source: Microcomputing, januari / februari 1981

by Hal Chamberlin

Hal Chamberlin is vice president of Research and Development for Micro Technology Unlimited, Box 12106, Raleigh, NC 27605.
Active in electronic sound synthesis since 1966 and in computer music synthesis since 1970, he has authored numerous magazine articles and has recently published a book entitled Musical Applications of Microprocessors.

Computer music synthesis in two parts.

Simulation Of Musical Instruments

By Hal Chamberlin

- The use of microcomputers for teaching, composing, transcribing and playing music is rapidly becoming a major application area. New synthesizer boards, music programs, and integrated systems with music capability are among the new products highlighted in the microcomputer field. It is now common for university and even high school music departments to acquire a quantity of microcomputers solely for musical purposes. It is even getting to the point where it is hard to find a microcomputer owner without some kind of music program, even if it only plays kazoo-like music through a built-in two-inch speaker.

The Computer Music Field

- Since any complete discussion of microcomputer music is impossible within the confines of a magazine format, this article deals with a much narrower subject area. First, however, we need to characterize the field somewhat to see how the topics of musical instrument simulation and software digital synthesis fit in. Computer music systems cover a broad range of sophistication, application, capability, sound quality and cost. At one extreme, we have the limited-range, tinny, one-voice, "gee whiz" type of system mentioned earlier that can be set up for a dollar's worth of parts and a program whose listing would not even fill a page. At the other extreme, we have experimental computer music systems in some universities that have a range beyond human perception, quadraphonic sound quality exceeding that of the best recording equipment, virtually unlimited synthesis capabilities and practically infinite voice count at a cost (if measured by industrial standards) in the millions. Using a microcomputer, you can set up a system with reasonably wide range, good stereo sound quality, good synthesis flexibility and 32 voices for a few thousand dollars. The important point is that there is a definite need for systems addressing these extremes and many points in between.
  The simple one-voice systems are certainly the most common and are fully adequate for teaching elementary music concepts as well as for impressing friends and neighbors. In fact, they are probably preferred for getting started because their very simplicity makes them easy to learn and use. Since only pitch and timing can be controlled, there are only two variables to worry about. Harmony, timbre, envelope and dynamics are either absent or predetermined.
  Note that this type of music system is easily implemented either purely in software using timed loops, which toggle an output port bit, or through a combination of software and hard- ware where control bytes are sent to a simple divide-by-N counter which may even be part of the I/O interface chip used by the computer.

With this level of system, you either quickly outgrow it and move on or are content to file the program alongside the Lunar Lander and Star Trek cassettes.
The next step up is generally either a synthesizer board or an inexpensive eight-bit digital-to-analog converter. The synthesizer board is a set of several oscillators which at a minimum are programmable for pitch and amplitude. (There are a couple of very sophisticated single-voice synthesizer boards, but they are intended to be used in multiples.!
The simplest type of synthesizer board has three square-wave oscillators with pitch and amplitude registers for each and sometimes an overall volume control. Typically, these boards are implemented using programmable timer integrated circuits as the oscillators (normally intended for use in process-control-oriented microcomputers) and discrete circuitry for the volume control function.

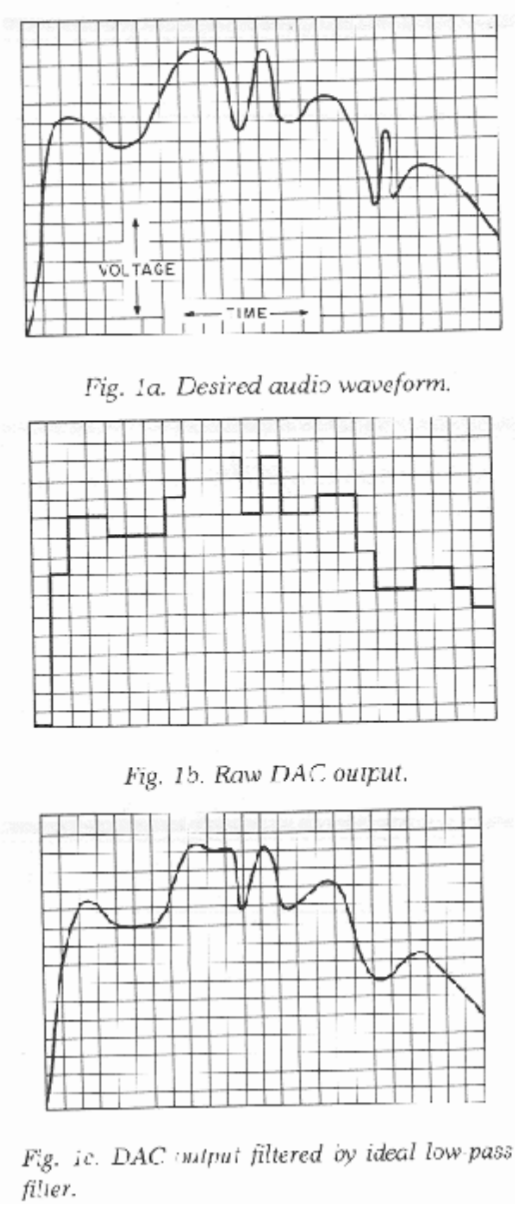


Fig. 1a. Desired audio waveform.

Fig. 1b. Raw DAC output.

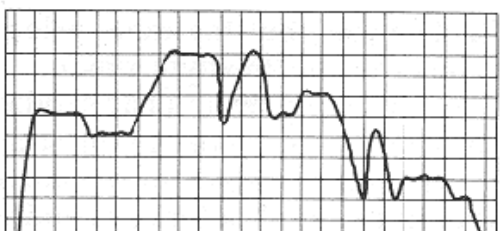
Fig. 1c. DAC output filtered by ideal low pass filter.

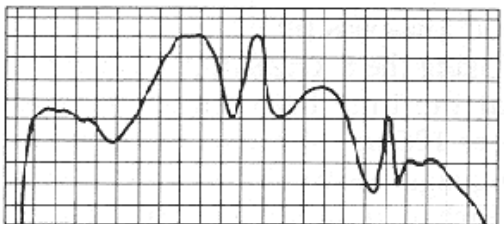
Recently, General Instrument introduced a synthesizer chip that has the three oscillators and the volume control circuitry integrated on a single chip along with a noise generator useful for limited percussion effects. These chips, usually in trios for a total of nine voices, are appearing in the latest batch of synthesizer boards. Prices range from a little over $100 to nearly $300 with little connection between capability and price.
With these synthesizer boards, the computerist musician gains a great deal of flexibility, since he can play complex chords and control dynamics and tone envelopes. As a result, these boards are a great deal more difficult to master, although you can choose to ignore some of the variables initially.
There is one serious shortcoming, however: all synthesizer boards in this class produce square waves exclusively. (One three-voice board on the market has the capability of combining two of the voices into a single variable width rectangular wave, which increases the tonal variety somewhat.) Square waves have a

