San Jose State University
SJSU ScholarWorks

Master's Theses | Master's Theses and Graduate Research

1989

DMX-1000 user guide and tutorials

Eric Gatzert
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation
Gatzert, Eric, "DMX-1000 user guide and tutorials" (1989). *Master's Theses*. 3069.
DOI: https://doi.org/10.31979/etd.zu8n-nqys
https://scholarworks.sjsu.edu/etd_theses/3069

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 1337803

DMX-1000 user guide and tutorials

Gatzert, Eric William, M.S.

San Jose State University, 1989

Copyright ©1989 by Gatzert, Eric William. All rights reserved.

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

DMX-1000 USER GUIDE AND TUTORIALS

A Thesis

Presented to

The Faculty of the Department of Music and

The Faculty of the Department of Cybernetic Systems

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Eric Gatzert

May, 1989

APPROVED FOR THE DEPARTMENT OF MUSIC
AND THE DEPARTMENT OF CYBERNETIC SYSTEMS

Allen Strange

Devayani L. Smith

Devayani Smith

Larry Wendt

Larry Wendt

APPROVED FOR THE UNIVERSITY

M. Lou Lewandowski

DMX-1000 USER GUIDE

AND TUTORIALS

Version 1.0

By

Eric Gatzert

(c) 1989

CONTENTS

Appendices

ILLUSTRATIONS

Chapter 6

Chapter 7

Chapter 8

CHAPTER 1

INTRODUCTION

The purpose of this interactive tutorial is to provide the user with useful information concerning the DMX-1000 signal processing computer and its operation. This guide describes the various functions and musical applications of the DMX-1000 Digital Signal Processor and its current controlling language, MUSIC 1000. The content of this guide is not intended to replace the present documentation. Rather it is to be used as a supplement which may clarify certain areas of procedure.

A number of different synthesis techniques are supported by MUSIC-1000. These techniques include additive and subtractive synthesis, linear frequency modulation synthesis, and amplitude modulation synthesis. While overviews of the above will be given at various points within this manual, it is assumed that the user possesses at least an intuitive understanding of the mentioned synthesis techniques. If these processes are not familiar, the included bibliography section provides adequate resources for further inquiry.

MUSIC 1000 is a programming language which requires two separate collections of textual information or *files*. Together, these files contain information for the DMX-1000 which allow for the generation of audio signals or processing. The two files are named Orchestra (ORCH.USR), and Score (SCORE.USR). The Orchestra is an area in which instrument definitions are assembled to create voices or patches. The Score represents a list of events which are executed by the Orchestra section of MUSIC 1000. The Score is a passive section containing numbers which correspond to specific conditions which exist in the active portion of the program, the Orchestra.

SYSTEM OVERVIEW

The DMX-1000 is a sophisticated 16-bit computer which has been designed specifically for digital processing of audio signals. In a general sense, the DMX-1000 can be compared to a synthesizer without any keyboards, factory preset voices, or built in synthesis schemes. The instrument is a blank slate waiting for instructions. These instructions must be generated elsewhere, by a "host" or master computer. The role of the host computer is to send the programmed information or instructions over to the DMX-1000. The program contains the information that the DMX-1000 requires in order to generate sound in real-time.

Real-time digital synthesis requires very high speed computation, i.e., "number crunching." This process is possible for the DMX-1000 due to its use of two separate high-speed memories: the *program memory* and the *data memory*. The host computer has the ability to write to either memory at any time, and therefore it is not limited to the sequential access of memory as with more conventional computer architectures. The program memory is used to hold the main program which operates the DMX-1000 as it is sent from the host computer. The data memory is used to hold the various constants, parameters, waveform lookup tables, and numerical values in general. This two memory parallelism is the most important factor in the DMX-1000's ability to produce digital sound in real-time. Illustration 1.1 is a diagram of the simplified architecture of the DMX-1000.

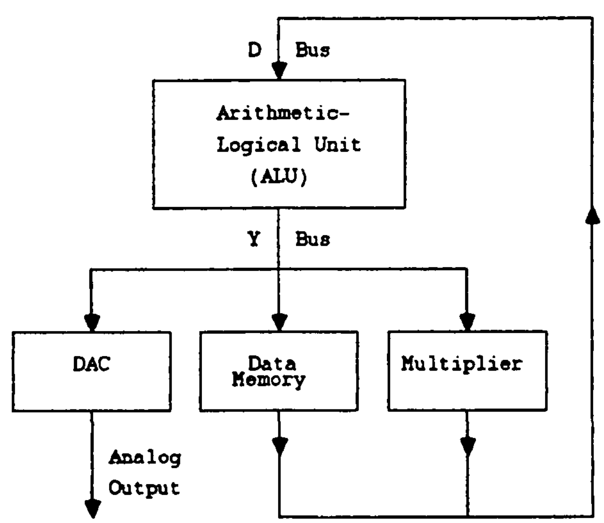


Illustration 1.1 DMX-1000 Architecture

The DMX-1000 creates sound by executing a digital calculation of waveform values. These values are then converted to analog signals by means of one or more *digital to analog converters* (DAC's). The host computer provides the information required to assemble the desired waveform which is then passed over to the DMX-1000 for synthesis.

The host computer which is currently in use at San Jose State's Music Department is a Terak 85101a minicomputer running RT-11 Version IV as its operating system. This computer is capable of running MUSIC 1000, a language designed specifically to control the DMX-1000. The programming must be done within the context of an available text editor. The most accessible to use in the present environment is A Screen Oriented Text Editor (ASOTE). Documentation concerning ASOTE exists as an appendix within this manual (see Appendix A).

HOW TO USE THIS MANUAL

The organization of this manual is developed in such a way as to allow the user to explore various applications of the DMX-1000 by audio, program, and flowchart illustrated examples. Spectra of the audio examples obtained through Fast Fourier Transforms (FFT) will also be used. For each of the examples included within this manual accompanying information will exist in the following format:

1. Two separate flowchart illustrations of the "patch."

2. A listing of the appropriate Music 1000 program.

3. A disk which contains the Music 1000 program for a real-time demonstration of the example.

4. A cassette of the audio example.

The sources listed above should provide the user with adequate information regarding the basic functions of the DMX-1000 and the MUSIC 1000 language. Additional information will also be necessary at times and resources will be included as needed.

HOW TO LISTEN TO THE DMX-1000 EXAMPLES

The DMX-1000 is currently rack-mounted with its Terak host computer. There are three switches located on the front panel: one for power to the DMX-1000, one for power to the Terak, and the third switch is used to reset the Terak. Two 8" disk drive units are also located on the front panel of the rack mount case. As illustration 1.2 shows, the "A" drive is on the left and the "B" drive is on the right.

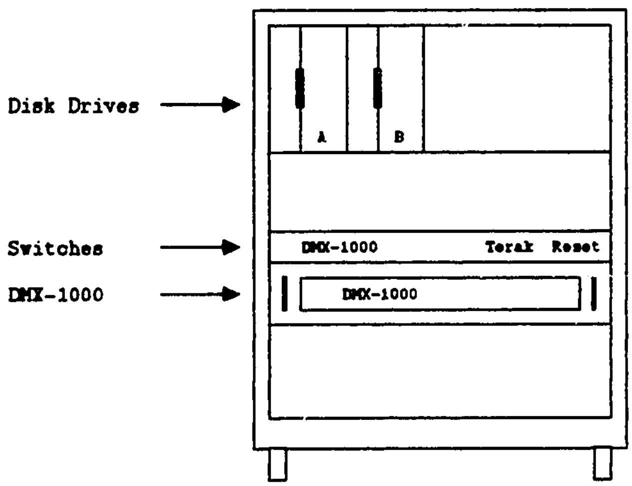


Illustration 1.2 System Hardware Locations

To boot the system, a *MUSIC 1000 Systems Disk* must be placed in the "A" drive. The power may then be switched on for both the DMX-1000 and the Terak. The system will then boot itself. The appropriate example disk is then placed in the "B" drive and the following must be done in order to hear the example:

1. Be sure that the audio system is turned on.

2. Type ***RUN SINE*** SINE is only used for the first example, other names must be used for different examples.
   Press ***<return>***

3. Upon the prompt SCORE NAME?
   Press ***<return>***

4. Upon the prompt DO YOU WANT THE FUNCTIONS DISPLAYED?

   Type *N*

   Press *<return>*

The audio example should then be heard. These instructions will be the same for each diskette with the exception of the program name. Each example disk will be labeled with the appropriate program name necessary for execution.

