STAN-M-7

REAL TIME INTERACTIVE COMPUTER MUSIC SYNTHESIS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

by

F. Richard Moore

September, 1977

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

James B. Angell

(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

(Music)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

M V Mathews

(Bell Laboratories)

Approved for the University Committee on Graduate Studies:

(Dean of Graduate Studies)

PREFACE AND ACKNOWLEDGEMENTS

Art is, by definition, artifice. A piano is as much a machine as an automobile except, of course, for certain differences in their emissions. Even the sound of a piano is the product of human imagination; it does not imitate anything that occurs naturally, or the sound of other inventions. With respect to modern history, the piano is a logical consequence of a combination of the harpsichord with hammer-string instruments such as the clavichord and the cimbalom. Early pianos did not sound like modern ones, though by the late nineteenth century the piano had reached modern proportions both physically and sonically, as had most of the "traditional" instruments. Only recently has the means of music begun to change once again. The new instruments are electronic, and they are as interesting for the questions they raise as for the sounds they produce. It is now possible to "design" sound itself, and we are faced with the problem of resolving such questions as what makes a sound musically interesting, or how we distinguish the sound of a clarinet from that of a loudspeaker. Computers act as tools for investigating such questions in ways never before possible, because they give us the means to create virtually any sound that we can understand and describe accurately. But that understanding is the crux.

Since computer music is both a new and a highly interdisciplinary field, it is difficult to describe accurately to non-practitioners. We will often find ourselves having to take into account, simultaneously, such diverse considerations as the mathematics of digital signal processing, the psychology of hearing, the science of computers, the rigors of musical composition, and the mysteries of aesthetics. We often have to leave behind what can be comfortably understood with little effort, perhaps too far sometimes. With this in mind I have attempted to use the simplest possible descriptions consistent with accuracy, rather than, perhaps, the fullest descriptions; in this way the content will hopefully be as accessible as possible to non-specialist readers from the various constituent fields of computer music.

So many years have gone into the development of the ideas presented here that it seems impossible to acknowledge all of the sources of support and encouragement which I have encountered. I would especially like to thank John Chowning and Loren Rush for their companionship, and their leadership in creating the stimulating environment embodied in the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford. Countless conversations with Andy Moorer, John Grey, and Leland Smith have provided a rich source of technical and philosophical insight. Ted Panofsky and Jeff Rubin of the Stanford Artificial Intelligence Project have been invaluable sources of engineering information and aid. Interested fellow students who have contributed their time and energies have included Doug Wilson, Collin Park, Christine Shadle, and Barbara Hoey. Construction of the music synthesizer was speeded by the joint efforts of Michael McNabb, Patte Wood, Paul Winecke, Kip Sheeline, and Rick Taube. Julius Smith was the first person besides myself to build a synthesizer module, and has made many contributions to the design of the amplitude envelope generator module. Robert Poor and Ken Shoemake have been instrumental in realizing the connection between the synthesizer and the computer with its associated software. Special thanks go to Christine Peutsch for correcting my misspellings and reuniting my infinitives. Albert Cohen of the Department of Music provided many useful criticisms of the sections of this thesis dealing with music history. James B. Angell of the Electrical Engineering Department has provided not only the support and encouragement of a good thesis advisor, but invaluable insights into the process of achieving as well. Finally, I would like to thank Max V. Mathews of Bell Telephone Laboratories, not only for his encouragement over the past many years, not only for having started the computer music field in the first place, not only for demonstrating to me how to get things done, but for his continued friendship as well.

F. Richard Moore
September 1977
Stanford, California

- TABLE OF CONTENTS -

- LIST OF TABLES -

• LIST OF FIGURES •

"There is no excellent beauty that hath not some strangeness in the proportion."

CHAPTER 1 - INTRODUCTION

Music is "the science or art of ordering tones or sounds in succession, in combination, and in temporal relationships to produce a composition having unity and continuity." [Webster's New Collegiate Dictionary, 1973]. Therefore, to produce music, it is necessary to have a) a means of generating sounds, b) a means of controlling the synchrony and succession of the generated sounds, and c) sufficient insight into human psychology to understand what is meant by "unity and continuity" in a musical composition.

This presentation is concerned with methods of satisfying the first two requirements, the third is left as an exercise to musical investigations. The sound generation process has been accomplished traditionally with such materials as wood, air columns, and stretched strings, which can be made to vibrate in the audible range when excited appropriately by striking, plucking, bowing, or with rushing air. Extensive experimentation has resulted in a variety of sounds, obtained from different materials, methods of excitation, and various means of amplifying and modifying the sounds produced by the vibrating substance. We refer to the overall quality of sound as its *timbre*, which concept may be broken down into the following non-independent attributes:

PITCH, which is the subjective impression of the placement of a musical tone within the audible range of frequencies. Generally, tones which result from rapid vibrations have a higher pitch than tones which result from slower ones. Pitch is often taken to be a subjective attribute of a periodic vibration which is proportional to the logarithm of its perceived fundamental frequency. It should be noted, however, that pitch is also affected by other attributes of a sound than its frequency, such as its loudness in some cases (e.g., when the tone is roughly sinusoidal in character), or when the distance between the tonal source and the listener is changing (Doppler shift).

LOUDNESS, which is the subjective impression of the strength, or amplitude, of a sound. Loudness is clearly affected by the particular frequency components of a sound, since the ear is not equally sensitive to all frequencies in the audible range, and by the distance between the sound source and the listener. Loudness can also be affected by the duration of a steady tone, for example, due to the phenomena of auditory fatigue and forward and backward masking.

TONE QUALITY — Nearly every sound which occurs in nature, including musical sounds, is composed of a combination of vibrations at several frequencies, amplitudes, and phases, all of which change in a characteristic way throughout its duration. It is this characteristic *temporal evolution* of the spectral content of a sound which is responsible for its overall subjective tonal quality (including, really, its pitch and loudness). Recent investigations have focused precisely on this evolutionary microstructure of musical sounds, since it has been found that the classical model of "relative harmonic strength" (due to Helmholtz) is an inadequate description of tone quality.

SPATIAL RELATIONSHIPS, which refer to the perceived directionality, distance, and trajectory of a sound source in the "acoustic space" surrounding the listener. The human auditory mechanism is constructed in such a way as to be sensitive to these aspects of a sound (probably because it is necessary to know whether the dinosaur or automobile one is hearing poses any danger or not). The radiation pattern of instruments, acoustic enclosures, and relative position of instrument and listener all play a part in the perception of musical sound.

A Brief Review of the Development of Electronic Music

The evolution of music since the sixteenth century has generally been towards controlling each of these subjective aspects of sound more precisely, and more extensively, thereby enriching the musical language with new possible utterances. It is not that composers of earlier times were unaware of most or all of the subjective properties of sound, but simply that they lacked the technological tools to control them in a very precise way. New technological developments of the past, however, often found their way into musical instrumentation rather quickly. Thus the pianoforte, with its greater range of controllable loudnesses, found favor among musicians over the earlier keyboard instruments (it is interesting to note that the earlier instruments were not *replaced* by the piano, since music written for the harpsichord still sounds most convincing when played on the harpsichord, etc.). New instruments such as the Wagner tuba extended the useful pitch range of the brasses, and organs with ever-expanded timbral and spatial capabilities were constructed. The nineteenth century was a time of great technological innovation, and musical language expanded along with the capabilities of new instruments. A kind of practical limit seemed to be achieved in the monster-music productions around the turn of the twentieth century (Stravinsky's *Le Sacre du Printemps* calls for a variegated orchestra of over 120 musicians, the works of Mahler, Wagner and Strauss reached gargantuan proportions, both in physical size and length), and a new interest in more practicable works developed, with great attention paid to the details of structure and organization, perhaps more than ever before (Boulez).

Although several new musical instruments were developed during the past 100 years, only a few have found general acceptance (such as Adolph Sax's contribution to the woodwinds), and even then more by so-called "pop" musicians than by "serious" musicians. The schism in music between "pop" and "serious" modes of expression was exacerbated by the fact that while the "pop" musicians were searching out new means for traditional kinds of expressions, the "serious" composers were creating new kinds of expressions — utterances based on the ability to differentiate among phonemes in a new, or at least radically changed, musical language (Webern). Inevitably, the music of the post-Schönberg era became intelligible to a declining portion of the population, and there ensued a possibility that "serious" music might be in danger of extinction, as with any other highly specialized field: concert music was often as unintelligible to most concert-goers as was Latin to congregations in the Renaissance.

The development of mass communication technology around the middle part of this century allowed many people to become familiar with the new sonic art/science in the form of recordings audited through a device capable of producing an enormous variety of sounds: the loudspeaker. At first, the electronic media were viewed simply as a means of dispersing information such as news and music to a wide audience who could choose from an available program menu. But soon thereafter some musicians took note of the new medium as a potential boon to the sonic art/science, and so-called *electronic music* was born shortly after the advance of the tape and wire recording techniques of the late 1940's. Proponents of *musique concrète* in France found that the requirements of a musical language could be satisfied through manipulation of tape recordings of natural sounds — the pitch, loudness, spectrum, and spatial aspects of a bird song could be manipulated by varying the tape speed, volume level, filtering, and loudspeaker placement of its tape recording. Synchronization and succession of the obtained sounds were manipulated by the painstaking processes of tape splicing and mixing. And though the techniques were technologically primitive by today's standards, compositions of great unity and continuity (Varèse: *Poème électronique*) were achieved through the application of sufficient patience. In Germany, a school of "pure" electronic music dispensed with the microphone altogether (analogous to the idea of producing photographic images without a camera) and used a more analytic approach: it was known that music could theoretically be constructed out of electronically generated "elemental" sounds, such as pure sinusoidal oscillations, and "white noise," which contains all audible frequencies in equal proportions. But even though the sound source was fundamentally different, the same picayune procedures were necessary to control a composition: dozens, hundreds, or thousands of bits of magnetic tape had to be spliced together in order to create a satisfyingly complex whole.

Regardless of the means, an essential result of the early electronic music attempts was the development of a new point of view regarding the loudspeaker: it could be treated not only as a *re*producer of musical sounds, but as the original source, as a new kind of generalized musical instrument. And it became obvious that he who controlled the loudspeaker had access to a vast universe of sonic possibilities, which, happily, included many of the traditional ones as well. So the problem of how to control musical sound became a question of how to control the loudspeaker. Objections were (and are still!) raised to this approach to music, often based on the fear of "replacement" of something more desirable with something less desirable, even though this seems both historically and rationally improbable. More important seems to be the new potential for musical expression and understanding, and the opportunity to "humanize technology," rather than "dehumanize music."

A significant attempt to improve the means for controlling the loudspeaker came in the form of the analog music synthesizer (e.g., the BUCHLA, the MOOG — named after their designers -- and the ARP, the SYNTHI, etc.). These ingenious musical instruments are based on the concept of voltage control of electronic devices, such as oscillators, filters, and amplifiers. Control voltages can be generated by musical keyboards, knobs, switches and dials, or by the oscillators and amplifiers themselves in some cases, so that it is possible to "play" these electronic devices in real time like any other musical instrument. It is also possible to achieve a kind of "programmed" control of musical events through the use of sequencers (a discrete voltage stepping circuit), and other techniques borrowed from analog computer technology. The popular success of analog synthesizers is well-known — Walter Carlos' *Switched-On Bach* sold more copies than any other classical disc in recording history. What is perhaps less well known is their limitations.

A fundamental limitation of analog music synthesizers is accuracy. For example, the human ear can detect frequency differences of about 5 parts per 1000 when the frequency of a pure tone is slowly varied about an average value [Zwicker, Flottorp, and Stevens, 1957], and sudden changes in frequency as much as 30 times smaller than this can be detected [Rakowski, 1971]. If two complex tones are played simultaneously and electrically mixed together (i.e., if their waveforms are added in a reverberationless environment) then the slightest mistuning gives rise to audible beats. Also, to have a voltage-controlled oscillator with a musically useful frequency range such as the range of fundamental frequencies of a piano, both the signals and the responses to them have to be accurate to within 0.05% or so over a range of more than 7 octaves. Such accuracy is possible with analog circuits, but not at the cost-effective level for musical instruments. Another difficulty is stability. Recent voltage controlled oscillators are very well regulated by electronic instrumentation standards, but still the ear is quite sensitive to the discrepancies caused by frequency drift, for example. These accuracy and stability constraints also make it very difficult to repeat anything exactly, which for a performance instrument is perhaps less important than being out of tune, but the creation of electronic music on tape makes repeatability highly desirable, as does the study of sound qualities.

These limitations have a strong effect on the musical applications of analog music synthesizers. As a real time performance device, it is fairly difficult to manage due to the problem of interconnecting the various voltage controlled modules so as to achieve different sound qualities, called "patching" — analogous to setting the registration of stops on an organ, but with infinitely many more stops. Clever crossbar switching techniques have been developed which both eliminate patch cords and make it fairly easy to change from one patch to another, but even with practice it could take 5 to 10 seconds to repatch a moderate-sized synthesizer. This is often rather inconvenient, musically.

Another interesting problem is the one of decoding the information generated by a piano-like keyboard. Synthesizer patches usually tend to simulate a small collection of different sounds which are available at any one time. To control this small collection of "instruments" with a keyboard implies the ability to associate each key being depressed with each available instrument. This problem has been attacked through the use of multiple keyboards (again analogous to organ technique), a limited and expensive solution. "Polyphonic" keyboards have been built, and as long as they are used to control more than one voice of the same tonal quality (as on the organ) they are fine. But consider even two musical voices of differing tone qualities connected to a single keyboard. If a single note is depressed, should one or the other or both voices sound? This problem is somewhat alleviated with real time computer processing of the keyboard information, but no general solution has been found to date. It seems very likely that new types of real time performance devices will become desirable to take full advantage of the synthesizer's potential as a musical instrument. But it would be of questionable value to sever the connection with present performance technique — thousands of musicians are already quite adept at the piano-like keyboard, for example. Many contemporary musicians are leery of the prejudices associated with the keyboard, fearing that the ingrained patterns every keyboard performer has so painstakingly acquired will limit his imagination to *clichés* which he has already learned. It seems fairly clear, however, that the ability to use the keyboard in both old and new ways will be forthcoming when the meaning of each key can be programmed in a general way.

The problem of controlling many different sounds and of repatching are most notably solved by multichannel tape recording techniques, and it has been as a non-real time studio instrument rather than a performance instrument that the analog music synthesizer has made its mark on the direction of new music. All of the traditional tape manipulation techniques may be combined with the flexibility of the synthesizer, which is powerful enough to obviate the need for excessive post-processing.

Digital Music Synthesis

The analog music synthesizer presents a powerful though limited means of controlling a loudspeaker, since the loudspeaker can produce many sounds which the analog synthesizer cannot, such as human speech. Digital computers, however, are capable of controlling the loudspeaker with precisely the accuracy, stability, and repeatability that are lacking in the analog music synthesizer.

Computers have been used for digital sound processing since the early 1960's. A great deal of research has been done on computer speech processing and generation, and volumes on the digital processing of audio signals now abound. Computer music has also been researched for a similar amount of time, but much less thoroughly, due to the comparatively limited number of people capable of doing such research (in the past) and the accessability of computer systems sophisticated enough to allow such research -- essentially the same hardware is required as for digital speech processing, but signal bandwidths are considerably greater for music. The first computer music sound synthesis programs were written by Max V. Mathews at Bell Laboratories in the early 1960's. (This was by no means the first application of computers to music, however: Lejaren Hiller had programmed a computer to compose music as early as 1955. cf. L.A. Hiller's *The Illiac Suite for String Quartet*, published in 1957.) Mathew's programs have remained the basis for most computer music programs to this day; in particular the MUSIC V program [Mathews et al., 1969] was coded in FORTRAN IV and was thus exportable to nearly any computer of sufficient size.

In MUSIC V parlance, the user writes a "score," analogous to the traditional musical score, except that in this score it is also necessary to specify all of the acoustic properties of each "instrument" employed. Two fundamental concepts of MUSIC V are the *unit generator*, and the *stored function*. The unit generator is a computer algorithm which simulates the signal processing or producing properties of an imaginary electronic device. For example, the "oscillator" unit generator (see Figure 1-1) is an imaginary device with three inputs, one controlling frequency, one controlling amplitude, and one controlling the waveform being generated. The output of the oscillator unit generator is a digital representation of the specified waveform at the specified frequency and amplitude. An important computational expediency is the storage of exactly one period of any waveform to be produced as a table of amplitude values, typically 512 numbers long. This stored function may then be referenced by any unit generator which requires it, and the process of computing many successive values of *sin(t)*, for example, is replaced by the much faster table look-up procedure. Thus the oscillator unit generator can produce *any* periodic

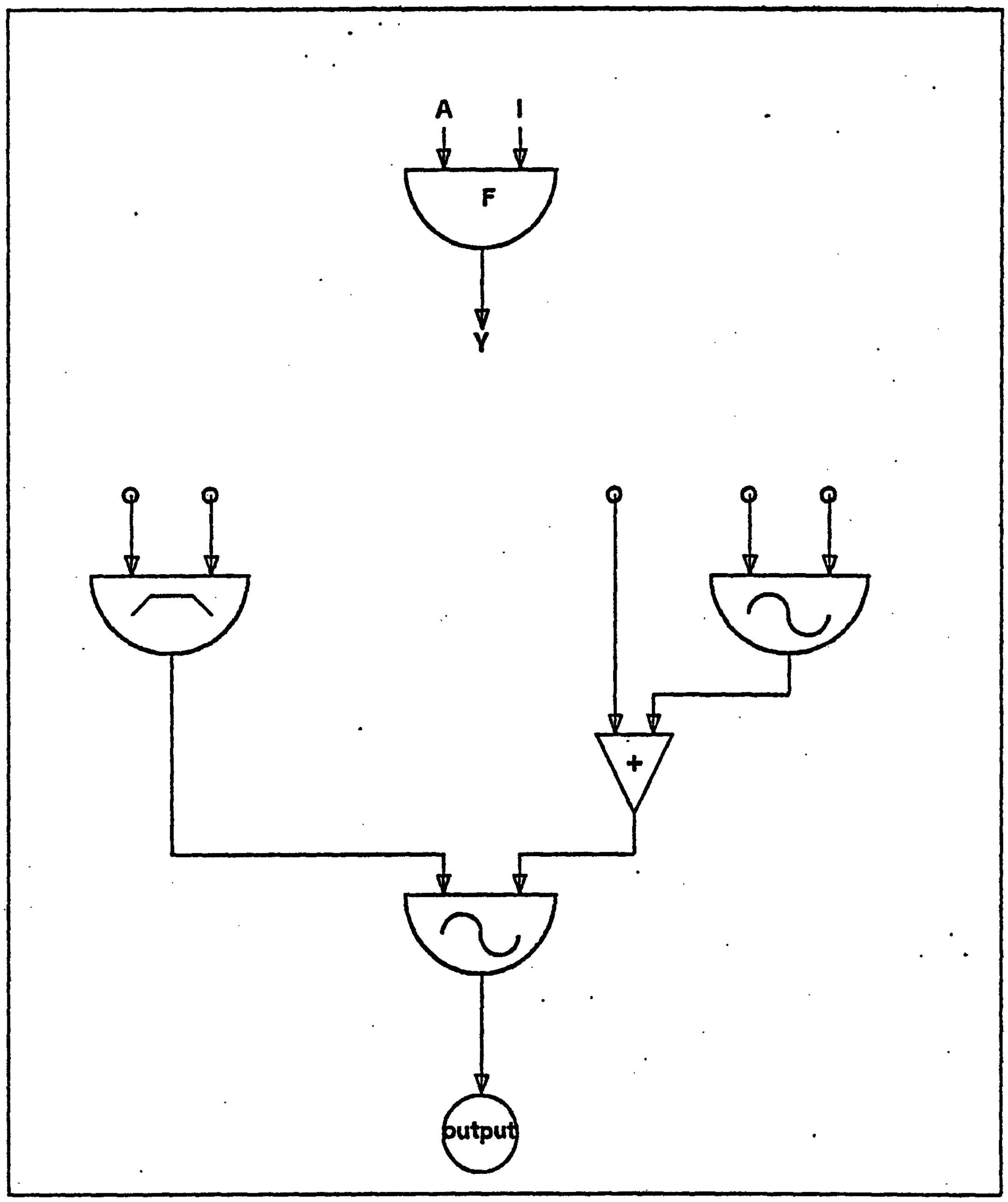


FIGURE 1-1: The MUSIC V unit generator notation: the symbol for the oscillator unit generator is shown at the top. It represents a computer algorithm for producing a digital signal, Y, which has an amplitude A, a frequency determined by I, and a waveform F. A typical MUSIC V instrument is shown below (in this case, the instrument produces a frequency-modulated waveform). The small circles represent "inputs" to the instrument, which are supplied as numerical values in the MUSIC V "score." The frequency of the upper left hand oscillator in this instrument is set so that it produces exactly one period of its waveform during the duration of one note; it thereby controls the amplitude envelope of the generated sound.

waveform at the rate and amplitude specified at its inputs. These inputs may be constants, or they may themselves be functions which change over time. For example, to produce a vibrato, the frequency input may be the sum of a constant and a slowly-varying, low amplitude sinusoid.

Other unit generators have been designed to perform other signal processing tasks, such as amplitude envelope generation, filtering, and reverberation. The user of the program "designs" instruments by combining unit generators in a graphical language similar to electrical circuit notation, and it is the specification of this "circuit of unit generators" which is typed as an instrument definition in the score. The stored functions must also be specified, which is typically accomplished by typing the specifications of either a sum of sinusoids (additive waveform synthesis) or the specification of a piecewise-linear waveform (e.g., for the amplitude envelope). Since any signal processing algorithm method may be programmed, the unit generators available depend only on the imagination and sophistication of the user.

"Notes" are then "played" on these instruments as specified in the rest of the score. Each time an instrument is "played," all of the "note parameters" of that instrument must be given; a single instrumental note might typically require values for the starting time, duration, frequency, amplitude, attack time, decay time, vibrato rate, vibrato deviation, and sound quality, to list only a few. It is evident that the traditional composer has relied on the performer to "supply values" for most of these aspects of a tone. The MUSIC V kind of score requires the user to be both composer *and* performer, since he must specify everything *in advance* of the audition of the sound.

The advantages of the MUSIC V-type of direct digital synthesis include the ability to produce virtually any sound which can come from a loudspeaker. Furthermore, the "synthesis model" can be described in terms of a circuit composed of predefined or specially designed unit generators. The program takes as much time as it needs to perform the calculations necessary for each sample of the musical waveform, and since there is no intrinsic limit on this amount of time, any reasonably efficient algorithm may be employed.

Therein lies a rub, however. If the amount of time needed to calculate each sample exceeds the sampling period of the output waveform, the program cannot run in real time, i.e., it requires more time to calculate the sound than it takes to hear it. This problem is solved in the MUSIC V context by storing the samples as they are calculated in some form of bulk computer memory such as magnetic tape or disc, and playing them back in real time after they all have been calculated. The well-known Nyquist sampling theorem states that in order to accurately represent a waveform containing frequency components up to f_{max} Hz. it is necessary to represent that waveform by *at least* $2f_{max}$ samples for each second. So in order to cover the audible range of frequencies up to, say, 20,000 Hz., the music program must calculate at least 40,000 numbers for each second of sound. For stereophonic or quadraphonic sound, of course, the minimum sampling rate is multiplied by the number of channels, since the sound emanating from each channel is presumably different.

Further, the precision with which each sample is calculated is responsible for the overall signal-to-noise ratio of the resultant sound, due to quantizing effects. This factor can also affect the useful "dynamic range," or overall available range of usable loudness levels, of the music. Assuming that the samples are represented in the computer to N binary digits of precision, and that the "quantization noise" can be characterized by a random signal whose amplitude is not greater than $\pm$ 1/2 of the least significant bit, then the theoretical signal-to-noise ratio is approximately equal to $6N$ dB. Standard, professional quality audio equipment operates with signal-to-noise ratios of about 60 to 70 dB., so in order to avoid introducing additional noise due to quantization effects, 12 or more bits typically have been chosen to represent each waveform sample. Combining this measure with the sampling rate restriction given above, we can see that the total bandwidth of the digital music signal has a lower bound of approximately:

$$12\ cNf_{max} \text{ bits/second}$$

where:

c is the number of audio channels,

N is the number of bits per sample, and

f_{max} is the maximum frequency of any component contained in the signal waveform.

For a typical stereophonic disc recording with a frequency bandwidth of about 15 kHz., then, a digital signal bandwidth of about 7×10^5 bits per second would be required to generate such a sound in real time. Assuming sequential processing of each sample value (i.e., that the computer is not a parallel machine), each value would have to be calculated and output within 1/60,000 second, $\approx$ 16 μs. Whether a high speed digital computer can perform the necessary calculations within this amount of time depends on the calculation methodology, which in turn depends on the synthesis algorithm used, though in fact 16 μs is generally not enough. Since these algorithms are crucial to the calculation requirements of computer music, we shall review them now.

Synthesis Methodology

Most of the current synthesis methods fall into one or more of the following four categories.

