important features, capabilities, and pitfalls present in existing computer-assisted music systems. **VERSATRACS** was designed to improve upon the current state of computer music using innovative software and relatively low-cost hardware. This section briefly considers the successes and failures of the system.

>> WHAT WORKS WELL

The main features of the program, such as recording and playback, work very well. The software effectively masks the complexities associated with moving data values, controlling the synthesizer hardware, and handling the bookkeeping needs. For example, a user can record a piece of music by simply pressing R , playing the clavier keyboard, and pressing RETURN to finish.

The method of record-keeping using "events" has been very useful. Although the data could be compressed more, the basic structure allows for straight-forward manipulation of all of the recorded information.

Entry of notes using the Pitch/Rhythm system has received favorable comments from many observers. The feature provides synthesizer capability to persons who are not sufficiently facile with a piano-style keyboard to play music directly. The ability to mix tracks played in real-time with tracks played using the Pitch/Rhythm method is also a helpful and versatile feature.

Provision for changing musical parameters before, during, and after a live-keyboard performance or recording has been a significant advantage for experimentation and orchestration. The opportunity for expanded musical expression is provided by the fact that all parameter changes are

(>> WHAT WORKS WELL cont.)

recorded as timed events, so dynamics, speed changes, etc., can be built into a composition.

The menu-driven method of moving from one function to another has proved to be quite satisfactory. The **VERSATRACS** system contains an extensive repertoire of software routines, and the menu format obviates the memorization of dozens of control codes or key sequences. The menu format also provides an elegant way for the user to keep track of which functions are available at a certain time.

>> WHAT NEEDS IMPROVEMENT

The major complaint about **VERSATRACS** has been the quality of sound produced by the hardware. The sound quality is limited by the construction of the Mountain Hardware synthesizer boards and the resolution of the 256 byte wavetables they employ. The frequency response characteristic is not adequate for truly high fidelity sound. One possible solution to the hardware problem is to use the digital oscillators for experimentation, preparation, and production of background music, and then use the MIDI output capability to drive a MIDI-equipped performance synthesizer. In this way, the recording and editing features of **VERSATRACS** could be used, while the outboard synthesizer could provide the higher sound quality desired. This configuration has the advantage that the performance synthesizer could be rented or borrowed only for a particular performance engagement or recording session. In this way, the particular capabilities required could be tailored to the application, without the expense of purchasing several synthesizers.

The cost of the **VERSATRACS** system hardware is another problem. The Apple computer system, memory expansion, synthesizer cards, etc., result in a price tag of several thousand dollars. That price pushes the system out of the "casual purchase" domain. However, in comparison to other computer music systems, the performance/cost ratio is much better

(>> WHAT NEEDS IMPROVEMENT cont.)

for **VERSATRACS**. It is unlikely that the cost of the current hardware requirements will drop dramatically, so a move to a lower-cost configuration retaining or extending the software features would be a necessity for widespread interest.

The limitations of the Apple][computer itself are frequently mentioned by observers and encountered in programming. The 1 MHz clock rate -- cut to 500 kHz due to the synthesizer's DMA -- and eight-bit architecture of the Apple are difficult barriers with which to cope. A user of **VERSATRACS** is free to use the computer for other tasks besides music, so many potential buyers might question whether an Apple][system would be the wisest investment in today's discount market.

The limit on directly addressable memory in the Apple forces the movement of program code between the extended memory card and the active memory when a new function is to be made available. Although the process is quite brief, the system would benefit if more of the software features could be active at a given moment. For example, the Auto-Sequencer module must be terminated if the Compositions module is to be invoked. This means a delay of a few seconds as the Compositions module is loaded, then a few more seconds delay as the Auto-Sequencer is re-initialized following the composition function.

A feature frequently mentioned for inclusion in **VERSATRACS** is a means of printing standard music notation for hardcopy output. Automatic transcription of a recording from the clavier keyboard is in demand because of the tedium of hand-printing a musical score. Printing music can be a more complex undertaking than a word processing program, particularly if an extensive font of musical symbols is included. This is due to the fact that music notation is more context-dependent than printed text. A simple music printing program has been started, but more time for development will be required before it is at the level of the rest of the system.