There are several diskettes available for use which include examples that will be described throughout the remaining chapters of this manual. The examples can be listened to at any time and the previous instructions will apply for every diskette.

It may be useful to view the MUSIC 1000 code for the various audio examples. Once the diskette is inserted in the drive unit, the Score can be printed on the screen by typing the following:

TYPE SCORE.USR <return>

In order to view the Orchestra type:

TYPE ORCH.USR <return>

The textual files named SCORE.USR and ORCH.USR contain the MUSIC 1000 code necessary for synthesis on the DMX-1000. It is recommended that the audio examples are run to provide a basic familiarity with the Terak computer system, the audio system, and the DMX-1000 itself. The following chapters will provide both general and detailed information regarding synthesis with the DMX-1000 and the MUSIC 1000 language.

CHAPTER 2

THE DEVELOPMENT OF MUSIC 1000

A number of programming languages for computer related musical instruments have been in existence since the 1950's when experimental programs for musical realization were initially developed. According to Dodge (1985), the first of many general-purpose programs for computer synthesis was generated at Bell Laboratories in the early 1960's by Max V. Mathews. The language that Mathews created was known as MUSIC 3, and was developed from its predecessors MUSIC 1 and MUSIC 2. A fourth version of the language titled MUSIC 4 was then created and used by various universities throughout America. MUSIC 4 was a language which made the first true generation of computer related music possible as a wide spread activity.

MUSIC 4 was generally considered to be a somewhat definitive language. Further modifications, however, were necessary and numerous updated versions of the language surfaced at different locations. Princeton University claimed MUSIC 4B and MUSIC 4BF while Stanford created MUSIC 6 and MUSIC 10 (see page 6). Although these languages shared a number of similarities, there were drawbacks as well. The largest of these drawbacks was the fact that none of these various MUSIC languages supported real-time synthesis. MUSIC 1000 was created by Dean Wallraff specifically for the DMX-1000 as a real-time alternative to the earlier MUSIC language environment. MUSIC 1000 has its roots in similar languages including MUSIC V, MUSIC 11, and MUSIC 360. Like the programs developed before it, MUSIC 1000 also possesses some idiomatic problems which can make certain musical procedures somewhat difficult. MUSIC 1000, however, does provide the user with a tangible and useful format for creative expression and real-time synthesis.

MUSIC 1000 OVERVIEW

(As stated in Chapter 1,) MUSIC 1000 is designed to make use of two separate sections of textual information which must be created independently. The interaction between the two is critical. These independent sections are known as the *Score* and the *Orchestra*. The Score contains data for musical realization including pitch specifications for the instrument(s), tempo and amplitude values, and additional information such as location and delay time parameters. The Orchestra section of MUSIC 1000 is used to create instruments in a fashion which is conceptually not unlike creating an instrument by "patching" on a traditional synthesizer. The instrument must be defined according to the rules imposed by MUSIC 1000 but the flexibility is significant. The instrument developed within the Orchestra section then looks for the values (known as parameters) located within the Score section and executes these instructions accordingly. An analogous process might involve the construction of a violin (the Orchestra), and the sheet music which instructs the violinist what to play (the Score).

SYSTEM PARAMETERS

The Score section of MUSIC 1000 contains various parameters which are numerical values specified by a 2's compliment depiction of a 16 bit number (within a range of -32767 and +32767). Any values used in the Score must fall within this range or an error message will result. The Score appears as lines of text and each line of the Score section is executed before the computer moves on to the next. Each of these lines may contain as many as 50 parameters. The first three parameters (known as p1, p2, and p3 respectively) are fixed with a specific meaning. The remaining parameters, p4 through p50, are available for user definition.

The first parameter, p1, refers to the instrument reference number. The number of possible instruments which can be used at any one time depends upon the complexity of the Orchestra and the available memory space within the DMX-1000 itself. In general, the first instrument used in the Score will be known as 1, the second as 2 and so forth. This does not, however, imply order of appearance since instrument numbers may be of any value as long as duplication is avoided.

The second parameter, p2, represents the starting time of the instrument in arbitrary units known as *beats*. The instrument will not become active until it is instructed to do so. Parameter p2 allows for a delay or pause to be inserted before the instrument is made active. If the value 10 for example is used at p2, a ten second pause would occur before the instrument becomes active.

The third parameter, p3, refers to the duration of the event in beats. The instrument will become active for the duration specified at p3 which defaults to time in seconds. A value of 10, therefore, when used at p3 will create a duration of 10 seconds for the specified instrument. The rest of the parameters, p4 through p50, are available to the user for various implementations which will be discussed in detail throughout the remaining portions of this manual.

In order to illustrate the first three parameters, an *i* (event) statement from the Score will be used. The first character in the line is the opcode *i* which creates an event (this will be elaborated upon in following chapters). The three characters which follow the *i* are the first three parameters, p1, p2, and p3 respectively. The values used in the example below would create an event for instrument number 1 which delays for 10 seconds then becomes active for 10 seconds.

<i statement	*inst#*	*delay*	*duration>*
i	*1*	*10*	*10*

SYSTEM TIMING RELATIONSHIPS

The DMX-1000 utilizes three separate timing features which are available for use in both the Score and the Orchestra sections of MUSIC 1000. These three timing clocks are referred to as the *I-rate*, *X-rate*, and *K-rate* variables. These variables can be used for different types of instrument control computation according to a selected rate. These clocks can also be used to trigger certain events at specified times throughout the execution of a MUSIC 1000 program.

The *I-rate* variable refers to the *initialization rate* of an event for any single instrument. This initialization rate is executed within one millisecond at the time that the instrument becomes active. *I-rate* instructions allow for the computation of numerical values before the instrument is made active. This may be necessary for the calculation of envelope attack and delay times, pitch parameters and so forth. *I-rate* addition, for example, is accomplished with an *IADD* statement which adds two values together and places the sum in a specified location.

The *X-rate* variable represents the audio rate or the *sampling rate* of the DMX-1000. Two sampling rates are possible, either 20kHz or 40kHz. The *X-rate* clock runs at the selected audio rate and can be used for various high speed computations and changes effecting the audio output of the DMX-1000. *X-rate* calculations such as *XADD* can be used to add audio signals for various implementations including additive and modulation synthesis.

The third available timing clock is known as the *K-rate*, or control rate. This clock has a frequency of 100 Hz and it is used for continuous control signal generation. Envelopes for example can be created with *KLINE* statements which produce line segments (linear events with a starting point, an ending point, and a duration). Illustration 2.1 shows the relationships of the three timing clocks described above.

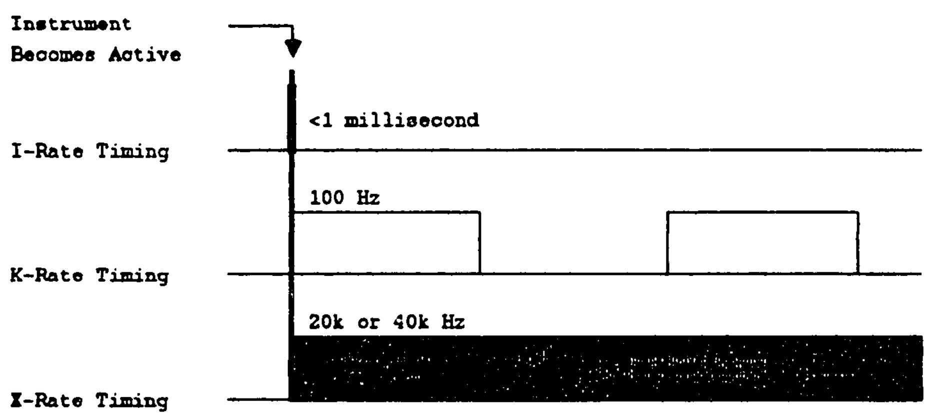


Illustration 2.1 System Timing Relationships

FLOWCHART DESCRIPTION

In order to illustrate the instrument created on the DMX-1000, two styles of flowchart representations will be presented. These flowchart diagrams will be used to display the patch for any particular instrument in standard formats for audio synthesis. The various instruments can then be studied and evaluated according to their architecture as illustrated by the flowchart diagrams.