PHYSICAL MODELLING -- In order to synthesize the sound of a traditional musical instrument, it is possible to write down a description of that instrument in terms of the mathematical equations describing its physical operation (see Figure 1-2). Such a physical-mathematical-acoustical description can then be modified and improved until a satisfactory match between the physical instrument and the mathematical "model" is obtained, as determined by listening to the sounds produced by the model and comparing them to those of the original instrument [Ruiz ,1969]. This method is limited because it is essentially independent of the *psychoacoustic* characteristics of the musical sound -- it is difficult to see intuitively how to modify such a mathematical model to extend the sonic capabilities of an instrument, for example. Also, the solution of a sufficiently complicated set of differential equations often requires a great deal of computation time, and it is not usually evident how to speed such computations without harming the model. This method may be characterized as an expensive means to achieve excellent imitation of a real musical instrument, which is typically not the main objective of users of computer music programs -- hence it will not be considered further.

ADDITIVE SYNTHESIS (see Figure 1-3) -- Rather than examine the properties of the musical instruments themselves, most computer music research to date has examined the sound itself, in an attempt to discover what is *perceptually* important about a particular waveform as heard by a listener in a musical context. With Fourier's theorem as a starting point, the additive synthesis model views any and all sound as a collection of sinusoidal "building blocks," each of which varies in frequency, amplitude and phase over time in some characteristic way. Thus if we can both determine the structure of a sound and reassemble a new sound composed of synthetic components of the same description, then we

The WAVE EQUATION for an ideal string:

$$\partial^2 y/\partial t^2 = T/\rho S \cdot \partial^2 y/\partial x^2$$

with general solution:

$$y = g_1(x - at) + g_2(x + at)$$

is modified to include the effects of non-ideal characteristics of real strings:

$$\underbrace{\partial^2 y/\partial t^2 = T/\rho S \cdot \partial^2 y/\partial x^2}_{\textit{(ideal string)}} \underbrace{- ER^2/\rho \cdot \partial^4 y/\partial x^4}_{\textit{(stiffness)}} \underbrace{- b_1 \cdot \partial y/\partial t + b_3 \cdot \partial^3 y/\partial t^3}_{\textit{(friction \& damping)}}$$

where:

S is the cross-sectional area of the string in cm^2,
T is the string tension in dynes,
ρ is the string density in gr/cm^3,
E is Young's modulus of elasticity in dynes/cm^2 for string material,
I is the moment of inertia of the cross-section about a plane going through the center of the string and perpendicular to the direction in which the string vibrates, in cm^3,
$R = I/S$, the radius of gyration of the cross section, in cm,
b_1 is a positive constant accounting for heat dissipation, and
b_3 is a positive constant accounting for sound radiation.

The above differential equation is converted into a difference equation and solved for a given set of boundary conditions at times 0, Δt, $2\Delta t$, ... , yielding a digital waveform resembling that of the physical instrument with sampling rate $1/\Delta t$.

FIGURE 1-2: The PHYSICAL MODELLING synthesis method illustrated for string tone synthesis [after P. M. Ruiz: A Technique for Simulating the Vibrations of Strings with a Digital Computer, Master's thesis, Department of Music, University of Illinois, 1969]: The physical system of the instrument is mathematically modelled first as a differential equation, then converted into a difference equation. The difference equation is then solved as a function of time on a digital computer. The calculations usually require considerable computer time, but the correct mathematical model can produce excellent sonic results.

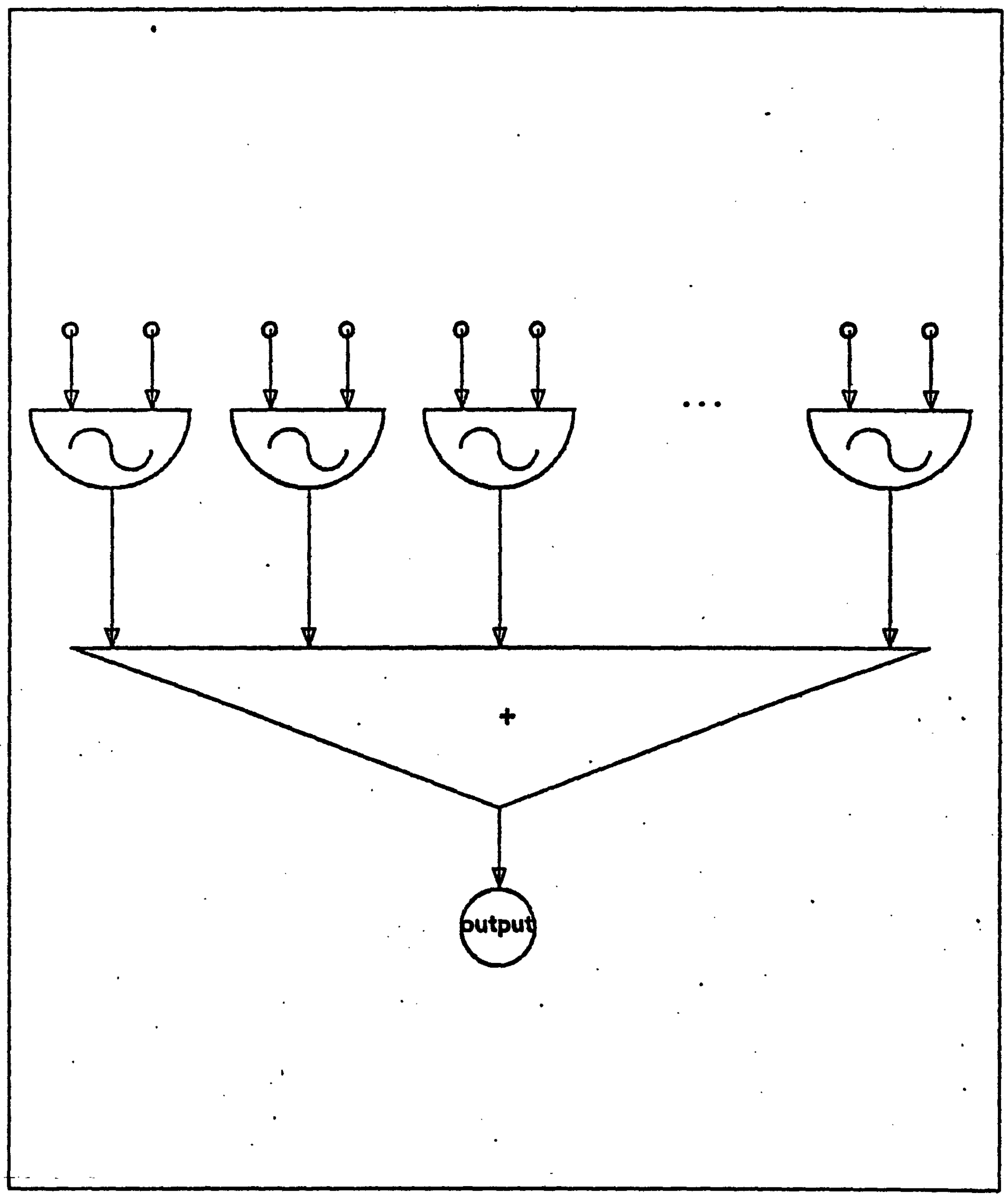


FIGURE 1-3: The ADDITIVE synthesis instrument in MUSIC V notation: Each of the small circles represents an input for controlling either the amplitude or frequency of a sinusoidal waveform. These inputs are generally functions of time which describe the amplitude and frequency fluctuations of each sinusoidal component during a single note played on the instrument. The instrument is analogous to a generalized version of the Fourier series in the sense that the amplitudes and frequencies are not necessarily constants, nor are the frequencies necessarily harmonically related to one another.

can verify the efficacy of this synthesis model in general. This analysis-based synthesis has been accomplished in certain cases with impressive results (see below), and it seems that additive synthesis is the most powerfully general synthesis algorithm available. For example, Moorer [Moorer, 1975] was able to extract the frequency and amplitude characteristics (phase was discarded due to its apparent unimportance) of components of single notes played on various musical instruments. Tones resynthesized from the data obtained from this analysis proved to be musically indistinguishable from their original counterparts. (This criterion deserves a special remark: it is not true that in every case the resynthesized tone was *absolutely* indistinguishable from the original tone, but simply that the difference was judged to be musically immaterial, i.e., smaller than the difference between two consecutive tones played by a skilled performer on the same instrument which are as nearly identical as possible. Thus *musical* indistinguishability is not absolute, but is often a more appropriate criterion in studies of this type.)

Following Moorer's work, psychoacoustic investigations of Grey [Grey, 1975] demonstrated that the component descriptions obtained from Moorer's analysis could be grossly simplified without sacrificing much, if any, of the tone quality of the original sound. Thus he was able to reduce the size of the necessary description by a factor of 100 or more by discarding what seemed to be perceptually irrelevant in the description obtained from the analysis. The amount of complexity involved in additive synthesis is difficult to characterize because it varies so widely, but generally it is quite complicated if high quality tones are desired. Grey was able to adequately mimic the sounds of several standard musical instruments using 12 to 16 components, each described by piecewise-linear functions of time for frequency and amplitude variations consisting of 5 to 7 line segments each. The production of any single component is of course a simple task, but with several complex tones sounding at once the sheer bulk of the computation involved soon becomes quite hefty. In computer terms, this type of synthesis procedure would be most easily carried out by a highly parallel, distributed computing system, with each computing element having rather simple calculating capabilities and a modest amount of memory. The typical, large scale modern computer facility is based on a serial-instruction, centralized processor with vast amounts of computational power and memory which can simulate, *à la* Turing machine, the properties of the former, but at a much slower operating speed. Thus the additive synthesis scheme is unlikely to run in real time on typical computers. Special hardware, such as a large array of rather simple, inexpensive microprocessors controlled by a central computer could provide an ideal environment for additive synthesis.

SUBTRACTIVE SYNTHESIS (see Figure 1-4) — Additive synthesis is not particularly well suited to analog music synthesizers due to the inherent lack of precision with which they can control individual components of a sound. The additive synthesis model more than any other requires the precision inherent in digital synthesis. Waveform synthesis is often accomplished in analog synthesizers by starting with a complex waveform (thus assuring a harmonic relationship between individual components, for example), and then modifying this waveform with filters whose characteristics may vary in time. Subtractive synthesis is also the general approach used in most speech synthesis research. The excitation waveform might be harmonic, inharmonic, or a noise source, and the time-variant filters allow a fairly fine degree of control of the spectrum of the resultant sound. Here the problem becomes one of specifying the filters to be used, and we are usually concerned with the computational complexity of the filtering system and the choice of an excitation function with desirable characteristics.

Unfortunately, specifying filters involves some rather formidable mathematical analysis and the criticism has been advanced that filter descriptions, unlike component

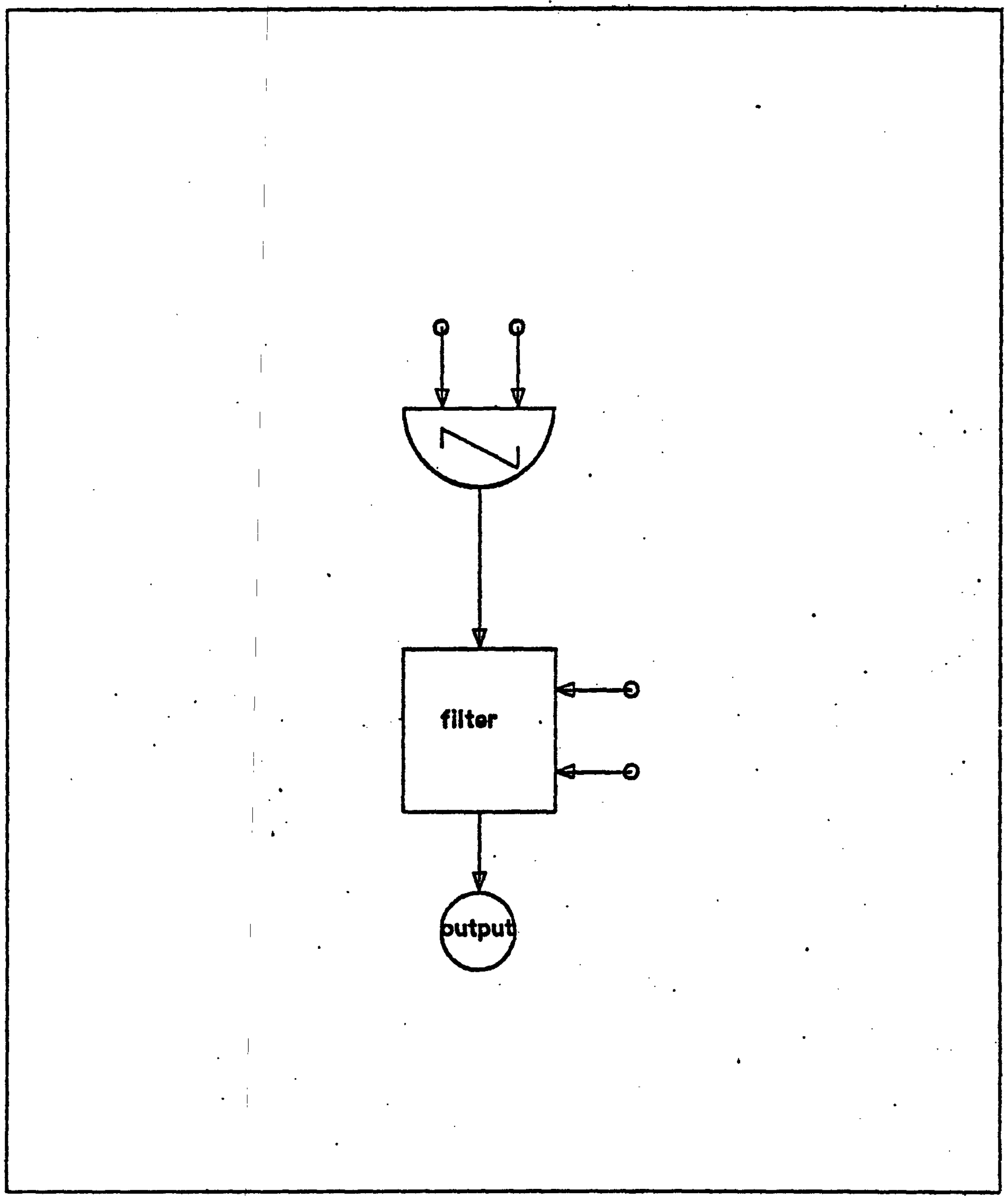


FIGURE 1-4: The SUBTRACTIVE synthesis instrument in MUSIC V notation: A complex waveform (in this case a sawtooth wave which is presumably band-limited) containing several harmonics is fed into a unit generator which performs the digital filtering operation, thereby modifying the spectrum of the complex waveform. The inputs for the filter control its the width and position of its passband. Several filtering units may be used in tandem to achieve the desired spectral response. All inputs may be time-varying.

descriptions, have no particular perceptual meaning, and hence we gain little intuitive insight into the nature of musical sound modelled in this way. Nor can we experiment easily with new sounds due to the mathematically capricious behavior of filter coefficients. The complexity of the calculation depends almost totally on the complexity of the filter, and is probably about the same as, or slightly less than, that of additive synthesis for comparable quality. A model of the computational process would include a complex waveform source driving either a complex signal processing machine, or a series of simpler, identical machines representing the decomposed filter. It is interesting to note that the overall delay introduced by this series of simultaneously-operating machines (this is a so-called *pipelined* computational structure) is immaterial to real time operation of the system as long as the total delay is perceptually unimportant (less than, say, 100 ms.). Thus the speed of the calculation is limited only by the speed of the slowest computing element in the serial path. A high speed, time multiplexed second order filter stage such as the one built by H. Alles at Bell Laboratories provides the ideal environment for subtractive synthesis. The only essential disadvantages of subtractive synthesis are its unintuitive nature and the assumption that the excitation function which provides correct results may be easily found.

NONLINEAR SYNTHESIS — Synthesis models which do not fall into any of the above categories are generally nonlinear in nature. One of the most powerful nonlinear synthesis methods was first introduced by Chowning when he described complex waveform synthesis based on frequency modulation techniques [Chowning, 1973]. Since then, many other similar techniques have been uncovered, some of which are described by Moorer [Moorer, 1975]. These tend to be the most efficient methods — that is why they are chosen — and they seem to lie somewhere in between additive and subtractive synthesis as to their "intuitiveness." They make various assumptions about the nature of the sound being synthesized, and they tend to require fewer but more sophisticated steps than the other methods. For example, the band-limited pulse waveform promises to become useful in conjunction with additive and subtractive synthesis as well as in its own right. It is simply obtained from a trigonometric identity relating the sum of a finite number of angles:

$$f(t) = \sum_{k=0}^{n-1} \sin(\theta + k\beta) = \sin\{\theta + (n-1)\frac{\beta}{2}\} \sin(\frac{n\beta}{2}) \csc(\frac{\beta}{2})$$

If we set θ to $2\pi f_1 t$ and β to $2\pi f_2 t$ then this relation allows us to assemble a waveform with a finite number (n) of equally spaced (f_2) components positioned starting at any frequency (f_1), all with unity amplitude, with a fairly small, fixed amount of computation independent of n. In subtractive synthesis, such a waveform allows an economic method of gaining flexibility in the excitation function, while in additive synthesis it could easily represent a portion of a spectrum. The calculation does involve some sophistication due to the presence of the discontinuous cosecant function, and normalizing the output amplitude presents added difficulty. Other nonlinear techniques allow similar control over the placement and number of frequency components, along with a rough control over their relative amplitudes in some cases, and a similar amount of calculation is usually required, though its nature will vary depending on the method. A set of fast, complex microprocessors with high speed arithmetic ability would provide a suitable environment for most nonlinear methods. It is here that the requirements are changing and growing most rapidly.

Three Ways to Make Brass-like Tones

As an example of the conceptual difference between these synthesis methods, consider a brass-like tone as modelled by each one. Risset discovered and then verified by means of additive synthesis that a brass-like tone is one in which the frequency components are

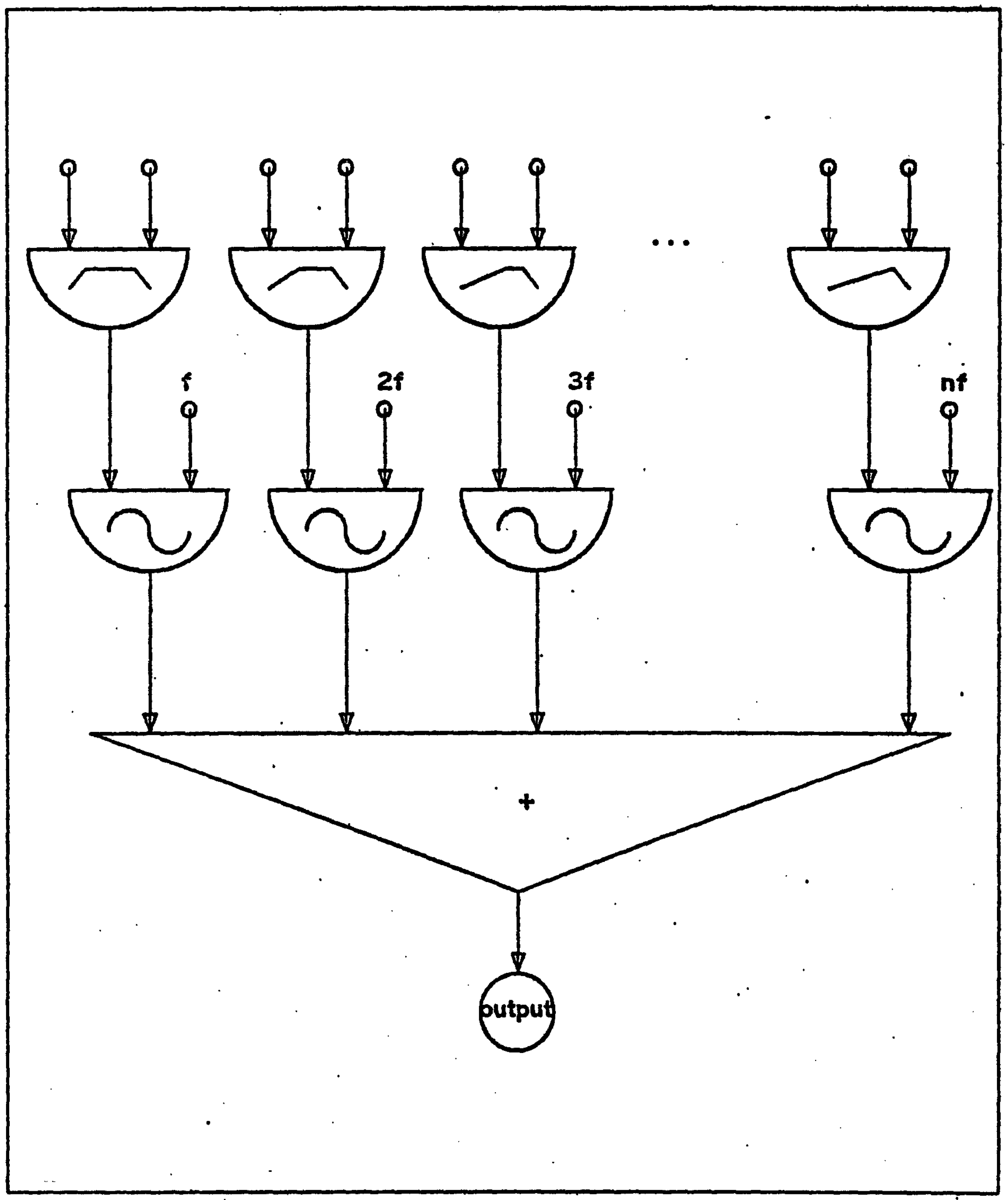


FIGURE 1-5: ADDITIVE synthesis instrument for brass tone synthesis with fundamental frequency f: The top row of unit generators produce amplitude envelopes for each of the harmonics of the note. The amplitudes are arranged so that the higher frequencies attain full amplitude after lower frequency components. Thus as the overall amplitude increases more high frequency energy is incorporated in the tone, causing it to have a brass-like character (see text).

roughly harmonic, and the bandwidth is proportional to the amplitude envelope of the sound [Risset and Mathews, 1969]. The details and location of the spectrum will determine the exact timbre, but just this much specification is sufficient to insure an unmistakable brass-like character. Risset defined a set of piecewise linear amplitude functions composed of 3 segments, an attack, steady state, and decay portion; he then varied the slopes of the attacks so that higher frequency components became audible after lower ones (see Figure 1-5). This sum of harmonically-related sinusoids with this set of amplitude characteristics produced a waveform which obeyed the brass-tone hypothesis given above, and it does indeed sound brass-like. Somewhere between 10 and 20 components seem to be necessary before additional improvement cannot be obtained, typical for most additive synthesis results to date.

An equivalent application of Risset's brass tone hypothesis to subtractive synthesis results in a complex waveform, such as a band-limited pulse or sawtooth waveform, fed into a lowpass filter whose cutoff frequency is made proportional to the amplitude envelope of the desired note (see Figure 1-6). The output of the filter, which obeys Risset's hypothesis and thus sounds brass-like, can be implemented with voltage controlled equipment in an analog synthesizer, since the sources and interconnections are available and the details of the amplitude envelope and excitation waveform are not too critical.

In nonlinear synthesis (see Figure 1-7), the brass tone can be accomplished with, for example, frequency modulation:

$$f(t) = A(t)\, sin\{\, \omega_c t + \frac{\Delta\omega}{\omega_m} sin(\, \omega_m t\,)\}$$

where:

$A(t)$ is the amplitude envelope function,

ω_c is the carrier frequency ($= 2\pi f_c$),

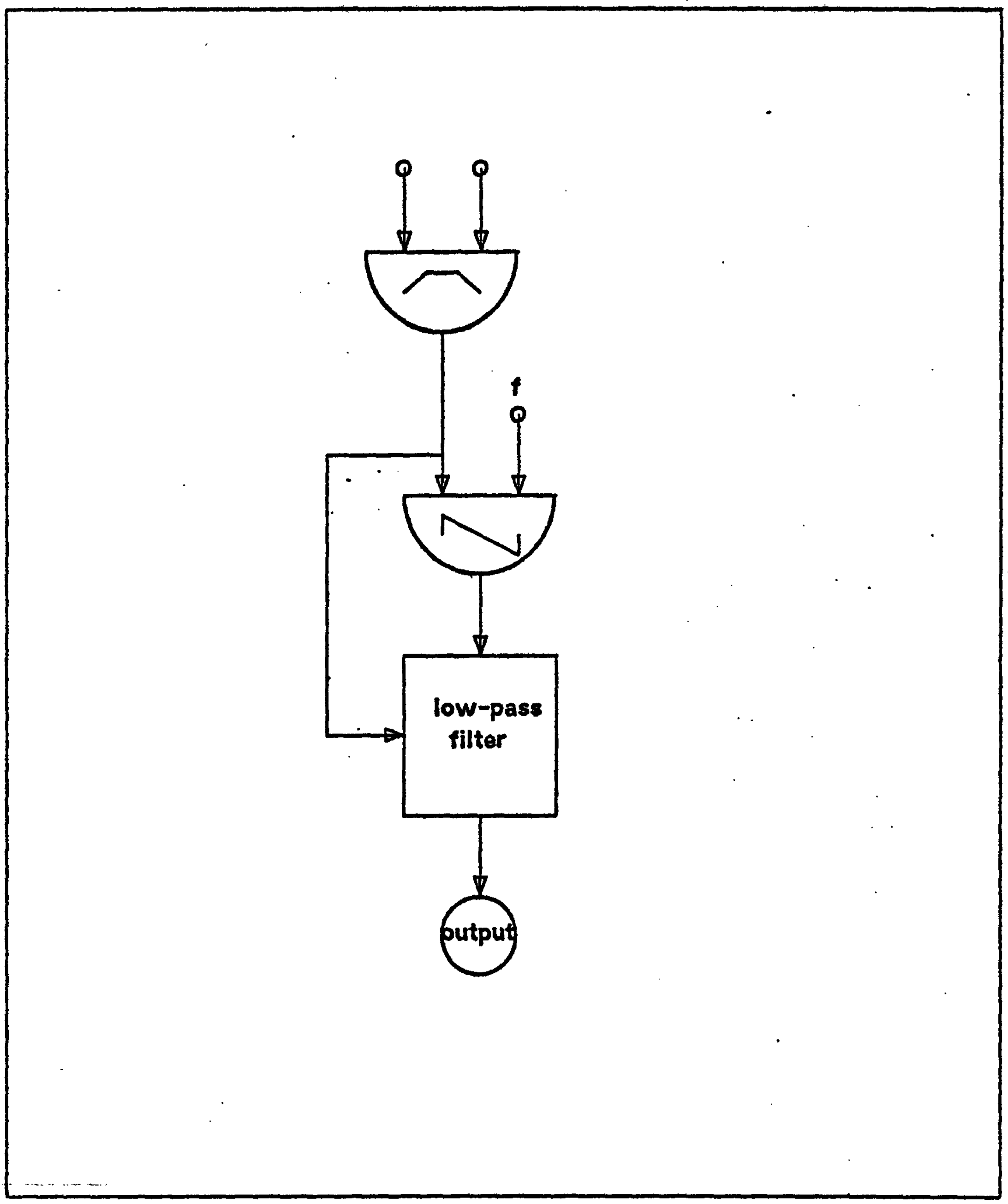


FIGURE 1-6: SUBTRACTIVE synthesis instrument for brass tone synthesis with fundamental frequency f: The same envelope function is used to control both the overall amplitude of the note and the cutoff frequency of a low-pass filter which has a complex wave input. Thus higher frequency components of the output gradually become more prominent as the amplitude is increased during the note, and it sounds brass-like.

