

A SOUND EXPERIMENTATION FACILITY FOR CHILDREN

by

Mark M. Preissler

Submitted in Partial Fulfillment

of the Requirements for the

Degree of Bachelor of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May, 1976

Signature redacted

Signature of Author _____

Department of Electrical Engineering, 5/7/1976

Signature redacted

Certified by _____

Thesis Supervisor

Signature redacted

Accepted by _____

Chairman, Departmental Committee on Theses



TABLE OF CONTENTS

<u>SUBJECT</u>	<u>TOPIC</u>
Abstract	2
Introduction	3
Figure 1	4
Figure 2	5
Figure 3	6
System Description	9
Figure 4	11
Figure 5	12
Conceptual Organization	14
Engineering	16
Bells and Whistles	22
Conclusion	26
Table of Macro-codes	Appendix 1
Program Listings	Appendix 2
References	References

ABSTRACT

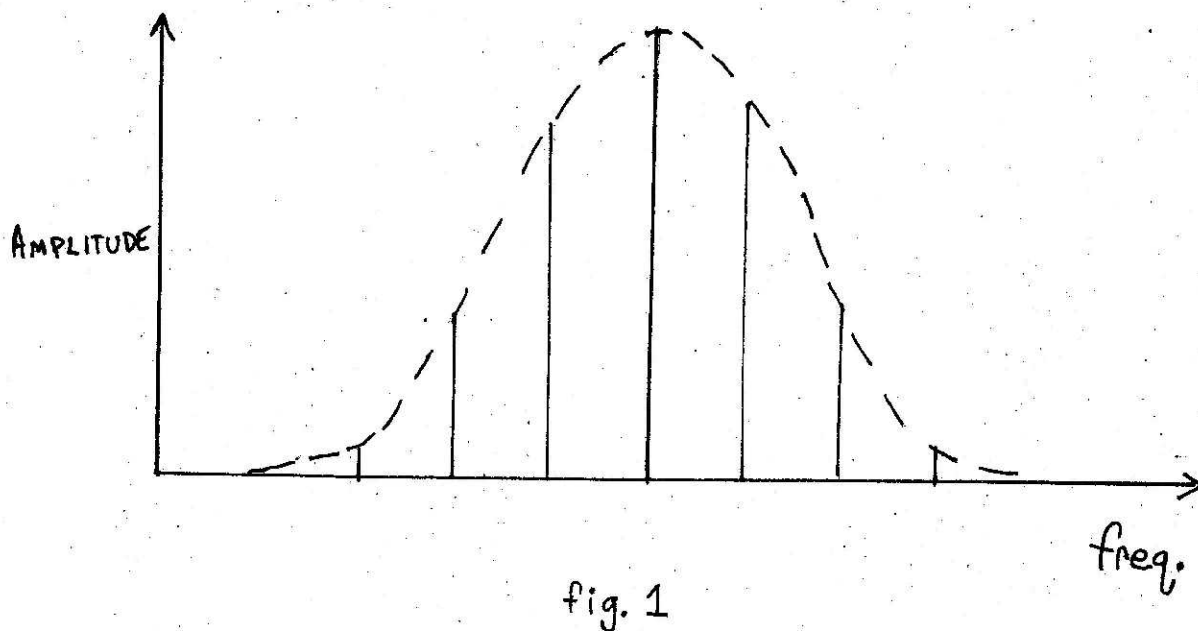
A sound experimentation facility has been implemented for the LOGO world on a GTI - TT2500 mini-computer. The system allows definition and modification of waveforms, sequencing of sonic events, and high quality sound to be produced. Music is representable on a somewhat primitive level. The organization and implementation of the system is described, and modifications and extensions are proposed.

INTRODUCTION

This report describes a sound experimentation facility for children. A better term would be sound experimentation environment, since the facility exists as a set of programs rather than a locus. The environment is part of the LOGO world and is hoped to fulfill certain needs in the aural medium, allowing simple procedural definition and manipulation of sonic events, as the now-classic turtle and its extensions (the seeing eye turtle, touch-sensitive turtle) have allowed simple procedural definitions for behavioral functions and provided a learning environment for children to come in contact with these procedural definitions. It is hoped that the sound experimentation environment will provide as rich an atmosphere in its sphere.

The system was originally intended to allow (relatively) arbitrary sonic events for interrelated purposes of instruction and entertainment. One very instructive sonic illusion is Stewart's

illusion in which a note (whose spectra is shown below) is played for a few seconds (figure 1):



Then, each of the harmonics is shifted up in frequency, still maintaining the envelope (figure 2):

This process is iterated. Eventually the spectra returns to its original state: no increase in frequency has occurred. What obtains perceptually is a rising tone that never gets any higher.

Another perceptual trick is the flipping of the phase of one harmonic

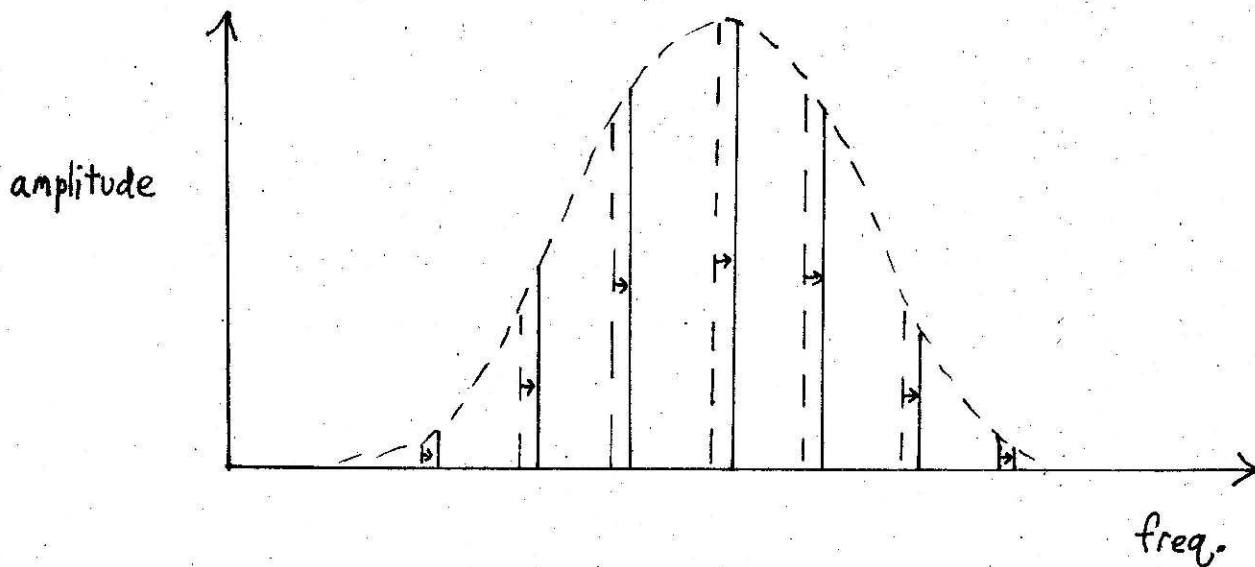


fig. 2

of a note and hearing the harmonic's presence where before it was integrated into the note.

There are several interesting things to be done with multiple channels: causing sounds to fly around in three space by varying the amplitudes and phases of four channel reproducers arranged in a tetrahedron (figure 3):

Simple harmonic motion "thru one's ears" would be an interesting effect rather easily obtained. A more advanced task would

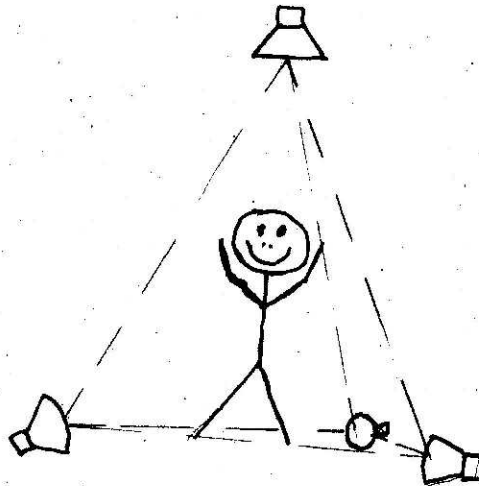


fig. 3

be the creation of sound "jungles" in which sounds would advance and recede as it was "physically" (actually or metaphorically) walked thru. Such a facility would provide an interesting excursion into the rules governing placement of sounds, pursuable topics might be what happens when we change the laws of intensity and distance, what happens if we fit some sort of sensor onto the head and move sounds opposite to where they should be going.

Such a sonic facility has need for a sound wave sculpting

facility. Why does a sinusoid sound like a "pure" tone. What happens if I lop off the top and bottom of a sine wave? What does this waveform sound like? The ability to see a waveform on a display screen, to hear it, then change the waveform, see it, hear it,...