rather sharp, yet hollow, sound that most closely resembles that of a kazoo. By suitable control of the amplitude envelope, you can produce continuous organ-like tones and percussive plucked-like tones, but the basic character of square waves remains.
With the proliferation of this type of board in recent months (and its constant demonstration at computer shows!, the public may very well come to associate square wave sound with computers, just as a piano is associated with its own tone color. This would be unfortunate indeed, since the ultimate value added by computers in music is a wider range of timbres than any other instrument. Nevertheless, there is sufficient expressive power available so that the difference between a piece programmed by a novice and one programmed by an experienced musician is readily apparent.
Music systems based on these synthesizer boards seem to satisfy many users whose goal is to learn music, enjoy transcribing music into the computer and even perform simple composition. They typically will not satisfy a musician attempting to do serious performance work with the computer.
Much more sophisticated synthesizer boards with programmable waveforms are also available in the $500 to $2000 price range. These overcome the lack of tonal variety of the square wave boards by providing programmable waveforms, usually with provisions for a different waveform for each voice.
An important consideration that will be discussed later is whether the board allows dynamically variable waveforms; that is, the ability to smoothly alter the waveform while a note is being played with it. This is a requirement for many effects such as the "wah" of a muted trombone, and, as might be expected, the less expensive units do not provide for it. In either case, the tonal variety is far greater than the square wave units and is sufficient to satisfy many musicians as well as casual users.
On the other side of the hardware/software fence are music systems based on digital-to-analog converters (DAC). As we shall see later, a digital-to-analog converter simply translates numbers into voltages. A very rapid string of numbers produces a rapidly varying voltage; that is, an audio waveform.
In theory, appropriate software can calculate the necessary number sequence to produce literally any sound. The capabilities of a music system based on a digital-to-analog converter are determined solely by the sophistication of the software involved rather than the capabilities designed and frozen into a hardware synthesizer. DAC boards also tend to be less expensive. A good eight-bit DAC board sells for less than $70, while an experimental home brew unit can be put together for half the price of a movie ticket.
The remainder of Part 1 will describe how sound generation software works in a DAC-based system.

Numbers to Sound

- The fundamental principle behind digital sound synthesis is that a string of numbers from a computer program may be converted into a high-fidelity audio signal. As you might expect, the rate at which the numbers are supplied and the precision of the numbers both determine the fidelity of the resulting sound. In synthesis applications, fidelity's usual definition, i.e., faithfulness to the original, does not apply I since there is no original. Instead, fidelity is used to refer to the frequency range that can be produced and the relative freedom from undesired noise and distortion.
  Fig. 1 shows how a DAC can produce a smooth audio waveform from a string of numbers and the errors in': volved. The grid in the figures represents time in the horizontal direction and voltage in the vertical direction.
  Fig. la shows a greatly magnified drawing of a small portion of a typical audio waveform. Notice that it wiggles and curves through the figure without regard for the grid.
  In Fig. 1 b we have the raw output of a DAC being fed the string of numbers representing the waveform in Fig. la.
  Each vertical grid line represents the point in time that the DAC receives a new number; thus, it stands to reason that the DAC output can only change up or down at vertical grid lines. Each horizontal grid line represents a possible numerical value that the DAC can receive. For example, an eight-bit DAC can only accept 256 (28) different numbers, so the complete grid for such a DAC would have 256 horizontal grid lines. As a result, the DAC output can only dwell at a horizontal grid line. Needless to say, the smoothly curved waveform of





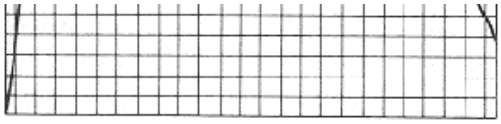


Fig. 1d. Filtered DAC output at twice the sample rate.

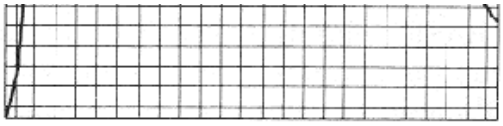
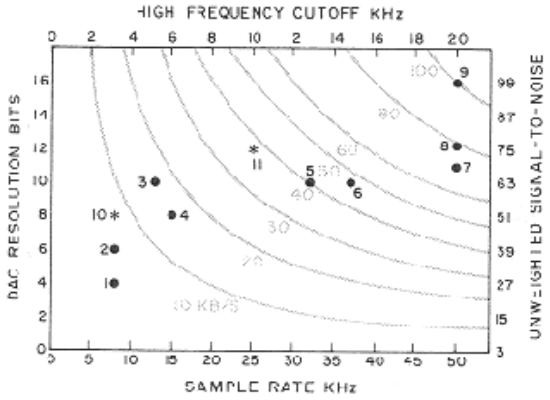


Fig. 1e. Filtered DAC output at twice the resolution and sample rate.

Fig. 1a is severely distorted in shape by this two-dimension quantization imposed by the DAC.

In Fig. 1c we have passed the raw DAC output through an ideal low-pass filter. Usually such a filter is described as allowing all frequencies below a certain cutoff frequency to pass through while blocking those above it. Here, however, we are using it to smooth the DAC output into something that more closely resembles Fig. 1a.

As you can see, the filter does a nice job of smoothing the curve between the vertical grid lines (that is, between new numbers from the computer). Closer examination, however, reveals that when the curve crosses a vertical grid line, it is also precisely on a horizontal grid line as well. Thus, since it is still constrained by the horizontal grid, there is still some error remaining. The only way to reduce this error is to make the horizontal grid lines denser by adding significant bits to the numbers and the DAC.



Comparing Fig. 1c with Fig. 1d, it should be apparent that the horizontal spacing of vertical grid lines determines the shortest waveform wiggle that can be reproduced. Since the horizontal direction is time, this is the same as saying that the horizontal spacing limits the highest audio frequency that can be reproduced. In particular, it takes at least two grid lines to reproduce one cycle of a sine waveform, so it follows that the highest audio frequency must be equal to or less than one-half of the rate at which numbers are fed to the DAC. Actually, you can go this high only when an ideal low-pass filter is used to do the smoothing.

With a practical low-cost filter, audio frequencies should be kept to 40 percent or less of the DAC update, or sample rate. If higher frequencies are attempted, the filter cannot properly smooth the waveform and an unpleasant distortion called alias distortion occurs. Note that there is no low-frequency limit when producing sound with a DAC; you can go clear down to dc if desired.

No matter how dense the vertical grid lines become, the waveform accuracy is always constrained by the horizontal grid line spacing. This constraint leads to background noise and distortion that cannot be filtered out.

Fig. 1e illustrates the effect of adding another bit of precision to the DAC and the numbers it receives. In terms of audio noise level, adding the bit reduces noise by six decibels, a significant but not dramatic amount. In most cases the noise itself is basically white, resembling somewhat the sound of ocean surf.

Fig. 2 shows how the fidelity of sam-

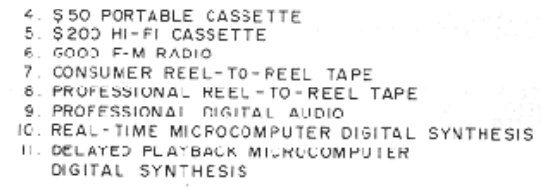


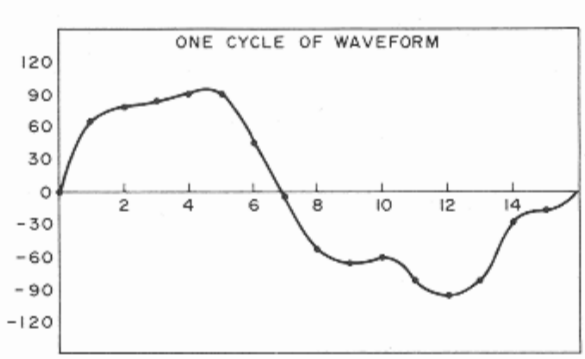
Fig. 2. Required DAC resolution, sample rate and data required for varying degrees of fidelity.