The first method of illustration was codified and published by Strange (1972), and has since become a well recognized format for patch notation. This method involves the use of various symbols which represent conventional synthesizer modules such as oscillators, function generators, and amplifiers. These modules are then connected with lines which initially represented physical patch cords. Digital synthesizers, however, require no physical patching with cords as this process is now simulated entirely with software. While electronic instruments have evolved considerably in recent years, this method of patch notation is still valid regardless of the type of synthesizer (or computer) involved. Illustration 2.2 includes symbols which will be necessary for this type of illustration as well as an example of a simple audio patch.

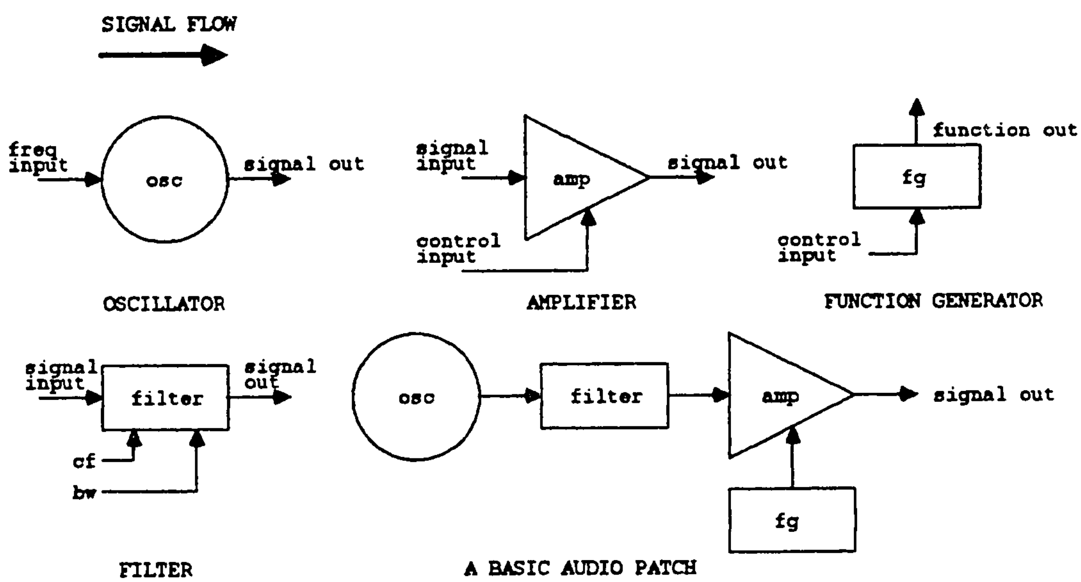


Illustration 2.2 Modular Flowchart Components

The second type of flowchart diagram to be presented is currently the accepted format at computer music institutions such as The Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University, and the Institut de Recherche et Coordination Acoustique/Musique (IRCAM) in Paris. This method of instrument illustration is based upon accepted graphic representations of mathematical processes and it utilizes "unit generators," which are essentially modules or digital signal processors. These modules exist only as numbers within the digital computer and are not physical elements. Each unit generator has at least one input and one output and may perform signal generation, modification, or a combination of the two. Unit generators may also be patched together in order to create complex synthesis algorithms. Illustration 2.3 is an example of a unit generator patch notation.

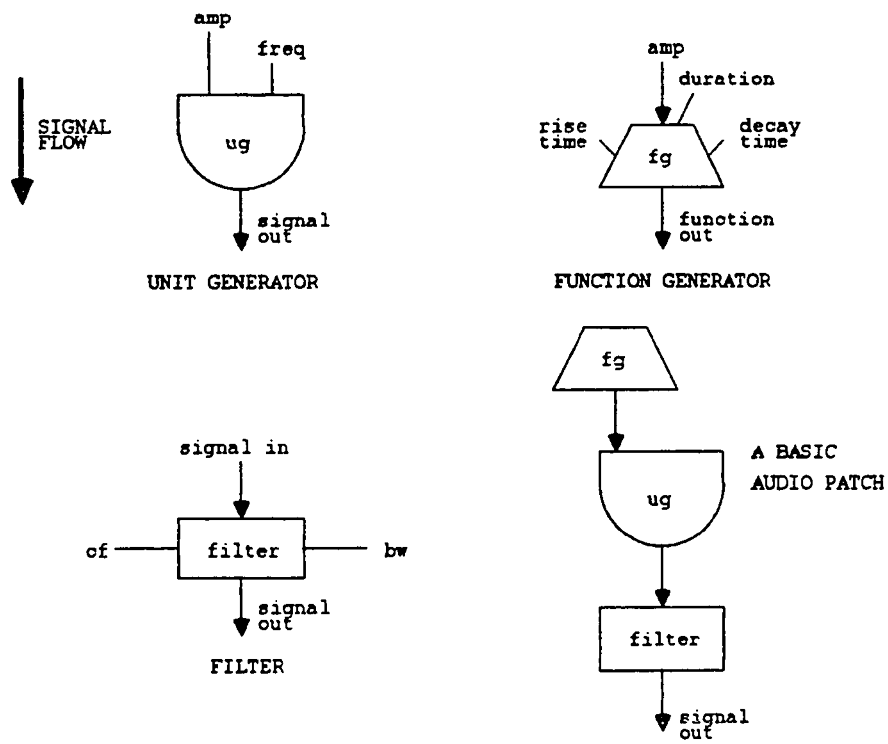


Illustration 2.3 Unit Generator Components

The two methods of patch notation introduced above will be used to illustrate the MUSIC 1000 code in a graphic fashion. The illustrations are intended to clarify the examples presented in following chapters. Each example will be diagrammed with specific parameters from the appropriate MUSIC 1000 code in order to provide detailed information regarding the patch. Two styles of graphic notation are used to reinforce the examples construction and the relationship of the MUSIC 1000 code to the actual software patch.

CHAPTER 3

BASIC SYNTHESIS CONVENTIONS

Additive synthesis, subtractive synthesis, and modulation synthesis can be demonstrated with audio examples programmed in MUSIC 1000 for the DMX-1000 computer. These synthesis techniques can also be represented graphically with flowchart diagrams, musically notated illustrations of the spectral content of the example, and spectral analyses obtained through fast fourier transforms (FFT). A number of MUSIC 1000 conventions will be used throughout the examples presented in following chapters and these conventions can be summarized in advance.

The Score section of MUSIC 1000 contains a set of numbers which correspond with various parameters defined in the Orchestra. These parameters are numerical values which are assigned to specific functions or processes such as modulation index, oscillator frequency, etc. The Score is organized with parameters beginning at p1 (or parameter #1) on the left, followed by p2, p3, etc. with a maximum limit of 50 parameters allowed. The Score is executed one line at a time from the top down. A different value for any parameter from one line to the next will create a change in some aspect of the output of the DMX-1000. The following illustration diagrams this process.