One very powerful way to produce waveforms is to add weighted sinusoids of varying frequencies to one another, a la Fourier series. To see and hear sums of sinusoids become square waves is a potentially simple way of giving an intuitive grasp of Fourier series in particular and frequency domain analysis in general. Other ways of manipulating waveforms are apparent: digital filtering techniques, simple addition of two tones yielding the sound of both tones (I remember confusion as a child as to how could both a guitar and a singer come over that one wire!), phase, frequency, and amplitude twiddling, etc.

Manipulation must also be allowed to take place on sequences of such events. In order to have the ability to do music, illusions that evolve over time, or any sounds that depend on external events we must be able to reprogram the device. First we need a representation for a

series of sonic events referencing user definable and manipulable waveforms. A macro facility is provided for this purpose. The details of this will be given in the Engineering section.

Such a system needs to produce good audio. To represent 20 khz frequency we at very minimum need a sample rate of 40 khz or 25 usecs between samples. And in order to do music our representation of frequency needs to be fairly accurate. [Backus] reports 3 cycles is the minimum percieveable difference (below 1 khz) in pitch. Also, large number of output channels should be provided, at least four to represent 3-space.

Another necessary property of the system is ease of use. If the user doesn't care to know about all the properties of the system (eg. the stored program facility, multiple voices) he shouldn't be bothered by them. The representation of sequences of sonic events should be reasonably easy to use. And the "flavor" of the programming ability should be interpretive - allowing things to be tried, modified, retried, and then made somewhat permanent.

As with any software, the components need to be as decouplable as possible. For example, if at some future point in time the programming feature is found to be inappropriate, a modification of the macro facilities should not affect the other software.

SYSTEM DESCRIPTION - WHAT THE USER SEES

A voice is an assignable channel. Physically, a voice is an output register and a DAC. The hardware can support 4 voices at a 40 khz sample rate and 8 voices at a 20 khz sample rate, each playing arbitrary waveforms. Conceptually a voice is a triple consisting of a frequency, a phase, and a waveform pointer. Due to a lack of hardware multiplies, an amplitude is not a changeable attribute of a voice, it must be changed on the waveform level (see below). The operations that are permitted on a voice are changing phase, changing frequency, assigning a different waveform, and activation or deactivation (albeit the last occurs indirectly).

A waveform is a double consisting of a 240 element array of 16 bit words representing a single cycle of a periodic signal and a 16 bit frequency multiplier that is invisible to the user and whose purpose will be explained below. Conceptually, a waveform is an oscillator whose characteristics can be changed at will. Operations permitted on waveforms are creation (such as making one a sinusoid, sawtooth, etc.), scaling (takes about 4 msec), display (on a graphics device) and combination.

Combination is a 7 argument operation: wvfm1, wvfm2, wvfm3, a, b, k, l. Essentially, wvfm1 and wvfm2 are scaled by a and b, then are expanded or contracted in time according to k and l. That is, letting wvfm_i(f) be a waveform i at frequency f (remember that frequency is not bound to the waveform until it is played: "f" is just a reference frequency), this function will perform the operation

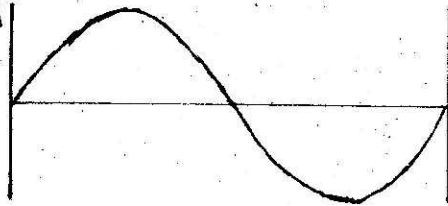
$$\text{wvfm3}(f) \leftarrow a * \text{wvfm1}(f) + b * \text{wvfm2}((k/l)*f)$$

let $k=3, l=1$

1 copy of wfm 1

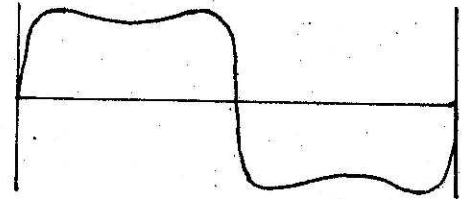
11

SYSTEM DESCRIPTION

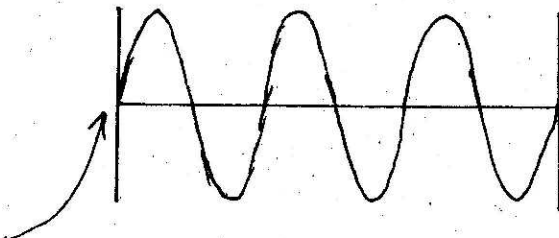


:freq "f"

let $a=1, b=\frac{1}{3} \rightarrow$



:freq "f"



:freq "f" $\approx k/l$

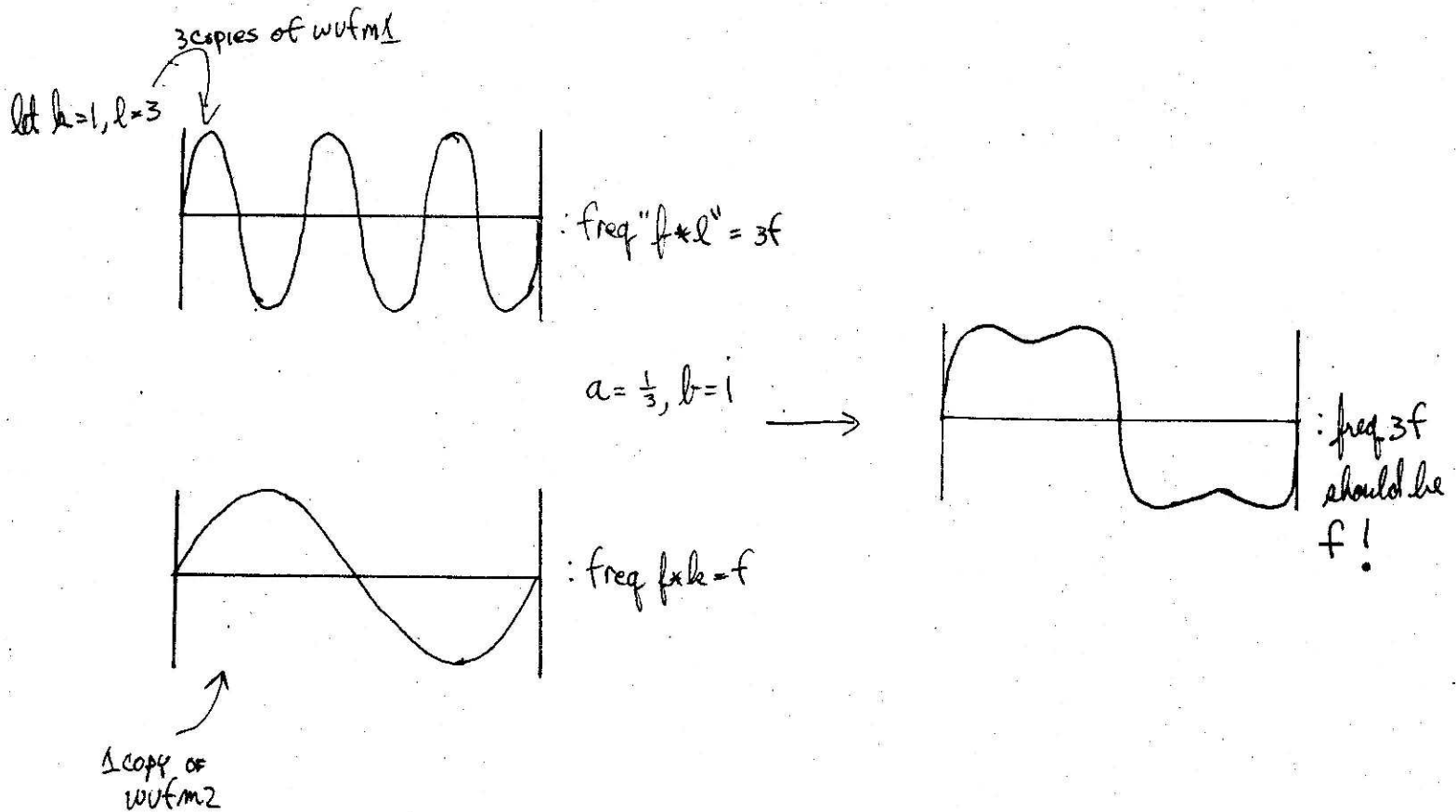
3 copies of wfm 3

fig.4

Graphically (figure 4):

A problem occurs if l doesn't equal 1, that is when the frequency of wfm2 will be at a non-integral frequency multiple of wfm1. This will be where the frequency multiplier helps us out. What we do now is make l copies of wfm1 and k copies of wfm2, scale them by a and b and add them together (figure 5):

But now if we play this waveform at frequency f we will actually be playing it at frequency $l*f$. This is due to the fact that we didn't



know there were 1 copies of wvfml, the waveform at the reference frequency. In order to keep track of this, we keep the "1" around. When we initialize a voice, we will divide our frequency by this number. The combination operation gives us a lot of mileage - providing many functions for sound experimentation such as addition of sinusoids, overlaying of waveforms, etc.