Fig. 2 shows how the fidelity of various combinations of sample rate (horizontal grid spacing) and DAC resolution (vertical grid spacing) compares with familiar audio devices. Also shown are contours of constant data rate (total kilobytes per second) to give an idea of required

system speed. The important points are that frequency range and background noise level are independently adjustable system parameters and that greater fidelity is accompanied by a higher data rate. Note that it is considerably less expensive in terms of data rate to reduce the noise level than it is to increase the high-frequency limit. The two stars in Fig. 2 represent the two software digital music synthesis system that will be discussed in this article.

Where the Numbers Come From

- The real trick in a DAC-based music system, then, is to compute the string of numbers, or samples, repreenting the desired sound and then send it to the DAC at the required ate. In all of the cases that will be considered here, the sample rate will be constant because that assumption greatly simplifies the computations. Conversely, when the rate is assumed to be constant, it must be to rather close tolerances to avoid excessive jitter noise.
  At this point you can choose to go in either of two directions. In real-time digital synthesis, the samples are computed at the rate required by the DAC and sent to it immediately. The advantage, of course, is that the sound is heard in its final form as the program is running. The disadvantage is that practical sample rates are relatively high, which means that a very efficient program using an uncomplicated synthesis technique running on a fast microcomputer is required.
  The other choice is delayed playback digital synthesis, where the computed samples are first written into a mass storage device at relatively low speed and then later reread and played through the DAC at the necessary high speed. The advantages here are that the synthesis program can be more accurate (and thus slower!, any synthesis technique of any complexity can be utilized, and the higher sample rates and DAC resolutions necessary for high fidelity can be utilized. The main disadvantages are a rather long delay between program execution and audible results and the need for a large capacity, high-speed mass storage system.
  It is also possible to combine the two philosophies-real time for composition and experimentation with the orchestration and delayed playback for a high-fidelity final result.



WAVEFORM TABLE

ADDRESS	CONTENTS
0	0
1	67
2	79
3	81
4	92
5	93
6	46
7	-7
8	-55

9	-64
10	-60
11	-80
12	-93
13	-86
14	-27
15	-14

Fig. 3. Example waveform table.

The emphasis in this article will be on the real-time type of system, since that is of interest to most people at this time.

It should be noted that the software techniques that will be described actually perform the same functions as the hardware programmable waveform synthesizer boards mentioned earlier. In fact, a study of the history of computer music reveals that the techniques were developed first in software and then later implemented in hardware when it became practical to do so.

Let's begin by choosing a microprocessor, a sample rate and a DAC word size and then determine what can be done within those constraints.

The microprocessor will be a 6502 because of its usage in a large number of computers (PET, Apple II, Atari, KIM-1, SYM-1, AIM-65, OSI and Mattel, to name a few) and its effective high speed in this application (at a standard 1 MHz, approximately 60 percent faster than a 2 MHz 8080 or Z-80). About the lowest sample rate of interest is 8 kHz. This would allow audio frequencies up to a little beyond 3 kHz, which is just below the highest note on the piano keyboard. More important, it might limit the high harmonics of many instrumental sounds and thus give them a somewhat muffled character. Nevertheless, it is high enough to be useful.

Needless to say, the DAC word size will be eight bits because of the microprocessor chosen. This would give an audible but acceptably low background noise level with a DAC

having true eight-bit linearity and a

good low-pass filter.

Using the 8 kHz sample rate means that the time between samples is a mere 125 us, or around 40 machine-language instructions per sample. Clearly you cannot write the program in BASIC, use the SIN function to compute samples, use floating-point arithmetic or even perform a simple multiply operation and get a sound sample computed in 125 us, much less several for multi-part harmony. Although there are simple (and thus fast) methods of computing sawtooth, triangle and square wave samples directly, only three different waveforms would not be very much variety.

The answer is to compute the waveform ahead of time and put the results in a waveform table. The music playing program then merely looks up the waveform samples in the table—an operation that takes almost no time on a 6502—and sends them to the DAC. Since the waveform is precomputed, even a very slow BASIC program is acceptable for computing it.

A waveform table is nothing more than a string of bytes in memory where each byte is a sample along the stored waveform. For simplicity, which means speed, waveform tables will be made 256 bytes long, or one memory page. The 256-byte content of the waveform table represents exactly one cycle of the stored waveform. Fig. 3 aids in understanding the relation between a waveform and its image in the table. For illustration purposes the table length is assumed to be 16 bytes, but the same principles hold for the 256-byte length.

Now let's see how a music synthesis program might look up, or scan, the waveform table to produce sound. In its simplest form, scanning consists of reading the first entry, sending it to the DAC, reading the second entry, sending it to the DAC, etc., in a loop. When the end of the table is reached, it is necessary to wrap around to the beginning for the next cycle of the waveform.

If you did this at an 8 kHz rate with a 256 entry table, approximately 31 scans would be performed each second, which corresponds to 31 waveform cycles per second, a very low frequency note indeed. For higher frequency notes the scanning could be altered so that every other entry was read, which would give 62 Hz, every third for 93 Hz, etc. Although not intuitively obvious, skipping

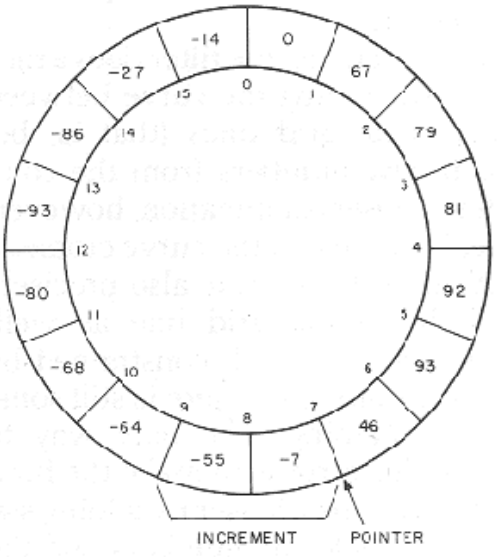
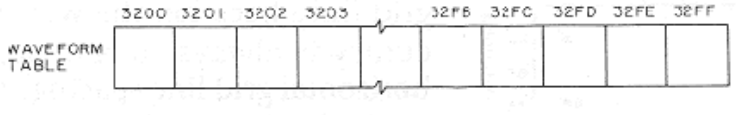
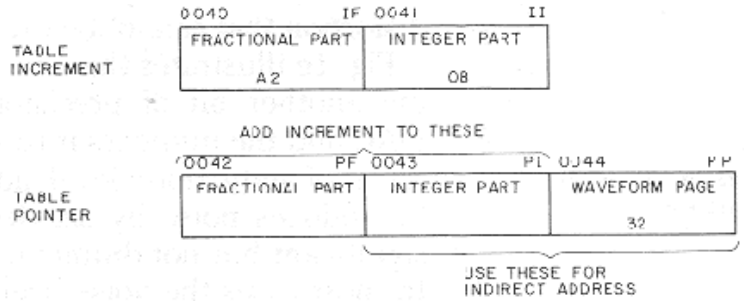


Fig. 4. Waveform table scanning.