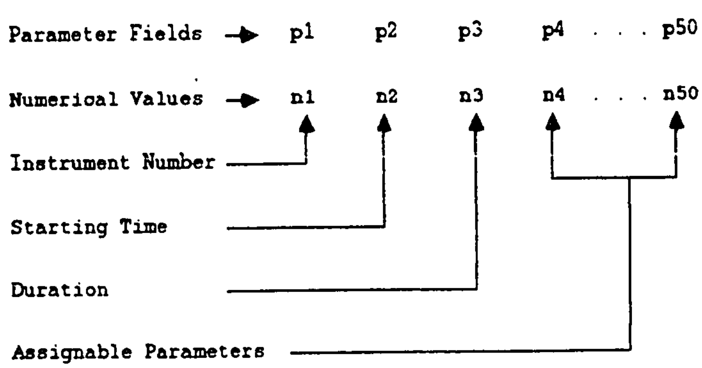


Illustration 3.1 Score and Parameter Relationships

The Orchestra portion of MUSIC 1000 is used to build various instrument configurations. These instruments then refer to the Score for parameters which represent controlling aspects of the audio output. Only the first three parameters of the score are not user definable. The remaining parameters, p4 through p50, can be assigned to control some part of the instrument. For example, parameter p4 can be used to control the audio output of the DMX-1000. As p4 is varied from one line of the Score to the next, the amplitude of the audio output also changes.

Both the Score and the Orchestra section require that certain MUSIC 1000 words are included for proper operation of the DMX-1000. In addition, a number of words will be common to each example listed in subsequent chapters. The following section addresses various MUSIC 1000 words necessary for the Orchestra and the Score.

ORCHESTRA CONVENTIONS

Orchestra Syntax

The Orchestra consists of a number of statements which are executed sequentially in the following format:

ORCH *[fast]* *[,lead]*
functions
instruments

The MUSIC 1000 word *ORCH* is a required element of the Orchestra. The word *ORCH* begins the Orchestra. Two options (which are identified by brackets) are also possible, *[fast]*, and *[,lead]*. The word *FAST*, if present, allows the DMX-1000 to operate at a sampling rate of 40kHz which establishes a frequency range of 0Hz to 20kHz (the maximum obtainable frequency equals one-half of the sampling rate). If *FAST* is not present, the DMX-1000 will default to a sampling rate of 20kHz establishing a 0Hz to 10kHz range. The advantage of using the default sampling rate, 20kHz, is improved frequency resolution throughout the entire 10kHz range.

The second option *[,lead]* refers to a lead time for processing in which numerical calculations can occur. *LEAD* is represented by a numerical value preceded by a comma. Lead time is necessary when there are large quantities of numerical calculations required. Typical lead values range from 10000 to 25000. The *ORCH* statement may have the following format:

ORCH fast ,25000

Note: MUSIC 1000 does not make a distinction between upper and lower case text formats. Both are acceptable.

Function Declarations

MUSIC 1000 uses functions that can generate waveforms, envelopes, tables, and various types of filters. Function declarations create the function by setting up tables of numbers to be used for processing or synthesis. The word *FNCTN* begins the function declaration which has the following format:

FNCTN NAME,SIZE,TYPE [,NORMAL]
[Nargs, argument list]

NAME refers to a user selected title for the function by which it will be referred to elsewhere in the Orchestra. A function which generates a sine wave for example could be named *sine*. A partial *FNCTN* declaration with the *NAME* sine would look like this:

FNCTN sine,

Due to the fact that functions are essentially tables of numbers, memory space must be reserved for each function declaration. *SIZE* is the amount of memory which will be allotted to an individual function. *SIZE* must be a power of 2 between 8 and 2048, and the larger the size, the better the resolution of its function. A maximum limit of 4096 is imposed by the hardware, therefore, the sum for all *SIZEs* used in any Orchestra must be 4096 or less. Functions utilize both the DMX-1000 data memory and the host computers internal memory. An incomplete *FNCTN* declaration with the *NAME* sine and a *SIZE* of 1024 would appear as follows:

FNCTN sine,1024,

TYPE refers to the nature of the function to be generated. There are many *TYPEs* available which allow for filtering, fourier synthesis, enveloping etc. (refer to the MUSIC 1000 manual for a complete listing). Fourier synthesis for example is achieved by selecting *fourier* as the *TYPE*. The above *FNCTN* declaration with the addition of fourier as *TYPE* would be:

FNCTN sine,1024,fourier

The optional argument *NORMAL* is used to eliminate aliasing or overflow by proportionally scaling numerical values so they do not exceed 32767. The numerical values, known as the argument list, are located on the second line of the function declaration and can not exceed 32767 when added together. If the sum of the numbers in the argument list exceeds 32767, the word *NORMAL* can be used to scale the values so overflow does not occur. *NORMAL*, when added to the *FNCTN* declaration from above would appear as follows:

FNCTN sine,1024,fourier,normal

Nargs represents the *number of arguments*, or the number of elements which will be used to generate specific functions. *Nargs* is a value followed by a comma and the *argument list*. The *argument list* must contain as many values as determined by *Nargs*, each of which must be separated by commas. The *argument list* can be used to specify individual harmonic amplitudes, the points of an envelope, the contour of a filter, etc. The included audio examples will provide additional information concerning this process. The complete *FNCTN* declaration including *Nargs* and an *argument list* would appear as follows:

FNCTN sine,1024,fourier,normal
1,32767

It is important to note that the argument *normal* is not needed in this application since the value on the second line of the function declaration does not exceed 32767. It has been retained, however, for the sake of continuity and it would not affect the operation of the program.

Instrument Declarations

Instrument declarations are used to create sound-generating instruments and are often referred to as one voice or one part. Instruments are called upon to become active or played by the Score. Instrument declarations have the following format:

INSTR N
storage allocation statements
unit statements
ENDIN

INSTR designates the beginning of a user defined instrument declaration. *N* refers to the number of the instrument. Instrument numbers should not be repeated in the Orchestra and they are usually consecutive in nature. Consecutive numbers, however, are not imperative.

Storage Allocation Statements are used to set aside memory space for variables which are identified by user selected names. The variable names must begin with a letter and be no longer than six characters. Variables can be used to hold the output of certain *I-rate* or *K-rate* computations as they are essentially memory locations that hold the results of mathematical operations.

Storage may be allocated for variables which can be either local or global in nature. Local variables are used only within the instrument for which they are defined. Global variables may be used with any instrument created in the Orchestra. The format for storage allocation is as follows:

LOCAL <var1,var2,...>

GLOBAL <var1,var2,...>

The words in brackets, *var1* and *var2*, represent variable names. The name for each variable should be descriptive of its purpose. For example, an amplitude envelope could be named *ENV*. A number of variable names are reserved and must not be used (refer to the MUSIC 1000 manual for a complete listing). An example of a *LOCAL* statement which reserves memory space for two variables, *env1* and *env2*, is as follows:

LOCAL <env1,env2>

Unit Statements represent specific functions or processes which generate and control sound. Units consist of oscillators, modifiers, output control etc. Unit statements are used to create audio patches by connecting a number of units together with software.

Unit Timing refers to the rates at which various units operate, (See Chapter 2, page 3). There are three timing rates available, *I-rate; K-rate* and *X-rate*. *I-rate* occurs only once during each event. *K-rate* is executed every 10ms while the instrument is active and is similar in nature to a control voltage. *X-rate* code occurs at the sampling rate of the instrument, either 25 or 50 microseconds (20kHz or 40kHz). *X-rate* events generate audio rate signals.

Unit statements are preceded by a letter which designates its timing frequency, either *K, I,* or *X*. The following section is a list of unit statements are used in the examples to be presented later. Refer to the MUSIC 1000 manual for a complete listing of unit statements.

KLINE ***KOUT,Istart,Idur,Iend***

KLINE is a unit statement which is used to generate a single line segment. This line segment begins at the value *Istart,* travels to value *Iend* over a time specified by *Idur*. Simple envelopes can be created with *KLINE* as shown in illustration 3.2.

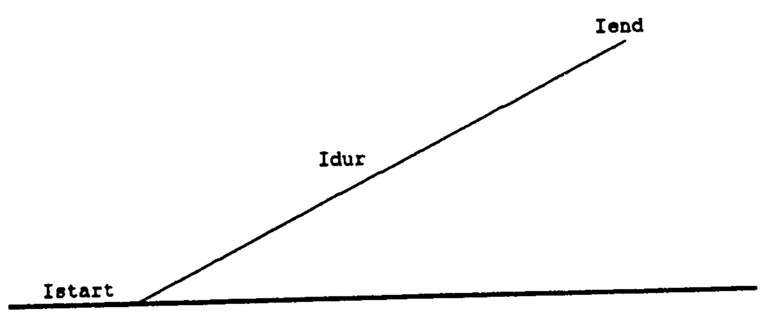


Illustration 3.2 KLINE Envelope Diagram

A simple *KLINE* statement which begins at 0, (*Istart*) travels to 100, (*Iend*) over a duration of 10, (*Idur*) would have the following format, while the output of the *KLINE* statement is placed at *KOUT*, or in this example, placed into the variable *env1*.