The sequencing feature needs description. The allowed operations are listed in Appendix 1. There are two modes of operation:

command mode and interpret mode. In command mode every command that is given to the system is executed immediately. I would imagine this to occur if someone was sound-sculpting or trying to debug a sonic sequence. In interpret mode the commands would be stored as macros for subsequent execution. Then from command mode these stored sequences could be recalled and played and used as building blocks for larger sequences.

The programs written for the system are divided into two major categories, the code that runs the voices in the 2500 (a mini-computer that handles the real-time stuff), and the code that handles miscellaneous LOGO functions. The LOGO functions include the combination operation, display upon a graphics device, and a few other operations needed to interface the 2500 with LOGO. The 2500 program besides the real-time sampling of the waveforms also provides the macro capability allowing interpretable sequences of sonic events.

CONCEPTUAL ORGANIZATION OF THE SYSTEM

The code written for the 2500 (the minicomputer that runs the "guts" of the system) is centralized upon the routine oscstart. An alternate entry point, oscinit, sets up some constants. This routine takes 1 argument : the number of active voices. The voices are treated as an ordered set, and if oscstart is called with n then voices 1, 2, ... n are treated as active and the inner loop is iterated n times. The routine does not return until a 40 khz clock fires, providing the time base for the system.

There are several routines that provide time measuring facilities. Play is a two argument subroutine that plays the current voice configuration of n voices for s seconds. Playm is an alternate entry point that plays for s milliseconds. Wait and waitm should be self explanatory. Playforever allows the current configuration to be played forever (until an interrupt occurs).

Another class of routines are provided for creation of waveforms: impulse, zerowvm, sin, square. These are generally 2

argument functions: amplitude and waveform number. The proper waveform with the specified amplitude is put into the location specified by the waveform number. The waveform number is a member of the set $\{0, 1, \dots\}$.

There are several miscellaneous routines: `nsamples` plays `n` samples of the active waveforms. `Scalearray` multiplies a waveform by an 8.8 bit fixed binary number. `Mapwvfm#` takes the waveform number and produces the location of the beginning of the waveform. `Posmultiply` performs multiplication of 2 16 bit numbers.

The macro facility is simply an instruction decode and a call to the routines that are specified. As of now, the only control structure allowed is the unconditional branch. Future extensions are covered in the Bells and Whistles section to come.

A scratchpad register, `pc`, is used as a pointer into the macro-code. The format of the macro is as follows:

	instruction	
	arg1	
	arg2	
	.	
	.	
	.	
	.	

The macro-code is stored in main memory locations 5670-7634 (octal).

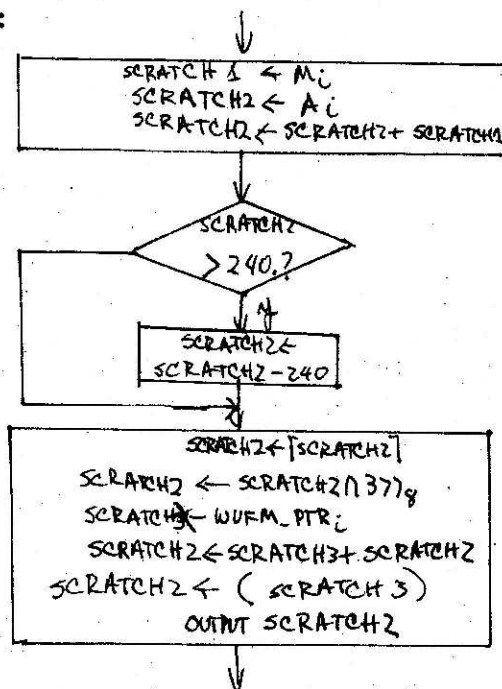
Two routines, fetch and store, do all the accessing of the macros.

ENGINEERING

The restrictions to the routine have been mentioned before: high frequency resolution and fast execution speed. The high frequency resolution is provided by the DDA algorithm. As mentioned before, a voice is a triple consisting of a waveform name (pointer), a phase, and a frequency. The phase is represented as a special index into the waveform: it is not simply an integer, but as an 8 bit integer and 8 bit fraction (called a). Only the 8 bit integer portion of a is used

for the index. The frequency is also represented as an 8.8 bit fixed binary number (called m). The frequency is actually a displacement to be added into the phase each time. The phase is truncated to an integer, compared with the size of the waveform (for wraparound) and then used as an index into the waveform for the sample. Below is a

flowchart (figure 6):



The various a's, m's, and waveform pointers are kept in a voice state vector, a contiguous block of 24 words allowing up to 8 voices in the

2500's scratchpad address space. A feature of the 2500 allows indexing of these voice parameters. In order to increase execution speed of the inner loop, the active voices are assumed to be the low order "n" in the block (n is the single argument to the oscstart routine). Therefore, activation and deactivation of a voice is done only via changing "n". In order to deactivate a voice in the middle, it would either have to be assigned a zero waveform or be reshuffled in the block (by calls to voice).

For our purposes, the DDA algorithm is accurate enough. If s is the sampling frequency, m is the frequency representation (as discussed above) and f is the frequency of the waveform (that is, every 1/f seconds the waveform is reiterated) we have

$$f = s / (240/m) = s * m / 240 .$$

Since m is quantized and has 8 bits of fractional precision we can denote

$$m' = m + e$$

to be the "true" frequency representation. It is clear that

$$|e| \leq 1/512$$

and that the bounds for the frequency f with frequency representation m are

$$s^{*(m+e)}/240 \Rightarrow f \Rightarrow s^{*(m-e)}/240.$$

If we denote $c_{\text{sub } i}$ to be the fraction of error in units of

a semitone (if $c_{\text{sub } i} = 1$ we would be one semitone off), $i = 1$ meaning the next lower available frequency and $i = 2$ meaning the next higher one,

then

$$f = (2^{(1/12)})^{c_2} s^{*(m+e)}/240$$

$$= (2^{(1/12)})^{c_1} s^{*(m-e)}/240$$

or

$$c_2 = \log_{\text{base}_2} (1/12) (f / (s^{*(m-e)}/240))$$

$$= \log_{\text{base}_2} (1/12) (m / (m-e)). \quad \text{and}$$

$$c_1 = \log_{\text{base}_2} (1/12) (m / (m+e)).$$

It is easy to see that our maximum error will occur at the lowest frequency (small m). At 50 hz (to take an arbitrary lower bound)

$$m' = 240 \cdot 50 / 40 \cdot 10^3$$

$\approx .3$

so

 $c1 = .033780$ $c2 = -.113079$

The worst case frequency error is then about 11 % of a semitone at 50 hz. As mentioned before, the minimum perceivable change of pitch is about 3 hz, which at 50 hz is 6 percent. This means a barely perceptible frequency error at the low end of the sonic range.

The decoding of the instructions for the macro facility is very simple. Basically, the first 4 bits of the sixteen bit op-code specify the type of opcode: see appendix 1 for a list of opcodes and their meanings.

A tag opcode is used only for branch macros: when they are encountered in the normal flow of execution they are treated as a no op. If, in the flow of execution, an unimplemented op-code is detected we simply hang up (a jump .) as an error condition.

These macros are admittedly hard to handle, as they are to

music structure what assembly language is to PL/1. They're saving grace is that they can be decoded in less than 10 usec. Their use should be, eventually, as an object code description of a higher level description. A LOGO (or other host-environment) program could "compile" a suitable user-level representation into this admittedly inhospitable code, thus benefitting from ease of use and run-time efficiency.

As an example, let us say that we wished to vary the volume of a C tone, starting at a loud level and progressively becoming softer and softer, the amplitude decreasing each time by a factor of 2. We could represent that in symbolic macro-code as follows:

```

sin(sinusoid,30000)           ;create sin at sinusoid
voice(c,1,sinusoid)          ;initialize voice 1
                             ;to "sinusoid" at pitch c
setvoice#(1)                  ;only 1 voice active
loop: play(2)                  ;play for 2 seconds
scalearray(sinusoid,sinusoid,.5);scale it by .5
jump loop

```

or if we want channel 1 getting louder and channel 2 getting softer, we

can use the following:

```

sin(sinusoid1,30000)
sin(sinusoid2,1000)           ;2 sinusoids, cause
                               ;we will be changing their
                               ;amplitudes.
voice(c,1,sinusoid1)
voice(c,2,sinusoid2)
setvoice#(2)
loop: play(2)
      scalearray(sinusoid1,sinusoid1,.5)
      scalearray(sinusoid2,sinusoid2,2.)
      jump loop

```

BELLS AND WHISTLES

The system is capable of supporting 8 voices at a sampling frequency of 20 khz. This is assuming that not too much is happening in between samples. By reducing the number of voices, we can do a lot in the extra time. One nice extension is frequency modulation. The applications of f.m. to audio synthesis is reported by [Chowning]. The technique allows a wide variety of timbres by perturbing the frequency of a sinusoid. Chowning reports brass, woodwind, and

percussive tones are obtainable.