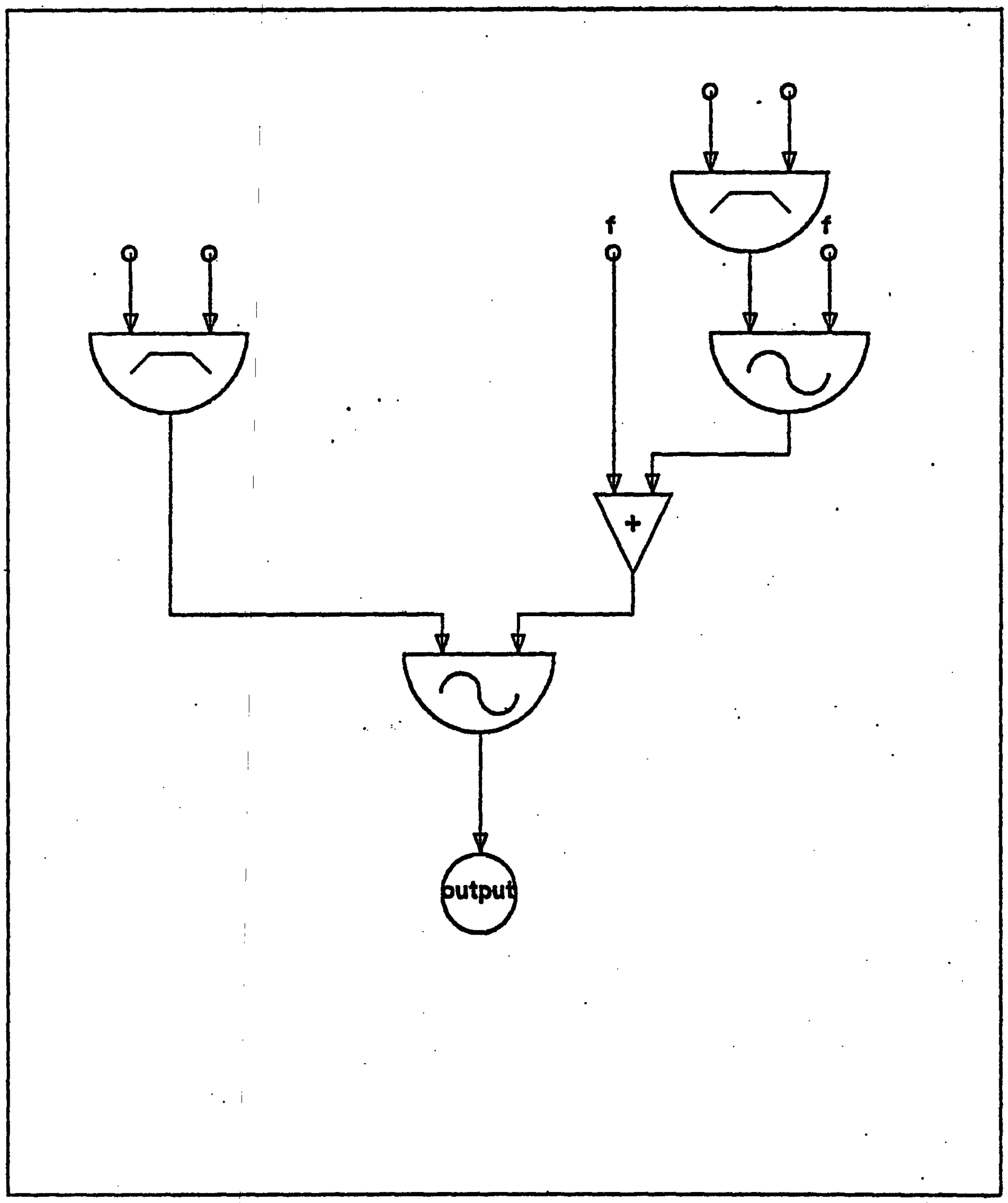


FIGURE 1-7: NONLINEAR synthesis instrument for brass tone synthesis with fundamental frequency f: Setting the modulating frequency equal to the carrier frequency produces a harmonic spectrum whose bandwidth is proportional to the amplitude of the modulating wave. In this instrument the same function is used to control both the overall amplitude and the frequency deviation, causing the bandwidth to increase with amplitude, producing a brass-like sound.

$\Delta\omega$ is the peak frequency deviation, and

ω_m is the modulating frequency.

The quantity $\Delta\omega/\omega_m$ is called the "modulation index" and more or less directly controls the bandwidth of the frequency modulated signal (though not its amplitude — which must be separately controlled via $A(t)$, as shown in the equation) and the values of ω_c and ω_m affect the placement and spacing of the frequency components. By setting ω_c = ω_m we can obtain a harmonic spectrum with fundamental ω_c, and if the modulation index is controlled through variation of $\Delta\omega$ as a function of time, namely $A(t)$, then the spectrum of the waveform would obey Risset's hypothesis, and the tone also would sound brass-like.

Real Time Music Synthesis

Real time synthesis has been hampered in the past by the inability to perform all of the needed calculations for a given set of notes by *any* of the available synthesis methods within one sampling period of the musical waveform. A very high speed general purpose computer and a great deal of clever programming has allowed a limited amount of real time synthesis with software [Saunders, 1974]. But unless computers become several orders of magnitude faster than they are, special hardware must be used to achieve more than a tiny digital synthesis capability in real time. The computation scheme of the digital synthesizer must fit the algorithms involved rather exactly, making it difficult to include even modest amounts of generality in the form of microprocessors and similar flexible, low-level devices. The specialized hardware tends to the rapidly varying characteristics of the sound, such as its waveshape, while a slower, more flexible general purpose computer controls the slowly varying characteristics such as overall loudness, channel placement, etc. The approximate dividing line between the operating time scales of the synthesizer and computer is the fastest speed of human action, to which the computer must be able to respond, such as depressing a key, or twisting a knob. It seems that most manual human actions can be adequately described by time functions which sample these actions between 100 and 200 times per second [Mathews and Moore, 1970]. Thus anything that changes more often than once every, say, 10 ms. is in the province of the synthesizer, while more slowly varying quantities are handled by the computer, unless there is a special reason to do otherwise. Basically, the sounds are generated by the synthesizer hardware, and the computer controls the synchrony and succession of the generated sounds.

The GROOVE system developed at Bell Laboratories was, among other things, an attempt to study the properties of these slowly-varying functions (GROOVE is a dedicated digital computer controlling a medium-sized analog music synthesizer via a bank of digital-to-analog converters). Experience with it tends to confirm that the division of labor should occur at about 10 ms. As a real time system, GROOVE has also indicated the importance of the interactive user environment, including real time graphics displays and conveniently mounted knobs and switches, and the acoustic condition of the room in which the interaction takes place.

Small-scale, specialized attempts to construct digital real time music synthesis hardware have succeeded in the past [Appleton, 1975], but the first really large scale attempt to build a powerful all-digital music synthesizer is the Systems Concepts' design by P. Samson. It is a stream machine in concept, processing a continuous stream of samples much as the analog synthesizer runs continuously, and it embodies the virtual capabilities of up to 256 "generators" and 128 "modifiers" which together can perform any of the known synthesis algorithms, as well as a fair amount of real time signal processing. It includes a sizable amount of random-access memory for holding function descriptions and for performing reverberation algorithms. Because of its complexity and speed, though, the Systems Concepts synthesizer will be extremely difficult to modify, should the desire to do so arise, and it is quite expensive (approximately $80,000).

It is the author's thesis that the desire to modify any music synthesis hardware will indeed arise, and soon, and repeatedly thereafter. Therefore let us next consider the properties of a digital music synthesis system which has modifiability as a central specification.

CHAPTER 2 - PROBLEM STATEMENT

The "FM Lesson"

Music is inherently a highly parallel process, even at the intranote level where individual components behave in an independent manner. Coupled with the fact that it is easier to modify small, simple machines than to change large, complex ones, the parallel nature of music suggests that a device designed for its production could benefit from modularity. Rather than conceive of the music synthesis system as an integrated, fixed, and highly optimized processor it is useful to consider it as a collection of simple devices, each of whose function could be easily changed without affecting the operation of any other part of the system. Such flexibility is gained at the price of a certain amount of optimality in the operation of the device for any one algorithm. But the function of such a device as a musical sound synthesizer is ill-defined at best, if only because it cannot be completely specified. The different synthesis methods described earlier each have slightly different computational requirements, so no one device can operate optimally for all three. Even if it could, it is very likely that new synthesis algorithms will be developed. For example, the earliest versions of music synthesis programs such as MUSIC V contained an oscillator algorithm as follows:

$$O_i \leftarrow F[\,S_i\,] \times A_i\,;$$
$$S_{i+1} \leftarrow (\,S_i + I_i\,) \bmod L\,;$$

where:

O_i is the i^{th} output sample,

A_i is the i^{th} amplitude control value,

I_i is the i^{th} increment value (frequency control value),

S_i is a cumulative sum of increments,

$F[k]$ is the k^{th} value stored in array F (F might typically contain values for *sin x*), and

L is the length of table F.

This algorithm works by stepping periodically (due to the modulus operation) through a table of stored values. The step size (I_i) can be varied to allow differing output periods to be obtained. Thus if L values of one period of *sin x* were stored in the F array, this algorithm would generate a sampled sinusoid with a controllable frequency and amplitude. (The frequency of the output waveform is equal to $\rho I_i/L$, where ρ is the sampling rate.) Since computer time is often at a premium, and because of the bulk of computation involved, it is usually necessary to implement this algorithm in the most efficient possible way on a given machine. In particular the modulus operation (α *mod* β), in order to avoid a time-consuming division, was often implemented by subtracting β from α if α exceeded β:

$$\alpha \; mod \; \beta ::= \text{IF } \alpha < \beta \text{ THEN } \alpha \text{ ELSE } \alpha - \beta$$

Because the increment value was typically positive and small compared to L in the above algorithm, this implementation worked quickly and well. It was in fact the fastest method, given the stated assumptions. But the introduction of frequency modulation as a sound synthesis technique changed one important assumption: the I_i were still typically small compared to L, but they were no longer necessarily positive. In other words, the oscillator might be asked to generate a negative frequency during frequency modulation (this occurs whenever the peak frequency deviation exceeds the carrier frequency, $\Delta\omega > \omega_c$). It was a simple matter to change the software to allow for negative increments, but the point is that it was necessary to do so to take advantage of a major breakthrough in nonlinear synthesis technique. To change a hardwired device would not have been so easy, if it had been possible at all.

Synthesizer Design Specifications

New synthesis procedures, and modifications and improvements to old ones, are being discovered at an increasingly rapid rate as computer music research progresses. It seems fairly certain that the fixed, large scale synthesizer design has enough advantages to insure its utility for a long time to come. But it also seems likely that there will soon exist synthesis methodology incompatible with its capabilities. Furthermore, its complexity is achieved at fairly high cost, which makes it unlikely that such a machine will become available as a performing instrument. A less expensive, portable, and flexible device will be useful for both music performance and as a research tool, even if the total capacity of the smaller machine is less. The desirable properties of the modular synthesizer may be summarized as follows:

FUNCTION -- the music synthesizer should provide a reasonably powerful capability in two realms: as a musical performance instrument, and as a tool for research. As a musical instrument, the hardware synthesizer runs under control of a small, portable digital computer system (such as a PDP11/34 with a floppy disc memory) and a set of real time controls such as knobs and keyboards. As a research tool, a more powerful computer would probably be needed for control, because of the desire to connect the digital signal output of the synthesizer back into the computer, which is not the case during typical performance situations. In either situation a set of modules would be available out of which the exact configuration of the machine would be selected. Multiple synthesizers should be allowed to operate together intercommunicatively in order not to limit the overall capability of the device.

COST — a research machine design (prototype) usually costs more than a design intended primarily for mass production. It seems that a fair price objective would be a replication cost on the order of the price of a concert grand piano. This price would not include the cost of the control computer system, but neither is such a system restricted to service the hardware synthesizer. Some properties of the control computer system are likely to make it intrinsically more expensive than the synthesizer.

SIZE — again the controlling computer system presents more of a physical size problem than the synthesizer. Since the synthesizer shouldn't cost more than a grand piano, it shouldn't be harder to move either!

FLEXIBILITY — the device should be capable of performing all known synthesis algorithms in a reasonably efficient manner. It is not necessary that they all be performable at once, however, since the device is to be easily modifiable. The synthesizer should place as little restriction as possible on its own applications.

SELF-EXTENSIBILITY -- an extremely useful property of a modular system would be no absolute restrictions on size or complexity, i.e., it should be possible to keep adding modules to perform arbitrarily complex synthesis procedures.

EASE OF MODIFICATION -- the device should be as easily modifiable as possible in keeping with other requirements, such as speed. Module design should be as simple so that new modules may be developed easily.

AUDIO FIDELITY — the device should be in keeping with current professional audio standards in order to take full advantage of the precision afforded by digital representation and processing of audio signals.

REAL TIME — the device should be able to perform in real time, but if this is not possible, it should degrade in performance speed in such a way that it remains useful as a non-real time signal generator and processor.

It is possible to satisfy all of these criteria with the hardware synthesizer described in the remainder of this thesis. It should be noted, however, that the development of the synthesizer is only one portion of the complete music generating system, which also includes a controlling computer with requisite software, and a set of real time interactive control devices with which the user may "perform" on the system.

CHAPTER 3 - SYSTEM DESIGN

Overall System Description

An overall view of a real time interactive computer music synthesis system which meets the design criteria stated in Chapter 2 is shown in Figure 3-1. The system may be subdivided into the following subsystems:

SYNTHESIZER — a real time digital sound synthesizer capable of generating sounds by any known synthesis method under computer control,

COMPUTER — a real time digital computer subsystem with its associated software, bulk memory, and peripheral devices which provides the control data for the synthesizer hardware,

INTERACTIVE CONTROLS — a set of interactive controls which can be manipulated in real time by the user of the system to achieve a musical performance capability, and

LOUDSPEAKERS — an audio playback facility and environment in which the interactive portion of the system is imbedded, including some standard electronic music facilities, such as tape recorders, mixers, etc.

All of the above facilities exist as part of the GROOVE system, for example, except for the digital synthesizer. It is not known at this time whether the GROOVE software will work as optimally for a digital synthesizer as it does for an analog synthesizer. Certainly some changes will be required to take full advantage of the capabilities of digital synthesis procedures, but these considerations are relegated to future investigations. We now turn to the properties of the digital sound synthesizer.

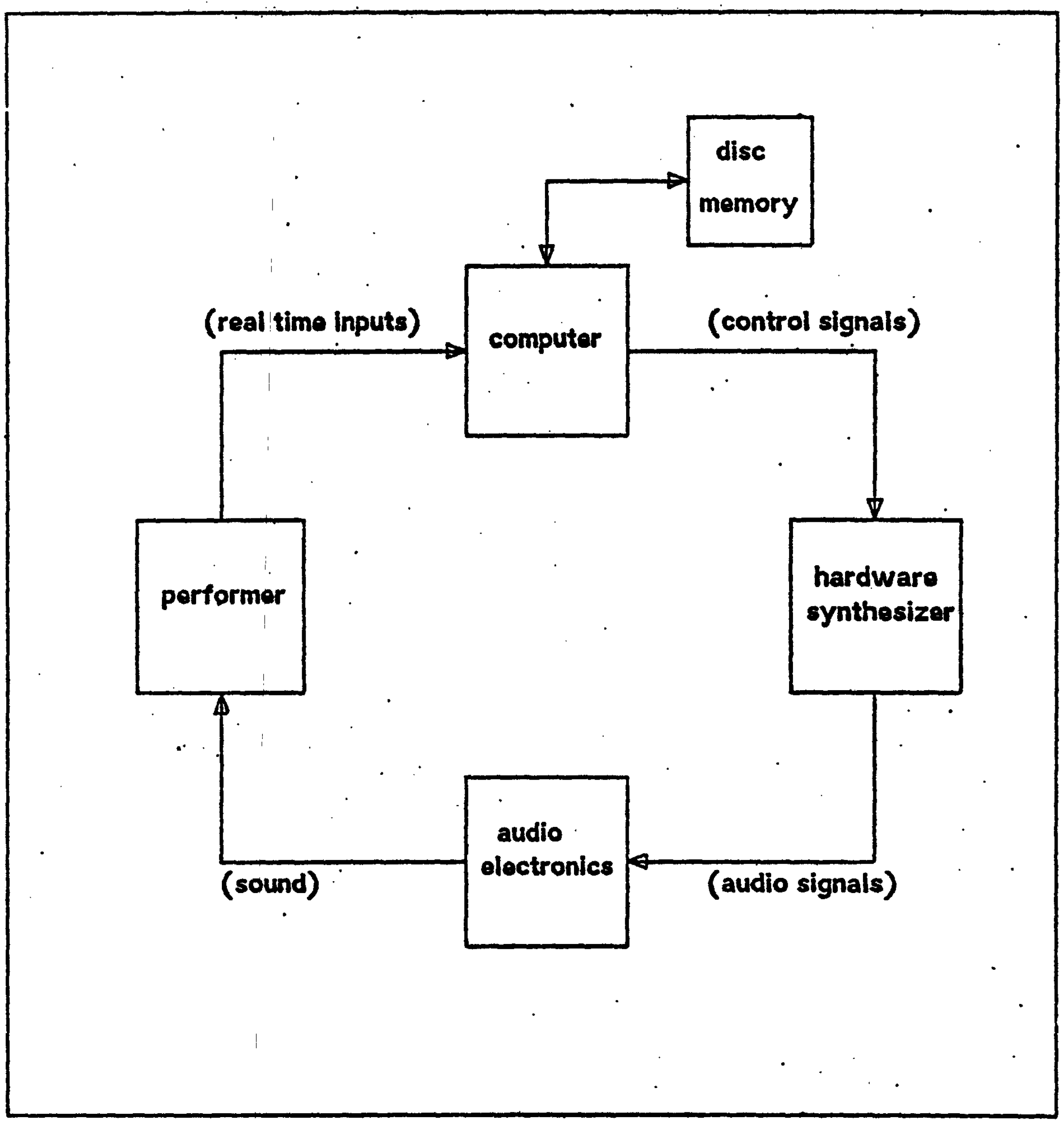


FIGURE 3-1: Block diagram of a real time interactive computer music facility: The performer generates input information to the computer by manipulating real time input devices (such as knobs, keyboards, switches, etc.). The computer makes a record of the actions of the performer for later reference, combines the inputs currently being generated by the performer with previous inputs, and generates control signals which are fed to the digital hardware sound synthesizer. The synthesizer produces a digital waveform which can be either fed back to the computer or output through a digital-to-analog converter to the audio electronics and loudspeakers (or both). Loudspeakers produce the sound being generated, which can be heard by the performer in real time.

The Synthesizer Control Unit

The real time digital music synthesizer hardware -- from now on simply "synthesizer" unless otherwise indicated -- is shown in block diagram form in Figure 3-2. It is conceptually divided into a control unit, and a set of functional modules. These modules may be thought of (for the moment) as devices which perform functions analogous to the unit generators of MUSIC V, such as oscillators, filters, envelope generators, etc. Information is supplied to the modules by an input bus connecting the control unit to all module inputs in parallel; the information produced by the modules is likewise routed along another bus back to the control unit. Physically, the modules are digital circuits wired on circuit boards which are plugged into slots provided for them on a module mounting rack (see Figure 3-3), in a manner analogous to the plug-compatible units found in analog synthesizers. In the current implementation provision is made for up to 8 module inputs and 8 module outputs to be routed through a single control unit.

The control unit performs two basic functions. One is time-multiplexing the operation of each module connected to the system so that its function is replicated several times for each output waveform sample. Thus a single module represents several "virtual" devices which perform the function of that module: in the current implementation the oscillator module provides the equivalent of up to 32 oscillators, etc. No special requirements are made on the design of the modules themselves, except that if a given module requires memory in order to perform its function (e.g., a filter may have to "remember" one or more previous output values), then the module must contain a separate memory for each of its virtual functions. Small random access memories can be used instead of memory registers in this case, and present no particular problem to the module designer.

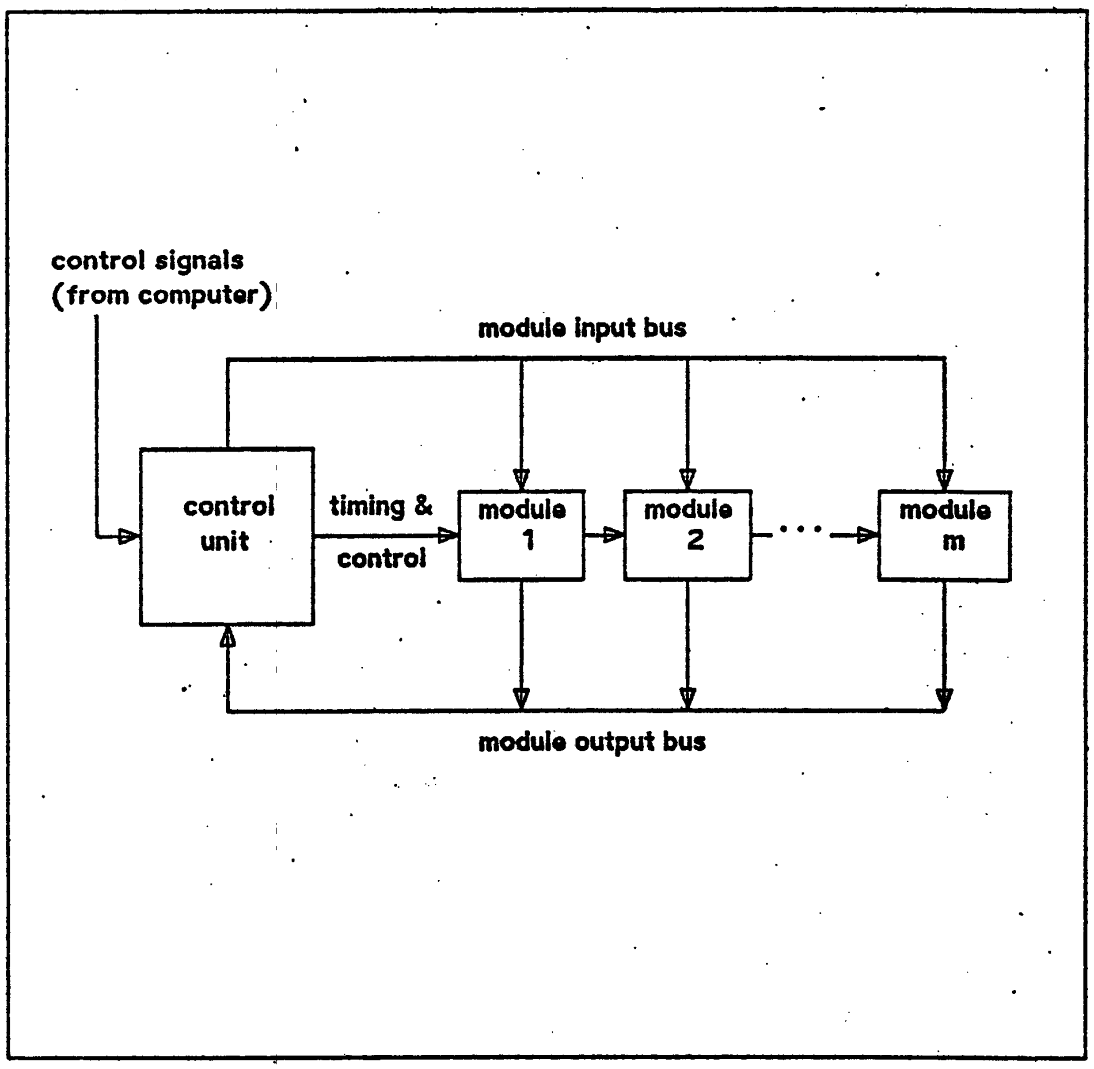


FIGURE 3-2: Block diagram of the real time digital hardware synthesizer: The user selects a set of modules according to the synthesis algorithm he wants to employ. Each module performs some function such as oscillation, filtering, etc. They are plugged into the synthesizer, which can accommodate up to m modules at once (m=8 in the current implementation). The control unit interchanges the signals among the modules and time-multiplexes their operation, yielding v "virtual" occurrences of each function (v=32 in the current implementation). The computer supplies information to the control unit specifying how the signals are to be interchanged, and the exact function of each virtual occurrence of the modules. The computer also supplies real time signals to the synthesizer via one of the module output connections (i.e., the computer "impersonates" a module producing signals in real time). One module drives the digital-to-analog converter system, which generates the analog signals which are fed to the audio electronics.