KLINE ***env1,#0,#10,#100***

Note: the <#> symbol must be used in front of numerical values in the Orchestra section of MUSIC 1000.

The *OSCIL* statement is used to generate either audio-rate or sub-audio oscillators. The *OSCIL* component contains four elements which determine the overall output of the designated oscillator. The first element, *XOUT*, refers to the register or location of the oscillators signal. *XOUT* is typically assigned to a register within the DMX-1000 such as *x6* or *x7*. *IFUNC* represents a function generated with the *FNCTN* statement. The function name must be preceded by the # symbol. *XKSI* refers to the frequency of the oscillator as specified by either an *X-rate* or a *K-rate* value. *XKVOL* also may use an *X-rate* or a *K-rate* value to specify the amplitude of the oscillator.

OSCIL XOUT,IFUNC,XKSI,XKVOL

The following is an example of the *OSCIL* unit statement as it may appear in the Orchestra using *x7* as an output location, (x7 is a memory location or register within the DMX-1000), the *FNCTN* sine, the parameter p4 for frequency input, and p5 for volume (p4 and p5 are located in the Score).

OSCIL x7,#sine,p4,p5

An *XNICE* statement is included mainly to eliminate "clicks" at the beginning and end of a sound. This is achieved by the generation of very short ramps (50ms unless otherwise specified by *[IDUR]*), at the start and end of the output. The signal is brought into *XNICE* at *XIN*, and is placed at the location specified by *XOUT*.

XNICE XOUT,XIN,[IDUR]

The *XNICE* unit statement might appear in the following manner using register x7 for both the input and output locations:

XNICE x7,x7

Note: Registers may be compared to patch cords as various signals can be routed through the available registers in the same way a signal can be transferred through a wire. The available registers which appear to work with most applications are x6, x7, x8 and x9. Other registers do exist, but they may be in use, depending on the complexity of the instruments in the Orchestra.

OUT is used to place the signal located at *XIN* at the specified channel(s). Channels are determined by the value *ACHNL*. If *ACHNL* = 1, output occurs through channel

A only. If *ACHNL* = 2, output is at channel B only. If *ACHNL* = 3, output occurs at both channels.

OUT ***XIN,ACHNL***

An example of the *OUT* statement which uses register *x7* as its output, and the value 3 for channel location would appear in the Orchestra as follows:

OUT ***x7,3***

A comment statement is technically not a unit statement since it has no effect on the DMX-1000. The semicolon (;) is used in the Orchestra for documentation. Anything following the semicolon is ignored by the DMX-1000. Comments are for informative purposes only and they do not affect the operation of the MUSIC 1000 program.

COMMENT STATEMENT ***;*** ***comment***

A *COMMENT* statement for example may be:

; ***Frequency Modulation Voice 1***

ENDIN is a word which must conclude each instrument created in the Orchestra. *ENDIN* signals the *end of an instrument* and it is an essential element.

SCORE CONVENTIONS

The Score section of MUSIC 1000 contains information which is used to control various events. Events must have at least three properties: an instrument number, a starting time, and a duration. These properties are listed in the form of parameters and as many as fifty parameters can be associated with any single event.

The Score is organized in a sequential manner with a single alphabetic character (or opcode), followed by parameter fields, or pfields. Pfields may be positive or negative numeric values and floating point numbers are allowed. The format for score is as follows:

opcode ***p1 p2 p3 ... p50***

The following Score Statements are used throughout the examples presented in following chapters. For a complete list of possible score statements refer to the

MUSIC 1000 manual.

COMMENT STATEMENT *C* *comment*

The Score uses the C character to identify comment statements. Anything following the C will have no effect on the operation of the MUSIC 1000 program. Comments are essential sources of documentation and may be used at any point within the Score.

An example of a *COMMENT* statement as it might appear in the Score is this:

C *Frequency Modulation Score1*

SCALING STATEMENT *X p1 p2 p3...p50*

The *X* statement is used to scale various numerical values to make them more manageable by multiplying the numbers by some factor. This process is important when certain conditions, such as the expression of oscillator frequency in Hertz, is desirable. Scaling statements are used to make this possible. The *X* statement may contain numerical values or any of the MUSIC 1000 scaling values which are provided for the user. The following MUSIC 1000 scaling values are used in following chapters:

SCPS *This scaling value allows for the specification of oscillator frequency in Hz.*

DB *The value DB allows for the specification of amplitude in decibels.*

A scaling statement which uses the scaling value *SCPS* at p4 and *dB* at p5 would appear as follows:

X *1* . . *SCPS* *dB*

Note: The dots located at p2 and p3 are characters which are used to fill the space at these parameters since the scaling statement ignores them. Any character should work; however, the locations must be filled.

EVENT STATEMENT *I p1 p2 p3...p50*

Event statements are used to call upon a certain instrument to become active for a specified duration.

The *I* statement may include values for oscillator frequencies, amplitudes, modulation frequencies, etc. These values are placed in the various pfield locations and used in connection with the Orchestra. An *I* statement might appear as below:

<opcode	*p1*	*p2*	*p3*	*p4*	*p5>*
I	*1*	.	*10*	*-25*	*100*

Note: The words enclosed in brackets above the *I* statement are for reference only.

REST STATEMENT *R p1 p2 p3*

The *REST* statement is used to create a rest for instrument p1. The *R* statement creates a timed silence by adding the duration, p3, to the p2 value of the next *I* statement. A ten second *REST* for instrument 1 would have the following format:

<opcode	*p1*	*p2*	*p3>*
R	*1*	.	*10*

END STATEMENT *E*

This statement signifies the *END* of a Score. All Scores must end with the *E* statement on the final line of code.

The conventions described above are common to the examples which will be presented is the following chapters. The definitions of these conventions will be reinforced by applications appearing in the various demonstrations. The examples are intended to illustrate general operations of the MUSIC 1000 language.

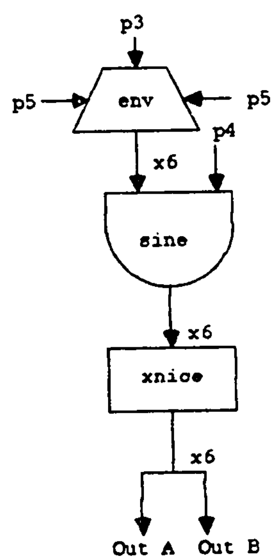
CHAPTER 4

ADDITIVE SYNTHESIS

The demonstrations of additive synthesis are located on Disk #1, titled ***Additive Synthesis Examples.*** Four examples are included: Sine, Square, Saw, and Invert.

Additive, or Fourier synthesis is achieved by combining sine waves with different frequencies and amplitudes. This process enables the generation of complex waveforms by the addition of simple sine tones. As more frequencies are added to the fundamental, the resulting spectrum changes thereby varying the overall timbre of the sound. By combining the proper frequencies and amplitudes, additive synthesis can be used to create basic waveforms such as the square wave, and the sawtooth wave.

The following example demonstrates a sine tone at a frequency of 220 Hz for a duration of ten seconds. The flowchart patches are shown in illustration 4.1 while illustration 4.2 is the MUSIC 1000 code for the patch. Illustration 4.3 shows the spectrum (or lack thereof) and the waveform of the first example.