ADDRESS	CODE	ASSEMBLY STATEMENT			EXECUTION TIME
C200	A5 42	LDA	PF	ADD FRACTIONAL PART OF INCREMENT TO	3
C202	65 40	ADC	IF	FRACTIONAL PART OF POINTER	3
C204	85 42	STA	PF		3
C206	A5 43	LDA	PI	ADD INTEGER PART OF	3
C208	65 41	ADC	II	INCREMENT TO INTEGER PART OF POINTER WITH CARRY	3
C20A	85 42	STA	PI		3
C20C	B1 43	LDA	(PI),Y	DO THE TABLE LOOKUP REGISTER Y=0, A HAS TABLE ENTRY WHEN DONE	5
					23

Fig. 5. Table scanning on the 6502.

waveform table entries does not contribute to distortion if the tabulated waveform conforms to certain rules that will be discussed later.

Fig. 4 aids in understanding the scanning process. Here the example 16-point waveform table has been bent into a circle, which is one way to view the wrap-around process mentioned earlier. The arrow represents the waveform table pointer, which contains the contents of a machine register or memory location. The bracket represents 'the value of the waveform table increment, which indicates how far the table pointer is advanced every 125 us sample period.

Thus, if the increment is one, the pointer will take on values of 0, 1, 2, . . ., 14, 15, 0, . . . (0-255 in real life) and give us a low note. If the increment is 3, the pointer will go through the sequence 0, 3, 6, 9, 12, 15, 2, . . . and give us a three-times-higher note. Thus, the increment is proportional to the pitch of the synthesized tone. Note that in this case successive trips around the table are not exactly the same. Again, this does not lead to distortion if the waveform meets certain requirements.

Returning to the real case of a 256 point table, it is apparent that the frequency resolution of 31 Hz when using integral waveform table increments is not sufficient for most musical applications. What is needed is the ability to specify an increment with a fractional part such as 7.04 to produce a precise A below middle C. This is quite possible but requires that the waveform pointer also take on a fractional part, which leads to a problem. How should the table be read when the pointer says "read the 78.645th entry"?

A sensible answer would be to look at both the 78th and 79th entries and then interpolate between them. Unfortunately, even simple linear interpolation is fairly complex (requires a multiply), which means it is slow. For real-time digital synthesis on a microcomputer, we will be forced to ignore the fractional part of the pointer when reading the table but include it when adding the increment to compute the next value of the pointer. Taking this shortcut leads to a distortion called interpolation noise, which is significant but generally tolerable.

Now how might a program segment be set up to manipulate the pointer, increment and table to generate sample values for the DAC? Fig. 5 shows the arrangement of a waveform table, its pointer and its increment in memory. For illustration purposes, the waveform table is assumed to be in memory from 3200-32FF, which is page 32, while the pointer and increment are kept in memory page zero for fast access. The increment is a two-byte value with an integer byte and fraction byte as mentioned above. The decimal equivalent of the increment value shown is 11.633. The pointer is actually a three-byte value.

The most significant byte is the page number (32) of the waveform table and normally remains constant but can be changed to select a different waveform. The middle byte is the integer byte of 1he pointer into that table, while the least significant byte is the fractional part of the pointer.

Every sample period (125 us) the increment is double-precision added to the integer and fractional parts of the pointer, and the pointer is replaced with the result. Any overflow is simply ignored, since it is merely an indication of wrap-around from the end to the beginning of the waveform table. Actual table lookup is extremely simple in the 6502; you simply use the rightmost two bytes of the pointer (the waveform table page address and the integer part of the pointer) as the indirect address of an indirect load instruction. Thus, only one instruction is needed to look up in the waveform table. The 6502 machine-language code shown

requires only 23 us to do all of this.
Since the 23 us figure is considerably less than the 125 us allowable, you can have several waveforms, pointers and increments for several simultaneous tones. There is enough time to handle four tones with some left over for housekeeping, You could also have fewer voices and a higher sample rate, or more and a lower rate.

```
A00F            DAC     =       X'A00F      ; OUTPUT PORT ADDRESS WITH DAC
3000            WAV1TB  =       X'3000      ; WAVEFORM TABLE FOR VOICE 1
3100            WAV2TB  =       X'3100      ; WAVEFORM TABLE FOR VOICE 1
3200            WAV3TB  =       X'3200      ; WAVEFORM TABLE FOR VOICE 1
3300            WAV4TB  =       X'3300      ; WAVEFORM TABLE FOR VOICE 1

0000                    .=      0           ; STORAGE STARTS AT PAGE 0 LOCATION 0

0000 00         V1PT:   .BYTE   0           ; VOICE 1 WAVE POINTER, FRACTIONAL PART
0001 00                 .BYTE   0           ; INTEGER PART
0002 30                 .BYTE   WAV1TB/256  ; WAVEFORM TABLE PAGE ADDRESS FOR VOICE 1
0003 00         V2PT:   .BYTE   0           ; SAME AS ABOVE FOR VOICE 2
0004 00                 .BYTE   0
0005 31                 .BYTE   WAV2TB/256
0006 00         V3PT:   .BYTE   0           ; SAME AS ABOVE FOR VOICE 3
0007 00                 .BYTE   0           ;
0008 32                 .BYTE   WAV3TB/256
0009 00         V4PT:   .BYTE   0           ; SAME AS ABOVE FOR VOICE 4
000A 00                 .BYTE   0
000B 33                 .BYTE   WAV4TB/256

000C 0000       V1IN:   .WORD   0           ; VOICE 1 INCREMENT (FREQUENCY PARAMETER)
000E 0000       V2IN:   .WORD   0           ; VOICE 2
0010 0000       V3IN:   .WORD   0           ; VOICE 3
0012 0000       V4IN:   .WORD   0           ; VOICE 4

0014 00         DUR:    .BYTE   0           ; DURATION COUNTER
0015 B6         TEMPO:  .BYTE   182         ; TEMPO CONTROL VALUE, TYPICAL VALUE FOR
                                            ; 4:4 TIME, 60 BEATS PER MINUTE, DURATION
                                            ; BYTE = 48 (10) DESIGNATES A QUARTER NOTE

                ;       4 VOICE PLAY SUBROUTINE
                ;       ENTER WITH VARIOUS TABLE POINTERS ALREADY SET UP
                ;       LOOPS TEMPO*DUR TIMES
0300                    .=      X'0300
0300 A000       PLAY:   LDY     #0          ; SET Y TO ZERO FOR STRAIGHT INDIRECT
0302 A615               LDX     TEMPO       ; SET X TO TEMPO COUNT
                                            ; COMPUTE AND OUTPUT A COMPOSITE SAMPLE
0304 18         PLAY1:  CLC                 ; CLEAR CARRY
0305 3101               LDA     (V1PT+1),Y  ; ADD UP 4 VOICE SAMPLES
0307 7104               ADC     (V2PT+1),Y  ; USING INDIRECT ADDRESSING THROUGH VOICE
0309 7107               ADC     (V3PT+1),Y  ; POINTERS INTO WAVEFORM TABLES
030B 710A               ADC     (V4PT+1),Y  ; STRAIGHT INDIRECT WHEN Y INDEX = 0
030D 8D0FA0             STA     DAC         ; SEND SUM TO DIGITAL-TO-ANALOG CONVERTER
0310 A500               LDA     V1PT        ; ADD INCREMENTS TO POINTERS FOR
0312 650C               ADC     V1IN        ; THE 4 VOICES
0314 8500               STA     V1PT        ; FIRST FRACTIONAL PART
0316 A501               LDA     V1PT+1
0318 650D               ADC     V1IN+1
031A 8501               STA     V1PT+1      ; THEN INTEGER PART
031C A503               LDA     V2PT        ; VOICE 2
```

Listing 1. Core sound generation routine for organ-like music.