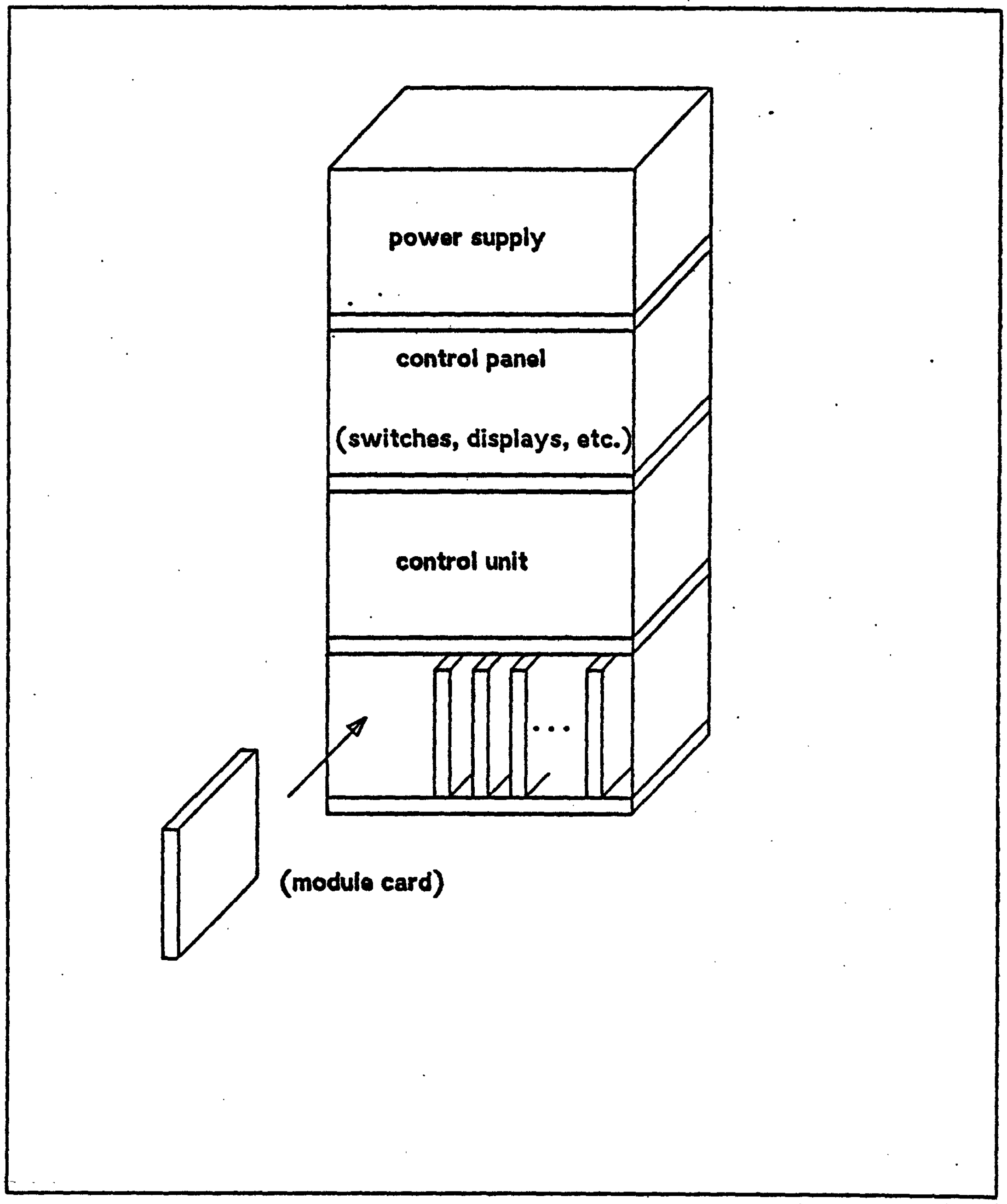


FIGURE 3-3: Block diagram of the physical construction of the hardware synthesizer: The hardware synthesizer is constructed in such a way as to facilitate plugging and unplugging module cards. Provision is made for some modules to occupy more than one circuit card.

The control unit also provides the means by which these virtual devices can intercommunicate, i.e., it allows the $8 \times 32 = 256$ possible virtual unit generators to be "patched" together to form "instruments" in the MUSIC V sense. In addition, the control unit is used to supply "control" information to each of the 256 possible virtual functions, which may be used to determine the exact function to be performed, for example.

In order to accomplish the signal interchange and time multiplexing functions, the control unit contains two high speed, random access memories, a signal memory, M_s, and a control memory, M_c. Each memory is $8 \times 32 = 256$ words long, the signal memory words containing 20 bits of information, and the control memory words containing 18 bits each. The format of these memories is shown in Figure 3-4: each word of signal memory M_s holds the signal output of a particular virtual function. The modules are serviced in a round-robin fashion corresponding to the address format of M_s: first the output of module 0 is stored in $M_s[0]$, then the output of module 1 is stored in $M_s[1]$, etc., up to $M_s[7]$; then the next output of module 0 is stored in $M_s[8]$, module 1's next output goes into $M_s[9]$, etc. This approach allows the value of the address used for M_s to be thought of as an 8 bit quantity which is divided into two fields: the least significant 3 bits correspond to the number of the module whose output is stored at this address, a value between 0 and 7, and the most significant 5 bits correspond to the "virtual occurence number," a value between 0 and 31, specifying which of the 32 possible virtual functions represented by this module has its output stored at this address. Thus M_s may also be interpreted as a 2-dimensional array indexed by quantities M, a module number between 0 and 7, and V, a virtual occurence number between 0 and 31. The absolute address in M_s is then given by:

$$A = [8(V+1) + M] \bmod 256$$

SIGNAL AND CONTROL MEMORY ADDRESS MAPS

			CONTENTS OF M_s: (current signal output of virtual occurrence V-1 of module M)		CONTENTS OF M_c: (current input source plus control bits for virtual occurrence V of module M)		
ADDRESS	V	M	←→ 20 bits →→		←5 b→	←3 b→	←←← 10 b→→→
0	0	0	output of M0V31		Vin	Min	CB for M0V0
1	0	1	output of M1V31		Vin	Min	CB for M1V0
2	0	2	output of M2V31		Vin	Min	CB for M2V0
3	0	3	output of M3V31		Vin	Min	CB for M3V0
4	0	4	output of M4V31		Vin	Min	CB for M4V0
5	0	5	output of M5V31		Vin	Min	CB for M5V0
6	0	6	output of M6V31		Vin	Min	CB for M6V0
7	0	7	output of M7V31		Vin	Min	CB for M7V0
8	1	0	output of M0V0		Vin	Min	CB for M0V1
9	1	1	output of M1V0		Vin	Min	CB for M1V1
10	1	2	output of M2V0		Vin	Min	CB for M2V1
⋮	⋮	⋮	⋮		⋮	⋮	⋮
253	31	5	output of M5V30		Vin	Min	CB for M5V31
254	31	6	output of M6V30		Vin	Min	CB for M6V31
255	30	7	output of M7V30		Vin	Min	CB for M7V31

FIGURE 3-4: ADDRESS MAPS of the signal (M_s) and control (M_c) memories: In the normal use of the machine the first 5 bits of the 8-bit memory address may be interpreted as the virtual occurrence number, and the last 3 bits as the module number with which a memory location is associated. Thus the 20-bit word located at $M_s[10]$ is the current signal output value for module 2, virtual occurrence 0 (all counting starts from 0). The location of a signal output in M_s may be modified by the details of a particular module's operation, such as pipelining (see text). The 18-bit word located at $M_c[10]$ is interpreted as follows: the first 8 bits specify which of the 256 locations in M_s contains the signal to be used as the input for module 2, virtual occurrence 1; the last 10 bits specify control information for module 2, virtual occurrence 1.

where A is the absolute address in M_s of the output signal associated with module M, virtual occurence V (the +1 factor comes from the fact that the output of a module occurs one module service period *after* the module receives an input).

Each 18-bit word of control memory M_c may be interpreted as follows: $M_c[A]$, where $A = 8V + M$, contains an 8 bit address Y, and an 10 bit control byte X. The Y address specifies which of the 256 possible signals stored in M_s is to be connected logically to the input of virtual function A as a signal input via the module input bus. Thus the input of each virtual function may be connected to the output of any other virtual function (an "input gather" technique as opposed to an "output scatter" technique), making it possible to "patch" the virtual functions together without making any special physical connections in the synthesizer. The 10 bit control byte X is fed directly to the associated virtual function as input. The meaning of these bits is determined entirely by the module; they are typically used to specify scale factors, or operation codes for selecting a subfunction for a particular module, etc. Thus the function of any particular module is not fixed, since each of the 32 virtual occurrences may select among a theoretical maximum of $2^{10} = 1024$ possible operations.

One of the modules is the output collector, which drives the digital-to-analog conversion (DAC) system. This module can use its control byte to determine such things as signal amplitude scaling or channel placement (any number of audio channels is possible without affecting the sampling rate of the output waveform). A real time input (RTI) module is connected via an interface to the control computer so that the computer can supply control signals to the synthesizer as if it were a module. The control computer communicates directly with M_c, as well as controlling the running mode of the synthesizer.

The computer may be optionally connected to the output collector module via an interface, so that it may read signal information from the synthesizer, allowing the synthesizer to be used as a high speed, though not necessarily real time, signal processing device and calculating aid to the computer. The running modes of the synthesizer include "free" run, where the synthesizer processes data independently of computer timing (the normal "stream" mode), and single sample mode, where the synthesizer steps through each of the 256 virtual functions just once, producing a single sample of the output waveform. The run mode may be set either by the computer or manually on the synthesizer control panel. Modules normally perform their operations within one module service period (about 1 μs), during which time the control unit services all 8 modules exactly once. Since there are 32 virtual functions to be performed by each module, it takes 32 module service periods, or about 32 μs, to step through all 256 virtual operations, yielding an output waveform sampling rate on the order of 32 kHz. This sampling rate provides a usable audio bandwidth of about 14 kHz., which is quite sufficient for most computer music applications. In fact, if the standard oscillator algorithm given in Chapter 2 is used, and the operation speed of the synthesizer is adjusted to yield an output sampling rate of $32{,}768 = 2^{16}$ Hz., then by virtue of the fact that the length of the stored table is chosen to be a power of 2 (typically 512 or 2048), the oscillator increment value is readable directly in Hz., possibly scaled by a power of 2. This consideration leads to a clock rate of $2^{24} = 16{,}777{,}216$ Hz., which is a reasonable choice for high-speed Schottky logic, and results in a computational convenience saving a great deal of software calculation by eliminating the need to calculate increments for the oscillator frequencies. It also yields a module service period conveniently equal to 953.67 ns.

Constraints on Control Unit Design

The maximum values of M and V are determined as follows. First, the MV product is equal to the total number of virtual functions during a sample period of the output waveform, Δt. For each virtual function to be performed, it is necessary to store one module output value and to read one module input value from M_s. Assuming that it takes t_r seconds to read information from M_s and t_w seconds to write information in it, and that these two operations are performed sequentially, we see that the following constraint must be satisfied:

$$\Delta t/MV \geq t_r + t_w$$

or,

$$MV \leq \Delta t/(t_r + t_w)$$

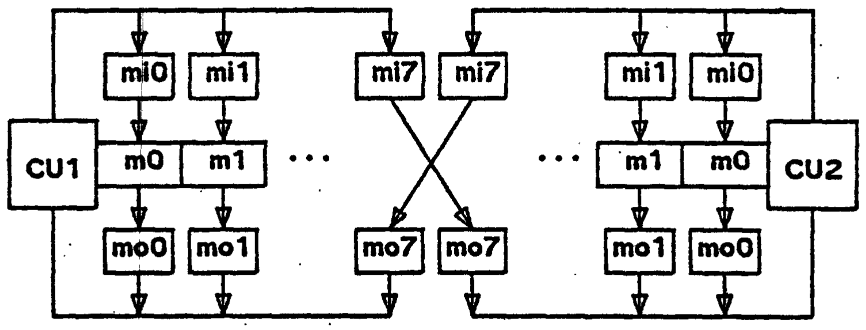
It is convenient to choose MV as the largest power of 2 which satisfies this constraint. Δt is chosen to provide sufficient audio bandwidth in the output signal, and the minimum values for t_r and t_w are determined by available semiconductor memory technology. Choosing $\Delta t = 2^{-15}$ second as discussed above, we find that currently available random access semiconductor memories using standard TTL technology have the characteristic:

$$t_r + t_w \approx 100 \text{ ns.} \approx 2^{-23} \text{ second}$$

Thus we have $MV \leq 2^8 = 256$. Choosing $MV = 256$ maximizes the number of virtual functions available. We now have to decide on the M vs. V tradeoff, keeping in mind that increasing M increases the amount of hardware that has to be built (module input-output connections), and increasing V increases the amount of hardware that does *not* have to be built. Choosing M too small unduly restricts the number of simultaneously available modules in the synthesizer, while choosing M too large both wastes hardware and limits the economy gained through time multiplexing the available functions. Setting $M = 8$ and $V = 32$ seems to be a reasonable compromise, and yields a module service period equal to $\Delta t/V \approx 1$ μs. The other leading possibility was to choose $M = V = 16$, which would have doubled the number of module connections (a significant portion of the cost of the synthesizer), relaxed the module service period to 2 μs, and yielded half the time multiplexing factor per function. It was felt that since most synthesis algorithms seemed to require a large number of simple functions that setting $M = 8$ was a better initial choice. It is clear, though, that the basic architecture of the control unit can be used to service anywhere from 1 extremely speedy module, multiplexed by a factor of $MV = V$ times, or $MV = M$ slow modules which are not time multiplexed at all. The latter case may become more interesting as microprocessors become both more inexpensive and powerful enough to accomplish useful signal processing calculations within Δt seconds.

Properties of Interconnection Schemes

Two important properties of the control unit architecture are that it makes no particular assumption about the internal operation of any module, and that multiple control units may be easily interconnected via module input-output (I/O) connections, satisfying the self-extensibility design criterion and allowing a synthesis capability of arbitrary size to be "assembled." Assuming that one module I/O pair is used to interconnect each control unit to each of two other control units then N control units provide $7N$ module I/O connections. Any given control unit can easily exchange 32 signals directly with adjacently-connected control units (see Figure 3-5); other interconnection schemes are possible if this is insufficient, with the total number of possible modules being traded against the constriction of the intercommunication of control units.



LEGEND:
CUN - control unit N,
miN - module input register N,
mN - module N,
moN - module output register N.

FIGURE 3-5: Interconnection of two control units to achieve a more extensive synthesis capability: Two independent synthesizers may intercommunicate if one or more module input registers on one control unit are connected to module output register(s) on the other control unit. This figure shows a possible way of interconnecting two control units which allows two-way communication of up to 32 different signals from each synthesizer. Any input which is selected for module 7 on one machine appears as the output of module 7 on the other. Other possibilities include one-way communication, or, more than two control units may be interconnected (see text). Of course, the computer must supply control information to all synthesizers.

This particular control unit architecture was chosen because of its conceptual simplicity and ease of implementation, though several other data-interchanging and time-multiplexing schemes are possible. We might consider the function of the control unit from a more general viewpoint in order to see which subset of a more general capability seems most desirable.

The present control unit does not allow inputs to a particular module to come from more than one place at once: a single "input-gather" scheme, and outputs are placed in fixed locations in the signal-interchange memory. A more general scheme would allow any combination of, say, j module outputs to be connected to any combination of, say, i module inputs. For full generality this would require $i \times j$ connection points, with the property that if more than one module output is connected to a single input, then these signals are combined in some appropriate way (usually they would be added together). Since these $i \times j$ connection points are all either on or off (connected or disconnected), there are $2^{i \times j}$ possible states in the interconnection matrix (see Figure 3-6). In the present system, $i = j = 256$, and clearly the number of possible states would be rather large if a fully general interconnection scheme is chosen.

The way in which this general interconnection problem is solved depends on the types of interconnections which are expected to be commonly used. A typical scheme is simply to restrict the interconnections so that one module output may be selected by any number of module inputs ("input gathering"), or one module output may be directed to multiple module inputs ("output scattering"). Either of these schemes correspond to allowing either only one connection per row or per column in the interconnection matrix, respectively. The choice here is between letting the addresses in the control memory M_c represent where

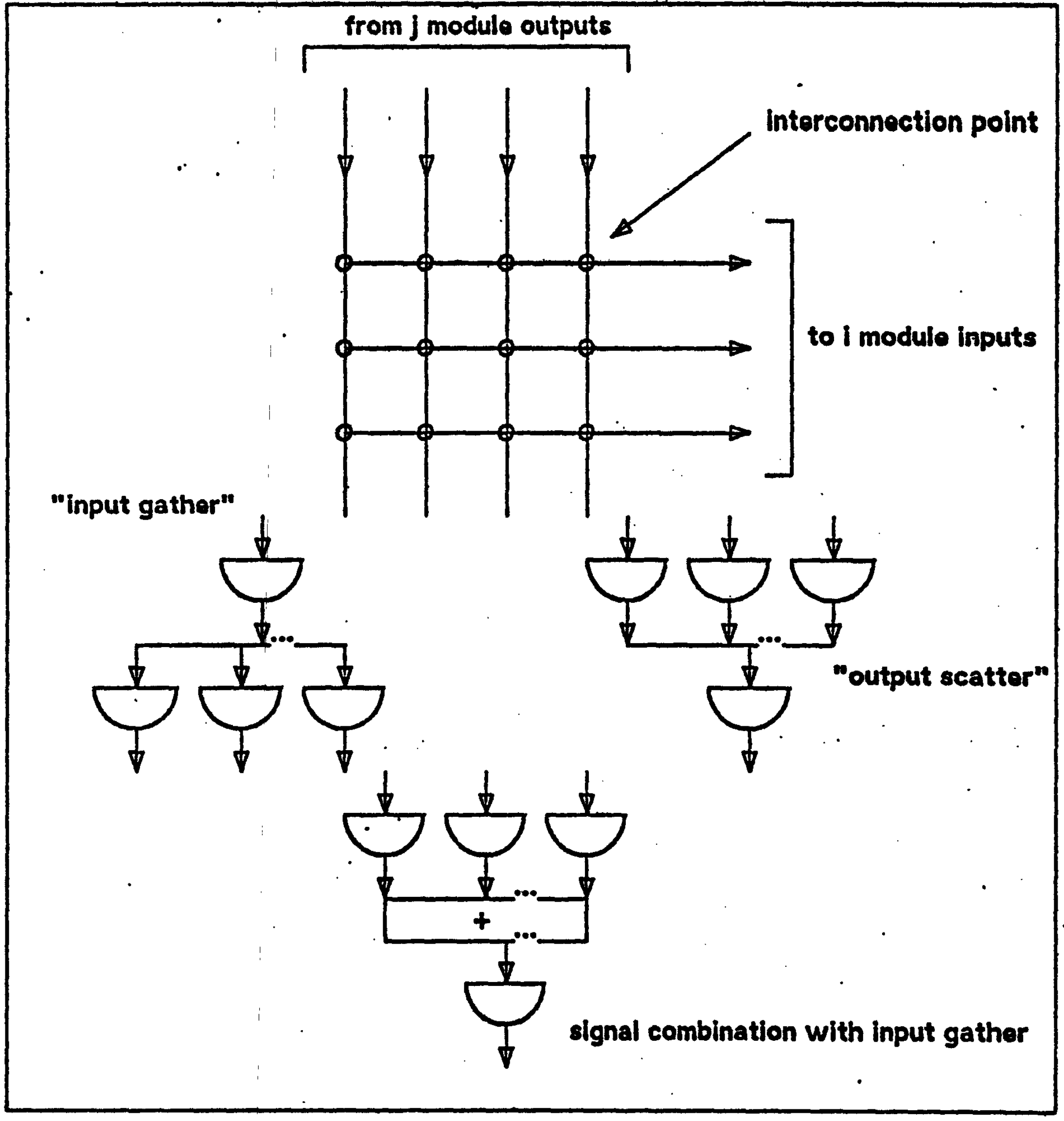


FIGURE 3-6: General solution to the interconnection problem for j module outputs and i module inputs is shown at the top: Any signal coming into the switching matrix can be connected to any combination of lines leaving the matrix; if more than two signals leave on the same line, they are "combined" (e.g., added) first. "Input gathering" (left middle) allows multiple fan-out connections, while "output scattering" (right middle) facilitates fan-in connections. Using "input gathering" requires signal combination to be performed by the module functions (bottom).

the input for a particular virtual function comes from (out of a fixed menu of choices) or where its output is to be placed (into a fixed set of input slots). In either of these cases the number of possible interconnection states is reduced to either $i \log_2 j$ or $j \log_2 i$ depending on whether input-gather or output-scatter is used. Two problems arise with the output scatter technique, however: two outputs may be directed towards the same input, requiring that some special combining operation such as addition be performed, and a single output may not be connected to several inputs in any convenient way. The latter problem is clearly not present with the input-gather technique, and the problem of combining several outputs can be handled by circuitry external to the signal interchanging mechanism. Since the ways in which musical signals are combined often requires something other than simple addition (such as weighted sums, multiplication, etc.) it is clear that input-gathering is the preferable technique for most cases. The only synthesis technique for which a large number of additions must routinely be performed is additive synthesis, and this can still be accomplished via module functions.

Several other solutions to this interconnection problem exist, usually involving considerably more hardware than the present synthesizer design. For example, the Systems Concepts synthesizer contains two so-called "sum" memories to allow for both input and output addressing simultaneously: any of the 392 signal sources can direct its output signal to any of 64 "sum" memory locations, where it is added to the signals already accumulated there since the last sample was output by the synthesizer. Any of these 392 processes can also "read" the contents of any of these memory locations as well, so 64 perfectly general interconnection paths are available in this machine.

The only other simple scheme for interconnection known to the author at present has been suggested by P. DiGiunnio of I.R.C.A.M. in Paris: a number of fast modules are connected to a single two-way data bus. A control unit for the bus is also connected directly to every module; the control unit can cause any module to either accept data from the bus and begin processing, or to output a previously calculated piece of data onto the bus. The control unit contains a list of sequencing commands of the form:

Move output of module A *to input of module* B .

Thus each module can be activated whenever it is needed; in fact, virtually any function mixture can be obtained in this manner. However, it is difficult to take advantage of parallel operation of the modules, since this would make the programming of the control unit dependent on the module timing.

Programming the Control Unit

The advantages of the control unit design presented here are its economy and simplicity. It allows the modules to operate more slowly than a sequential operation scheme such as the one presented above by operating all modules simultaneously — most high-speed operations are quarantined in the control unit. It is conceptually simple and easy to program, and provides the basic interconnection capability needed for all synthesis schemes. Its main disadvantage is that it requires the number of virtual occurrences of each module per sample period to be constant (32), but this can be partially offset by the use of modules with more than one function, which will be fully discussed below.

Since the control unit makes no assumption about any module's internal operation, it is possible for the modules to use the capabilities of the control unit to the fullest extent. For instance, if a module function cannot be performed within a module service period, two solutions are possible. One is simply to perform the operation in whatever time it takes, not producing a valid output every module service period. If 2 periods are required, then the module simply operates as if it had a time multiplexing factor of 16 instead of 32, with the result that only half the signals supplied to M_s by the module would be valid. In general the operation speed of a module should be a power of 2 times the module service period so that fixed locations in M_s will contain valid data. For such a module,

$$A = [8S(V + 1) + M] \bmod 256$$

where A is the absolute address of the *output* of the Vth virtual occurence of module M, which requires S module service periods per operation ($1 \le S \le 32$; S a power of 2).

Another possible approach is to pipeline the module, so that a new output is produced every module service operation, but that output is associated with the inputs received by the module P + 1 module service operations ago, $0 \le P \le 31$. Then

$$A = [8(V + 1 + P)] + M\} \bmod 256$$

This technique, where applicable, does not reduce the number of virtual operations available, though it does increase the complexity of the module design somewhat. It is expected that both slower-operating and pipelined modules will be employed to implement functions which cannot be performed entirely within one module service period.