>> CONCLUSION

This paper has described the development and features of a digital synthesizer system called **VERSATRACS**. The hardware requirements, programming environment, software design, and a critical evaluation are included in the body of the paper. Additional information is included in several appendices on the pages following.

The use of digital equipment in music is another example of the way in which computers are used. The computer-assisted synthesizer described in this paper is not an attempt to imitate human creativity, but a tool to facilitate expression and extend the range of musical possibilities. This project has been quite challenging. The requirements of real-time programming have been overcome to a great extent, and the system is capable of producing some excellent music. Most of the goals established prior to the project have been accomplished, and the experience gained in this development will be valuable as new features are designed and new hardware becomes available. The major lesson learned has been that the "ultimate" computer music tool is yet to be designed.

REFERENCES

Benade, A. H., Fundamentals of Musical Acoustics, Oxford University Press, 1976

Gayler, W. D., The Apple II Circuit Description, Howard W. Sams & Co., 1983

Maher, R. C., et al., "A Low-Cost Digital Synthesizer System for Music Applications and Psychoacoustic Research" (abstract), Journal of the Acoustical Society of America 77 (1985): S75 (supplement)

"MusicSystem Operating Manual", Mountain Computer, Inc., 1981

"SY6500/MCS6500 Microcomputer Programming Manual", Synertek, 1976

"model JUNO-106 Operating Manual", MIDI implementation, Roland Corp. US, 1985

APPENDIX A

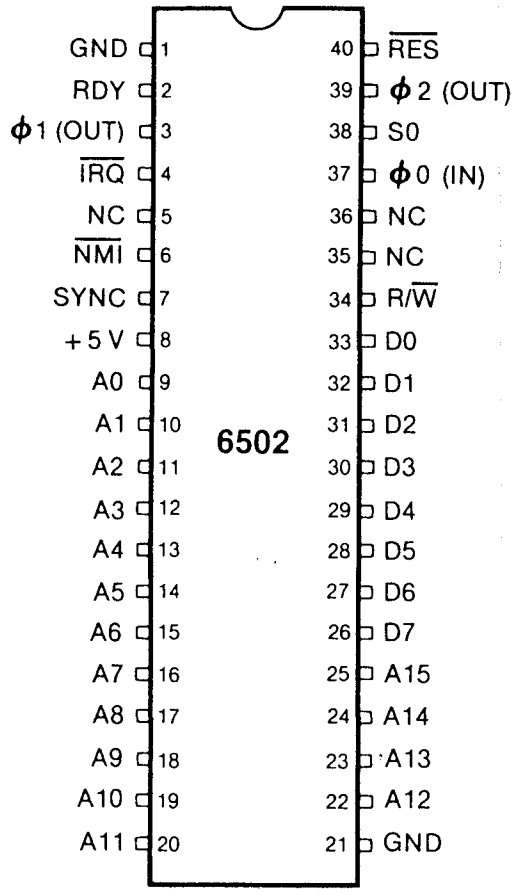


Fig. 6-1. 6502 Microprocessor.