Since we have the phase of the voices (in the block of a's, m's, and waveform pointers) and the samples from waveforms (from the last iteration in the output registers) we can perform a transformation from output samples to perturbation of a phase. [Chowning] and [Saunders] have proposed frequency modulation schemes based on sinusoids modulating sinusoids (in the former article) and triangle waves modulating sinusoids (in the latter). In the current system, we would allow an arbitrary waveform modulating an arbitrary waveform, a waveform modulating a waveform modulating a waveform, ... , cross modulation, co-modulation, etc. The only constraint will be computational complexity as concerns time: in the Saunders paper this was not a constraint since the perturbation of the frequency was a precomputed constant. In the current system, it is hoped that simple (ie. not necessarily using multiplies or divides) transformations will allow a sufficient parameter space for interesting tones.

As mentioned previously, the macro code for the system is somewhat unwieldly : the operations involve the physical manipulations that the system allows, and these operations are intentionally primitive. I would imagine a more conceptual approach could be easily implemented, a la [Smoliar]. In his approach each voice is conceptualized as a process that plays a "score" consisting of such constructs as subroutine invocation, inversion and transposition of a sonic sequence (ie. as parameters to a subroutine), iteration, playing a note for a given period of time, allowing tempo and intensity as parameters, etc. Such a system could be constructed in the LOGO language with a modicum of effort, using a sort of "scan-conversion" (for lack of a better term). That is, the sequences would be considered as a parallel representation of events that would be mapped into a linear string of macros for the 2500, eg:

breve c	semibreve c
semibreve	breve
breve d	semibreve d
semibreve	breve

.	.
.	.
.	.
.	.

becomes

```

voice(c, 1, sinusoid)
voice(c, 2, sinusoid)
playm(500)
voice(dont_care, 2, zero)
playm(500)
waitm(500)
voice(d, 1, sinusoid)
voice(d, 2, sinusoid)
playm(500)
voice(dont_care, 2, zero)
playm(500)
waitm(500)

```

.
 .
 .

This process is one of scanning across each of the voices, looking for the next event, and changing any requisite parameters. Some events are necessarily broken up into two or more events (eg. the breve c, breve

d in the previous example), so the process becomes one of looking for crossings of such events with the scan line, thus the term scan conversion.

CONCLUSION

A system that allows arbitrary sequences of sonic events to occur and almost arbitrary waveform manipulation has been presented. The system is hoped to be fleshed out over the summer and whipped into a more usable form by yours truly. It is hoped that the approach taken will allow for great flexibility and expansion in the sonic medium.

APPENDIX 1 : OP CODES

TYPE	OPCODE	MEMBERS	ARGUMENTS
tag	0	tag (used as a label for jumps)	none
creation routines	2	sin, square, saw, impulse, ...	wvfm#, amplitude
4 argument routines	6	voice	m, a, wvfm#, voice#
time sequencing	8	play, playm, wait, waitm	time to be played.
3 argument routines	10	scalearray	from_wvfm#, to_wvfm#, scale
misclarg	12	zero_wvfm, wvfm_lod, branch, setvoice#	wvfm# or # of voices or tag
zeroarg	15	hang, playforever	none

APPENDIX 2 : LISTING OF PROGRAMS

```

(& THIS VERSION OF THE 2500 STUFF TAKES CARE)
(& OF THE STEPPING THRU OF ARRAYS PLAYING)
(& THE CURRENT CONFIGURATION INTERPRETING)
(& OSCILLATOR OP-ODES (SOMEWHAT SIMILAR))
(& (TO DISPLAY CODE) CREATION ROUTINES ETC . #51)
(CALL 74 SAVEREGSAVE)
(CALL 75 SAVEREGSAVE2)
(CALL 76 SAVEREGSAVE3)
(CALL 76 JOT)
(CALL 7777 TOPMEM)
(CALL 70 #WVFM)
(CALL 71 WVFM SIZE)
(CALL 72 DACPRECIS)
(CALL 73 STACKPTR)
(CALL 27 11BUFF)
(CALL 3 DACCONV)
(CALL 0 SCRATCH1)
(CALL 1 SCRATCH2)
(CALL 2 SCRATCH3)
(CALL 3 SCRATCH4)
(CALL 4 SIXTYFOUR)
(CALL 5 #ITERS)
(CALL 6 THIRTYK)
(CALL 7 EIGHTBITS)
START      200      1200      (LOD SCRATCH3)
            201      360      (360)
            202      76271     (PUT SCRATCH3 WVFM SIZE)
            203      1200      (LOD SCRATCH3)
            204      400      (400)
            205      76272     (PUT SCRATCH3 DACPRECIS)
            206      1200      (LOD SCRATCH3)
            207      7777      (TOPMEM & INIT STACK POINTER TO TOP)

(& OF AVAILABLE CORE)
            210      76273     (PUT SCRATCH3 STACKPTR)
            211      41114     (PUSHJ HANG & AFTER WE HAVE)

(& INITIALIZED WAIT FOR)
(& SOMETHING TO HAPPEN)
(&)
(& SCRATCH1:WVFM#)
(& NOTE: SINGLE ARGUMENT FUNCTION AMPLITUDE IS CONSTANT)
SAW         212      40550     (PUSHJ MAPWVFM#)
            213      74271     (GET SCRATCH3 WVFM SIZE)
            214      7202      (SUBI SCRATCH3 SCRATCH3)
            215      4001      (4001)
            216      76223     (PUT SCRATCH3 XR)
            217      1100      (LOD SCRATCH2)
            220      -72460     (-72460 & WE MAKE OUR OWN AMPLITUDE)

(& FOR SAWTOOTH)
            221      1200      (LOD SCRATCH3)
            222      372      (372 & OUR INCREMENT)
SAWLOOP     223      26041     (WRITEI SCRATCH1 SCRATCH2)
            224      4162      (ADD SCRATCH2 SCRATCH3)
            225      163775     (BXCI SAWLOOP)

```

	226	1100	(LOD SCRATCH2)
	227	1	(1)
	230	24001	(WRITE SCRATCH1 SCRATCH2 & DO NOT)
(& FORGET FREQ MULT)			
	231	75600	(POPJ)
(&			
(& SCRATCH1:WVFM#)			
(& SCRATCH2:AMPLITUDE)			
SQUARE	232	40550	(PUSHJ MAPWVFM#)
	233	7101	(SUB1 SCRATCH2 SCRATCH2)
	234	0	(0)
	235	1300	(LOD SCRATCH4)
	236	2	(2 & SWITCH TO MAKE THE LOOP EASIER)
SQUARELOOP1	237	74271	(GET SCRATCH3 WVFM SIZE)
	240	6261	(ARS SCRATCH3 1 & SET UP COUNT)
	241	7202	(SUB1 SCRATCH3 SCRATCH3)
	242	4001	(4001 & XR STUFF)
	243	76223	(PUT SCRATCH3 XR)
SQUARELOOP2	244	26041	(WRITE1 SCRATCH1 SCRATCH2)
	245	163776	(BXC1 SQUARELOOP2)
	246	7101	(SUB1 SCRATCH2 SCRATCH2)
	247	0	(0)
	250	4320	(DEC SCRATCH4)
	251	133765	(BNE SQUARELOOP1)
	252	75600	(POPJ)
(&			
(& SCRATCH1:WVFM#)			
(& SCRATCH2:AMPLITUDE)			
SIN	253	40550	(PUSHJ MAPWVFM#)
	254	74400	(GET SCRATCH5 SCRATCH1 & SAVE IT)
(& FOR OUR INDEX LATER)			
	255	1300	(LOD SCRATCH4)
	256	4001	(4001)
	257	74271	(GET SCRATCH3 WVFM SIZE)
	260	6302	(SUB SCRATCH4 SCRATCH3 & XR)
(& COUNT STUFF)			
	261	76323	(PUT SCRATCH4 XR)
	262	6202	(SUB SCRATCH3 SCRATCH3)
(& INITIALIZE "Y" VARIABLE)			
SINLOOP	263	74302	(GET SCRATCH4 SCRATCH3)
	264	40274	(PUSHJ CMULT & SCALE IT ACCORDING)
(& TO A MAGIC NUMBER)			
	265	6103	(SUB SCRATCH2 SCRATCH4)
	266	74301	(GET SCRATCH4 SCRATCH2)
	267	40274	(PUSHJ CMULT)
	270	4263	(ADD SCRATCH3 SCRATCH4)
	271	26442	(WRITE1 SCRATCH5 SCRATCH3)
	272	163770	(BXC1 SINLOOP)
	273	75600	(POPJ)
(&			
(& SCRATCH4:16 BIT INTEGER)			
(& RETURNS SCRATCH4: [.0216179190*16] BIT INTEGER (APPROX))			
CMULT	274	6000	(SUB SCRATCH1 SCRATCH1)
	275	6363	(ARS SCRATCH4 3)
	276	6363	(ARS SCRATCH4 3)
	277	4063	(ADD SCRATCH1 SCRATCH4)

```

300      6361      (ARS SCRATCH4 1)
301      4063      (ADD SCRATCH1 SCRATCH4)
302      6362      (ARS SCRATCH4 2)
303      4063      (ADD SCRATCH1 SCRATCH4)
304      6362      (ARS SCRATCH4 2)
305      4063      (ADD SCRATCH1 SCRATCH4)
306      6361      (ARS SCRATCH4 1)
307      4063      (ADD SCRATCH1 SCRATCH4)
310      6362      (ARS SCRATCH4 2)
311      4360      (ADD SCRATCH4 SCRATCH1)
312      75600      (POPJ)

(& *****)
(&)
(&)
(& THIS SECTION IS THE DIRTY WORK THAT IS INVOLVED WITH)
(& STEPPING THRU THE WAVEFORMS THE SECTION OF)
(& CODE FROM OSCLOOP TO THE END OF THE LOOP SHOULD)
(& RUN IN ABOUT 4 4 USEC)
(& #ITERS:THE NUMBER OF VOICES CURRENTLY ACTIVE)
(&)
(& *****NOTE***** THIS ROUTINE DOESN (QUOTE T) SAVE REGISTERS)
(&)
(&)
(& *****)
OSCINIT      313      74671      (GET THIRTYK WFMMSIZE)
              314      4610      (ROT THIRTYK 10)
              315      5666      (ADDI THIRTYK THIRTYK)
              316      -1        (-1 & THE PRECEDING GARBAGE)

(& SETS UP A CONSTANT THAT WE)
(& NEED TO SEE IF OUR INDEX WRAPS AROUND)
              317      1400      (LOD SIXTYFOUR)
              320      100       (100 & THIS IS NEEDED TO INC)

(& BITS 6-11 OF THE XR)
              321      1700      (LOD EIGHTBITS)
              322      377       (377 & USED AS A MASK)
              323      74372     (GET DACCONV DACPRECIS)

(& DUMB ABSOLUTE DA"S)
OSCSTART      324      1000      (LOD SCRATCH1)
              325      2040      (2040 & TO MAKE THE XR)

(& POINT TO A"S M"S OFFSETS)
(& AND DAC"S)
              326      76023     (PUT SCRATCH1 XR)
              327      10000     (NOP)
OSCLOOP      330      74012     (GET SCRATCH1 12 & GRAB M)
              331      160000    (BXC I NEXT1)

(& INC OUR SCRATCHPAD POINTER)
NEXT1      332      74112     (GET SCRATCH2 12 & GRAB A)
              333      4160      (ADD SCRATCH2 SCRATCH1)

(& CALCULATE NEW FLOATING ADDRESS)
              334      6263      (ARS SCRATCH3 3)

(& MAKE PREVIOUS SAMPLE A 9-BIT INTEGER)
              335      6263      (ARS SCRATCH3 3)
              336      6261      (ARS SCRATCH3 1)
              337      4263      (ADD SCRATCH3 DACCONV & ME TOO)
              340      76211     (PUT SCRATCH3 11 & WRITE OUT LAST)

(& SAMPLE (SINCE READS))