There are two ways to combine the four table-lookup values into a single eight-bit value for the DAC. One is to simply add them up and send the sum to the DAC, which is the equivalent of audio mixing. When this is done the waveform table values must have been adjusted when the table was computed to avoid overflow (which can lead to horrendous distortion) when the four voices are added up.
The other method is to immediately send each value to the DAC when it is found and let the lowpass filter smear them together, thus effecting mixing. One disadvantage of this approach is that the dwell time of each voice in the DAC must be the same or there will be differences in loudness among the voices. Another disadvantage is that certain DAC distortions are accentuated, although they are usually not significant at the

eight-bit level. It is also a simple matter to have two DACsand direct two voices to each for an approximation to stereo.

Listing 1 shows the core sound generation routine used in a digital synthesis program first published in 1977. It is capable of generating four tones simultaneously, where each tone can use a different waveform table. It uses the "add-ern-up" technique of mixing the four voices into a single sample value for the DAC. A separate routine is expected to store the appropriate values in each of the four increments for the desired pitches and also set TEMPO and DUR for the desired duration of the chord.

Each time through the main loop takes 115 us and represents one sample period, thus the sample rate is 8.7 kHz. Also, each time through the loop decrements a copy of TEMPO, which is held in the X register. When X decrements to zero, it is restored from TEMPO, and DUR is decremented directly in memory. If DUR also decrements to zero, the chord is complete and a return to the setup routine is taken. Thus, the total chord duration is proportional to the product of TEMPO and DUR. This property makes it possible to change the speed of the music without recoding it.

Note the presence of time-equalizing instructions at TIMWAS so that the loop time is the same whether or not register X decrements to zero. This is necessary to eliminate jitter distortion mentioned earlier. The setup routine would look at coded music in memory to determine what successive values of the four increments, DUR and possibly TEMPO should be to produce the desired music. Typically, music data would be set up in memory as a set of five bytes for each musical"event" (note or chord) in the piece. The first byte would be the duration, while the other four would represent the desired pitch of each of the four voices.

A note frequency table would be used to determine the proper two-byte value of the increment from the one-byte pitch code. This routine must also be as fast as possible because sound generation is stopped when it is in control. If the flow of samples is stopped for too long, an objectionable click between notes is introduced. See references for a further explanation of the setup routine.

Next month we will continue our discussion of synthesizing multiple tones using waveform table data, explore the capabilities of existing DAC software and examine some of the prospects for the future.

Part 2 of this music synthesis article explores the computation of data for waveform tables and describes an experimental KIM-based synthesis system.

Simulation of Musical Instruments

By Hal Chamberlin

- In Part 1 we devised a method of synthesizing multiple tones with any waveform desired. The question now becomes, "How do you determine what samples to put into a waveform table?"

  Perhaps the simplest method is to draw one cycle of the waveform on graph paper and then laboriously read off 256 sample values and enter them into the table. The drawn shape could come from an oscilloscope photo of a musical instrument sound or from imagination. The drawn shape must span exactly 256 grid lines in exactly one cycle to be valid. You could also make use of a light pen or graphic digitizer in conjunction with a drawing program to do the same thing with much less effort.

  The biggest problem, when using imagination is that there is no simple relation between the appearance of the drawn shape and the resulting timbre. Thus, if a particular shape produces a sound that is close to what is desired, there is no way to know what must be changed to make it sound even closer.

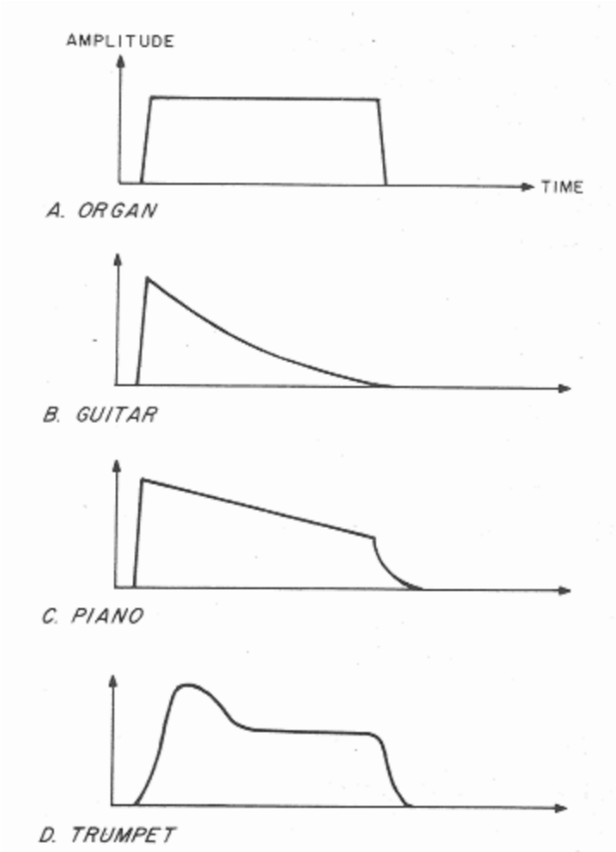


Fig. 1. Typical instrument amplitude envelopes.

Filling the Waveform Tables

Probably the best way to fill waveform tables is to write a program that accepts harmonic specifications, computes the corresponding wave-shape and automatically enters it into memory. There is a very definite correlation between the harmonic makeup of a tone and its timbre. You can also occasionally find published harmonic analyses of musical instrument tones, particularly organ pipes.

Listing 1 shows a very simple BASIC program that can be used to create waveform table data and poke it directly into memory. The statements starting at line 3000 first amplitude-normalize the waveform, convert the samples into integer form in the range of 0 to 63 (to avoid overflow when four are added up) and then poke them into memory.

The biggest advantage of using harmonics to specify waveforms is that alias distortion can be readily avoided. Alias distortion occurs whenever any frequency component of a waveform exceeds one-half of the sampling frequency. This can easily happen with high notes using waveforms rich in harmonics.

For example, if you attempt to play high C (523 Hz) using a waveform with ten significant harmonics through an 8 kHz sample rate system, the eighth, ninth and tenth harmonics will alias, since they will be 4184, 4707 and 5230 Hz, respectively, all above four kHz. Aliasing means that intended frequencies are altered and usually produces an objectionably harsh sound. Thus, waveform tables used to play high notes should have their upper harmonics restricted, while those for low notes may have dozens of significant harmonics if desired.

Musical Instrument Synthesis.