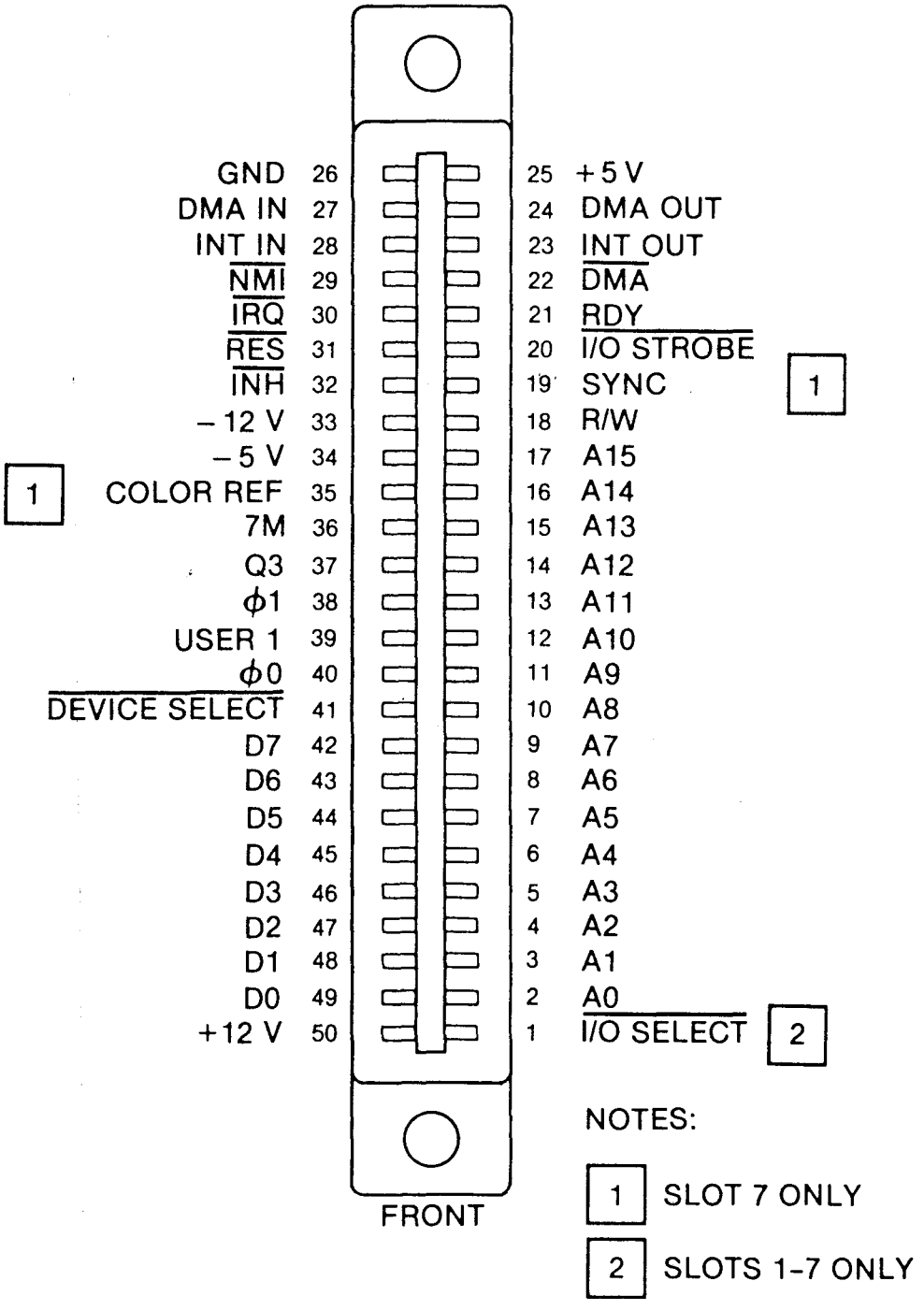


Fig. 6-17. Peripheral connector pinout. (Courtesy Apple Computer, Inc.)

Table 6-1 The Apple II Bus

Pin Number	Signal Name [1]	Mode [2]	Description
1	$\overline{\text{I/O SEL}}$	I	INPUT/OUTPUT SELECT. Seven individually decoded lines for slots 1–7. $\overline{\text{I/O SEL}}$ at any one connector goes low during $\phi 2$ when $CN00–$CNFF is accessed; N = slot number.
2–17	AD0–AD15	B	16-bit address bus. Address becomes valid during $\phi 1$ and remains valid throughout $\phi 2$.
18	R/W	B	READ/WRITE. Becomes valid during $\phi 1$ and remains valid throughout $\phi 2$. High for read; low for write.
19	SYNC	I	Video synchronization signal (C13-8). Connects to slot 7 only.
20	$\overline{\text{I/O STB}}$	I	INPUT/OUTPUT STROBE. Common select line that goes low during $\phi 2$ when $C800–$CFFF is accessed.
21	RDY	B	READY. Taken low during $\phi 1$ to insert wait states. The 6502 recognizes low on RDY only during read cycles.
22	$\overline{\text{DMA}}$	O	DIRECT MEMORY ACCESS. Taken low at the beginning of $\phi 1$ to halt the processor and make the address, data, and R/W lines high impedance.
23	INT OUT	O	INTERRUPT OUT. Daisy chain to lower priority slots. Normally driven high if used; connects to pin 28 if not used.
24	DMA OUT	O	DIRECT MEMORY ACCESS OUT. Daisy chain to lower priority slots. Normally driven high if used; connects to pin 27 if not used.
25	+5 V		+ 5 volts
26	GND		Ground
27	DMA IN	I	DIRECT MEMORY ACCESS IN. Daisy chain input from higher priority slots. Normally high.
28	INT IN	I	INTERRUPT IN. Daisy chain input from higher priority slots. Normally high.
29	$\overline{\text{NMI}}$	O	NON-MASKABLE INTERRUPT. Taken low to start a 6502 non-maskable interrupt.
30	$\overline{\text{IRQ}}$	O	INTERRUPT REQUEST. Taken low to start a 6502 maskable interrupt. Recognized only if interrupt disable flag is not set.
31	$\overline{\text{RESET}}$	B	As an output, taken low to reset the 6502 and other peripherals. As an input, goes low during resets from the keyboard, on power-up, and from other peripherals.