```

(& (TAKE MULTIPLE CLOCKS WE ARE DOING THE))

(& (PREVIOUS ONE HERE))

	341	74223	(GET SCRATCH3 XR & GRAB XR FOR UPDATING)
	342	4264	(ADD SCRATCH3 SIXTYFOUR & INC BIT 6)
	343	76223	(PUT SCRATCH3 XR & BACK FOR MORE)
	344	16601	(CMP THIRTYK SCRATCH2 & DO WE WRAP)
	345	124001	(BPL NOSUB & NO)
	346	6106	(SUB SCRATCH2 THIRTYK & NOW WE AGAIN POINT)
(& INSIDE WAVEFORM)			
NOSUB	347	76112	(PUT SCRATCH2 12 & SAVE A FOR THE NEXT TIME)
(& AROUND)			
	350	160000	(BXC1 NEXT2 & INC XR)
NEXT2	351	4110	(ROT SCRATCH2 10 & TRUNCATE TO INTEGER ADDRESS)
	352	147	(AND SCRATCH2 EIGHTBITS & CLEAR OUT CRAP)
	353	74212	(GET SCRATCH3 12 & GRAB OFFSET IN MAIN MEM)
	354	4162	(ADD SCRATCH2 SCRATCH3 & GRAB SAMPLE AND SAVE)
(& IT IN)			
(& SCRATCH3 FOR NEXT TIME AROUND)			
	355	20102	(READ SCRATCH2 SCRATCH3)
	356	160000	(BXC1 NEXT3)
NEXT3	357	4525	(DEC #ITERS #ITERS)
	360	133747	(BNE OSCLOOP)
	361	76211	(PUT SCRATCH3 11 & THERE WILL STILL BE ONE)
(& SAMPLE LEFT)			
(& OVER IN SCRATCH3)			
	362	75617	(POPJ1 & BACK WHERE WE CAME FROM)
(&)			
(& SAVES REGISTERS USING STACK POINTER)			
(& (73 OF SCRATCHPAD))			
(& IF OVERFLOW THEN HANG UP)			
SAVEREGS	363	76074	(PUT SCRATCH1 SAVEREGSAVE & OUR OWN SAVE AREA)
	364	74073	(GET SCRATCH1 STACKPTR)
	365	17000	(CMP1 0 SCRATCH1)
	366	7633	(- TOPMEM 144 & STACK OVERFLOW??)
HERE	367	127777	(BPL HERE & HANG UP)
	370	24027	(WRITED SCRATCH1 SCRATCH8)
	371	24026	(WRITED SCRATCH1 SCRATCH7)
	372	24025	(WRITED SCRATCH1 SCRATCH6)
	373	24024	(WRITED SCRATCH1 SCRATCH5)
	374	24023	(WRITED SCRATCH1 SCRATCH4)
	375	24022	(WRITED SCRATCH1 SCRATCH3)
	376	24021	(WRITED SCRATCH1 SCRATCH2)
	377	76175	(PUT SCRATCH2 SAVEREGSAVE2)
	400	74174	(GET SCRATCH2 SAVEREGSAVE)
	401	24021	(WRITED SCRATCH1 SCRATCH2)
	402	76073	(PUT SCRATCH1 STACKPTR)
	403	74175	(GET SCRATCH2 SAVEREGSAVE2)
	404	74074	(GET SCRATCH1 SAVEREGSAVE)
	405	75600	(POPJ)
(&)			
(& COMPLEMENTS ABOVE)			
RESTOREREGS	406	74073	(GET SCRATCH1 STACKPTR)
	407	6040	(INC SCRATCH1 SCRATCH1)
	410	22041	(READ1 SCRATCH1 SCRATCH2)
	411	76174	(PUT SCRATCH2 SAVEREGSAVE)
	412	22041	(READ1 SCRATCH1 SCRATCH2)