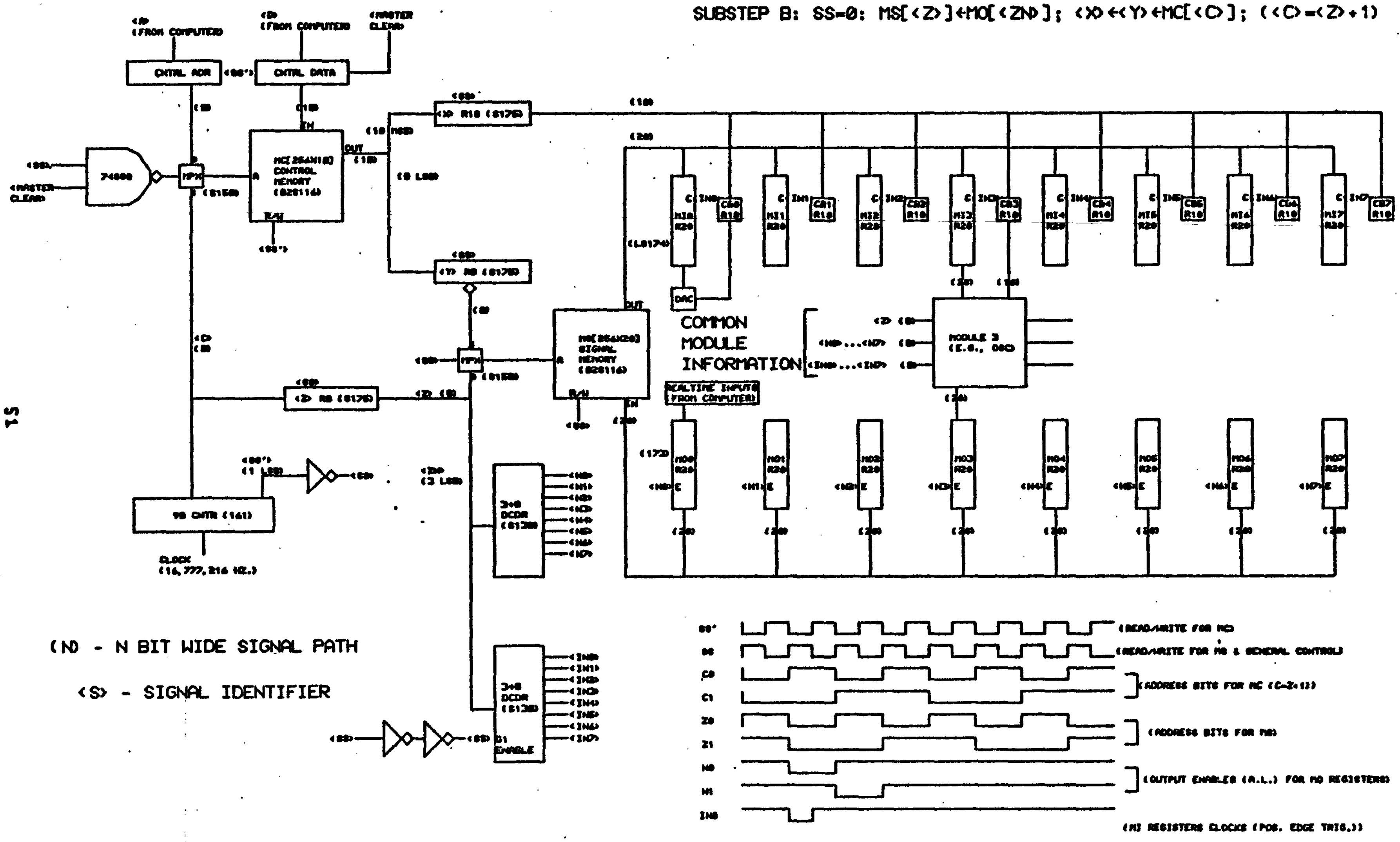
The general formula relating the address A in signal memory M_s of the output of module M, virtual occurence V, with operation speed S and pipeline factor P can be stated:

$$A = [8S(V + 1 + P) + M] \bmod 256$$

BASIC STEP (DONE EVERY 119.20 NS FOR 32,768 SAMPLING RATE):

SUBSTEP A: SS=1: MI[<ZN>]←MS[<Y>]; CB[<ZN>]←<X>; MC[<A>]←<D>;

SUBSTEP B: SS=0: MS[<Z>]←MO[<ZN>]; <X>←<Y>←MC[<C>]; (<C>=<Z>+1)



CNTRL ADR
CNTRL DATA
74S00
MPX
MC[256X18] CONTROL MEMORY (82S116)
MS[256X20] SIGNAL MEMORY (82S116)
R10 (S175)
R8 (S175)
9B CNTR (161)
CLOCK (16,777,216 HZ.)
3→8 DCDR (S138)
DAC
COMMON MODULE INFORMATION
REALTIME INPUTS (FROM COMPUTER)
MODULE 3 (E.G., OSC)
(READ/WRITE FOR MC)
(READ/WRITE FOR MS & GENERAL CONTROL)
(ADDRESS BITS FOR MC (C=Z+1))
(ADDRESS BITS FOR MS)
(OUTPUT ENABLES (A.L.) FOR MO REGISTERS)
(MI REGISTERS CLOCKS (POS. EDGE TRIG.))

(N) - N BIT WIDE SIGNAL PATH

<S> - SIGNAL IDENTIFIER

SYNTHESIZER CONTROL UNIT | FIGURE 3-7 | 04-APR-77 21:10 | CU[H, FRM]

Figure 3-7 shows a detailed block diagram of the synthesizer control unit as currently implemented. A 9 bit counter is used to control the M_c and M_s memories, and to provide timing and control signals to the modules. The least significant bit of the counter (<SS'>) determines which of 2 basic substeps is executed. On the first substep, the module inputs are supplied with information read from M_s and the control byte portion of M_c. While this is being done, M_c is also updated with information supplied by the control computer. During the second substep, information is written from the module output into M_s, while addressing and control information from M_c is being read. Representative control signals are shown, as well as representative TTL part numbers for each of the integrated circuits used in the control unit's fabrication.

CHAPTER 4 - MODULE SETS

The function of the hardware synthesizer is determined by the set of modules which are connected, and by information supplied from the control computer. It is necessary for the software in the control computer to be compatible with any particular set of modules, but no great difficulty is presented in making the control software general enough to handle most module functions. The partitioning of the synthesizer into control unit and functional modules makes it straightforward to design the control software to accept information about 1) the position of any module (slot 0 through 7) inputs and outputs, 2) its operation speed (i.e., whether it produces a valid output every module service period, and/or whether the output is delayed due to pipelining), and 3) what control bits are accepted by the module, for multifunction or multimode modules.

Stream Processing

One of the most important assumptions which has been made in the design of this synthesizer is that it is intended to be used as a "stream processor." Much specialized hardware for digital signal processing has been constructed around the "block processing" idea in which a "block" of digital samples is placed into a the memory of the special hardware. This data is then processed as rapidly as possible according to some fixed, programmed algorithm of which the Fast Fourier Transform (FFT) is a common example. If the algorithm can be executed quickly enough for a given size of data block, then real time operation is made possible.

In a "stream" processor everything happens on a sample-by-sample basis, and it is uncommon to operate on a large amount of data at any one time. In fact, the only time a large number of samples is held in the synthesizer's memory is when a sizable delay is required for such operations as filtering or reverberation. The timing of the "stream" processor is also totally independently of the computer to which it is attached; data is sent to the synthesizer *at the actual moment* when it is desired to change a parameter, such as the frequency of an oscillator. The oscillator affected will then produce that frequency continuously until it is set to another value by the computer. It is difficult to characterize the bandwidth of the data channel from the computer to the synthesizer because of its time varying requirements, but the worst case occurs when the computer has to update (i.e., change) every parameter in the synthesizer at a particular instant in time. Fortunately this occurs only infrequently, typically during the initial setup of the synthesizer at the beginning of a real time run.

Module Sets and System Function

There is absolutely nothing about the architecture of the synthesizer which restricts its use to music. The control unit makes no assumptions about the internal workings of any module (other than general speed), or the types of data it accepts or produces. The 20-bit input and output data may represent a 2's complement fractional values, floating point numbers, or sets of binary-coded characters. Also, modules need not be — in fact they are typically not — independent of each other. In principle, a single-input module can perform any realizable function on that input:

$$y \leftarrow m(x, c)$$

where: y is the output (signal) of the module,

x is the input (signal),

c is a (control) parameter, and

m is the (module) function.

Since the module is capable of making conditional decisions, a general formulation of the effect of control parameter c is as follows:

$$\text{if } c = 0 \text{ then } y \leftarrow m_0(x)\,,$$
$$\text{if } c = 1 \text{ then } y \leftarrow m_1(x)\,,$$
$$\vdots$$
$$\text{if } c = j \text{ then } y \leftarrow m_j(x)\,,$$

or, more compactly:

$$y \leftarrow m_c(x)$$

If c is a k-bit number, then any of 2^k different functions may be specified to determine how x is mapped into y. For example, $m_0(x)$ may be $\sin x$, and $m_1(x)$ may be $\cos x$. Such functions could be implemented using straightforward table look-up techniques. Simple digital arithmetic circuitry allows the implementation of such functions as $m(x) = -x$, or $m(x) = x^2$. If the module contains memory, such functions as $m(x_n) = x_{n-1}$ (the unit sample time delay operation), $m(x_n) = x_{n-d}$ (an arbitrary time delay of d samples), or $m(x_n) = (x_n + x_{n-1})/2$ (a simple averaging low-pass filter) may be implemented.

Low Level Arithmetic and Memory Modules

Memory on an arithmetic module can be arranged for use as a stack, which allows the computation of results which require more than one operand, such as the product of two numbers. A stack is a LIFO (last-input-first-output) list; items are "pushed" onto the top of a stack as if it were a pile of pancakes, where only the top pancake is available at any particular time. Items can be "popped" from the top of the stack (removed), revealing the item lying "underneath." Typically, the top two items on a stack are combined with a binary (i.e., two-operand) arithmetic operation, such as addition, the stack is "popped," and the top item on the stack is replaced with the sum of the previous two top items on the list. Modules can "push" values onto such a stack (the output, y, produced by this operation is usually not defined), while other functions can combine the input value with the top item on the stack in a number of ways. This provides a quite general calculating capability using a single-input module.

Since the control parameter in the current implementation is specified with ten binary digits, there are 1024 possible functions specifiable for any module, and each of the 32 virtual occurrences of this module may choose independently from this size menu (not all 1024 functions are necessarily defined). The choices must be consistent with the way the module memory operates, since some memory may be assigned uniquely to each virtual occurrence (such as a delay memory for a digital filter), while other memory can be shared among all 32 virtual occurrences of the module (such as a stack).

Finally, the input to any virtual occurrence of a module may come from the output of any virtual occurrence of any module, so module functions may be composed (in the mathematical sense) to create new ones. For example, if $m(x) = x^2$, then clearly selecting the output of this function as the input to another virtual occurrence of the module with the same function results in $m(m(x)) = m(x^2) = x^4$. Using a stack memory we can define module functions such as:

$m_{\downarrow}(x) =$ *(undefined)*, (push x onto stack S),

$m_{+}(x) = x + T$, (add x to T, the Top item on S, pop S, and replace T with the new sum),

$m_\times(x) = xT$, (multiply x by T, pop S,

and replace T with the new product), and

$m_s(s) = sin\ x$, (look up $sin\ x$, and replace T

with the new value).

We can then calculate the function $A\ sin\ \theta$ with the sequence: $m_\downarrow(A)$, $m_\times(m_s(\theta))$, meaning that we first execute $m_\downarrow(x)$ with x set to A, then execute $m_s(x)$ with x set to θ, and finally execute $m_\times(x)$ with x set to the output of the previous function. With one additional function for computing a sum-of-previous inputs:

$$m_\oplus(x_n) = x_n + y_{n-1}\ mod\ 2^p,$$

in which the output is the sum of the current input and the previous output (this function does *not* use the stack memory), and p is determined by the wordlength of the register used to perform the addition (the modulus operation is obtained by ignoring the overflow from the addition), we could perform the following sequence:

$$m_\downarrow(A),\ m_\downarrow(m_\oplus(\omega_c)),\ m_\downarrow(\Delta\omega),\ m_\times(m_s(m_+(m_\times(m_s(m_\oplus(\omega_m))))))$$

This says first to push A onto stack S, then to perform $m_\oplus$ on the operand ω_c and to push the result onto S, etc. The result of all of these operations will be a frequency modulated waveform with amplitude A, carrier frequency proportional to ω_c , modulating frequency proportional to ω_m , and modulation index $\Delta\omega/\omega_m$, as the reader may wish to verify. A repertory of just 5 module functions is required (namely $\downarrow$, $+$, $\times$, $\oplus$, and s), and 10 operations, or virtual occurrences, of which 32 are available from any one module.

A stack is not needed to perform multiple-operand operations if two or more modules cooperate in generating a result. If two modules intercommunicate with each other, addition or multiplication can be performed directly (i.e., without a stack). The output of one cooperating module could be the sum of the inputs to both, for example, leaving the other output either unused, or defined to have some valid, simultaneous meaning. We could arrange to have both the sum and product of two values at once. A dual-input, dual output "supermodule" would receive a total of 20 bits of control input, allowing over one million supermodule functions to be specified. The control unit "sees" this supermodule simply as two separate modules, each having its input and output serviced in the usual manner. Of course supermodules would take up two or more of the eight available module slots, but, as we shall see later, such extravagance can be well worth the price.

The use of low-level arithmetic and memory modules has two main properties: First, a high degree of generality is obtained due to the arbitrary ordering and interaction (through the stack) of up to 32 operations chosen from a possible repertory of 1024 functions. Second, almost any useful computation is likely to take a fairly large number of such low-level steps (virtual occurrences). As usual, generality turns out to be the foe of efficiency.

There is no absolute efficiency requirement, however, and even these rather inefficient but general low-level modules may be quite useful in many applications. They are simple to design and build (compared with some of the high-level modules discussed below), and they may provide a powerful tool for module design research. Also, a low-level module may be useful when imbedded in a set of higher-level modules such as oscillators or filters for performing occasionally needed operations not available from the other modules, such as direct table look-up, a one or two sample delay, subtraction, logical operations such as *and*ing and *or*ing, and so on.

The Semantics of Module Description

In order to discuss the operation of modules, we need to adopt a notation for describing their input-output connections:

$X_j[0{:}19]$ refers to bits 0 through 19 of the j^{th} signal input for a module, with bit 0 being the most significant (leftmost) bit,

$C_j[0{:}9]$ refers to the 10 control bits associated with the j^{th} signal input,

$Y_j[0{:}19]$ refers to the 20 bits of the j^{th} signal output of a module,

Z[0:7] refers to the 8 bits of timing information supplied to every module specifying the current module being processed, M[0:2], and the current virtual occurrence of that module, V[0:4], where M[0:2] = Z[5:7] and V[0:4] = Z[0:4], and

N0, N1, ... N7 and IN0, IN1, ... IN7, which are the 16 lines of decoded timing pulses (see Figure 3-7). These are wired so that each module is serviced when its own N0 line is pulsed, thus allowing the timing of each module to be independent of the slot into which it is plugged.

In addition to these 56 input-output connections, each module slot has the necessary power and grounds, an output register clock (the output registers are clocked by the module circuitry), and several "tie" connections (there are a total of 120 connections available) to allow for intermodule communication (at present, module slots 1 and 2, 3 and 4, and 5, 6, and 7 are "tied" together with 36 connections).

Unless otherwise stated, it will be assumed that the signals processed and interchanged in the synthesizer are 2's complement, fixed point binary fractions. While 20 bits are provided in the signal paths, most signals will be presumed to be represented to 16 bits of precision, right-justified in the 20 bit field. This allows 4 bits to be used as "head room" for overflow, which is often important in digital filtering and/or signal mixing. In most cases, a module will simply ignore the 4 most significant bits of its signal input and output connections.

Armed with these conventions, we can now discuss the operation of sets of modules to perform various real time functions. Our main interest is in the configuration of the machine as a music synthesizer. However, a brief description of the modules needed to configure the machine for several other uses will also be discussed, including a stream-oriented general real time signal processor and a speech synthesizer. The music synthesizer configuration is discussed first, and in most detail.

Music Synthesizer Modules

Ideally, a music synthesizer should be able to perform the additive, subtractive, and nonlinear synthesis algorithms conveniently. The following set of modules will fit into a single control unit synthesizer, and can perform these algorithms: 1) a dual input, dual output, multifunction oscillator module, 2) a dual input, dual output, multifunction envelope function generator module, 3) a dual input, dual output digital filtering module, and 4) a single input, single output reverberation module. The remaining module input slot is used for an output collector module which collects, scales, and distributes signals to as many channels of audio output as are available. The remaining module output slot is used to interface to the control computer for the purpose of supplying real time control signals to the rest of the modules (more than one output slot can be used for this purpose if available). Some of these modules will now be considered in detail.

Basic Method of Frequency Generation

Several means of frequency generation have been reported in the signal processing literature [Rabiner and Gold, 1975], including generation of sinusoids from difference equations, complex arithmetic involving the number *e* to an imaginary exponent, etc., but the most economical method by far seems to be the table look-up method already described in Chapter 2. The basic hardware elements required to implement this method are shown in Figure 4-1. A number determining the frequency, called the *increment*, is repeatedly added to a register (called the *sum-of-increments*, or *phase* register) at each clock pulse, where the clock frequency is here taken to be equal to the sampling rate. By a suitable choice of register width and table length, the modulus arithmetic is automatically performed by ignoring the register overflow, and the contents of this register are used to address a

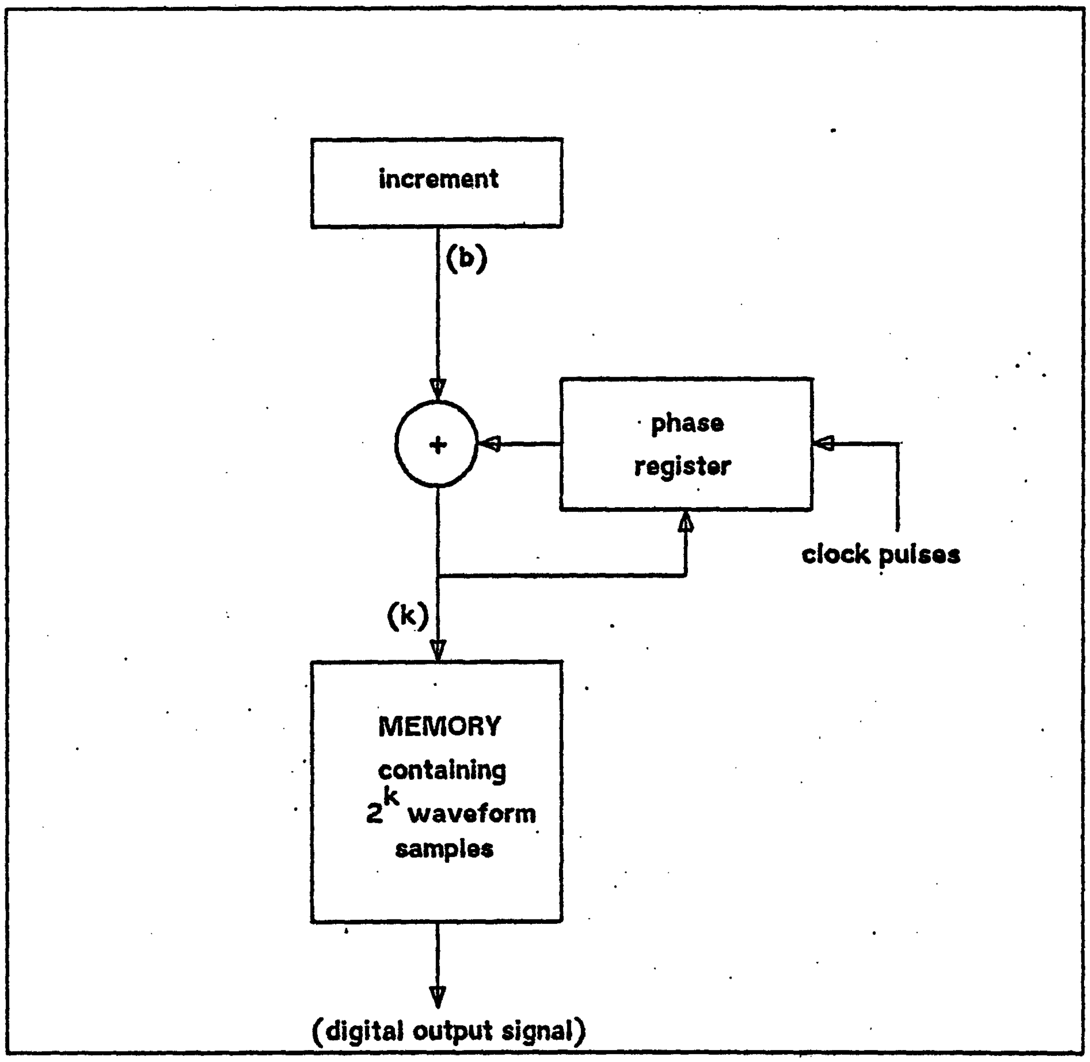


FIGURE 4-1: Block diagram of the hardware required for the table look-up oscillator. At each clock pulse (where the clock frequency is equal to the sampling rate), the signed, b-bit increment is added to the sum-of-increments, or phase, register. The uppermost k bits of this phase value are used to address a random-access (possibly read-only) table memory containing samples of one period of the waveform to be produced. Extensions to the basic method shown above include rounding the k-bit address by the (k+1)st bit of the phase value, or interpolating between adjacent table entries. The quality of the digital output signal is affected by slight errors due to the finite length of the table (called "phase jitter") and amplitude quantization errors due to the finite wordlength of the table entries. Measurements of the effects of these errors are given in tables 4-1, 4-2, and 4-3 for three different table-addressing schemes: truncation, rounding, and interpolation.

waveform memory containing 2^k samples of one period of the (periodic) waveform to be generated. The exact method of forming the table address is important in determining the quality of the digital output signal. The increment value is related to the frequency of the output waveform by the formula [Mathews et al., 1969]:

$$increment = \frac{(frequency)\,(table\ length)}{(sampling\ rate)}$$

which implies that

$$frequency = \frac{(increment)\,(sampling\ rate)}{(table\ length)}$$

We see that the increment represents the *size of the step* taken through the table for successive values of the output waveform. If the increment is equal to 1.0, each period of the output waveform will contain exactly as many samples as there are in the table, yielding a frequency equal to the sampling rate divided by the table length. Doubling the increment doubles the frequency, and vice versa. However, most frequencies do not yield integer values for the increment, so normally one of three methods is used to handle such cases: the *truncation* method, in which the integer part of the sum-of-increments is used directly to address the table, the *rounding* method, in which the sum-of-increments is rounded to the nearest integer, which is then used to address the table, or the *interpolation* method, in which some form of interpolation between successive table values is formed.

How Large Should the Sine Table Be?

In order to determine how large to make the sinusoid table (i.e., how many entries and how much precision it need have) we can compare the signal quality for each of the three methods given above, truncation, rounding, and interpolation, to the quality of an "ideal" digital waveform. This signal is "ideal" in the sense that it is computed to the full accuracy of a 36 bit general purpose computer at each sample, so that both "phase jitter" due to the finite length of the table and quantization noise are at minimum values. Further, it is assumed that these minimum values are well within the tolerance levels of acoustic perception, i.e., that the "ideal" digital waveform *sounds* as pure as the analog audio equipment will allow.

The method for making the comparison is as follows: the addressing mode is chosen, and the table length and width are set to selected values. Only table lengths which are powers of two are considered because of their convenience in hardware implementation. A computer program then simulates the operation of the hardware built to these specifications and the computed waveform is compared against the "ideal" waveform. The percentage root-mean-square amplitude of the error signal is determined according to:

$$\textit{Percentage RMS error} = \psi \sqrt{\frac{\sum_{n=0}^{k-1} \{ f_{ideal}(n) - f_{test}(n) \}^2}{k}}$$

where:

$f_{ideal}(n)$ represents the n^{th} of k samples of the "ideal" waveform at times nT, $n = 0, 1, \ldots (k-1)$, and T is the sampling period,

$f_{test}(n)$ is the analogous waveform generated by the hardware simulator, and

ψ is a normalizing factor, = 100/.707 to yield RMS in percent.

It is found that the RMS error depends on the relationship of the table length to the frequency of the generated sinusoid. By trying several frequencies the "worst case" error can be determined -- in most cases the worst case appears when the increment values are close to but not equal to 1.0 . If the error signal itself is considered to be a noise added to the "ideal" waveform, and if the amplitude of this noise is compared with the overall amplitude of the signal, we can infer a signal-to-error noise ratio (SN_eR) and the results of such calculations are shown in Tables 4-1, 4-2, and 4-3.