Table 6-1—cont. The Apple II Bus

Pin Number	Signal Name [1]	Mode [2]	Description
32	$\overline{\text{INH}}$	O	INHIBIT. Taken low to inhibit the ROM address space ($D000–$FFFF).
33	−12 V		− 12 volts
34	−5 V		−5 volts
35	COLOR REF	I	COLOR REFERENCE. Connects to slot 7 only.
36	7M	I	7MHz. 7.15909 MHz clock.
37	Q3	I	2.040968 MHz (average) clock.
38	ϕ1	I	PHASE 1. 1.020484 MHz (average) system clock. Used on bus in place of 6502 ϕ1.
39	USER 1	O	Taken low to inhibit the I/O address space ($C000–$C7FF).
40	ϕ0	I	PHASE 0. 1.020484 MHz (average) system clock and compliment of ϕ1. Used on bus in place of 6502 ϕ2.
41	$\overline{\text{DEV SEL}}$	I	DEVICE SELECT. Eight individually decoded select lines, one for each slot. $\overline{\text{DEV SEL}}$ at any one connector goes low during ϕ2 when $C0X0–$C0XF is accessed; X = N + 8, N = slot number.
42–49	D7–D0	B	Eight bit bidirectional data bus. Data becomes valid during ϕ2 and remains valid to the end of ϕ2.
50	+12 V		+ 12 volts

Notes: [1] Unless otherwise specified, all signals appear on all eight connectors.

[2] I: Input
O: Output
B: Bidirectional
With respect to peripheral card.

APPENDIX B

EXTENDED MEMORY UTILIZATION - VERSATRACS - MARKET INTERFACE, INC. 1 OF 2

256 K LEGEND INDUSTRIES, LTD. R. MAHER 6/28/85
'IS' CARD

1-MAIN BANKS (16 @ 12K BYTES)

0-3	PRIMARY EVENT RECORDS - 4 BANKS @ 1536 EVENTS = 6144 EVENTS
4-7	SECONDARY EVENT RECORDS "
8	MEMORY BLOCK SAVE AREA - DYNAMIC ALLOCATION
9	VIRTUAL DISK AREA*
10	$D000 - $EFFF - HRG BUFFER IMAGE (8K) $F800 - $FBFF GETICS EXEC $E000 - $E3FF - "SETT" EXEC, RCVI $FC00 - $FFFF AUXILLIARY LOADS $E400 - $E7FF - OKCHK WORDS
11-13	VIRTUAL DISK AREA*
14	ENVELOPES
15	WAVES

* SEE DESCRIPTION BELOW

SUB-BANKS (16 @ 4K BYTES)