	413	22042	(READI SCRATCH1 SCRATCH3)
	414	22043	(READI SCRATCH1 SCRATCH4)
	415	22044	(READI SCRATCH1 SCRATCH5)
	416	22045	(READI SCRATCH1 SCRATCH6)
	417	22046	(READI SCRATCH1 SCRATCH7)
	420	22047	(READI SCRATCH1 SCRATCH8)
	421	4020	(DEC SCRATCH1 SCRATCH1)
	422	76073	(PUT SCRATCH1 STACKPTR)
	423	74074	(GET SCRATCH1 SAVEREGSAVE)
	424	75600	(POPJ)
(&)			
(& THIS IS A DIAGNOSTIC DUMP ROUTINE ENTERABLE ONLY)			
(& THRU SWITCHES)			
DUMP	425	74272	(GET SCRATCH3 DACPRECIS)
	426	6000	(SUB SCRATCH1 SCRATCH1)
	427	1100	(LOO SCRATCH2)
	430	3421	(- 4001 360)
	431	76123	(PUT SCRATCH2 XR)
DUMLOOP	432	22041	(READI SCRATCH1 SCRATCH2)
	433	10000	(NOP)
	434	10000	(NOP)
	435	6163	(ARS SCRATCH2 3)
	436	6163	(ARS SCRATCH2 3)
	437	6161	(ARS SCRATCH2 1)
	440	4162	(ADD SCRATCH2 SCRATCH3)
	441	76121	(PUT SCRATCH2 21)
	442	163767	(BXC1 DUMLOOP)
	443	50425	(JUMP DUMP)
(CALL 4 SCRATCH5)			
(CALL 5 SCRATCH6)			
(CALL 6 SCRATCH7)			
(CALL 7 SCRATCH8)			
(&)			
(& SCRATCH1:16 BIT INTEGER)			
(& SCRATCH2:8 0.8 POSITIVE FIXNUM)			
(& RETURNS SCRATCH1:16*8 0.8 IN 16 BIT FORM)			
SCALENUM	444	4110	(ROT SCRATCH2 10 & SET IT UP LIKE FRACT INT)
	445	74300	(GET SCRATCH4 SCRATCH1)
	446	1200	(LOO SCRATCH3)
	447	3771	(- 4001 10)
	450	76223	(PUT SCRATCH3 XR)
	451	6202	(SUB SCRATCH3 SCRATCH3)
MULLOOP1	452	4101	(ROT SCRATCH2 1 & DO WE HAVE A BIT)
	453	124001	(BPL NOADD1 & NO)
	454	4260	(ADD SCRATCH3 SCRATCH1 & ADD IN INTEGER)
NOADD1	455	4060	(ADD SCRATCH1 SCRATCH1)
	456	163773	(BXC1 MULLOOP1)
	457	4107	(ROT SCRATCH2 7)
	460	1000	(LOO SCRATCH1)
	461	3771	(- 4001 10)
	462	76023	(PUT SCRATCH1 XR)
MULLOOP2	463	6361	(ARS SCRATCH4 1 & SHIFT MULTIPLICAND)
	464	4161	(ADD SCRATCH2 SCRATCH2 & DO WE HAVE A BIT??)
	465	124001	(BPL NOADD2 & NO)
	466	4263	(ADD SCRATCH3 SCRATCH4)
NOADD2	467	163773	(BXC1 MULLOOP2)

```

      470      74002      (GET SCRATCH1 SCRATCH3)
      471      4117      (ROT SCRATCH2 17 & SET THE SCALER BACK UP)
(& FOR CALLER)
      472      75600      (POPJ)
(&)
(& SCRATCH1:IN WVF#)
(& SCRATCH2:OUT WVF#)
(& SCRATCH3:SCALE (8.8 FIXNUM))
SCALEARRAY      473      74501      (GET SCRATCH6 SCRATCH2)
      474      40550      (PUSHJ MAPWVF# & TURN IN WVF# TO LOCATION)
      475      74400      (GET SCRATCH5 SCRATCH1)
      476      74005      (GET SCRATCH1 SCRATCH6)
      477      40550      (PUSHJ MAPWVF#)
      500      74500      (GET SCRATCH6 SCRATCH1)
      501      74102      (GET SCRATCH2 SCRATCH3)
      502      74671      (GET SCRATCH7 WVFMSIZE)
      503      7606      (SUBI SCRATCH7 SCRATCH7)
      504      4001      (4001)
      505      76623      (PUT SCRATCH7 XR)
SCALELOOP      506      22440      (READI SCRATCH5 SCRATCH1)
      507      40444      (PUSHJ SCALENUM)
      510      26540      (WRITEI SCRATCH6 SCRATCH1 & PUT ANSWER IN OUT ARRAY)
      511      163774      (BXCI SCALELOOP)
      512      75600      (POPJ)
(&)
(& SCRATCH1:#ITERATIONS)
(& SCRATCH2:NUMBER OF VOICES)
NSAMPLES      513      76037      (PUT SCRATCH1 SAVE1 & SAVE IT FOR A RAINY DAY)
      514      76136      (PUT SCRATCH2 SAVE2 & SAVE THE NUMBER OF)
(& WAVEFORMS TOO)
      515      74501      (GET #ITERS SCRATCH2)
      516      40313      (PUSHJ OSCINIT)
NSAMLOOP      517      74536      (GET #ITERS SAVE2)
      520      40324      (PUSHJ OSCSTART)
      521      74037      (GET SCRATCH1 SAVE1)
      522      4020      (DEC SCRATCH1 SCRATCH1)
      523      76037      (PUT SCRATCH1 SAVE1)
      524      133772      (BNE NSAMLOOP)
      525      75600      (POPJ)
(&)
(& THIS SETS THE BLOCK OF 3 WORDS DEFINING THE STATE OF
(& THE OUTPUT CHANNEL)
(& SCRATCH1:M)
(& SCRATCH2:A)
(& SCRATCH3:O)
(& SCRATCH4:VOICE NUMBER)
VOICE      526      17003      (CMPI 0 SCRATCH4)
      527      7      (7 & THIS WILL TELL US IF WE (QUOTE RE) TOO BIG)
      530      124001      (BPL VOICEOK)
      531      75600      (POPJ & BAD ARG JUST RETURN)
VOICEOK      532      6404      (SUB SCRATCH5 SCRATCH5)
      533      4463      (ADD SCRATCH5 SCRATCH4)
      534      4363      (ADD SCRATCH4 SCRATCH4 & MULTIPLY BY TWO)
      535      4364      (ADD SCRATCH4 SCRATCH5)
      536      5363      (ADDI SCRATCH4 SCRATCH4)
      537      40      (40 & NOW IT POINTS TO VOICE DESCRIPTOR BLOCK)

```

```

      540      76323      (PUT SCRATCH4 XR & CAUSE WE (QUOTE LL) WANT)
(& TO USE XR AS AN INDEX)
      541      18000      (NOP)
      542      76012      (PUT SCRATCH1 12 & INDIRECT THRU 0-5 OF XR)
(& THIS IS M)
      543      160000     (BXCI VNEXT1 & INC XR)
VNEXT1      544      76112      (PUT SCRATCH2 12 & THIS IS A)
      545      160000     (BXCI VNEXT2)
VNEXT2      546      76212      (PUT SCRATCH3 12 & THIS IS O)
      547      75600      (POPJ & ALL DONE)
(&)
(& SCRATCH1:WVFM#)
(& RETURNS SCRATCH1:WVFM LOCATION)
MAPWVFM#    550      76174     (PUT SCRATCH2 SAVEREGSAVE)
      551      6101      (SUB SCRATCH2 SCRATCH2)
      552      4160      (ADD SCRATCH2 SCRATCH1)
      553      4060      (ADD SCRATCH1 SCRATCH1)
      554      4060      (ADD SCRATCH1 SCRATCH1)
      555      4060      (ADD SCRATCH1 SCRATCH1)
      556      4060      (ADD SCRATCH1 SCRATCH1)
      557      4160      (ADD SCRATCH2 SCRATCH1)
      560      4060      (ADD SCRATCH1 SCRATCH1)
      561      4160      (ADD SCRATCH2 SCRATCH1)
      562      4060      (ADD SCRATCH1 SCRATCH1)
      563      4160      (ADD SCRATCH2 SCRATCH1)
      564      4060      (ADD SCRATCH1 SCRATCH1)
      565      4061      (ADD SCRATCH1 SCRATCH2)
      566      74174     (GET SCRATCH2 SAVEREGSAVE)
      567      75600      (POPJ)
(CALL 77 PC)
      570      1000      (LOD SCRATCH1)
      571      5670      (5670 & INITIALIZE PC)
(&)
(& THIS DOES THE INTERPRETATION OF THE CODE TO DRIVE THE)
(& OSCILLATORS)
INTERPRETLOOP 572      41212     (PUSHJ FETCH & RETURNS (PC) IN SCRATCH 10)
      573      74707     (GET SCRATCH8 SCRATCH8 & SET BUS BITS)
      574      72000     (DIS BUS 17 & WHICH OPCODE?)
      575      50572     (JUMP INTERPRETLOOP & THIS IS A TAG OPCODE)
(& TREAT AS NOP)
HERE1      576      50576     (JUMP HERE1 & HANG UNIMPLIMENTED OP CODE)
      577      50615     (JUMP CREATION & CREATE WVFM)
HERE2      600      50600     (JUMP HERE2 & UNIMPLIMENTED)
HERE3      601      50601     (JUMP HERE3 & DITTO)
HERE4      602      50602     (JUMP HERE4)
      603      50767     (JUMP 4ARG)
HERE5      604      50604     (JUMP HERES)
      605      50713     (JUMP TIMESEQUENCE)
HERE6      606      50606     (JUMP HERES)
      607      50754     (JUMP 3ARG)
HERE7      610      50610     (JUMP HERE7)
      611      50644     (JUMP MISCIARG)
HERE8      612      50612     (JUMP HERE8)
HERE9      613      50613     (JUMP HERE9)
      614      50742     (JUMP ZEROARG)
(&)