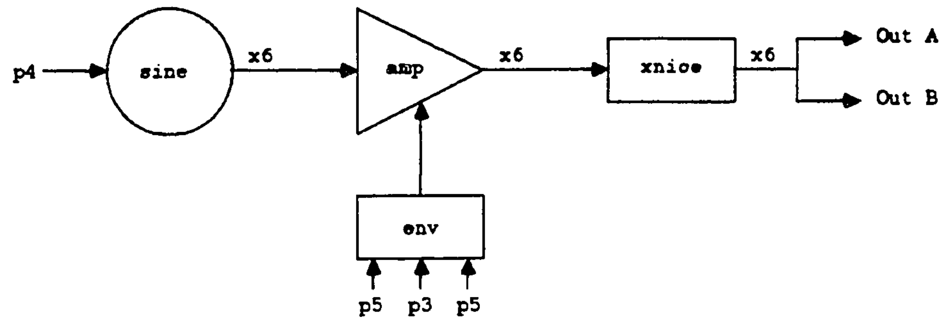


Illustration 4.1 Flowchart Diagrams for Additive Synthesis

SINE WAVE SYNTHESIS

The Orchestra

```
;       Sine Wave Demonstration

orch

fnctn   sine,1024,fourier
        1,32767

instr   1
local   <env>
kline   env,p5,p3,p5
oscil   x6,#sine,p4,env
xnice   x6,x6
out     x6,3
endin
```

The Score

```
c    Fourier Synthesis Example

c    p1     p2      p3     p4     p5
c    ins#   start   time   freq   ampl

x    1      .       .      scps   dB
i    1      0       10     220    -20
e
```

Illustration 4.2 MUSIC 1000 Code for Sine Wave Generation

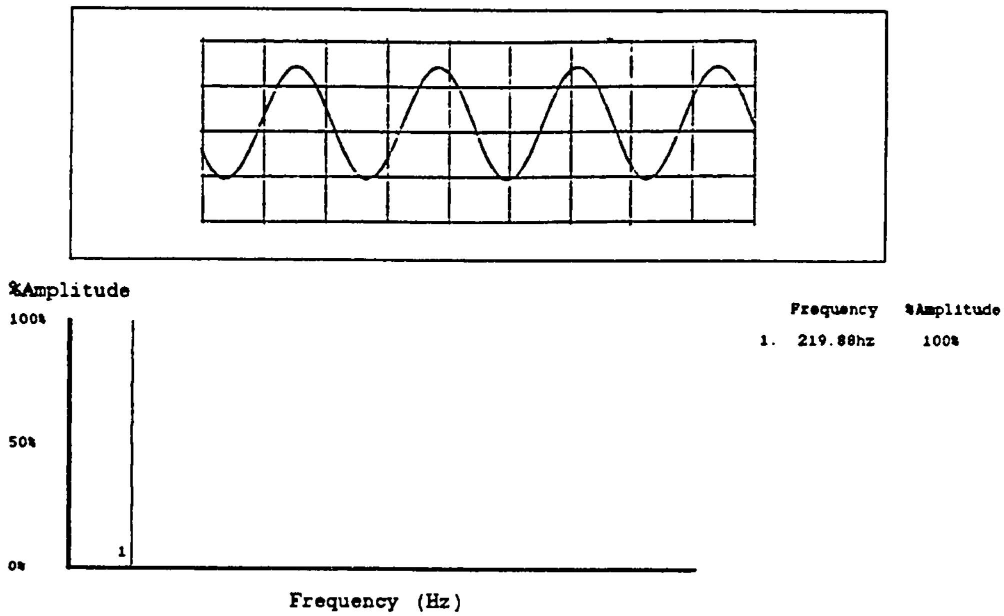


Illustration 4.3 Sine Wave and FFT Results

The Orchestra section which is used to create the sine tone consists of eleven lines of code. The numbers at the left in brackets are for reference only and are not a part of the MUSIC 1000 language. The Score section contains six lines of instructions and, like the Orchestra, the numbers are for identification purposes only. A line by line analysis of these two sections will assist the user in developing a basic understanding of the MUSIC 1000 format.

The Orchestra

<1> *;* ***Sine Wave Demonstration***

Line #1 of the Orchestra section is a comment which provides a title for the example. The semicolon is used for comment statements in the Orchestra.

<2> ***orch***

Line #2 activates the Orchestra section with the word *orch*. This is a required element of the Orchestra and it must not be omitted. It is important to note that both upper and lower case text formats are allowed.

Function Declaration

```
<3>          fnctn      sine,1024,fourier
<4>                     1,32767
```

Line #3 begins the *FNCTN* statement. These function declarations are used to build numerical tables which, in this case, represent wavetables. The function generated in lines three and four of the code is that of a sine wave. Line #3 is used to declare the name, size, and type of function. In this example the name of the function is *sine*, its allotted memory size is *1024*, and its type is *fourier*.

Line #4 is used to create the *number of arguments, (Nargs)*, and an *argument list*. The sine wave is a single element and is generated through the Fourier process. When the type is Fourier, the argument list refers to the individual harmonics and their relative amplitudes. In order to create a sine wave a single argument is needed; therefore, the number of arguments must equal one. The value 32767 refers to the amplitude of the sine wave. Since 32767 is the maximum value allowed due to a 16 bit limit, the sine wave will be at its maximum possible amplitude. The output of the sine wave, however, may still be attenuated by other parameters before reaching the loudspeaker.

Instrument Declarations

```
<5>          instr      1
```

Line #5 uses the word *INSTR* to refer to the selected instrument and reference number. *INSTR* must be followed by a numerical value which corresponds to a certain patch or voice. This value is then used in the Score section when the instrument is called upon to become active.

```
<6>          local      <env>
```

The *LOCAL* statement is a storage allocation statement which allocates a storage area for the variable named *env*. The variable *env* is local and will be recognized and used only by instrument #1. The brackets which enclose the word *env* are essential.

```
<7>          kline      env,p5,p3,p5
```

Line #7 is used to create a function or envelope by generating a line segment. The envelope generated in this example uses the variable name *env* established in the previous line (notice that no brackets are used in this statement). This envelope is used to control the output of the instrument by generating an amplitude function.

In the sine wave example a simple on-off type of envelope is created with the *KLINE* statement. The function titled *env* uses only two parameters from the Score section. The parameters which are a part of the Score, p3 and p5, represent numerical values. Parameter p3 is a duration which in this case equals ten seconds and the value, 10, can be found under p3 in the Score. Parameter p5 is an amplitude value expressed in decibels in the Score. This example uses an amplitude value of -20 dB which is found under p5. Illustration 4.4 shows the envelope *env* with the values from the Score section.

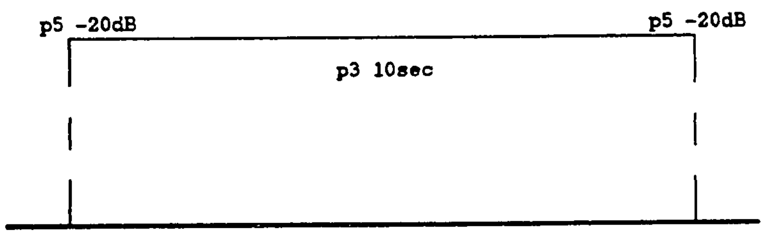


Illustration 4.4 Amplitude Envelope

<8> *oscil* *x6,#sine,p4,env*

Line #8 contains the word *OSCIL* which is used to generate an audio-rate oscillator. The *OSCIL* component contains four elements which determine the overall output of the designated oscillator. The first element, *XOUT* refers to the output location of the oscillator. *XOUT* has been assigned to *x6* which is a register within the DMX-1000. These registers may be compared to traditional patch cords which are used to transfer signals to and from various units.

The second component of the *OSCIL* statement is reserved for *IFUNC*. The function titled *sine* is preceded by the number symbol (#) and used for this example. The number symbol is used in the Orchestra to signify constant numerical values, or functions, and it can not be omitted. The frequency of the oscillator is determined by the third element of the *OSCIL* unit. The tone to be produced is to have a frequency of 220 Hz. The value, 220, is located within the Score section of the example at parameter p4. The Orchestra looks for p4 in the Score, finds a value of 220 in this location, and applies it accordingly.

The final portion of the *OSCIL* statement represents the amplitude of the oscillator. The *XKVOL* component determines the output amplitude. The output of this oscillator is controlled by the envelope *env*.