0	$D000 - $D3FF - ACTIVE SCALES, KSEQ, P/ DISPLAY, P/ NOTE LOOKUP (@ 256 BYTES) $D400 - $D7FF - SCALE NAMES $D800 - $DAFF - INITIAL CONDITIONS DATA (INCL. RESERVE) $DB00, $DB51 - HELP HEADERS $DC00 - $DDFF - SECONDARY PARAMETER EXEC $DE00 - $DFFF - PRIMARY PARAMETER EXEC
1	$D000 - $D3FF - INSTRUMENT TITLES $D400 - $DBFF - INSTRUMENT DATA $DC00 - $DFFF - MAIN DISPLAY (ARCHIVAL)
2	$D000 - $D3FF - MENU DISPLAY $D400 - $D7FF - MAIN DISPLAY (SAVE) $D800 - $DBFF - HGCOD (RELOAD $0C00 - $0FFF) $DC00 - $DFFF - ASCOD (PROGRAM RELOAD $0C00 - $0FFF)
3	$D000 - $D3FF - INSTRUMENT ASSEMBLY DISPLAY $D400 - $DBFF - MAGNITUDE (AND PHASE) DATA (32 X 48) $DC00 - $DFFF - FUNDAMENTALS MENU DISPLAY
4	$D000 - $D3FF - WAVE, ENVELOPE, SCALE MAKER TEMP MENUS $D400 - $D7FF - FUNDAMENTALS MENU CODE $D800 - $DBFF - P/ HELP SCREEN $DC00 - $DFFF - WRTEX EXEC
5	$D000 - $D3FF - TITLES TEMP, SCORE TEMP, INS. TEMP, COMP. TEMP $D400 - $DBFF - INS. DATA TEMP. AREA $DC00 - $DFFF - COMPOSITION MENU DISPLAY
6	$D000 - $DFFF - TAV
7	$D000 - $D3FF - LOCAL DEFINITION $D400 - $DBFF - RWTS (DISK I/O CODE; $B800 - $BFFF) $DC00 - $DFFF - (-)

cont.

SUB-BANKS (CONTINUED)

8	$D000 - $D3FF - HRG CODE $D400 - $D7FF - SINE, TRI, SAW, SQUARE $D800 - $DBFF - ENVELOPE TITLES $DC00 - $DFFF - WAVE TITLES
9	$D000 - $DFFF - WAVE SAVE AREA ($B000 - $BFFF)
10-11	P/ STAFF AND CLEF DISPLAY
12	SCALES (16 @ 256 BYTES)
13-14-15	HELP SCREENS (VIRTUAL DISK)

VIRTUAL DISK AREAS

MAIN:	9	11	12	13
(1) $D000 -	99 DSXP, SETD, DPARM	125 } COMP LINKER	108 } P/ PITCH	131 - COMP MENU
(2) $D400 -	100 LOCATE	126	109	119 } SAVE COMP
(3) $D800 -	22(I) EDIT DISPLAY	127	110	120
(4) $DC00 -	135 - EDIT MENU CODE	117 EDIT, DELETE PARMS	111	121
(5) $E000 -	101 } EDIT	132 - SAVE INST.	112	122
(6) $E400 -	102	118 EXTRCT PITCHES	116 - SAVWF	39 - UPLOW, UPLOE
(7) $E800 -	103	105 P/ CHORD	113 } P/ RHYTHM	123 } GET COMP
(8) $EC00 -	101(I) MIDI DISPLAY	98 XTKX, FIXX	114	124
(9) $F000 -	100(I) MIDI SETUP CODE	128 } DELETE COMP	115	106 COUNT, FLCHD
(10) $F400 -	GILVL EXEC	129	9A SOUND ASSEMBLE	107 RSORT, P/SORT
(11) $F800 -		130	95 RLPS, SOUND ASSEMBLE	40 - INSTRL
(12) $FC00 -			96 RECX, SOUND ASSEMBLE	133 - SAVE INST.