In Table 4-1 the SN_eR values are given in dB for the truncation method, and the last value in each row indicates the largest ratio achievable for a given table length. Apparently adding more bits of precision does not increase this value because the error is then principally due to the finite length of the table, resulting in instantaneous phase errors in the test signal. Inspection of the table yields the rule that when the truncation method of table look-up is used, if the table is 2^k words long, then the signal can benefit from no more than k bits per table entry (*not* including a sign bit), and the limiting SN_eR is approximately $6(k-2)$ dB. This makes intuitive sense from an information theoretic viewpoint, since we would expect that k bits of information input cannot yield more than k bits of information output. Similar results are shown for the rounding and interpolation methods in Tables 4-2 and 4-3. It can be seen that rounding increases the SN_eR by about 6 dB compared to truncation for a given table size, since rounding allows the output signal to benefit by one more bit of table precision. (Again, from the standpoint of information theory, we "take account" of $k + 1$ bits of input, so we get $k + 1$ valid bits of output.) Interpolation greatly increases the output signal accuracy and is clearly of interest when memory must be conserved. Interpolation requires additional arithmetic and memory accesses, however, so it is intrinsically more time-consuming than either of the other methods. The method of interpolation is as follows: if we take i and f to be the integer and

SIGNAL-TO-ERROR NOISE RATIOS (SN_eR) FOR THE TRUNCATING OSCILLATOR

by waveform memory size .
(table entries are in dB)

MEMORY WORDLENGTH (in bits, not including the sign bit) →

MEMORY LENGTH (one period, in words) ↓

	4	5	6	7	8	9	10	11	12
32	18.4	18.6							
64	24.1	24.1	24.3						
128	27.9	29.8	30.0	30.4					
256	30.5	34.2	35.6	36.3	36.3				
512	31.7	36.6	40.0	41.6	42.1	42.4			
1024	32.0	37.7	42.7	46.1	47.7	48.2	48.4		
2048	32.1	38.0	43.6	48.6	52.1	53.6	54.2	54.4	
4096	32.1	38.0	43.9	49.6	54.6	58.1	59.7	60.2	60.4

Rule for truncating oscillator: if the table has 2^k entries, then each entry optimally should be specified to k bits of precision (not including the sign bit), and the SN_eR achievable will then be 6(k-2) dB.

Table 4-1: Worst Case Signal-to-Error Noise Ratios (SN_eR) for the truncating oscillator. If the truncating addressing scheme is used with a waveform table containing 512 samples of one period of a sinusoid, and each sample has 8 bits of magnitude, the noise due to "phase jitter" and quantization errors will be 42.1 dB below the signal level, according to this table (see text for an explanation of how these values are calculated). No improvement is gained by adding more bits of precision to the table beyond the last value in each row of the above table, since at this point virtually all the noise in the digital signal is due to the finite table length. Using this data it is possible to design digital hardware or software with "optimum" waveform memory sizes.

SIGNAL-TO-ERROR NOISE RATIOS (SN_eR) FOR THE ROUNDING OSCILLATOR

by waveform memory size
(table entries are in dB)

MEMORY WORDLENGTH (in bits, not including the sign bit) →

MEMORY LENGTH (one period, in words) ↓

	4	5	6	7	8	9	10	11	12
32	22.7	23.4	23.5						
64	27.4	28.9	29.7	30.2					
128	30.5	33.8	35.2	36.2	36.3				
256	31.6	36.7	39.9	41.7	42.1	42.3			
512	32.1	37.7	40.5	45.9	47.6	48.2	48.3		
1024	32.1	38.0	43.6	48.7	52.0	53.6	54.2	54.4	
2048	32.1	38.0	43.9	49.6	54.7	58.0	59.7	60.2	60.4

Rule for rounding oscillator: if the table has 2^k entries, then each entry optimally should be specified to k+1 bits of precision (not including the sign bit), and the SN_eR achievable will then be 6(k-1) dB.

Table 4-2: Worst Case Signal-to-Error Noise Ratios (SN_eR) for the rounding oscillator. If the rounding addressing scheme is used with a waveform table containing 512 samples of one period of a sinusoid, and each sample has 8 bits of magnitude, the noise due to "phase jitter" and quantization errors will be 47.6 dB below the signal level, according to this table (see text for an explanation of how these values are calculated). No improvement is gained by adding more bits of precision to the table beyond the last value in each row of the above table, since at this point virtually all the noise in the digital signal is due to the finite table length. Using this data it is possible to design digital hardware or software with "optimum" waveform memory sizes.

SIGNAL-TO-ERROR NOISE RATIOS (SN_eR) FOR THE INTERPOLATING OSCILLATOR

by waveform memory size

(table entries are in dB)

MEMORY WORDLENGTH (in bits, not including the sign bit) →

MEMORY LENGTH (one period, in words) ↓

	8	9	10	11	12	13	14	15	16	17	18
32	48.0										
64	57.1	60.1	58.9								
128	62.5	67.5	70.1	71.1	71.3						
256	56.3	61.5	68.9	73.7	79.5	82.3	83.4				
512	56.5	61.6	68.8	73.5	79.8	85.6	92.6	94.8	95.1		
1024	56.1	61.7	68.4	73.9	80.2	85.8	92.3	98.7	103.9	106.0	107.0
2048	56.0	61.9	68.0	74.1	80.2	86.0	92.4	98.4	104.0	110.2	115.4

Rule for interpolating oscillator: if the table has 2^k entries, then each entry optimally should be specified to $2(k-1)$ bits of precision (not including the sign bit), and the SN_eR achievable will then be $6k$ dB.

Table 4-3: Worst Case Signal-to-Error Noise Ratios (SN_eR) for the interpolating oscillator. If the interpolating addressing scheme is used with a waveform table containing 512 samples of one period of a sinusoid, and each sample has 8 bits of magnitude, the noise due to "phase jitter" and quantization errors will be 56.5 dB below the signal level, according to this table (see text for an explanation of how these values are calculated). No improvement is gained by adding more bits of precision to the table beyond the last value in each row of the above table, since at this point virtually all the noise in the digital signal is due to the finite table length. Using this data it is possible to design digital hardware or software with "optimum" waveform memory sizes.

fractional parts of the sum-of-increments register respectively, then interpolation involves evaluation of either:

$$sin(i + f) \approx sintable(i) + f \{ sintable(i + 1) - sintable(i) \} \qquad (1)$$

or

$$sin(i + f) \approx sintable(i) + f\, costable(i) \qquad (2)$$

The second method is based on the fact that $sin\ x \approx x$ for small x expressed in radians. Unfortunately f is not expressed in radians, so this otherwise attractive method does not compare favorably with method *(1)*, which was used to compute Table 4-3.

Frequency Precision

Next, the precision of the increments has to be considered. In order to represent both positive and negative frequencies (the latter occur in frequency modulation synthesis of complex audio waveforms) the increment must be a signed quantity of, say, m bits. If the increment and sum registers each contain m bits, then the frequency range of the oscillator from zero Hertz to the Nyquist frequency is simply divided into 2^{m-1} equal steps. For example, a sampling rate of $2^{15} = 32,768$ Hz. and an increment size of 16 bits (including sign) divides the frequency range of 0 to 2^{14} Hz. into 2^{15} equal steps, each 0.5 Hz. in width. This is sufficient accuracy to be psychoacoustically acceptable above about 200 Hz., but this resolution becomes increasingly intolerable at lower frequencies [Roederer, 1973]. For example, a smooth-sounding, slow *portamento* between about 50 to 100 Hz. would be impossible, since the abrupt changes of 0.5 Hz. are sufficiently large to be audible, producing a kind of "graininess" in the sound of the changing frequency.

Two solutions to this problem are either to make both the increment and sum register longer, or to simply make the sum register more precise and allow the increment to be arithmetically shifted to the left or right, thus making the range and frequency resolution of the oscillator variable. For each bit the increment is shifted to the right, the maximum frequency achievable is halved and the frequency resolution is doubled in accuracy — exactly the kind of relationship desired since frequency resolution is more of a problem at low frequencies than at high. The increment-shifting scheme also minimizes the bit-width of the required signal paths for controlling the oscillator with a time-varying increment control signal. Also, it is possible to apply not only scaling to the frequency specification, but ranging as well. Suppose we are shifting the 16-bit frequency value to the right by one bit. Then we have an upper frequency limit of about *8* kHz. instead of *16* kHz., and a frequency resolution (minimum step size) of *0.25* Hz. instead of 0.5 Hz. If we provide a mechanism for inverting bit *1* (the bit just to the right of the sign bit), then we can select whether the frequencies will be those between *0* and *8* kHz., or those between *8* and *16* kHz. This is a typical application of the control bits which come in with the frequency specification; if some are used to set the scale, then others can be used to set the range. (Note that the range bits must be *exclusive or*-ed into the scaled frequency bits in order to give proper results for 2's complement numbers.)

Economizing Memory

Finally, it is possible to take advantage of the quarter-wave symmetry of the sinusoid to reduce the amount of table memory by a factor of four, with only a slight increase in the amount of hardware needed to form the table address and to interpret the data read from the table. By storing only the first quadrant of the waveform it is also possible to reduce the width of the table by one bit since all of the values of the *sine* function in this quadrant are positive. The details of this method are as follows.

Assume that we have available a W-word memory, where $W = 2^n$, typically 512 or 2048. Assume also that each word in this memory contains B bits of data (the optimum number of bits may be determined from Tables 4-1, 4-2, or 4-3, according to the addressing mode). We fill this memory with positive binary fractions representing the values of the first quadrant of the *sine* function according to:

$$Memory[I] \leftarrow 2^B \sin\{(I\pi)/(2W)\}, \qquad I = 0, 1, 2, \ldots, W-1$$

where each fractional value is *truncated* to B bits of precision. For simplicity, let us assume that the truncation method is used to address the table. Referring to Figure 4-1, we take the uppermost k bits of the phase register with which to address the table. Since our quarter-wave table is only one fourth as long as a full-wave table, we use the uppermost $k = n + 2$ bits of the phase register. The most significant of these k bits (let us call it $k0$) provides the sign of the output value (normally, zero for positive and one for negative), and the second of these k bits ($k1$) determines whether we should negate the remaining n bits in the address for the table (note that $k0$ and $k1$ together form a quadrant number for our *sine* function: 00, 01, 10, or 11). The low-order n bits are used directly to address the table if bit $k1$ is a zero; the *2's complement* of these bits is used if bit $k1$ is equal to one, with one exception: when the low-order n bits are all zero and bit $k1$ is a one (i.e., the phase corresponds to either 90° or 270° *exactly*), then 2's complementing the low-order bits will result in an incorrect address of zero. This particular case must be treated specially by substituting the maximum address (all ones) for this value. This is accomplished in hardware by simply *or*ing the carry-out bit of the 2's complementer with the remaining n bits. It might seem that using a 1's complement rather than a 2's complement here would solve the problem, but unfortunately the more economical 1's complement leads to a distortion of the output waveform around the zero-crossing. The rounding operation may be included in this 2's complementation, if it is used.

The value read from the memory at the calculated address is a positive fractional value upon which we "superimpose" the sign bit by 2's complementing the B bits whenever bit $k0$ is equal to one, and retaining $B + 1$ bits of the result (i.e., we retain the overflow bit), resulting in a $B + 1$ result. Again 2's complementing is necessary to avoid generating the "illegal" fractional value -1.0. Thus a rounding oscillator with a 512 word by 12 bit quarter-wave table can achieve a worst case SN_eR of about 60 dB, which compares favorably to the noise level of high quality analog equipment. Tables 4-1, 4-2 and 4-3 can be used to determine the SN_eR for smaller widths than the optimum as well, for it can be seen that using a 512 word by 11 bit table (instead of 12 bits) has a very small effect on the quality of the generated signal.

A quarter- or half-wave table may be used whenever the waveform to be generated possesses the appropriate symmetry. The table look-up procedure provides the basic method for sinusoid generation needed for music synthesis, and its implementation in hardware is relatively straightforward. We can now consider its modifications and extensions.

AM, FM, and Band-limited Pulse Modes

For additive synthesis it is particularly useful to add a multiplier to control the amplitude of the generated waveform, thus allowing the generated waveform to be amplitude modulated by another waveform. One mode of the dual-input oscillator module is therefore the AM mode, in which one input controls the frequency and the other controls the amplitude of a waveform (see Figure 4-2). The AM mode is useful for the control of overall amplitude, for the generation of musical *tremolo*, and for spectral control, since

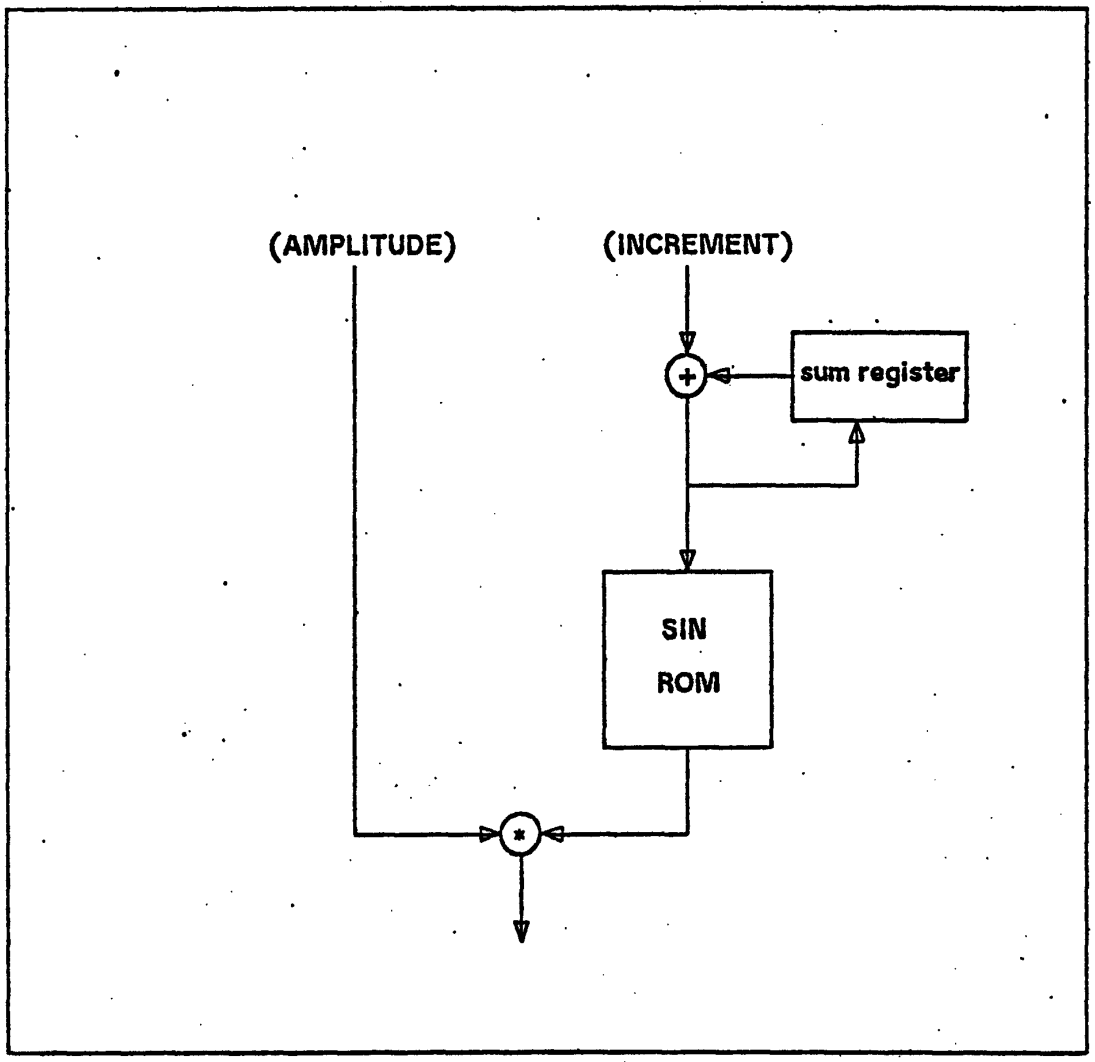


FIGURE 4-2: Block diagram of the hardware element interconnection for the table look-up oscillator running in AM (amplitude modulating) mode. (Modes are selected by setting appropriate control bits). The sinusoid generated is multiplied by the (possibly time-varying) value at the amplitude input. This allows general amplitude control, tremolo, and frequency-domain convolution of the signal present at the amplitude input with a sinusoid at the frequency specified by the increment, analogous to the electronic music effect called "ring modulation."

multiplying two waveforms in the time domain produces another waveform with a spectrum equal to the *convolution* of the spectra of the multiplied waveforms, analogous to the electronic music effect called "ring modulation."

Another simple and useful extension (the modes are selected for each virtual function of the oscillator module by setting appropriate control bits) is to allow the frequency of the generated sinusoid to be equal to that specified by the *sum* of the two input signals, thereby allowing frequency modulation (see Figure 4-3). This can be used to provide the musical effects of *vibrato* and *glissando*, as well as the realization of the FM equation, repeated here for convenience:

$$f(t) = \sin\{\ \omega_c t + \frac{\Delta\omega}{\omega_m} \sin(\ \omega_m t\ )\}$$

where: ω_c *is the carrier frequency* $(= 2\pi f_c)$,

$\Delta\omega$ *is the peak frequency deviation, and*

ω_m *is the modulating frequency* $(= 2\pi f_m)$.

One input is used to determine f_c , the carrier frequency, while the other is used to supply the modulation signal ($\Delta f \sin \omega_m t$), which is the output of another virtual oscillator function running in AM mode with a frequency of f_m and an amplitude of $\Delta f/f$. This FM procedure provides an efficient means to compute a rich variety of musically useful waveforms [Chowning, 1973].

A more complex circuit allows for the production of the so-called band-limited pulse waveforms, which have a finite number of components, thus allowing spectrally rich waveforms to be generated without foldover, which causes any frequency components higher than the Nyquist frequency to appear "aliased" at a lower, undesired, frequency. The equation is repeated here for convenience:

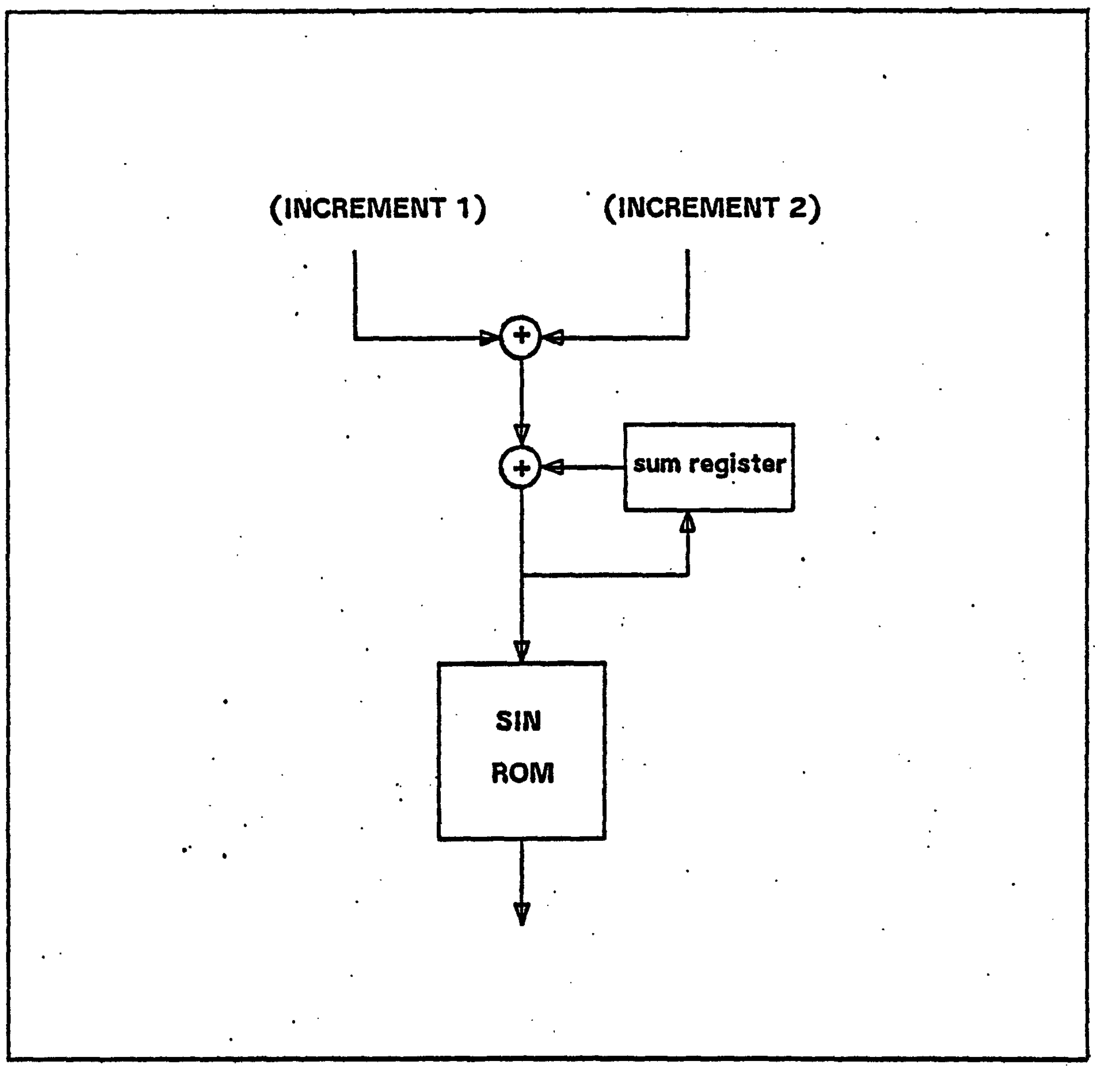


FIGURE 4-3: Block diagram of the hardware element interconnection for the basic table look-up oscillator, extended to allow frequency modulation (FM). (Control bits are used to specify to the module whenever the FM mode is to be employed.) If one of the inputs is constant and the other is sinusoidal, the FM equation discussed in the text is realized, which may be used to produce the musical effects of vibrato or glissando, as well as the generation of complex spectra.

$$S(\theta, n, \beta) = \sum_{k=0}^{n-1} sin(\theta + k\beta) = sin\{\theta + (n-1)\frac{\beta}{2}\}\ sin(\frac{n\beta}{2})\ csc(\frac{\beta}{2})$$

A cosine form for this identity also exists:

$$C(\theta, n, \beta) = \sum_{k=0}^{n-1} cos(\theta + k\beta) = cos\{\theta + (n-1)\frac{\beta}{2}\}\ sin(\frac{n\beta}{2})\ csc(\frac{\beta}{2})$$

If we set $\theta = 2\pi f_1 t$ and $\beta = 2\pi f_2 t$, we can use these relations to generate waveforms with spectra consisting of exactly n equal-strength components, starting at a frequency f_1 and spaced at intervals of f_2 Hertz (Figure 4-4 shows a spectral plot for f_1 = 1000 Hz., f_2 = 500 Hz., and n = 6). We will generally be concerned only with the sine form, for reasons discussed below. Clearly, if $f_1 = f_2$, the waveform will be harmonic and is a band-limited approximation to an impulse train with a period of $1/f_1$ second. If f_1 and f_2 are not integrally related the waveform will be inharmonic, and may be useful in the synthesis of (among other things) percussive tones, such as bells and drums. We may choose any integer value for n in the range

$$1 \leq n \leq \frac{\nu - f_1}{f_2} + 1$$

where ν is the Nyquist rate (equal to one-half the sampling rate).

Calculating the Band-limited Pulse Waveform

The actual calculation of this waveform in digital hardware is complicated by the fact that the amplitude of the waveform is a function of n and by the discontinuities present in

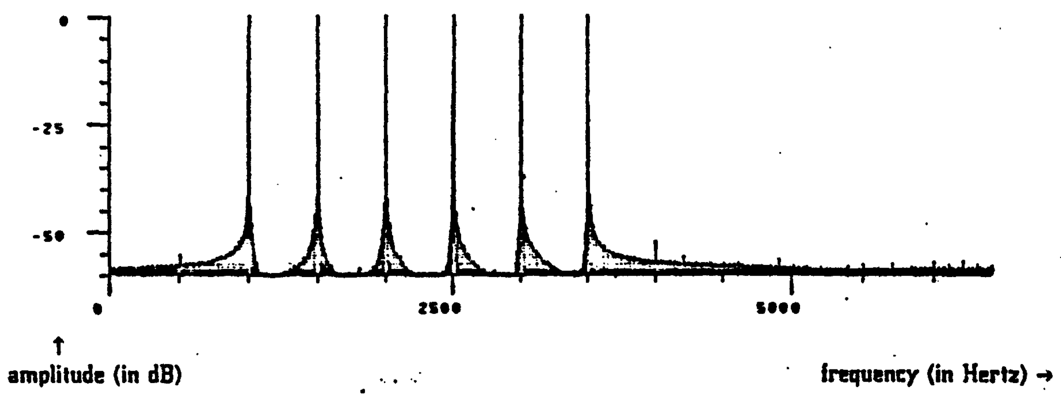


FIGURE 4-4: Spectral plot of a band-limited waveform (see text) with f1 = 1000 Hz., f2 = 500 Hz., and n = 6. Here f1 controls the frequency of the lowest-frequency component, f2 controls the intercomponent spacing, and n is the number of components. The "skirts" on each of the components in this plot are due to the "windowing" effect associated with taking the discrete Fourier transform of a finite number of samples of a waveform.

the cosecant function. In order to discuss the properties of this function let us first consider just the last two terms, which we will call the "band-limiting" function proper, since it is responsible for the generation of the components in the resulting waveform, but not their position on the frequency scale:

$$B(n, x) = \frac{\sin nx}{\sin x}$$

Written in this form, it is easy to see that

$$\lim_{x \to 0} \quad B(n, x) = n$$

since $\sin x \approx x$ for small x. In fact this limit is similar when x approaches any multiple of π, that is

$$\lim_{x \to k\pi} \quad B(n, x) = \pm n$$

where k is any integer and the sign of the limit depends on whether k and n are even or odd (the relationship among k, n, and the sign of the limit will be explained below). In order to determine the spectrum of $B(n, x)$, we can reason "backwards" from $S(\theta, n, \beta)$ by noting that the frequency denoted by $\{ f_1 + (n-1) f_2/2 \}$ is simply the mean frequency of the spectrum produced. By setting this mean frequency to zero (i.e., by multiplying $B(n, \beta/2)$ by a constant, say, 1.0, instead of a sinusoid), we observe that the resulting spectrum still has n components symmetrically spaced around zero Hertz with a spacing equal to $2x = f_2$ Hertz (see Figure 4-5).