<9> ***xnice*** ***x6,x6***

The ninth line of the Orchestra section uses the word *XNICE*. The *XNICE* statement is included mainly to eliminate "clicks" at the beginning and end of the sound. Register *x6* serves as both an input and output location for the signal.

<10> ***out*** ***x6,3***

OUT places the signal *x6* at the specified channel(s). The value 3, which follows *x6* allows for audio output through both channel A and channel B.

<11> ***endin***

The final line of the Orchestra is *ENDIN* which signifies the end of an instrument. *ENDIN* is a required final statement for *each* instrument developed in the Orchestra section.

The Orchestra is used to generate instruments by connecting a series of modules together with software. The type of synthesis to be used is based entirely upon the instrument definition developed in the Orchestra. The Score is then used to control and manipulate the instrument according to the configuration created by the user. The Score can not define synthesis schemes, it can only control the instrument(s) in the Orchestra. The following section describes the Score, and its interaction with the Orchestra defined above.

The Score

The Score section for the first audio example consists of six lines of MUSIC 1000 code. Lines #1 through #3 are comment statements as designated by the first character in the line, *c*.

```
<1>        c     Fourier Synthesis Example

<2>        c     p1     p2     p3     p4     p5
<3>        c     ins#   start  time   freq   ampl
```

The above comment statements provide information for the user which help to clarify the construction of the Score. Line #1 is used as a title for the example, while lines #2 and #3 document the parameter numbers and their assigned applications.

```
          <opcode  p1    p2    p3    p4     p5>
<4>           x    1     .     .     scps   dB
```

Line #4 is a scaling statement which is identified by the opcode *x*. Scaling statements allow for the use of manageable numerical values for various types of operations. In this example two pre-determined scaling factors (*scps* and *dB*) are used for frequency and amplitude. Parameter p1 refers to instrument number and since only one instrument is necessary for this example, the number 1 is located under p1.

The *x*, or scaling statement ignores parameters p2 and p3. These parameters in the Score must be filled with a character such as the period demonstrated here.

The MUSIC 1000 words *scps* and *dB* are scaling values for oscillator frequency and overall amplitude. The value *scps*, located at p4, allows for the specification of oscillator frequency in cycles per second or Hertz. The numerical equivalent for *scps* is 3.2768. The value *dB*, located at p5, refers to an amplitude attenuation value scaled in decibels. The range for the *dB* value is from -96 (no output) to -0 (maximum output). The scaling statements do not generate audio data but rather provide a format for subsequent score lines.

```
          <opcode  p1    p2    p3    p4     p5>
<5>           i    1     0     10    220    -20
```

This *i*, or event statement is used to call upon a specified instrument and activate it for a certain duration. The first parameter always refers to instrument number.

The second parameter is for starting time of the instrument in arbitrary units known as beats. The default time for one beat is one second. The value 0 is used because it is not necessary to generate a pause before the instrument is made active.

Parameter p3 represents the duration of the event. This corresponds directly with the Orchestra and the envelope created known as *env*. Parameter p3 was used to specify the duration of the envelope. The duration of p3 is expressed in beats, therefore, under default conditions the value 10 represents ten seconds.

The fourth parameter in this example is used for oscillator frequency, and the scaling statement in line 4 allows for it to be specified in cycles per second. The p4 value of 220 is used to generate an oscillator frequency of 220 Hz. This is possible due to the scaling factor *scps*, which generates the proper numerical value for frequency expressed in Hertz.

The fifth parameter, p5, is used here to represent amplitude attenuation and according to the scaling statement is expressed in *dB*. The dB value of -20 is applied to the envelope *env*. The value -20 when scaled by the *dB* factor is 3277. This generates a moderately loud output.

<6> *e*

The final line of the Score, line #6, is an end statement. The opcode *e* is required to signify the end of the Score. All scores must end with this line.

The above example for sine tone generation can be easily modified in order to produce more complex waveforms. The Score which has been used for the first example will work equally well with the following Fourier synthesis examples. Modification of the Score is therefore unnecessary. The Orchestra section requires only slight changes in the *FNCTN* portion of instrument definition. The harmonic content of the generated sound is the only element which will be effected by alteration of the FNCTN statement.

SQUARE WAVE SYNTHESIS

In order to generate a square wave lines 1, 3, 4, and 8 of the Orchestra must be changed. Lines 3 and 4 are used to create the harmonic content for the selected instrument. The square wave contains only odd-numbered harmonics and the amplitude for each harmonic within the spectrum decreases as the harmonic number increases. Illustration 4.5 shows the MUSIC 1000 code necessary for square wave generation while illustration 4.6 shows the FFT results and waveform.

The Orchestra

```
<1>      ;       Square Wave Generation

<2>       orch

<3>      fnctn      square,1024,fourier,normal
<4>                 9,100,0,33,0,20,0,14,0,11

<5>       instr      1
<6>       local      <env>
<7>       kline      env,p5,p3,p5

<8>      oscil      x6,#square,p4,env

<9>       xnice      x6,x6
<10>      out        x6,3
<11>      endin
```

Illustration 4.5 MUSIC 1000 Code For Square Wave Generation

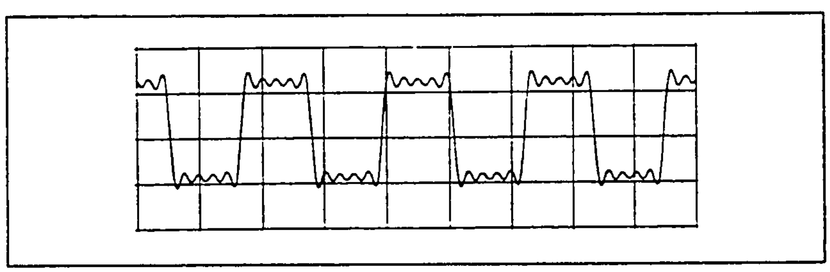
Line #1 in the above example has been changed for informative purposes only. Line #3 requires two alterations while line #4 has been expanded by the addition of numerical values. Line #8 has been changed to include the *FNCTN* name *SQUARE*.

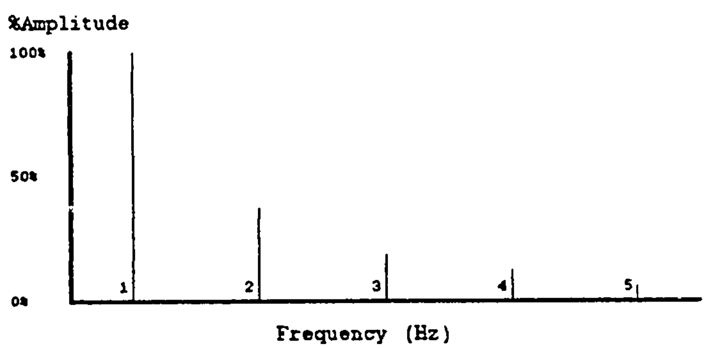
The *FNCTN* name in line 3 has been changed from *sine* to *square*. This line also contains the additional MUSIC 1000 word *NORMAL*. The statement *NORMAL* is used to proportionally scale the harmonic amplitudes of the function. The *NORMAL* statement also fills the table which creates the function in a proportional manner. This allows for the insertion of values based on percentages of amplitude as demonstrated in this example. The fundamental frequency has a value of 100, which means that the fundamental will have 100% amplitude. The third harmonic has the value 33 assigned to it, therefore, the third harmonic will possess 33% of the amplitude of the fundamental frequency.

Line 4 of the Orchestra contains the values necessary for the development of the waveform. The first value, *nArgs*, is equal to 9 and it represents the number of arguments or harmonics to be used. The following chart refers to the individual harmonics and relative amplitudes of each as generated by the argument list.