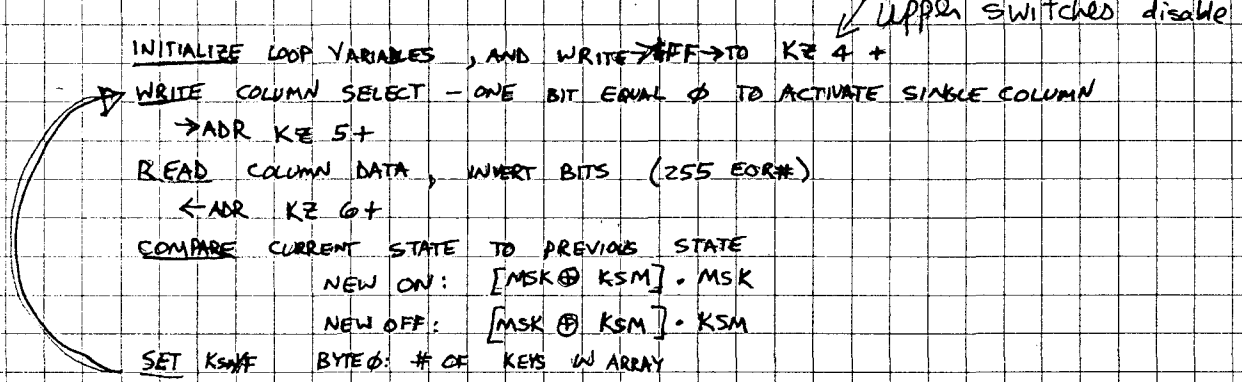
SUB:	13	14	15
(1) $D000 -	121(I) SYSTEM HELP	120(I) AUTO-SEQUENCER HELP	128(I) GLOBAL VERBOSE, TRACK
(2) $D400 -	122(I) GLOBAL HELP	125(I) GENERAL II	129(I) } TRACK VERBOSE
(3) $D800 -	123(I) TRACK HELP	126(I) GENERAL III	130(I)
(4) $DC00 -	124(I) GENERAL I	127(I) GLOBAL VERBOSE	131(I)

APPENDIX C

KBD SCAN TYPE 3

VARIABLES: COLM = COLUMN
ARRAY KSM = KEYBOARD SCAN MIRROR
" KSN = " " ON } NEW CHANGES (SINCE LAST SCAN)
" KSF = " " OFF }
RKS = 0-6D COUNT UP KEY#
XZ = 0-7 LOOP COUNTER
MSK = KEY PORT DATA
LL =
0, 1+ OFM = RESULT OF KSM COMPARISONS

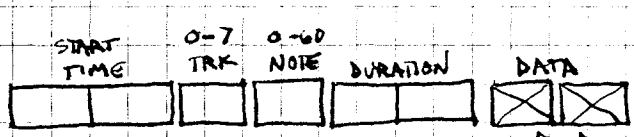
FUNCTION: READ KEYBOARD CURRENT CONDITION, COMPARE TO PREVIOUS CONDITION, AND SET ON OR OFF NOTES ACCORDINGLY



FORMATS

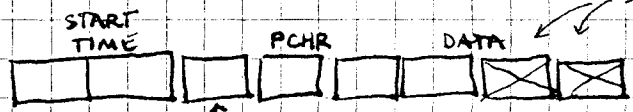
EVENTS

NOTE



SPARE

PARM



128 + TRK	10 —— XXX	TRK PARM
192	110 —— 0	GLOBAL
255	11 —— 1	MARKER

MARKER TYPES:

ϕ = LOOP ENDING

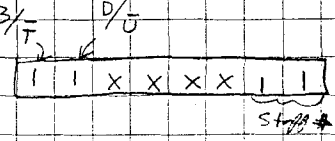
COMPOSITION START

INITIAL CONDITIONS 1/25/85 0 MB CHB $0800 +

bytes		Pos.				
2		0	1	SCALE	A-P	SCALE
2		2	3	ATRNS		TRANSPOSE POSITION
2		4	5	AVBR		VIBRATO RATE
2		6	7	BPMEAS		BEATS PER MEASURE
2		8	9	BBASE		BEAT BASE (NOTE GETTING ONE BEAT)
2	$0A	10	11	BPM		BEATS PER MINUTE
2	$0C	12	13	BINC	BINC H = 7020 / BPM	BINC = 7020 MOD BPM */ 256 BPM
8	$0E	14	21	TINS		INSTRUMENT ASSIGNED TO EACH TRACK
8	$16	22	29	TLVL		INSTRUMENT LEVEL ASSIGNED TO EACH TRACK
2	$1E	30	31	GVOL		OVERALL VOLUME
16	$20	32	47	SCORE*		(FOR SCORE PRINTING)
8		48	55	TOCT'	(TOCT = 2 * TOCT')	OCTAVE #
8		56	63	TENR		ENVELOPE RATES } TRACK
8		64	71	TDCR		DECAY RATES } TRACK
8		72	79	TVBD		VIBRATO DEPTHS } TRACK
8		80	87	TAMP'	(TAMP = 2.5 * TAMP')	AMPLITUDE } TRACK

*SCORE SEGMENT (PRELIMINARY)