To allow for the amplitude dependence on n, we can simply calculate the normalized

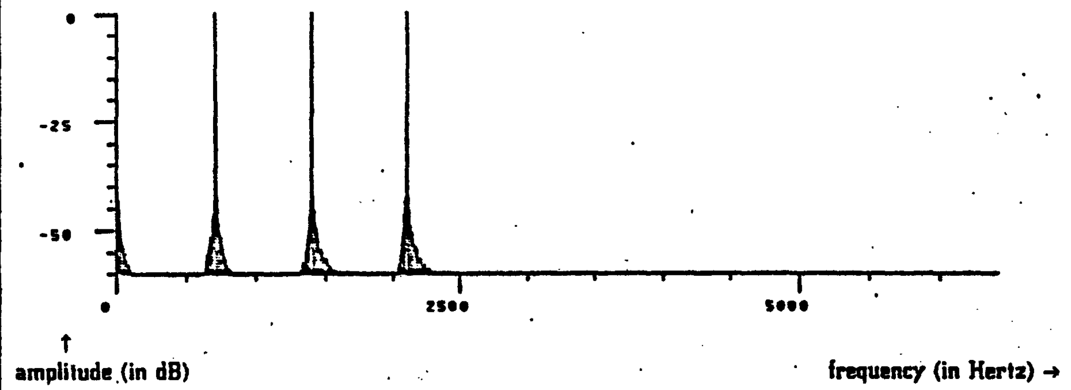


FIGURE 4-5: Spectral plot of the positive-frequency portion of the "band-limiting" function sin(nx)csc(x), for n = 7 (the negative-frequency portion of the spectrum is just the mirror-image of the positive-frequency part). Here x = 6.28ft, with f set to 350 Hz. This causes the spacing between adjacent components to be spaced symmetrically around zero Hz. at intervals of 2f = 700 Hz. Since n is odd, there is a component at zero Hz., which can be seen by the right half of its skirt in the plot.

function

$$B_{norm}(n, x) = \frac{B(n, x)}{n}$$

to ensure that the amplitude of $S(\theta, n, \beta)$ never exceeds unity. $B_{norm}(n, x)$ is shown in Figure 4-6 for values of n from 1 to 10. We pay particular attention to the points where $x = k\pi$, for this is where the denominator of the function becomes zero. From this function we can verify that the function at these points is always equal to plus or minus one, depending on whether n and k are even or odd. For odd multiples of n, $B_{norm}(n, x)$ is always equal to +1 when x is any multiple of π. If n is even, however, $B_{norm}(n, x)$ is equal to +1 at all even multiples of k, and -1 at all odd multiples of k. In order to avoid numerical problems in the hardware calculation of our band-limiting function, we can give it the final definition:

$$B_{norm}(n, x) = \begin{cases} +1.0 & \textit{if } x = k\pi \textit{ with } n \textit{ odd, or } n \textit{ even and } k \textit{ even} \\ -1.0 & \textit{if } x = k\pi \textit{ with } n \textit{ even and } k \textit{ odd} \\ n^{-1} \sin nx \csc x & \textit{otherwise} \end{cases}$$

where k is any integer.

Graphs of both the sine and cosine forms of the band-limited pulse function thus obtained are shown in Figure 4-7. Notice that the peak amplitude of the sine form decreases with increasing n, which is due to the fact that the peaks of the sine functions in the summation do not lie on top of each other. This could be remedied by using the cosine form of the identity (this waveform is shown in the bottom half of Figure 4-7) but this is deemed undesirable due to the transients associated with the onset of cosinusoids in digital sound synthesis. (At t=0, the cosine waveform can, and often does, produce an audible "click" for almost any non-zero amplitude. This problem is less severe if the sine function is used instead.)

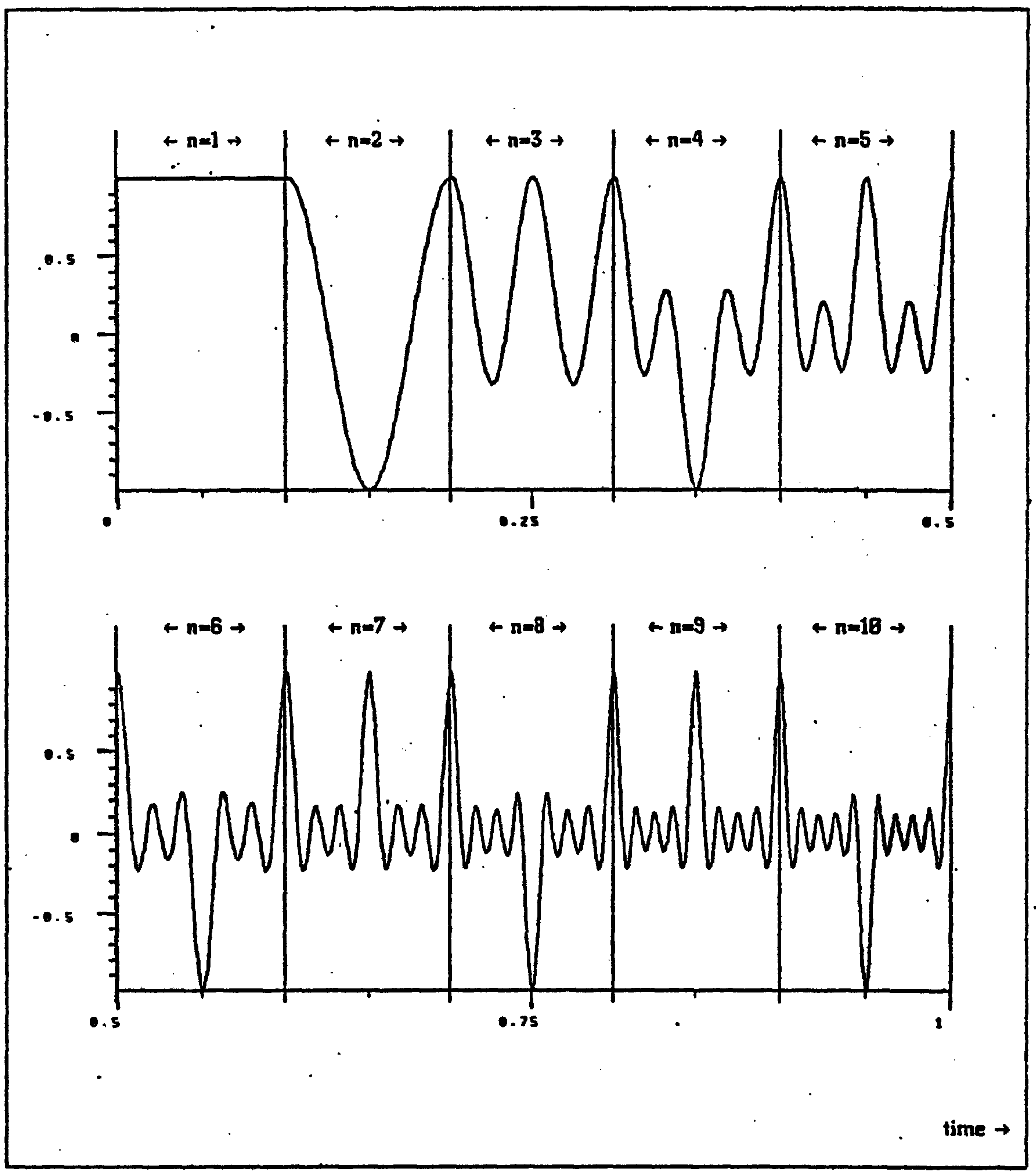


FIGURE 4-6: Waveform plot of the "band-limiting" function sin(nx)csc(x), for values of n from 1 through 10. Each tenth of a second, n is increased by one, starting from one for the first 0.1 second. Each tenth of a second represents exactly one full period of the function. The function shown has been normalized by dividing by n.

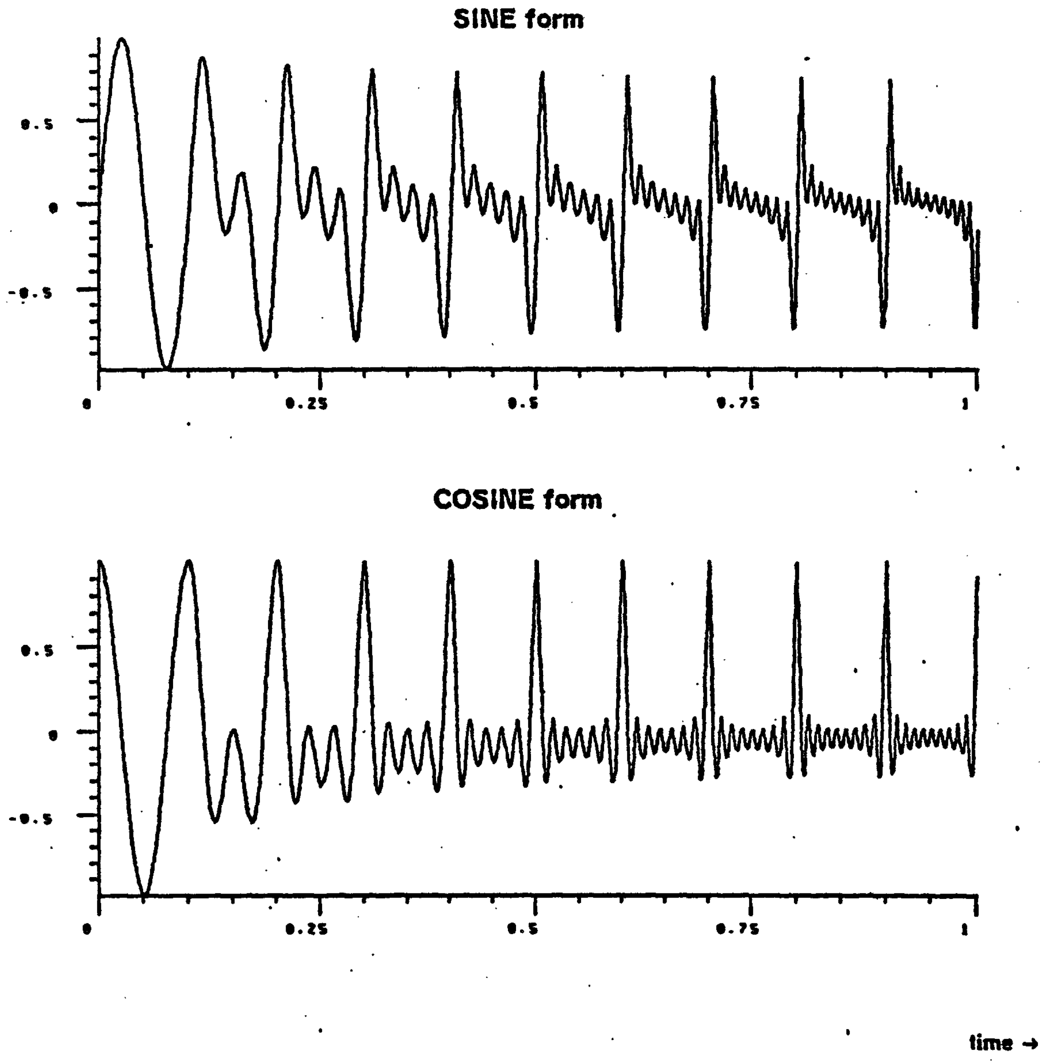


FIGURE 4-7: Waveform plots of both the sine (top) and cosine (bottom) forms of the band-limited pulse waveform for values of n from 1 through 10. Each tenth of a second, n is increased by one, starting from one, as in the previous figure. Each tenth of a second represents exactly one full period of the function. Here the spacing between components is equal to the frequency of the lowest component, resulting in a harmonic waveform. Notice that as n increases, the overall amplitude of these waveforms decrease slightly, but in different ways. In the sine form, both the positive and negative peaks are reduced, while in the cosine form, only the negative-going portion of the waveform is reduced in amplitude.

The evaluation of $B_{norm}(n, x)$ presents no further mathematical problems. However, if we consider the regions of the function where x is near some multiple of π, we observe that $csc\ x$ is quite large and $sin\ nx$ is quite small, which can produce a numerical instability due to roundoff error in the calculation, both in the multiplication of sine by cosecant and in the calculation of the phase angles x and nx.

Numerical Problems of the Cosecant

To insure that the multiplication itself goes well, care must be taken to "match" the values used for the sine and cosecant functions in the following way: set each value in the cosecant table to the *truncated* reciprocal of the corresponding value *actually used* in the sine table. This assures that the product $sin\ x\ csc\ x$ is no greater than 1.0 for any value of x (except, of course, $x = k\pi$, where $csc\ x$ doesn't exist). By far the more serious problem is phase jitter in the arguments of the sine and cosecant functions (phase jitter is caused by the fact that the table look-up procedure cannot use an infinite-length table). Since the cosecant function has infinite discontinuities at $x = k\pi$, if the relationship between nx and x is not exactly correct in the calculation of $sin\ nx\ csc\ x$, huge errors in the calculation can result. For example, let us assume that an oscillator increment value corresponds to 1.3, and we wish to calculate $sin\ nx\ csc\ x$ with n equal to 2. The sum-of-increments register for the cosecant-generating function would be set to the sequence:

0 1.3 2.6 3.9 5.2 6.5 7.8 9.1 10.4 11.7 13.0 . . .,

while the sum-of-increments register for the sine-generating function (with an increment of 2×1.3) would be set to:

0 2.6 5.2 7.8 10.4 13.0 15.6 18.2 20.8 23.4 26.0 . . .

in the same time span. Expressed in this way, the second sequence is just double the first, as it should be. If a truncating address scheme is employed in the table look-up procedures, the corresponding table addresses would be given by the two sequences:

$0 \ 1 \ 2 \ 3 \ 5 \ 6 \ 7 \ 9 \ 10 \ 11 \ 13 \ldots$ *and*

$0 \ 2 \ 5 \ 7 \ 10 \ 13 \ 15 \ 18 \ 20 \ 23 \ 26 \ldots$

It is clear that the second sequence is not always double the first, (even though the *average* difference between successive values is still equal to 1.3 for the first sequence, and 2.6 for the second), and when the numbers correspond to values of the argument close to $k\pi$, the $\sin nx \csc x$ product will be incorrect, causing a distortion of the waveform such as that shown in Figure 4-8. The same problem occurs if rounding is used, and, to a lesser extent, when interpolation is used as well. Therefore it is not possible in general to calculate the addresses for the sine and cosecant memories independently unless extreme accuracy is used. If, however, we calculate the second sequence directly from the first sequence *after* the truncation or rounding operation, the second sequence will always be exactly n times the first, and the $\sin nx \csc x$ product will be correct. Thus instead of using the hardware elements interconnected as depicted in Figure 4-9 at the top, we must use them as shown in the bottom half of Figure 4-9.

Finally, the problem of normalizing $B(n, x)$ may be solved by noting that n is always positive and can be reasonably limited to some maximum value, say 256. Thus n^{-1} may be calculated by table look-up as well using a small read-only memory. The values stored in this memory may then be multiplied by $B(n, x)$, to get $B_{norm}(n, x)$. An alternative procedure would be to simply scale $B(n, x)$ by shifting it right $\lceil \log_2 n \rceil$ bits, where n is the number of components being generated and $\lceil x \rceil$ is the "ceiling" function of x, defined as x if x has an integer value, and $x + 1$ if x has a non-zero fractional part. Such scaling would insure that instantaneous values of the waveform would never exceed unity, though in a

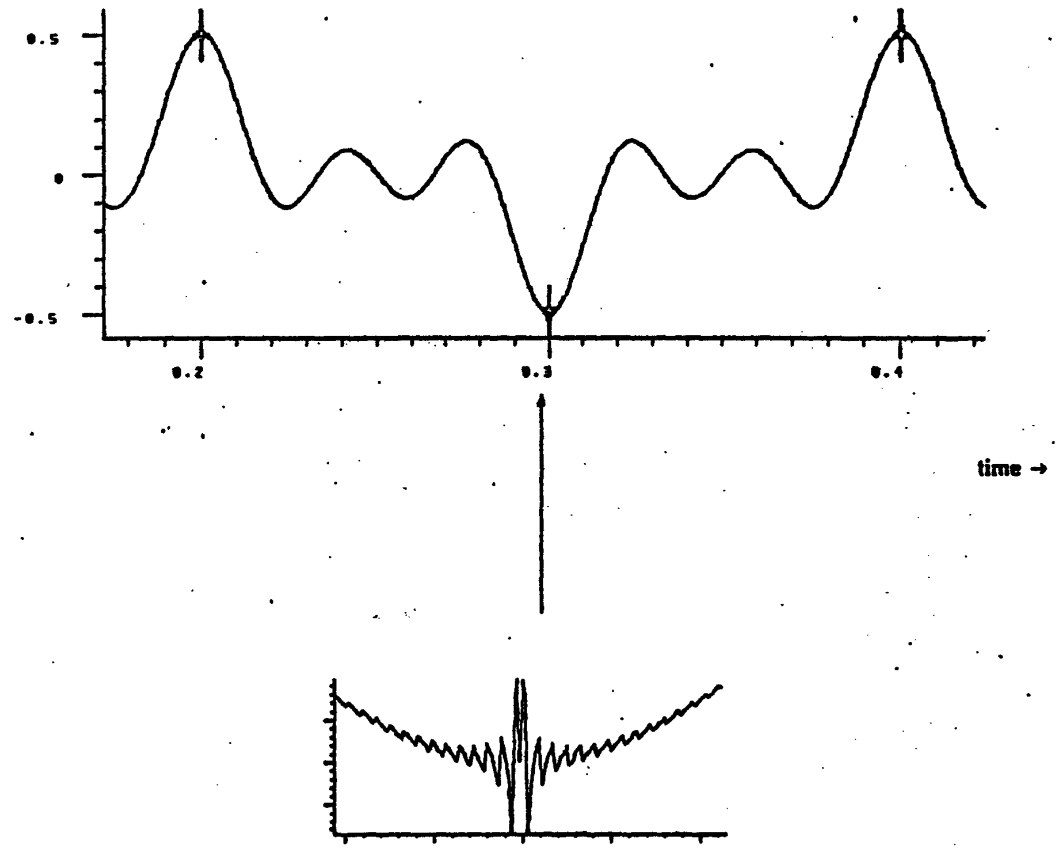


FIGURE 4-8: Illustration of distortion due to phase errors in the "band-limiting" function sin(nx)csc(x), here shown for n = 6. (The overall amplitude of the signal has been reduced in this plot since the errors would otherwise produce samples out of the -1.0 to +1.0 range.) As the value of x becomes close to any multiple of pi, amplitude errors result from the fact that the relation between nx and x is not always exactly n (see discussion in the text). Since the cosecant function has a very large slope around any multiple of pi, the resulting distortion can be quite severe. The lower plot shows a detail of the upper.

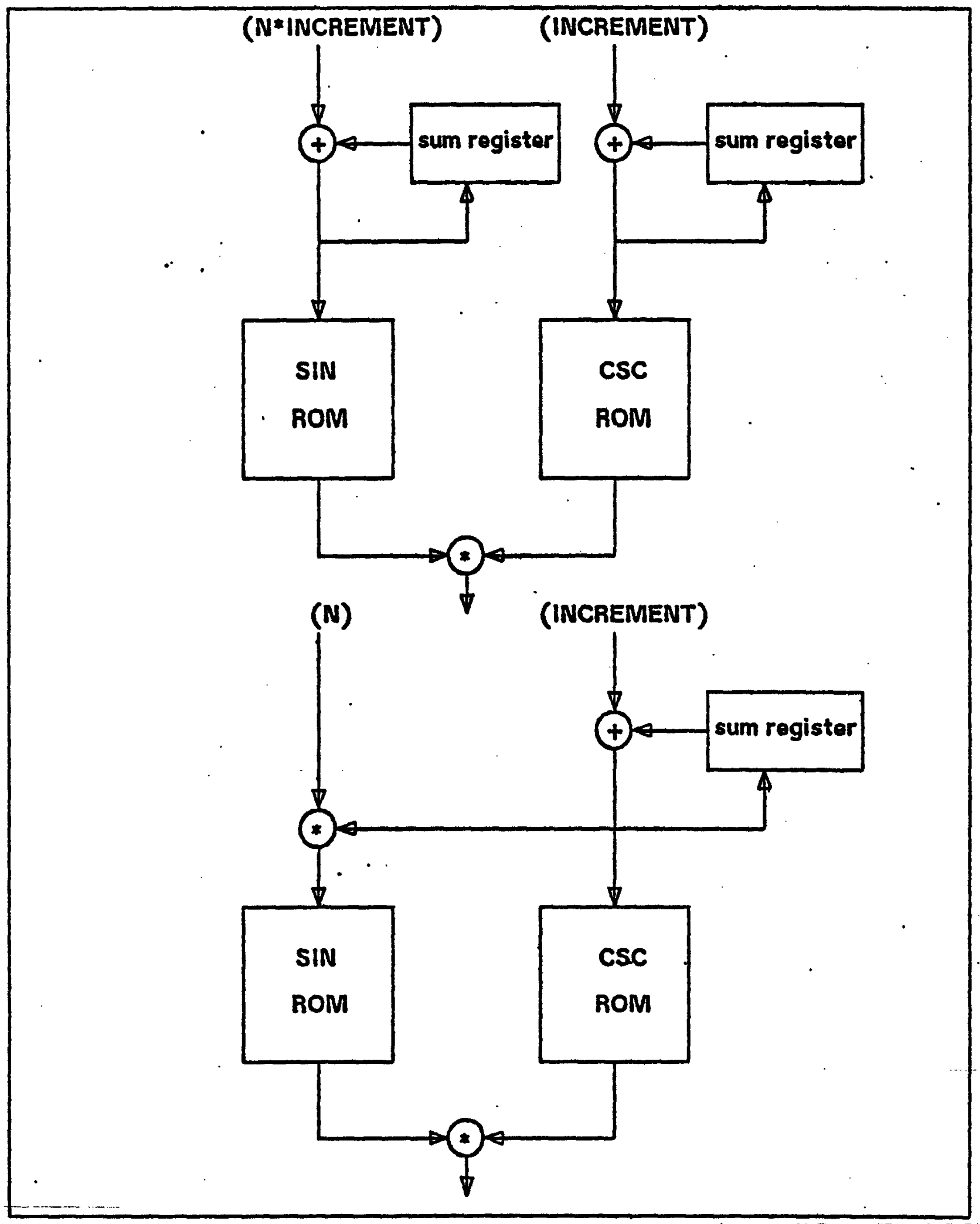


FIGURE 4-9: Block diagrams of two alternate interconnection schemes for realizing the calculation of the "band-limiting" function sin(nx)csc(x). The top figure depicts an incorrect realization which will lead to waveform distorion, while the bottom figure depicts a scheme for ensuring the correct relation between the addresses to the sin and csc tables, thus avoiding distortions such as those shown in Figure 4-8 (see discussion in text).

worst case, when n is one less than a power of 2, the amplitude could be as little as one half of full scale. A third alternative would be not to normalize $B(n, x)$ at all, and providing for a special multiplication by a user-specified amplitude value. Such a multiplication operation would indeed be "special" because the waveform operand will be larger than unity.

Description of a Useful Oscillator Module

We are now ready to functionally describe a useful oscillator module. It has two inputs, two outputs, and three basic modes: AM, FM, and BLP (for band-limited pulse). Using the notation developed at the beginning of this chapter, X_1 and X_2 are the two inputs, and Y_1 and Y_2 are the two outputs. Following the convention of using 16-bit, 2's complement specifications for quantities such as frequency and amplitude, we assume that the 16 bits are right-justified in the 20-bit field. Thus the first input is represented more precisely as $X_1[4:19]$; $X_1[0:3]$ are unused bits.

In the AM mode, X_1 and X_2 are taken to be a frequency and an amplitude specification, respectively. Let us then adopt the notion that frequency is the time derivative of phase, i.e., if $f(t) = \sin\theta$, then

$$\frac{d\theta}{dt} = \frac{d}{dt}(2\pi ft) = 2\pi f$$

We can then refer to the phase angle $\theta(t)$ as the time integral from arbitrary time 0 to the present time t of the frequency, f:

$$\theta(t) = 2\pi \int_0^t f\,dt$$

In our oscillator, the time integral of the frequency input is exactly what is contained in the sum-of-increments, or phase, register, except that this value is not permitted to grow arbitrarily large due to the modulus arithmetic used. This modulus phase value, which we will refer to as $\int X_1$, could be useful if it were available as an alternative output of the oscillator, since it has the shape of a ramp wave going from minus full scale to plus full scale, and a frequency equal to the one specified by X_1. It is non-band-limited, due to the "instantaneous jump" from plus to minus full scale, so it is not particularly useful as an audio waveform; however, it could be used to provide a frequency sweep if it were applied to the frequency input of another oscillator, or perhaps to control an amplitude rise during an attack. Clearly it will be most useful at low frequencies. This is precisely the kind of secondary information available from the oscillator which makes a second output desirable.