After some experimentation with different waveforms and types of music, you will discover that a wide

variety of tone colors is possible, but the tones always sound like an organ. Ofcourse, the organ is the most versatile of conventional musical instruments, but digital synthesis should be able to do better. One of the reasons for an organ-like sound is that only continuous, sustained tones can be generated by simple waveform table scanning. In other words, the ampli tude envelope is rectangular, as shown in Fig. la. Many instruments have other shapes, such as those in Figs. lb, lc and ld.
The standard method of adding an amplitude envelope to a sound is to pass it through a variable-gain amplifier and vary the gain in accordance with the desired envelope shape. In digital synthesis this is equivalent to multiplying the samples representing the sound by an amplitude factor that changes as the note progresses. The series of amplitude factors could come from an envelope table that is scanned just like the waveform table but much more slowly.

```
1000 REM WAVEFORM TABLE FILL PROGRAM
1001 REM ENTER HARMONIC NUMBER FOLLOWED BY RELATIVE AMPLITUDE
1002 REM HARMONIC NUMBER=0 FILLS THE TABLE AND EXITS
1010 DIM W(256): Z=6.283185/256
2000 FOR I=0 TO 255: W(I)=0: NEXT I
2010 PRINT "ENTER HARMONIC NUMBER ";: INPUT N
2020 IF N=0 GOTO 3000
2030 PRINT "ENTER RELATIVE AMPLITUDE ";: INPUT A
2040 FOR I=0 TO 255: W(I)=W(I)+A*SIN(N*I*Z): NEXT I
2050 GOTO 2010
3000 M=0
3010 FOR I=0 TO 255
3020 IF ABS(W(I))>M THEN M=ABS(W(I))
3030 NEXT I
3040 PRINT "ENTER ADDRESS OF WAVEFORM TABLE ";: INPUT A
3050 FOR I=0 TO 255
3060 POKE A+I,INT(31.5*W(I)/M+32)
3070 NEXT I
9999 STOP
```

Listing 1. Waveform Table Fill program in BASIC.

Adding overall envelope control certainly improves the variety of sounds available and is frequently enough to give reasonable simulations of common musical instruments. However, rather than spending a lot of time explaining how overall envelope control can be added to a table-scanning digital synthesis sys tem (which mainly involves methods for eliminating time-consuming multiplication), let's go all the way and include timbre envelopes as well.
To some extent the sound of all in- struments changes its waveform during the course of a note. Consider, for example, the' 'waaahhh" of a muted trombone or the "twaanng" of a guitar. The change in character of the sound during the notes is what makes these instrument sounds so distinctive. In terms of synthesizing these and similar sounds, it is the harmonic composition, as well as the overall amplitude, of the waveform that changes gradually.
The standard method of adding a timbre envelope to a sound is to pass it through a variable filter and vary the cutoff or center frequency and Q factor in accordance with the desired effect. In digital synthesis you have to use a digital filter, which involves several multiplications per sound sample. This is just not practical in a realtime microcomputer-based system, so some other method must be found. But first we need a way to visualize timbre envelopes so that they can be specified.

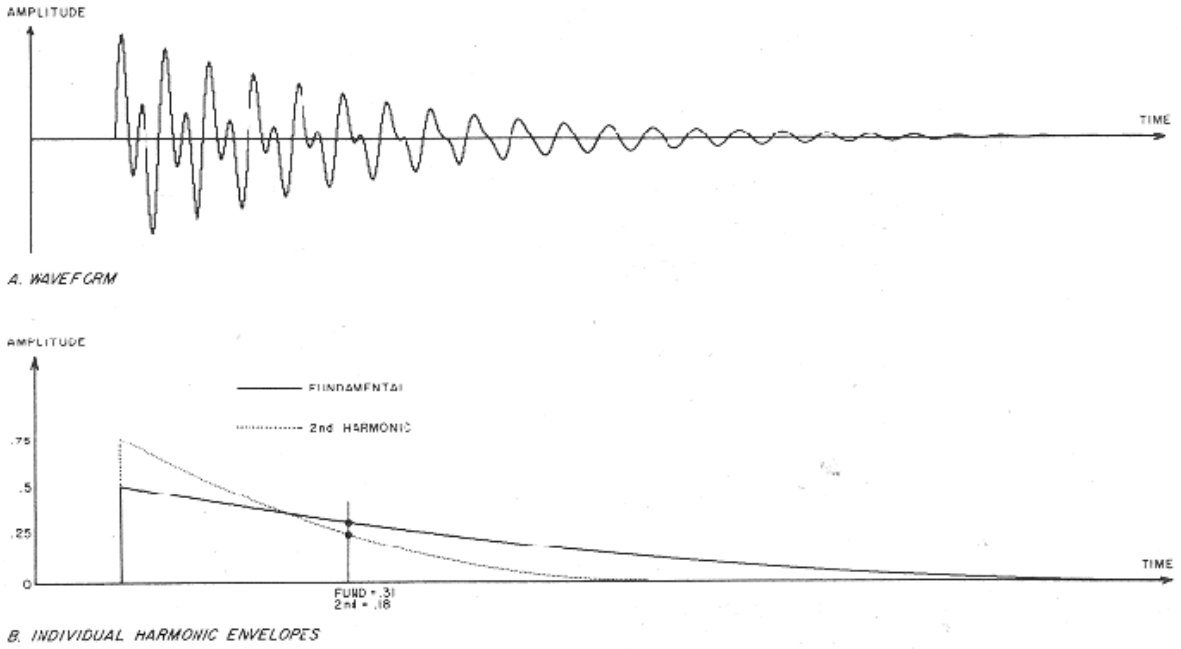


Fig. 2. Simplified characteristics of a plucked string.

Fig. 2a shows a simplified decaying waveform of a plucked string. The overall amplitude envelope is quite similar to that of Fig. lb, but the waveform itself also changes shape.

At the very beginning, the second harmonic is actually stronger than the fundamental. The second harmonic is responsible for the crook in the waveform near the baseline. However, as the waveform decays, the second harmonic decays faster than the fundamental and thus the crook gradually disappears. Eventually, the second harmonic fades out completely, leaving just a decaying sine wave. This is reasonable behavior for a plucked string because high- frequency vibrations encounter greater losses in strings than low-frequency ones do.

Fig. 2b shows one way of representing this behavior in meaningful terms. The solid line shows the amplitude envelope of the fundamental, while the dotted line shows the envelope of the second harmonic. We can find out the harmonic composition of the tone at any point in time by erecting a vertical scale at that point and reading off the amplitude of each harmonic as shown. The same idea will work for any number of harmonics.

Now, how can we modify the tone generator routine described last month for varying waveforms? The secret is to arrange for the waveform table address bytes, which are nor- mally constant, to change while the table scanning is taking place. Thus, while the tone is sounding, the synthesis program is actually switching through a sequence of waveform tables. If the switching is fairly rapid and the contrast between adjacent waveform tables is small, the audible effect is that of a smooth transition. The idea is not unlike that of a sequence of image frames giving the illusion of smooth motion in a movie.