32	$20	TRK 0-7	CLEF, STAFF, AND STEM ASSIGNMENT
39	$27		
40	$28 =	0-15	Key signature C=0 #'s, flat CGDAEBF#C# F B♭E♭A♭D♭G♭C♭F♭ (0)
41	$29 =		Resolution (4)
42	$2A =		SCORE FILE INDICATOR =1 if multi-track score, =0 if all on a play file



CODES - STATUS ARRAYS

TSTS:	VALUE	
	0 =	BLANK
	1 =	REC
	2 =	PLAY
	3 =	RHY
	4 =	KBD

TSTS = TRACK STATUS ARRAY

PARAMETERS

255 = MARKER

192 = GLOBAL

128 + TRK# = TRACK PARAMETER

OSCG = OSCILLATOR GROUP STATUS ARRAY FOR HUNT

192 + KEY#	ON NOTE - TRACKING ENVELOPE (KEY# 0-61)
128 + KEY#	ON NOTE - SUSTAIN AT END OF ENVELOPE
64	ON TRACKING RELEASE
63	OFF AND AVAILABLE
127	ON - START RELEASE TRIGGER (MIMETICS CARD)

MIDG = MIDI OUTPUT VOICE STATUS ARRAY

KEY#	MIDI ON
63	AVAILABLE

INSTRUMENT DATA SCN's 103 104

0	4	8	12	16	20	24	32	40	48
4	4	4	4	4	4	8	8	8	3
ENR	DCR	VBD	AMP	OCT	SPARE	OFFSET	WAVES	ENVS	SPARE

= 51 bytes

COMPOSITIONS DISK

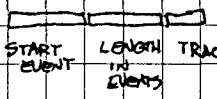
SCN ∅ TITLES: 17 CHARACTERS

DBUF 122+ I 40*+ DBUF 142+ I 40*+

USED: DBUF 2340*+ 14+

REMN: DBUF 2340*+ 33+

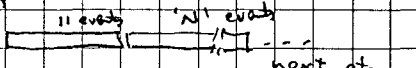
SCN 1 LOCATION: 5 bytes → DBUF 1022+ = TRK# FOR WORKFILE



FIRST 11 EVENTS ARE INITIAL CONDITIONS DATA (88 bytes)

see page 17 for IC data structure

SCN 2-91 DATA



11320 EVENTS (90K - 200 bytes)

SCN 92-139 48K WORKFILE (up to)

APPENDIX D

To start up the FIRST system, insert your FIRST diskette in the floppy disk drive.

Powering on the computer causes a bootstrap-read from the disk. The indicator light on the disk drive will come on, and you will hear a few clicks and scraping sounds, indicating that the read-operation is in progress. FIRST will type a copyright message, then display its prompt-symbol #. Now its ready and waiting for your typed input commands.

To transfer from FIRST to the APPLE MONITOR, enter MON, followed by a return. The prompt symbol changes from # to *. To re-enter FIRST, hit RESET key. To reboot from the monitor, type 6, then CTRL P followed by RETURN (the control-key is held down while P is typed). To reboot from FIRST enter COLD. To do a warm restart, which resets FIRST back to the startup condition but does not reload from the disk, enter EMPTY.

To list the directory lines for all the screens on your disk:

```
#20 139 DIR
```

(each line of the typed input words is acted upon when a RETURN key is pressed).

You can alter the startup actions of FIRST by editing Screen 20, which is automatically loaded (compiled and/or executed) after the initial FIRST vocabulary is fetched from the disk. You may compile new words and also directly execute words by editing them onto screen 20. You might, for instance, alter the startup message, or you might execute 20 30 DIR, which would list part of the disk directory.

When FIRST starts up, try these words:

DW	Lists all temporary words.
DPW	Lists all permanent words.
D	Lists all words, showing their memory locations.

NOTE: The previous list-words pause after filling the screen. To continue the listing, hit any key (except ESC, which aborts the listing)

DST — Displays the contents of the loop stack and the aritmetic stack.

MON — Breaks out of FIRST, back to the Apple monitor system.

COLD — Reboots FIRST from the disk (works with auto-start ROM).

EMPTY — Erases words created since boot-load, then loads screen 20, the startup screen.