```

```

(& WAVEFORM CREATION ROUTINES COME HERE)
CREATION      615      74207      (GET SCRATCH3 SCRATCH8 & SAVE SCRATCH8 FROM)
               616      41212      (PUSHJ FETCH & FETCH)
               617      74007      (GET SCRATCH1 SCRATCH8 & THIS IS WVFH#)
               620      41212      (PUSHJ FETCH)
               621      74107      (GET SCRATCH2 SCRATCH8 & THIS IS AMPLITUDE OR INC)
               622      17002      (CMPI 0 SCRATCH3 & SAW??)
               623      100000      (100000)
               624      130002      (BNE N01)
               625      40212      (PUSHJ SAW)
               626      50572      (JUMP INTERPRETLOOP & AROUND AND AROUND WE GO)
N01           627      17002      (CMPI 0 SCRATCH3 & SIN??)
               630      130002      (BNE N02)
               631      40253      (PUSHJ SIN)
               632      50572      (JUMP INTERPRETLOOP)
N02           633      17002      (CMPI 0 SCRATCH3 & RANDOM??)
               634      100002      (100002)
               635      40232      (PUSHJ SQUARE)
               636      50572      (JUMP INTERPRETLOOP)
N03           637      17002      (CMPI 0 SCRATCH3 & SQUARE??)
               640      100003      (100003)
N04           641      133777      (BNE N04)
RANDOM        642      40642      (PUSHJ RANDOM)
               643      50572      (JUMP INTERPRETLOOP)
MISC1ARG     644      74204      (GET SCRATCH3 SCRATCH5)
               645      41212      (PUSHJ FETCH)
               646      74007      (GET SCRATCH1 SCRATCH8)
               647      17002      (CMPI 0 SCRATCH3 & ZERO??)
               650      140000      (140000)
               651      130002      (BNE N06)
               652      41005      (PUSHJ ZEROWVFH)
               653      50572      (JUMP INTERPRETLOOP)
N06           654      17002      (CMPI 0 SCRATCH3 & WAVEFORM LOAD??)
               655      140001      (140001)
               656      130002      (BNE N07)
               657      41114      (PUSHJ HANG)
               660      50572      (JUMP INTERPRETLOOP)
N07           661      17002      (CMPI 0 SCRATCH3)
               662      140002      (140002)
               663      130022      (BNE N08)
               664      3000      (IORI SCRATCH1 SCRATCH1 & THIS IS A BRANCH)

(& TYPE LOOK FOR)
               665      170000      (170000 & LABEL AND SET PC TO IT)
               666      1100      (LOD SCRATCH2 & BOTTOM OF OSC OPCODES)
               667      5670      (5670)
               670      1200      (LOD SCRATCH3)
               671      7633      (7633 & TO OF OSCCODE)
BRANCHLOOP   672      22143      (READI SCRATCH2 SCRATCH4)
               673      10000      (NOP)
               674      10000      (NOP)
               675      10000      (NOP)
               676      16300      (CMP SCRATCH4 SCRATCH1)
               677      134003      (BEQ MATCH)
               700      16102      (CMP SCRATCH2 SCRATCH3 & ENTRY OUT OF BOUNDS)
               701      133770      (BNE BRANCHLOOP)
               702      50572      (JUMP INTERPRETLOOP & NO MATCH GO BACK AND)

```

(& INTERPRET)			
MATCH	703	4121	(DEC SCRATCH2 SCRATCH2 & POINT TO MATCHING)
(& SAMPLE)			
	704	76177	(PUT SCRATCH2 PC)
	705	50572	(JUMP INTERPRETLOOP & NOW PC POINTS TO NEW)
(& LOCATION)			
NO8	706	17002	(CMPI 0 SCRATCH3)
	707	140003	(140003)
NO314	710	133777	(BNE NO314 & UNIMPLEMENTED OP HANG)
	711	76270	(PUT SCRATCH3 #VOICES)
	712	50572	(JUMP INTERPRETLOOP)
TIMESEQUENCE	713	74207	(GET SCRATCH3 SCRATCH8)
	714	41212	(PUSHJ FETCH)
	715	74007	(GET SCRATCH1 SCRATCH8 & THIS IS THE SINGLE)
(& ARG OF TIME)			
	716	17002	(CMPI 0 SCRATCH3 & SEQUENCING ROUTINES)
	717	10000	(10000 & IS THIS A PLAY??)
	720	130002	(BNE NO9)
	721	41161	(PUSHJ PLAY)
	722	50572	(JUMP INTERPRETLOOP)
NO9	723	17002	(CMPI 0 SCRATCH3)
	724	10001	(10001)
	725	130002	(BNE NO10)
	726	41037	(PUSHJ WAIT)
	727	50572	(JUMP INTERPRETLOOP)
NO10	730	17002	(CMPI 0 SCRATCH3)
	731	10002	(10002)
	732	130002	(BNE NO11)
	733	41165	(PUSHJ PLAYM)
	734	50572	(JUMP INTERPRETLOOP)
NO11	735	17002	(CMPI 0 SCRATCH3)
	736	10003	(10003 & IS THIS A WAIT??)
NO12	737	133777	(BNE NO12 & HANG)
	740	41043	(PUSHJ WAITM)
	741	50572	(JUMP INTERPRETLOOP)
ZEROARG	742	17007	(CMPI 0 SCRATCH8)
	743	0	(0)
	744	130002	(BNE NO13)
	745	41114	(PUSHJ HANG & HANG JUST WAITS FOR AN 11 INTERRUPT)
	746	50572	(JUMP INTERPRETLOOP)
NO13	747	17007	(CMPI 0 SCRATCH8)
	750	1	(1)
NO14	751	133777	(BNE NO14 & HANG)
	752	41026	(PUSHJ PLAYFOREVER)
	753	50572	(JUMP INTERPRETLOOP)
3ARG	754	74307	(GET SCRATCH4 SCRATCH8)
	755	41212	(PUSHJ FETCH)
	756	74007	(GET SCRATCH1 SCRATCH8 & ARG1)
	757	41212	(PUSHJ FETCH)
	760	74107	(GET SCRATCH2 SCRATCH8 & ARG2)
	761	41212	(PUSHJ FETCH)
	762	74207	(GET SCRATCH3 SCRATCH8 & ARG3)
	763	17003	(CMPI 0 SCRATCH4)
	764	50000	(50000)
NO15	765	133777	(BNE NO15 & HANG)
	766	40473	(PUSHJ SCALEARRAY & THE ONLY 3 ARG ROUTINE FOR NOW)

4ARG	767	74487	(GET SCRATCH5 SCRATCH8 & SAVE OP)
	770	41212	(PUSHJ FETCH)
	771	74087	(GET SCRATCH1 SCRATCH8 & ARG1)
	772	41212	(PUSHJ FETCH)
	773	74107	(GET SCRATCH2 SCRATCH8 & ARG2)
	774	41212	(PUSHJ FETCH)
	775	74207	(GET SCRATCH3 SCRATCH8)
	776	41212	(PUSHJ FETCH)
	777	74307	(GET SCRATCH4 SCRATCH8)
	1000	17004	(CMPI 0 SCRATCH5)
	1001	120000	(120000)
NO16	1002	133777	(BNE NO16)
	1003	40526	(PUSHJ VOICE)
	1004	50572	(JUMP INTERPRETLOOP)
(&)			
(& SCRATCH1:WVFM#)			
ZEROWVFM	1005	40550	(PUSHJ MAPWVFM#)
	1006	74171	(GET SCRATCH2 WVFM SIZE)
	1007	7101	(SUBI SCRATCH2 SCRATCH2)
	1010	4001	(4001)
	1011	76123	(PUT SCRATCH2 XR)
	1012	6101	(SUB SCRATCH2 SCRATCH2)
ZEROLOOP	1013	26041	(WRITEI SCRATCH1 SCRATCH2)
	1014	163776	(BXCI ZEROLOOP)
	1015	75600	(POPJ & SHOULD BE WITH INTS ENABLED)
(&)			
(& SCRATCH1:WVFM#)			
IMPULSE	1016	41005	(PUSHJ ZEROWVFM)
	1017	1100	(LOD SCRATCH2)
((BYTE OVERLAP WARNING ONLY) (-4001 24))			
	1020	-3755	(-4001 24)
	1021	76123	(PUT SCRATCH2 XR)
	1022	76400	(76400)
IMPLOOP	1023	26041	(WRITEI SCRATCH1 SCRATCH2)
	1024	163776	(BXCI IMPLOOP)
	1025	75600	(POPJ & SOMETIME ENABLE INTERRUPTS)
(&)			
(& NEVER RETURN BUT CHECK FOR ELEVEN INTERRUPTS)			
PLAYFOREVER	1026	74570	(GET #ITERS #WVMS)
	1027	40313	(PUSHJ OSCINIT)
PLAYFLOOP	1030	74570	(GET #ITERS #WVMS)
	1031	40324	(PUSHJ OSCSTART)
	1032	72320	(DIS BUS 2 & 11 INTERRUPTS)
	1033	51030	(JUMP PLAYFLOOP)
	1034	10000	(NOP)
	1035	41117	(PUSHJ SERVE11)
	1036	75600	(POPJ)
(&)			
(& SCRATCH1:#SECONDS)			
WAIT	1037	1100	(LOD SCRATCH2)
	1040	500	(500 & THIS IS DUE TO 2 MSEC CLOCK)
	1041	41056	(PUSHJ POSMULTIPLY)
	1042	51045	(JUMP WAITADJUST)
(&)			
(& SCRATCH1:#MSECS)			
WAITM	1043	6061	(ARS SCRATCH1 1)