In the normal AM mode, output Y_2 can be the amplitude-scaled sinusoid, X_2 $sin(\int X_1)$, while Y_1 can be set to $\int X_1$, the "modulus phase" waveform. In the FM mode of operation, both X_1 and X_2 are taken to be frequencies, which are first added and then integrated to get the phase for the sine table look-up. Thus the useful alternative output in this case could be $\int(X_1 + X_2)$, the integral of the sum of the two inputs. In fact, just the *sum* of the two inputs could be useful, allowing the oscillator to operate as a mixer. This is especially true since the two inputs are both scaled quantities. The product of the two inputs is also a clear candidate for inclusion among alternative outputs, though this quantity is not already calculated anywhere by the oscillator. It is useful enough, however, to consider making it available in any case.

Finally, in the BLP mode, input X_1 is taken to be the fundamental frequency, and input X_2 is the frequency spacing for the n components to be generated. The quantity n itself is supplied to the module via control bits. (As an alternative to providing the complete

BLP signal, the BLP mode could use X_1 as the spacing frequency, and X_2 as an amplitude. The module could then provide $X_2 \times B_{norm}(n, fX_1)$ as an output, which could in turn be multiplied by the remaining *sine* term by another oscillator running in AM mode.)

Thus a formal description of the oscillator module is:

X_1[4:19] - input 1, always a frequency.

X_2[4:19] - input 2, a frequency in FM or BLP modes, an amplitude in AM mode.

$C_{1,2}$[0:1] - (true for both inputs) a 2-bit scale selection quantity:

- 00 leaves the input unaltered,
- 01 arithmetically shifts the input right 1 bit,
- 10 arithmetically shifts the input right 2 bits, and
- 11 arithmetically shifts the input right 3 bits.

When the inputs are frequencies, the following table relates the scale factor, f_{max}, and f_{min}:

Scale select	f_{max}	f_{min}
00	16384 Hz.	1/2 Hz.
01	8192 Hz.	1/4 Hz.
10	4096 Hz.	1/8 Hz.
11	2048 Hz.	1/16 Hz.

When then input is an amplitude, the scale factor simply selects whether to divide it by 1, 2, 4, or 8.

$C_{1,2}$[2:4] - (true for both inputs) three range selection bits which are *exclusive or*-ed into bits 1:3 of the *scaled* input.
Thus if the scale selection bits are 10, setting these three range bits to 110 will allow frequencies from 12288 to 16384 Hz. to be generated with 1/8 Hz. accuracy, etc.

C_1[5:6] - a 2-bit field selecting the main (Y_2) output:

- 00 for AM (i.e., $Y_2 = X_2 \sin\int X_1$),
- 01 for FM (i.e., $Y_2 = \sin\int[X_1 + X_2]$),

- 10 for BLP (X_1 = frequency, X_2 = spacing, n is supplied by control bits as shown below), and
- 11 is unused.

$C_1[7:8]$	- a 2-bit field selecting the secondary (Y_1) output:
	- 00 selects $Y_1 = X_1 + X_2$ as scaled above,
	- 01 selects $Y_1 = \int X_1$ in AM or BLP modes, and $Y_1 = \int(X_1 + X_2)$ in FM mode.
	- 10 selects $Y_1 = X_1 * X_2$ as scaled above, and
	- 11 is unused.
$C_1[9]$	- is unused.
$C_2[5:9]$	- a 5-bit field used to specify n for the BLP mode.

Control Function Generation

All of the synthesis methods require control functions for describing such time-varying parameters as the evolution of amplitude or a modulation index during a note. Such control functions are typically aperiodic, and usually describe changes which occur over several milliseconds, or even seconds. Thus they tend to provide the gross characteristics of a note, such as its attack transient shape, its variation (if any) during a pseudo-steady state, its decay, or how its bandwidth evolves. In additive synthesis such functions are used to describe the variation in time of the frequency, amplitude, and possibly the phase of each sinusoidal component of a particular sound.

Such functions might also be used to control a *portamento* between two pitches, or a general increase in loudness (*crescendo*) lasting several seconds. It is of course impossible to predict the exact shape of such functions, since the number of ways in which a *crescendo* can be made is equal to the number of paths between two points. Fortunately it usually is sufficient to define such shapes only approximately, since most of the ways of travelling from one point to another are equivalent, for all practical purposes.

Stored Control Functions

Several methods of generating control functions have been used in the practice of computer music, but these methods tend to fall into one of two categories: the stored function method *à la* MUSIC V, and the piecewise linear function method *à la* GROOVE.

In the stored function method, the desired pattern of variation is stored as a list of sample values in a table, just as a single period of a sinusoid is stored for use by the table look-up oscillator. The overall duration of this function may be controlled by reading it from the table at a variable rate in a manner analogous to the frequency control of the oscillator, since:

$$\textit{duration (period)} = \frac{\textit{(table length)}}{\textit{(increment) (sampling rate)}}$$

Thus we may "stretch" or "shrink" a function to fit any desired overall duration while retaining its characteristic shape. This is not a general solution to the amplitude envelope problem, however, since both the attack and decay times are typically independent of note duration for many musical sounds. If, by doubling the duration of a note, we also double its attack time, we will obtain a sound markedly different in quality. Mathews' solution to this problem in the MUSIC V context [Mathews, 1969] was to treat the stored function in three separate parts. The first part was used to describe the attack transient, the second described the steady state, and the third described the shape of the decay. A special unit generator for enveloping scanned the function at a variable rate during each of the three parts of a note in such a way that the increments used during the attack and decay were independent of the note duration. Thus we could specify an attack time of, say, 10 ms., and a decay time of 75 ms.; the envelope generator would automatically calculate the steady state time according to:

$$\textit{(steady state time)} = \textit{(note duration)} - \textit{(attack time)} - \textit{(decay time)}$$

The stored functions used in MUSIC V were typically 512 values long, which afforded enough resolution to describe the detailed shapes of the control functions. Disadvantages of this method include the large amount of memory needed to store simultaneously several different control functions, and the fact that as durations become long the necessary increment values can become quite small, the representation of which can cause numerical difficulties on many computers. Unfortunately, shortening the table length aggravates the numerical difficulty, since the increment is directly proportional to the table length in this case.

Piecewise Linear Control Functions

The second basic method of control function generation also uses a table, but a much smaller one. It is based on the fact that most useful control functions for music may be approximated fairly well by a small number of straight line segments drawn between the major inflection points of the more complex envelope functions observed in data describing musical instruments. For example, Grey showed in his investigation of musical timbre [Grey, 1975] that the temporal variations in amplitude of the individual harmonics of many musical tones may be represented by piecewise linear functions with small numbers of line segments, typically 5 to 7. Assuming that these functions always begin at zero value, to represent an n-segment function requires the specification of n points on a graph, i.e., $2n$ *pairs* of numbers. Each number pair specifies an ordinate value ("height"), and an abscissa value to the right of zero ("time"). If we plot these points on a graph, we have only to "connect-the dots" to obtain the function. By substituting these piecewise linear functions for amplitude and frequency curves obtained from the analysis of real tones played on standard musical instruments, Grey was able to reduce the amount of data required to specify a musical note by about two orders of magnitude. Furthermore, the sound of the data-reduced notes was judged to be musically indistinguishable from the original tone.

Experiments in music synthesis with computers have indicated that the minor fluctuations in such quantities as the amplitude envelope of a tone are far less important than their gross characteristics. Among the few instances where the straight line approach is detectable is in the decay transient of notes where the decay time is long, i.e., greater than about a half second. Most musical instruments which ring for such a long time, such as bells or pianos, have an exponential decay curve. If this exponential is approximated by two straight line segments, the difference is difficult to detect under most circumstances, and three line segments usually become indistinguishable perceptually, unless the ringing lasts for a very long time, as in the case of a loud bell or tam-tam note.

Undersampled Control Functions

Since the control functions under consideration generally vary rather slowly in time compared to the oscillations in the total waveform, some advantage could be gained if it were possible to deal with these control functions at a lower sampling rate than the waveform itself. For example, if control values were generated at a 1000 Hz. rate, then a new amplitude value could be specified every millisecond, which would be sufficient temporal resolution to describe most distinguishable tones. An unfortunate difficulty is encountered if this function, effectively sampled at 1000 Hz., is multiplied by a waveform sampled at a much higher frequency. Since the spectrum of any sampled waveform is periodic in frequency with a period equal to the sampling frequency, and since the sampling frequency is itself a component in the spectrum of a sampled waveform, we would have components at all integer multiples of 1000 Hz. in the spectrum of the control function waveform. When we multiply this function by another waveform, we *convolve* the spectra of the two waveforms, yielding a resultant waveform with many undesirable components in its spectrum. Since many of these components will be lower than the Nyquist frequency of either waveform, they cannot be removed by filtering either before or after digital-to-analog conversion.

Another way to describe this difficulty is in terms of the size of the steps between sample values in an undersampled control function. For a given time rate of change in such a function, the difference between successive sample values is inversely proportional to the sampling frequency. Only if these successive differences are very small will the discontinuous "jumps" in amplitude be inaudible, implying that the use of undersampled control functions is possible only for control functions which change extremely slowly. Therefore it is generally necessary to sample the control functions at exactly the same frequency as the rest of the waveform components, even though the control functions may change only rather slowly in time.

The Amplitude and Control Function Module

Since it seems more likely that a large variety of functions with grossly different shapes will be more useful in music synthesis than a small number of accurately defined functions, the straight line segment approach to function generation is adopted here to generate envelopes. The amplitude envelope and control function module has been designed to allow the specification of up to 32 piecewise linear functions, each consisting of up to 8 connected line segments. It has two inputs and two outputs; one input is usually a waveform which is to be scaled by a control function such as an amplitude envelope, and the other input specifies an overall amplitude (either input may be constant or time varying). One output is normally the product of the two inputs multiplied by the selected control function, the other output is the (unscaled) control function itself. Thus the main output is the product of three quantities which may be thought of as a waveform, an envelope function, and a volume control. Having the unscaled envelope function itself appear at the output allows it to be used to control the amplitude of several waveforms at once, perhaps scaled differently each time. Of course provision must be made for specifying the shape of each of the 32 functions, and a number of other module operations can be made available, such as straight multiplication of the two inputs.

The functions are each defined as a set of 8 slope/target value pairs (the beginning value of each function is assumed to be zero). Each sample of the defined function is generated by adding the slope to the previous function value. As soon as the target value is reached, the next slope/target values are used to generate the next straight line segment. The target values are signed, 2's complement, 8-bit quantities, which allow the entire range of positive and negative amplitudes to be broken up into 256 possible target levels. The slope is specified as an 8-bit magnitude and a range selection bit. The sign of the slope is derived from the relation of the previous target value to the current target value. The arithmetic internal to the module is carried out to 20 bits of (fractional) precision, and the target value always is used to specify the 8 most significant bits of the fractional target level. The range bit associated with each slope value determines whether it is added algebraically to the 8 least significant bits out of 20, or to the next higher 8 bits. Thus the minimum specifiable slope value is an increment of 2^{-20} units of increase per sample. At a sampling rate of 32,768 Hz., such a slope value would require 16 seconds to go from zero to full scale (32 seconds from minus to plus full scale). The steepest possible slope value is equal to $2^{-4} - 2^{-16}$, making it possible to go from zero to full scale in approximately 8 samples, or 0.24 ms. The entire set of 32 functions can be maintained in a random access memory consisting of only 256 words by 17 bits, where each slope/target pair is a 17 bit quantity, and each of the 32 functions requires 8 memory locations.

Of fundamental interest in amplitude envelope generation is the method of triggering (i.e., "starting") the envelopes under program control. The module provides for four specific ways of controlling the manner in which any function is triggered, according to the control bits, called the piano, organ, repeat, and threshold modes. The two input signals are each signed, 2's complement 16-bit quantities, leaving the four high-order bits of each input available for other uses. In the piano, organ, and repeat modes, the most significant bit of the scale input is used as a trigger bit and is normally controlled directly by the computer, although any module could control this bit, if desired.

In the piano mode, the envelope is triggered by a zero-to-one transition of the trigger bit. The entire envelope function is generated exactly once as specified, except that the final target value is "held" continuously until a new zero-to-one transition occurs in the trigger bit. In case a zero-to-one transition occurs before the envelope function has been completely generated, the function will be retriggered. This immediately resets the current slope and target values to the first slope/target pair, causing the amplitude, from wherever it was in the function, to head towards the initial target using the initial (attack) slope.

The organ mode differs from the piano mode only in that the fourth target value is also held while the trigger bit stays on. As soon as the trigger bit makes a one-to-zero transition, the rest of the envelope function is generated as in the piano mode, "sticking" on the final target value until a new trigger is received. Retriggering is accomplished exactly as in piano mode.

In repeat mode, the function does not "stick" at all: the function starts when a zero-to-one transition occurs in the trigger bit, and repeats continuously as long as the trigger bit remains equal to one. A one-to-zero trigger transition causes the function to complete its current cycle and then to stop.

The threshold mode does not use the trigger bit *per se*, but rather derives the trigger from the scale input, which may receive its signal either from the computer or from any module. Two control bits are used to specify one of four different threshold values; if the scaling signal is greater than or equal to zero, one fourth, one half, or three fourths, the function will be triggered (in any of the piano, organ, or repeat modes, according to the control bits). In the threshold mode, the main output is not usually scaled by the triggering input.

A formal description of the amplitude envelope generation module is as follows:

$X_1[4:19]$	- input 1, usually a waveform signal to be "enveloped."
$X_2[4:19]$	- input 2, usually a scale factor (volume control).
$Y_1[4:19]$	- output 1, usually the unscaled envelope function.
$Y_2[4:19]$	- output 2, usually the scaled, enveloped, waveform.

$X_2[0]$ - input 2 trigger bit in piano, organ, or repeat modes.

$C_1[0:2]$ - MODE selection bits:

- 000 PIANO mode
- 001 ORGAN mode
- 010 REPEAT mode
- 011 UPDATE function memory mode.

  In this mode, the second control input is used to specify an address into which a slope/target datum is to be written. This 17-bit datum must be first applied to $X_2[3:19]$, and consists of an 8-bit target and a 9-bit slope value as explained in the text above.
- 100 MULTIPLY mode ($Y_2 = X_1 * X_2$)
- 101 (unused)
- 110 (unused)
- 111 (unused)

$C_1[3]$ - trigger source selection bit

- if 0 then bit 0 of input 2 is used as trigger.
- if 1 then THRESHOLD mode using input 2 signal.

When the threshold mode is used, the following two control bits select the threshold level (see below).

$C_1[4:5]$	- threshold selection bits (in threshold mode only)
	- 00 trigger if $X_2[4:19] \geq 0$
	- 01 trigger if $X_2[4:19] \geq 1/4$ full scale
	- 10 trigger if $X_2[4:19] \geq 1/2$ full scale
	- 11 trigger if $X_2[4:19] \geq 3/4$ full scale
$C_1[5:9]$	- unused
$C_2[0:7]$	- the address in the 256-word by 17-bit function memory into which the datum applied to $X_2[3:19]$ is to be written.
$C_2[8:9]$	- unused

CHAPTER 5 - CONCLUSIONS AND OBSERVATIONS

An initial implementation of this real time interactive digital music synthesis system became operational at the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University during July, 1977. The synthesis hardware consists of a complete control unit as described in chapter 3, and an output collector module with provision for four audio channels, each serviced by a 16-bit digital-to-analog converter (initially only a single channel with a 12-bit DAC was used for testing purposes). A module for providing real time input data has been implemented which allows 32 time varying parameters to be set individually at any time by the control computer. A dual input, dual output oscillator module capable of generating up to 32 independent sinusoids in either AM or FM mode was built; a simple arithmetic module for performing multiplication and addition completed the initial module set. A simple two way interface allows the control computer to send information both to the control unit's control memory ("patches") and to the real time input memory ("signals") module. The computer may also receive the computed samples from the synthesizer for possible storage and/or further processing. This two way dialog can be carried on either in real time ("run mode"), or on a locked-step, sample-by-sample basis ("sample mode").

The control computer in this case consists of a Digital Equipment Corporation (DEC) PDP6, a medium scale, second-generation computer with 64k words of memory and a typical instruction time of about 3 microseconds. Even with its fairly slow instruction time (by 1977 standards), this computer has proven to be more than adequate to control fairly complex real time synthesis tasks on the hardware synthesizer. Software has been written which "performs" prepared scores, such as Bach chorals played on diverse instruments, and for real time performance at a piano-like keyboard, again with provision for many choices of sound quality. A great deal of future work on software is in the planning stages, including the ability to combine prepared scores with real time interactions at keyboards and knob panels, and provision for "remembering" the gestures and actions of the performer using the real time control devices for future playback and editing. Particularly important is the software facility for allowing the real time program to participate as an aid in the compositional process, thus providing for such features as automatic harmonization in real time.

The keyboard itself need not be used in the traditional manner. By interposing a computer between the keyboard and the sound producing elements of the system, it is possible to use the keyboard as a set of switches, each of which represents some "command" to the synthesis system. The interpretation of these "commands" is left to the discretion of the computer programmer, and is not at all limited simply to playing the notes corresponding to depressed keys. For example it may be desired that when the note "C" is depressed on the keyboard, an entire phrase, melody, or composition is caused to sound. The note "C sharp" might invoke a different melody, or perhaps cause the first melody to be transposed down by a constant 31 Hertz. The function to be performed may depend on a set of keys, or perhaps on a time sequence of key depressions.

Other forms of input besides keyboards and knobs may be considered, such as control of the computer with specially constructed devices including two or three dimensional "joysticks," or special interfaces with traditional instruments, such as violins and voices. It seems virtually certain that real time interactive computer music generation will have a strong effect on the manner in which music is performed, and it is in this area that some of the most fascinating future research topics lie.

Probably the most significant feature of this initial implementation is how well it works even though the system is far from complete. Since the amplitude envelope generator was not among the first to be built, its function was temporarily imitated by a combination of software in the control computer and an extremely simple arithmetic module. In other words, even a fragment of this system is already musically interesting and useful. Furthermore, the entire hardware implementation to this point took about six "man-months" of labor, including the ordering of parts, the manual construction of all circuits, the debugging of the hardware, and the development of skeletal software. The indication certainly seems to be that the modular approach adopted here yields a strong advantage by allowing the system to become operational one section at a time, and that the construction of each section is simplified by a standard set of "rules" for module construction (the oscillator module, for example, required approximately two weeks to design and implement from start to finish).

The remaining items in the basic set for music synthesis include filtering and reverberation modules. There is a vast literature on digital filter design from which candidate filter architectures may be chosen (see, for example, Rabiner and Gold, 1975) and no specific design will be discussed here. A suitable filter would be a straightforward second-order stage which can be cascaded via the control unit to obtain higher-order filters when needed. Full, 20-bit signal intercommunication paths might be used between the stages of a cascaded filter to allow intermediate values to have a numerical range exceeding the usual 16-bit range of -1.0 to $+1.0 - 2^{-16}$, if necessary. A desirable property of the filter design would be guaranteed stability for coefficients having magnitudes less than, say, 1.0, thus allowing the bandwidth or center frequency of the filter to be controlled dynamically in a convenient way. Any filter requiring the specification of several coefficients could be implemented using dual inputs to a module, where one input is the signal to be filtered, and the other is used to set coefficients, as selected by control bits. Dual outputs provide the necessary number of data paths for cascading many filters.

A reverberation module could consist of a basic all-pass stage [Schroeder and Logan, 1961] which can be cascaded in a manner analogous to the filter module. The main problem in this case is one of providing enough memory within the module itself to allow sufficient signal delays for usefully long reverberation times. At the 32,768 Hz. sampling rate used in the synthesizer architecture, this amount of memory could easily be as much as 16k words of 12 to 16 bits each. Current memory technology is just barely sufficient to allow such a large on-module memory, but it is very likely that this technology will improve considerably within the next few years. It may be desirable to make the reverberation facility a part of the output collector module, since the reverberation is typically controlled on a per-channel basis to achieve the illusory acoustic effects described by Chowning, *et al.* [Chowning, 1971, and Chowning, Grey, Moorer, and Rush, 1975].

Further module capabilities for music synthesis might include white noise and random number generation, delay memories for special "phasing" effects, table look-up memories for nonlinear transformations of waveforms, logarithmic conversion modules for extensive multiplication or division operations, and so on. Clearly it is possible to conceive of modules for other purposes than music production. Special modules could be implemented for speech synthesis and processing, such as vocoder and linear predictive coding modules. "Block" processes such as determining the short time spectrum of a speech or music signal in real time with the FFT algorithm could be implemented by modules which include sufficient buffer memory to hold a "block" of samples. The implementation of most of these functions within the architecture presented here is very straightforward, and the possibilities for experimentation seem endless. As microprocessor technology continues to improve, it seems likely that general purpose, programmable modules consisting of a fast, simple microprocessor with a high-speed multiplier and some random access memory will soon become feasible.

Finally, improvements in large scale integration and packaging techniques are likely to allow the entire synthesizer to become more powerful, physically smaller, and less expensive, making real time digital music synthesis available on a widespread basis for use in the concert hall, in recording studios, educational institutions, and in the home. High fidelity loudspeakers are already commonplace; low-cost digital music synthesizers are very likely candidates for turning these loudspeakers into musical instruments of remarkable interest and utility.

BIBLIOGRAPHY

Appleton, Jon, "A Simple Musical Language for the Dartmouth Digital Synthesizer," 2nd Music Computation Conference, Urbana, Illinois, Nov. 7-9, 1975.

Chowning, John M., "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation," *Jour. of the Aud. Eng. Soc.*, v 21, n 7, pp. 526-534., 1973.

Chowning, John M., "The Simulation of Moving Sound Sources," *Jour. of the Aud. Eng. Soc.*, v 2, n 6., 1971.

Chowning, John M., Grey, J.M., Moorer, J.A., Rush, L., "Computer Simulation of Music Instrument Tones in Reverberant Spaces," *CCRMA Technical Report STAN-M-1*, Dept. of Music, Stanford Univ., 1975.

Grey, John M., "An Exploration of Musical Timbre," *CCRMA Technical Report STAN-M-2*, Dept. of Music, Stanford Univ., (Ph.D. dissertation), 1975.

Mathews, M.V., with the collaboration of J.E. Miller, F.R. Moore, J.R. Pierce, and J.-C. Risset, *The Technology of Computer Music,* MIT Press, 1969.

Mathews, M.V., Moore, F.R., and Risset, J.-C., "Computers and Future Music," *Science*, v 183, pp. 263-268., 1974.

Mathews, M.V., and Moore, F.R., "GROOVE - A Program to Compose, Store, and Edit

Functions of Time," *Comm. of the ACM*, v 13, n 12, 1970.

Moore, F.R., "Computer Controlled Analog Synthesizers," *Bell Laboratories Comp. Sci. Tech. Rpt. #10*, 1973.

Moore, F.R., "Music--Film--Computers," *Filmmaker's Newsletter*, v 4, n 6, 1971.

Moore, F.R., "Music and Computers," *Enciclopedia della Scienza e della Technica/Mondadori*, Yearbook, pp. 490-498 (in Italian)., 1971.

Moorer, James A., "The Optimum Comb Method of Pitch Period Analysis of Continuous Digitized Speech," *IEEE Trans. on Acous., Speech, and Sig. Proc.*, v ASSP-22, n 5, pp 330-338., 1974.

Moorer, James A., "On the Segmentation and Analysis of Continuous Musical Sound by Digital Computer," *CCRMA Technical Report STAN-M-3*, Dept. of Music, Stanford Univ., (Ph.D. dissertation), 1975.

Moorer, James A., "The Synthesis of Complex Audio Spectra by Means of Discrete Summation Formulae," *CCRMA Technical Report STAN-M-5*, Dept. of Music, Stanford Univ., 1975.

Rabiner, L., and Gold, B., *Theory and Application of Digital Signal Processing*, Prentice Hall, 1975..

Rakowski, A., "Pitch Discrimination at the Threshold of Hearing," *Proc. 7th Int. Cong.*

Acoust. Budapest, v 3, p. 373., 1971.

Risset, J.-C., and Mathews, M.V., "Analysis of Musical Instrument Tones," *Physics Today*, v 22, n 2, pp 23-30., 1969.

Roederer, J.G., *Introduction to the Physics and Psychophysics of Music*, Springer-Verlag, New York, 1973..

Ruiz, P.M., *A Technique for Simulating the Vibrations of Strings with a Digital Computer*, Master's thesis, Department of Music, University of Illinois, 1969.

Schroeder, M.R., and Logan, B.F., "Colorless Artificial Reverberation," *J. Audio Eng. Soc.* v 9, n 192, July 1961.

Saunders, Steven E., "Real-Time FM Digital Music Synthesis," *Proc. of Music Computation Conf. II*, Urbana, Illinois, 1974.

Zwicker, E., Flottorp, G., and Stevens, S.S., "Critical Bandwidth in Loudness Summation," *J. Acoust. Soc. Amer., v 29, p. 548., 1957.*