20 LIST — Reads screen 20 and shows it on the display screen.

20 L — 20 L lists screen starting at 20.
-> pages forward
<- pages back.
^E enters editor, RETURN terminates L.
ESC shows the help display

2 3 + I. — Adds 2 and 3, then displays the sum. In detail, this means: Push 2, Push 3, ADD, POP and display the result (16-bit integer arithmetic).

2. 3. F/ F. Floating point arithmetic execise.

CLS — Clears display screen.

Now try a small exercise for creating a new word POSTMAN. Type in as follows:

```
#: POSTMAN BELL BELL ;
```

#DW — You will find the new word you created at the top of the vocabulary.

#POSTMAN — Executes the new word. Whenever POSTMAN is executed, the bell will always ring twice.

To power the system down, open the disk drive, remove your FIRST diskette, then turn off power to the computer and the display.

RESET key Hitting RESET stops whatever the computer is doing and restars FIRST. For instance, to return from Apple monitor mode just hit RESET.

If you accidently write a perpetual program (an easy mistake to make), you can break out of a loop by hitting RESET, which resets both arithmetic and loop stacks and signals readiness to resume by printing the PROMPT-sign.

COMPILING NEW WORDS

: : NNNN OOO PPP QQQ ; means, create the new word NNNN as a command equivalent to the sequence of exsting words OOO PPP QQQ.

: (colon), followed by the new word, is the start of a compilation.

; End of compilation of a new word.

Warning! It is easily possible to redefine the meaning of a word which is in common use. It would be dreadful, for instance, to redefine:

```
#: + BELL BELL ;
```

We have now lost the ability to add!

SPELLING RESTRICTIONS

New words can have 1 - 255 letters or symbols, with no interspersed spaces (the space is what seperates one word from the next).

Word spelling should avoid conflict with numerical constants. Here is a disastrous example:

```
: 14 BELL BELL ;
```

This really does have the meaning that 14, instead of pushing 14 on the A-stack, now causes bell to ring twice. No warning is given

when one redefines a number as a word!. Another pointless and misleading definition:

```
: 14 15 ;
```

Now both 15 and 14 will push 15 on A-stack!

Each new word created by : has this form in memory:

```
                    :
                    :
                    :
          Previously existing words
                    :
-------------------------------------------
                (PREVADR)  ADDR of prev. name
NAME =>         TEST       Spelling of new name
                 0         Null-termination
CODE =>         JSR XXX
                JSR XXX    For immed-mode words
                     :     the first OP in the
                     :     code-section is NOP
                     :
                   RTS
-------------------------------------------
Empty (available) Memory
                    :
                    :
```

ALLOCATION OF MEMORY IN FIRST

```
$0000 - $00FF   Constants, pointers
$0100 - $01FF   6502 JSR-stack
$0200 - $02FF   Keyboard buffer
$0400 - $07FF   Text-screen buffer
$0B00 - $0BFF   Disk-screen buffer
$0C00 - $0FFF   Hires graphics code
$1000 - $13FF   L-Stack
$1400 - $17FF   A-Stack
$1800 - $1FFF   Hash-code table
$2000 - $3FFF   Hires-Graphics buffer
$3C00 - $3FFF   Line-save buffer
$4000 - $XXXX   FIRST (restart addr = $4003)
:
:
:
$B800 - $BFFF   RWTS disk I/O code
$C000 - $CFFF   I/O device addresses
$D000 - $FFFF   ROM addresses
```

FREE — Displays amount (in bytes) of remaining available memory.

TOP — Shows amount of memory used. Top of used memory equals start of availabe memory.

SIZE — Displays the current size of vocabulary (total program size).

While writing procedures which have sizeable data-arrays, it is prudent to type FREE occasionally to avoid running out of memory. If you do exceed memory limits, fatal error messages will result, indicating either excessive array size or too many new words.

SHOW XXX — Disassembles the word XXX

FIND XXX — Finds XXX, if it exists and types its address. If XXX does´nt exist, an error message results.

LOAD — NN LOAD compiles and executes the words held on the disk screen NN. The effect is the same if a robot were entering the same text as is written on screen NN. (Text is written on a screen with the aid of the text editor).

LLD — MM NN LLD compiles screens MM through NN. Example:

```
: PROGV 25 29 LLD ;
```

creates a word which creates a vocabulary of words from definitions on screens 25-29.