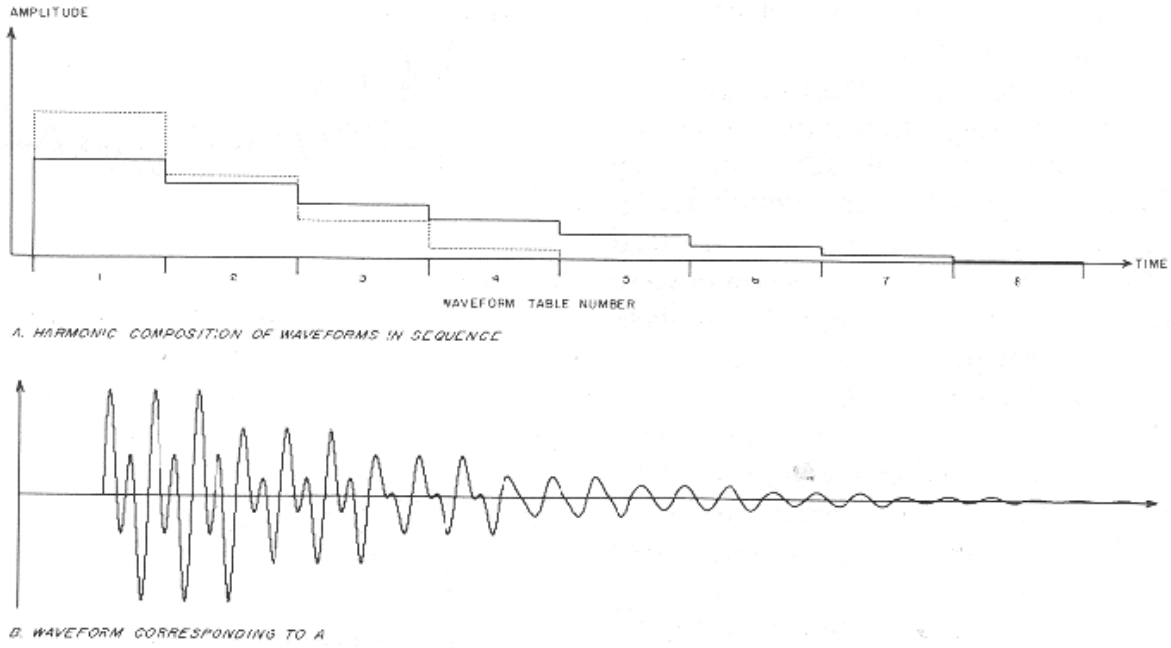


Fig. 3. Example synihesis of a plucked string.

Fig. 3a illustrates this concept by showing the resulting stair-step approximation to the smooth harmonic envelopes in Fig. 2b. In this example only eight waveform tables are used; in a practical situation it is common to use between 15 and 30 of them. Fig. 3b shows the resulting waveform, which even for this coarse example bears a remarkable resemblance to the ideal case in Fig. 2a.

In the actual implementation of waveform table switching, the concept of a waveform sequence table is introduced. The waveform sequence table is nothing more than a table of waveform table addresses. This extra level of indirection is very little problem in a microprocessor such as the 6502, and it has many benefits.

While a note is sounding, a pointer scans through the sequence table at uniform speed just as the waveform pointer scans through the waveform table, but more slowly. In the program implementation, the time equalization instructions are replaced with instructions to move four pointers through their respective waveform sequence tables at a rate of one increment each time register X (TEMPO) times out.

```
031E 650E            ADC    V2IN
0320 8503            STA    V2PT
0322 A504            LDA    V2PT+1
0324 650F            ADC    V2IN+1
0326 8504            STA    V2PT+1
0328 A506            LDA    V3PT          ; VOICE 3
032A 6510            ADC    V3IN
032C 8506            STA    V3PT
032E A507            LDA    V3PT+1
0330 6511            ADC    V3IN+1
0332 8507            STA    V3PT+1
0334 A509            LDA    V4PT          ; VOICE 4
0336 6512            ADC    V4IN
0338 8509            STA    V4PT
033A A50A            LDA    V4PT+1
033C 6513            ADC    V4IN+1
033E 850A            STA    V4PT+1
0340 CA              DEX                  ; DECREMENT & CHECK TEMPO COUNT
0341 D008            BNE    TIMWAS        ; BRANCH TO TIME WASTE IF NOT RUN OUT
0343 C614            DEC    DUR           ; DECREMENT & CHECK DURATION COUNTER
0345 F00C            BEQ    ENDNOT        ; JUMP OUT IF END OF NOTE
0347 A615            LDX    TEMPO         ; RESTORE TEMPO COUNT
0349 D0B9            BNE    PLAY1         ; CONTINUE PLAYING
034B D000    TIMWAS: BNE    .+2           ; 3  WASTE 12 STATES
034D D000            BNE    .+2           ; 3
034F D000            BNE    .+2           ; 3
0351 D0B1            BNE    PLAY1         ; 3  CONTINUE PLAYING
0353 60      ENDNOT: RTS                  ; RETURN
                                          ; TOTAL LOOP TIME = 115 STATES - 8695 HZ
```

Remainder of core sound-generation routine from Part 1.

One advantage of using a sequence table is that waveform switching can be rapid when there is rapid change in the harmonic envelopes and less rapid at other times, thus cutting down on the number of waveforms needed and memory usage. Another advantage is that waveforms do not have to be stored in memory in the order that they are used. This allows such tricks as playing through the attack sequence backwards for the decay sequence to save on memory.

Another trick is to cycle through a few waveforms during the sustain of a note to impart a sort of warble effect on notes. A strumming effect can also be created in this manner. You can even construct several sequence tables for the same set of waveforms to take care of differences in duration and articulation from note to note.

The results of adding waveform table sequencing to the earlier synthesis routine, which was done primarily by Frank Covitz, are astounding. Attempts at simulating plucked string sounds result in a real plucked sound, and you can easily tell the difference between a plucked string and a struck string (nqt possible without timbre envelopes!. Blown instruments sound blown, and bowed instruments sound bowed. You can even get reasonably nice-sounding

bells, even though true bell tones are decidedly inharmonic and therefore cannot be duplicated by simple waveform table scanning.

Many of the instrument definitions (sets of harmonic envelopes) that have been experimented with are based on computer analyses of musical instruments published in the Computer Music journal by James A. Moorer (see references)..

One particularly successful instrument simulation done by Cliff Ash- craft has been a piano. To cover the wide range of the piano, it is necessary to define several instruments, one for each octave. This is because the quality of piano sound varies in different pitch ranges due to differences in string construction and the fact that the sounding board has a finite mass. Music played with his piano definitions is amazingly realistic, just like a real piano in the next room. Consult the references for a full description of the system. "

This article is not primarily concerned with simulating existing musical instruments with a microcomputer. The real interest, and future of computer music synthesis, is in dreaming up entirely new instrumental sounds and composing scores that complement them.

Tone color as a musical variable is just as important as pitch and rhythm and may become more so, since pitch and rhythm composition has been experimented with for centuries, whereas timbre composition has only recently been possible. Convincing simulation of existing musical instruments is an important milestone because most conventional musical instruments produce very complex sounds. Doing a good job on them implies the capability to begin exploring timbre space without a lot of restrictions.

Delayed-Playback Digital Synthesis

While you can do amazing things with real-time software digital synthesis on a microcomputer, the compromises, shortcuts and relatively low sample rates necessary leave something to be desired in the area of fidelity. The faster microprocessors that are beginning to appear (both higher clock frequency standard units and the new 16-bit units) will certainly improve the capability of real-time software synthesis. A 6502 running at 3 MHz, for example (which is currently available), could produce eight voices at a 12 kHz sample rate for fidelity similar to good AM radio reception. However, there are still a number of musical features missing which are needed for a truly versatile system for interest to the majority of musicians and listeners. For example, bending notes {gradually changing their pitch), true vibrato, percussion instrument synthesis and singing voice synthesis are all needed to penetrate the contemporary music idiom (perhaps this is why Bach is so often performed with computers). With delayed playback, any or all of the compromises may be eliminated, the sample rate and DAC accuracy may be increased to true hifi levels, and any desired musical feature that can be defined can be implemented.