	1044	6101	(SUB SCRATCH2 SCRATCH2)
WAITADJUST	1045	1208	(LOD SCRATCH3)
	1046	1	(1 & OUR DECCER)
WAITLOOP	1047	73340	(DIS INTS 1 & 2 KHZ WAIT)
	1050	51047	(JUMP WAITLOOP & WAIT)
	1051	6002	(SUB SCRATCH1 SCRATCH3)
	1052	6023	(6020 SCRATCH2 SCRATCH3)
	1053	2001	(IOR SCRATCH1 SCRATCH2)
	1054	133772	(BNE WAITLOOP)
	1055	75600	(POPJ & SHOULD BE 1)
(&)			
(& SCRATCH1:NUMBER)			
(& SCRATCH2:POS NUMBER)			
(& RETURNS SCRATCH1:RESULT (LOW))			
(& RETURNS SCRATCH2:RESULT (HIGH))			
POSMULTIPLY	1056	76276	(PUT 2 JOT)
	1057	1208	(LOD 2)
	1060	-17	(-17)
	1061	76223	(PUT 2 XR)
	1062	10000	(NOP 0 & IF NUMBER IN ZERO IS NEG)
	1063	124007	(BPL BOTHPOS)
	1064	41070	(PUSHJ CMPL0 & COMPLEMENT IT)
	1065	41073	(PUSHJ BOTHPOS & MULTIPLY)
	1066	7101	(SUBI 1 1 & COMPLEMENT THE SHORT ANSWER)
	1067	0	(0)
CMPL0	1070	7000	(SUBI 0 0 & AND COMPLEMENT ANSWER)
	1071	0	(0)
	1072	75600	(POPJ)
BOTHPOS	1073	6202	(SUB 2 2 & RESULT WILL APPEAR HERE)
POSMULTLOOP	1074	4262	(ADD 2 2 & CLEARS THE CARRY)
	1075	4141	(XADD 1 1 & REG ONE MUST BE POSITIVE)
	1076	34260	(IFC ADD 2 0 & REG 0 CAN BE EITHER)
	1077	36140	(IFC INC 1 & TO PUT HIGH BITS IN 1)
	1100	167773	(BXSI POSMULTLOOP)
	1101	4206	(ROT 2 6 & GET NUMBER DIVIDED BY 100 IN 0)
	1102	1042	(ANDI 0 2 & TEN BITS COME FROM THIS WORD)
	1103	1777	(1777)
	1104	1141	(ANDI 1 1 & AND SIX FROM THIS ONE)
	1105	77	(77)
	1106	4106	(ROT 1 6)
	1107	4061	(ADD 0 1)
	1110	1140	(ANDI 1 0 & ONLY 11 BITS FOR MOVE)
	1111	777	(777 & DO IT HERE SO THE SIGN IS RIGHT)
	1112	74276	(GET 2 JOT)
	1113	75600	(POPJ)
(&)			
(& WAITS AROUND FOR 11 (AND ONLY 11) INTERRUPT)			
BODEOH	1114	73320	(DIS INTS 2)
	1115	51114	(JUMP BODEOH)
	1116	10000	(NOP)
(& USUALLY ENTERED VIA POPJ1 AT END OF OSCSTART)			
SERVE11	1117	41143	(PUSHJ GETWORD)
	1120	74200	(GET SCRATCH3 SCRATCH1)
	1121	1100	(LOD SCRATCH2)
	1122	7776	(- TOPMEN 1)
	1123	76177	(PUT SCRATCH2 PC)

SERVELOOP	1124	6707	(SUB SCRATCH8 SCRATCH8)
	1125	41143	(PUSHJ GETWORD)
	1126	74780	(GET SCRATCH8 SCRATCH1)
	1127	41221	(PUSHJ STORE)
	1130	4222	(DEC SCRATCH3 SCRATCH3)
	1131	133773	(BNE SERVELOOP)
	1132	1780	(LOD SCRATCH8)
	1133	140002	(140002 & THIS IS A BRANCH TO THE BEGINNING OF THE MACRO)
	1134	41221	(PUSHJ STORE & STORE JUMP)
	1135	6707	(SUB SCRATCH8 SCRATCH8)
	1136	41221	(PUSHJ STORE & STORE TAG (ARG FOR JUMP))
	1137	1000	(LOD SCRATCH1)
	1140	7777	(TOPMEM)
	1141	76077	(PUT SCRATCH1 PC)
GETWORD	1142	50572	(JUMP INTERPRETLOOP)
	1143	73320	(DIS INTS 2 & BYTE READY?)
	1144	51143	(JUMP GETWORD & NO)
	1145	10000	(NOP)
	1146	74127	(GET SCRATCH2 11BUFF)
	1147	1141	(AND1 SCRATCH2 SCRATCH2)
GETWAIT	1150	377	(377)
	1151	73320	(DIS INTS 2)
	1152	51151	(JUMP GETWAIT)
	1153	10000	(NOP)
	1154	74027	(GET SCRATCH1 11BUFF)
	1155	1040	(AND1 SCRATCH1 SCRATCH1)
	1156	177400	(177400)
	1157	2001	(IOR SCRATCH1 SCRATCH2)
	1160	75600	(POPJ)
(CALL 70 #VOICES)			
(CALL 37 SAVE1)			
(CALL 36 SAVE2)			
(CALL 35 SAVE3)			
(&)			
(& SCRATCH1:#SECS)			
(& #WVMS:HOW MANY ACTIVE VOICES)			
PLAY	1161	1100	(LOD SCRATCH2)
	1162	6	(6 & THIS FUNNY CONSTANT TAKES)
	1163	41056	(PUSHJ POSMULTIPLY & CARE OF TIME SPENT IN INNER LOOP)
	1164	51170	(JUMP PROJJUSTED)
(&)			
(& ALTERNATE ENTRY POINT MSECS INSTEAD OF SECS)			
PLAYM	1165	6063	(ARS SCRATCH1 3)
	1166	6062	(ARS SCRATCH1 2 & DIVIDE BY TIME IN INNER LOOP)
	1167	6101	(SUB SCRATCH2 SCRATCH2)
	1170	1200	(LOD SCRATCH3)
PROJJUSTED	1171	1	(1 & USED AS A DECCER)
	1172	76037	(PUT SCRATCH1 SAVE1)
	1173	76136	(PUT SCRATCH2 SAVE2)
	1174	76235	(PUT SCRATCH3 SAVE3)
PLAYLOOP	1175	40313	(PUSHJ OSCINIT)
	1176	74570	(GET #ITERS #VOICES)
	1177	40324	(PUSHJ OSCSTART)
	1200	74037	(GET SCRATCH1 SAVE1)
	1201	74136	(GET SCRATCH2 SAVE2)
	1202	74235	(GET SCRATCH3 SAVE3)

	1203	6002	(SUB SCRATCH1 SCRATCH3 & DEC LOW ORDER)
	1204	76037	(PUT SCRATCH1 SAVE1)
	1205	76136	(PUT SCRATCH2 SAVE2)
	1206	76235	(PUT SCRATCH3 SAVE3)
	1207	12001	(T IOR SCRATCH1 SCRATCH2 & BOTH ZERO?)
	1210	133765	(BNE PLAYLOOP)
	1211	75600	(POPJ & SHOULD BE 1)
(CALL 7777 MAXMEM)			
(&)			
(& RETURNS SCRATCH7:BOMBED)			
(& RETURNS SCRATCH8: (PC))			
(& RETURNS PC:PC+1)			
FETCH	1212	74677	(GET SCRATCH7 PC)
	1213	22647	(READI SCRATCH7 SCRATCH8)
	1214	17006	(CMP1 0 SCRATCH7)
	1215	7634	(- MAXMEM 143 & MEMORY OVERFLOW?)
PCOVF	1216	137777	(BEQ PCOVF)
	1217	76677	(PUT SCRATCH7 PC)
	1220	75600	(POPJ)
(&)			
(& SCRATCH8:VALUE TO BE WRITTEN)			
(& RETURNS SCRATCH7:BOMBED)			
(& RETURNS PC:PC+1)			
STORE	1221	74677	(GET SCRATCH7 PC)
	1222	26647	(WRITEI SCRATCH7 SCRATCH8)
	1223	17006	(CMP1 0 SCRATCH7)
	1224	7634	(- MAXMEM 143 & MEMORY OVERFLOW??)
	1225	134002	(BEQ PCOVFSTORE)
	1226	76677	(PUT SCRATCH7 PC)
	1227	75600	(POPJ)
PCOVFSTORE	1230	1600	(LOD SCRATCH7)
	1231	5670	(5670 & NOTE WE JUST WRAP AROUND ON A PC OVERFLOW)
	1232	76677	(PUT SCRATCH7 PC)
	1233	75600	(POPJ)

REFERENCES

John Backus, The Acoustical Foundations of Music,
(W. W. Norton N.Y., 1969)

John M. Chowning, "Synthesis of Complex Audio Spectrum
via Frequency Modulation", Journal of Audio Engineers Society
vol 21, no 7 pp 526-534 (1973)

Steve Saunders, "FM The Easy Way", Carnegie Mellon Univer-
sity (1974)

Steven W. Smoliar, "A Parallel Processing Model of Music
Structure", Project Mac TR91 (1971)