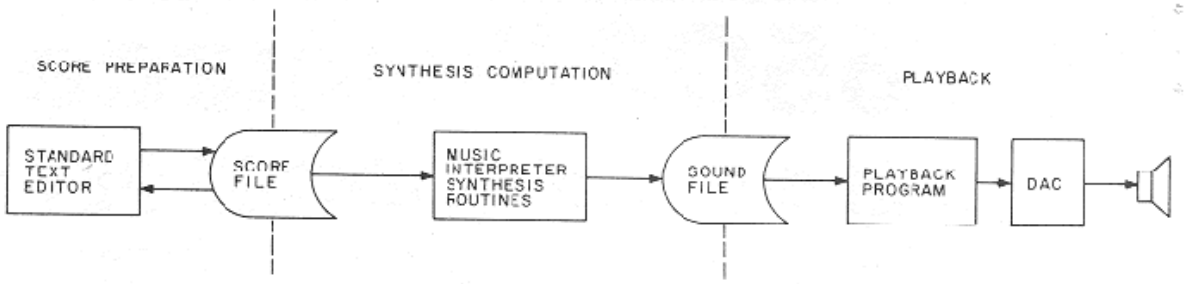


Fig. 4. Delayed-playback software synthesis system.

Fig. 4 shows a block diagram of a delayed-playback software synthesis system as it might be implemented on a microcomputer. Playing a musical selection is actually a three-step process.

In the first step a machine-readable score is entered or edited from a previous run. Typically, the score file on disk is just a standard ASCII text file, so a standard text editing program is sufficient. In advanced systems other methods of score entry, such as graphical input with a light pen, joystick or digitizer or even direct input from a music keyboard, are possible. In any case, the result of the first step is an integrated score and instrument definition file on disk.

In the second step, a music interpreter program, which also contains all of the synthesis routines, reads the score file, carrws out the indicated synthesis operations and writes a sound file on disk. While the majority of your work is spent creating and editing the score file, the vast majority of machine 'work is spent computing the sound file.

Computing a minute of final sound may take anywhere from five minutes to whatever CPO time you can tolerate, depending on the sample rate, number of simultaneous voices playipg and the sophistication of the synthesis techniques. Most of this time is spent in arithmetic subroutines, so a microprocessor with automatic multiply (such as the 6809, 9900 and all of the new 16-bit units) is a distinct advantage.

In the playback step, a highly specialized program reads the sound file from disk and sends the sound samples to the DAC at a uniform rate. When high-resolution DACs (ten bits or more) are used, the uniformity of sample rate becomes critical to minimize jitter distortion. In order to achieve such uniformity while the program is also handling data readback from the sound file, the DAC must generally be equipped with its own sample clock and at least one. level of data buffering.

A Delayed-Playback System

I implemented an experimental delayed-playback software digital synthesis system and demonstrated it at the PC '80 computer show in Philadelphia this fall. It runs on the 6502-based KIM-l microcomputer equipped with 16K of RAM and a Micro Technology Unlimited (MTU) disk controller, which adds another 16K. Two Siemens eight-inch floppy disk drives are used, and the double-density capability of the MTU controller is utilized.

An experimental 12-bit digital-to- analog converter with an additional three bits of gain control is used to get a theoretical dynamic range equivalent to a 16-bit DAC. The gain control is not yet utilized by the software, however. An important feature of the experimental DAC is a 256 sample first-in-first-out buffer which allows the sample stream from the computer to be interrupted for milliseconds at a time without affecting the smooth flow of data to the DAC itself.

When floppy disks are used to hold the sound file, the disk format is an important determinant of the maximum playback data rate. While the normal CODOS disk operating system software (which is used to prepare the score file) uses the standard IBM disk format of 26 sectors of 256 bytes each, the total diskette capacity is only about 512K bytes.

A different format consisting of 16 sectors of 512 bytes is used for the sound file and gives 630K bytes per disk, a 23 percent increase in potential data rate and capacity. In order to read through the sound file at high speed., it is mandatory to be able to read all of the sectors on a track in one revolution of the disk. In addition, you must be able to step to the next track without waiting for a whole revolution before reading again. Staggering the sector numbers by three on adjacent tracks is utilized to accomplish this. The resulting sustained average data rate from the disk can approach 40K bytes per second.

The actual playback program currently uses a 20 kHz sample rate with 12-bit samples for a total data rate of

30K bytes per second. At this data rate, an eight-inch diskette holds about 21 seconds of sound. Going to double-sided disks would double the capacity to 42 seconds. Minidisks have about half the capacity, but more important, only half the maximum data rate.

The synthesis and computation phase of a performance is relatively straightforward on the experimental system. The score file is read from drive 0 using CODOS, and the sound file records are written onto drive 1 using a set of specialized disk driver routines. When a sound disk is filled up, the synthesis program waits for a new disk to be inserted into drive 1.

When the playback program is called in, CODOS is disabled and the operator is expected to put the first sound disk in drive 0 and the second one in drive 1. When playback starts, the first 21 seconds of sound are read from drive 0 and then an immediate, inaudible switchover to drive 1 is performed. During the next 21 seconds, the operator must remove sound disk 1 from drive 0 and insert disk 3 to be read when disk 2 is exhausted. You can switch back and forth like this indefinitely for music of any duration; the performance at the PC' 80 show required 23 disks for eight minutes of sound.

The problem in using this system is not the disk jockeying required during playback but the changing of disks during computation. With the music selected for performance, a new disk was required about every 15 to 30 minutes, which means that the computation cannot be left to run overnight with any degree of benefit. Clearly, a 10 megabyte hard, disk would be advantageous here.

The experimental delayed synthesis program does about the same things as the real-time synthesis program mentioned earlier. The major differences are an essentially unlimited number of voices, interpolation between waveform table entries and interpolation between adjacent waveform tables in the sequence rather than sudden switching. It won't be considered complete until the musical features described previously are implemented.

The Future

While these developments may seem exciting now, the future is likely to see many more exciting things happen in the field of music synthesis on microcomputers. The sophisticated programmable synthesizer boards will undoubtedly become more sophisticated and gradually come down in price. Today's square-wave synthesizer chips will probably be supplemented by programmable waveform synthesizer chips that use direct memory access to automatically scan waveform tables in memory. The most exciting prospects are in the software synthesis area, how- ever. The processors used in personal systems will gradually get faster at the machine-language level, which will increase the capability and fidelity of real-time software synthesis. Even a simple step up to 16 bits, which is inevitable, will nearly double the speed of the core sound routine, giving both more voices.and a higher frequency range. Because of the very low cost of including a DAC in the circuitry of a computer most future systems will probably contain built-in DACs.

On the delayed-playback front, experimental systems such as the one just described will reach full development and make it possible to produce significant music of commercial value with microcomputers. Even the very general and powerful MUSIC-11 system (truly the ultimate in sound synthesis flexibility) has already been implemented on the LSI-11 microcomputer (used in the HeathH11 and Terak systems), and it is only a matter of time before it is available for the more common microcomputers.

The decreasing cost and increasing' capacity of small hard disks will also. make using a delayed-playback type- of system much more convenient and increase the fidelity even further..

References

1. Mathews, Max, *The Technology of Computer Music*, MIT Press, Cambridge, MA, 1969.
2. Moorer, J. and J. Grey, "Lexicon of Analyzed Tones," *Computer Music Journal*, vol. I, number 1, 1977, p. 4 and succeeding issues.
3. Chamberlin, Hal, "A Sampling of Techniques for Computer Performance of Music," September 1977, *Byte*.
4. Chamberlin, Hal, "Advanced Real-Time Music Synthesis Techniques," April 1980, *Byte*.
5. Chamberlin, Hal, *Musical Applications of Microprocessors*, Hayden Book Co., Rochelle Park, NJ.

The two real-time digital synthesis programs described in this article, along with the necessary digital-to-analog converter, may be obtained from Micro Technology Unlimited, Box 12106, Raleigh, NC 27605.

posted : 17 september 2003

Back *Back to* **Richard Davies NLnet Home Page**