Relative Amplitude		*Frequency, Amplitude Percent*
Value #1 = 100	---	*Fundamental frequency, 220 Hz*
Value #2 = 0	---	*2nd Harmonic #, no amplitude*
Value #3 = 33	---	*3rd Harmonic #, 660 Hz, 33%*
Value #4 = 0	---	*4th Harmonic #, no amplitude*
Value #5 = 20	---	*5th Harmonic #, 1100 Hz, 20%*
Value #6 = 0	---	*6th Harmonic #, no amplitude*
Value #7 = 14	---	*7th Harmonic #, 1540 Hz, 14%*
Value #8 = 0	---	*8th Harmonic #, no amplitude*
Value #9 = 11	---	*9th Harmonic #, 1980 Hz, 11%*

The spectrum resulting from the information above approximates that of square wave. The Fourier process of additive synthesis allows for the development of different waveforms with very few alterations to the MUSIC 1000 code. The following examples will reinforce this process.





	Frequency	%Amplitude
1.	219.88hz	100%
2.	660.88hz	33.2%
3.	1100.66hz	20.6%
4.	1541.66hz	14.4%
5.	1981.44hz	11.9%

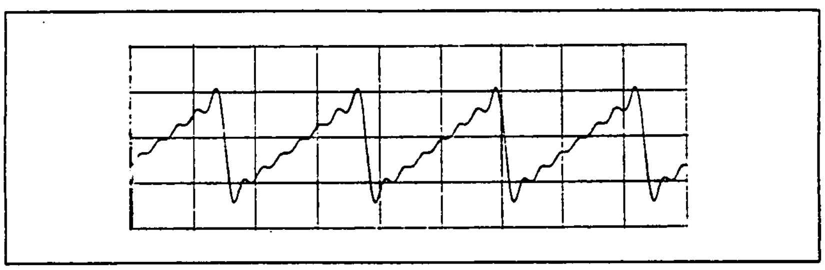
Illustration 4.6 Square Wave and FFT Results

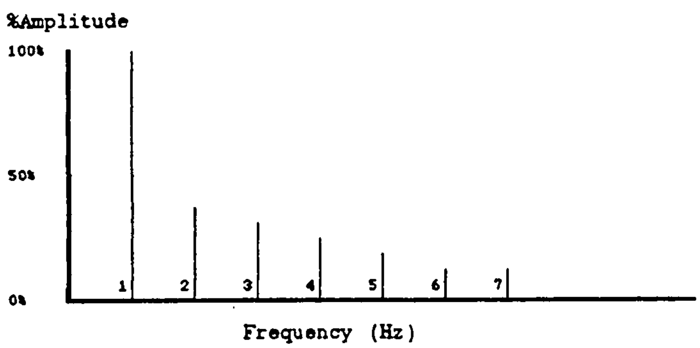
SAWTOOTH WAVE SYNTHESIS

The Orchestra

```
;       Sawtooth Wave Generation

  orch

fnctn      saw,1024,fourier,normal
           7,100,50,33,25,20,16,14

  instr      1
  local      &lt;env&gt;
  kline      env,p5,p3,p5

oscil      x6,#saw,p4,env

  xnice      x6,x6
  out        x6,3
  endin
```

Illustration 4.7 MUSIC 1000 Code For Sawtooth Wave Generation





	Frequency	%Amplitude
1.	219.88hz	100%
2.	439.77hz	36.7%
3.	660.88hz	33.2%
4.	880.77hz	27.9%
5.	1100.66hz	20.5%
6.	1320.55hz	14.3%
7.	1541.66hz	12.7%

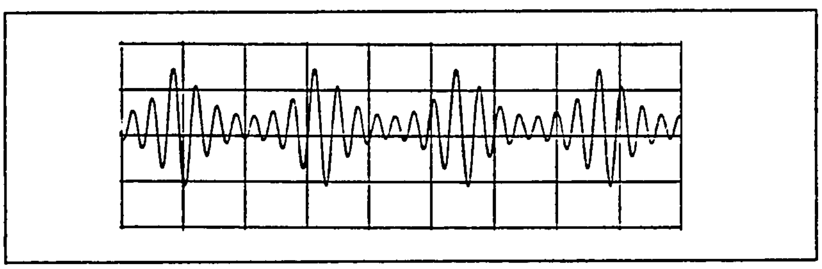
Illustration 4.8 Sawtooth Wave and FFT Results

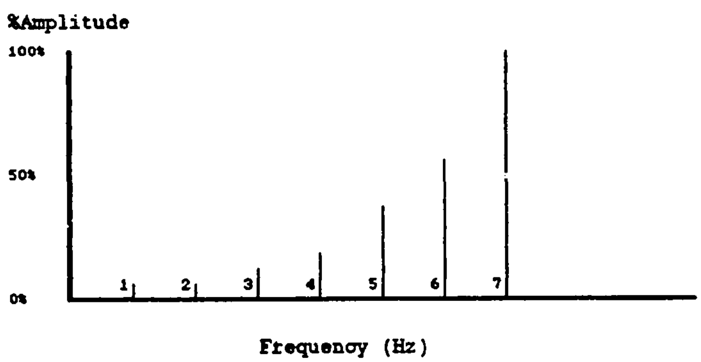
INVERTED WAVE SYNTHESIS

The Orchestra

```
;        Inverted Spectrum Waveform

           orch

fnctn      invert,1024,fourier,normal
           7,1,5,10,20,40,80,100

           instr    1
           local    <env>
           kline    env,p5,p3,p5

oscil      x6,#invert,p4,env

           xnice    x6,x6
           out      x6,3
           endin
```

Illustration 4.9 MUSIC 1000 Code For Inverted Spectrum Waveform





	Frequency	%Amplitude
1.	219.88hz	1.6%
2.	439.77hz	3.6%
3.	660.88hz	9.9%
4.	880.77hz	22.1%
5.	1100.66hz	40.7%
6.	1320.55hz	60.4%
7.	1541.66hz	100%

Illustration 4.10 Inverted Wave and FFT Results

Additive synthesis is a useful tool for creating various steady-state timbral configurations by altering the components that generate the waveform. MUSIC 1000 allows this to be accomplished on the DMX-1000 using the word *FOURIER*, or by building numerous, discrete oscillators.

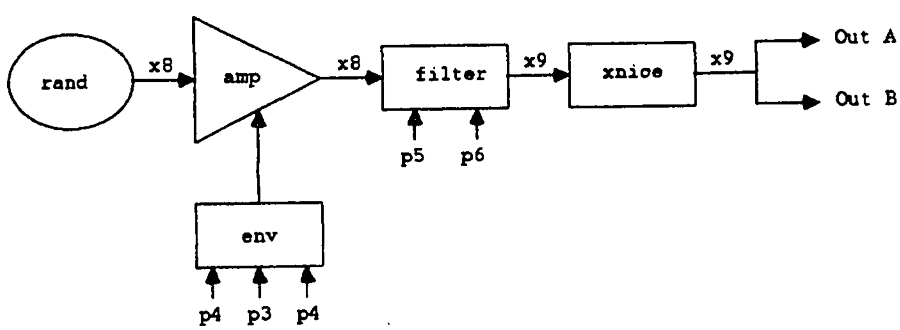
CHAPTER 5

SUBTRACTIVE SYNTHESIS

The models for subtractive synthesis are located on disks #2, #3, and #4 titled ***Subtractive Synthesis Examples***. There are three examples, one on each diskette: Band-pass Filtering; Hi-pass Filtering; and Lo-pass Filtering.

Subtractive synthesis involves the removal of unwanted frequency components from a complex source by the use of filters. Digital filters such as those found in the DMX-1000 are created by numerical calculations known as filter coefficients. The following subtractive synthesis example involves the use of a band-pass filter with variable center frequency and band width. The complex source to be filtered is a pseudo-random noise generator which allows filter changes to be easily recognized.

The first subtractive synthesis example produces a noise source and a band-pass filter. This example generates 5 separate 5 second events in which the center frequency of the filter changes with each event. A one second rest is used to separate the events. The band width of the filter is set at 20 Hz, while the center frequency assumes 100 Hz, 500 Hz, 1000 Hz, 3000 Hz, and 5000 Hz. Illustration 5.1 shows the flowchart patches for this example while illustration 5.2 shows the ideal (not actual) spectra of this example. Illustration 5.3 is the MUSIC 1000 code for the band-pass filtering demonstration.



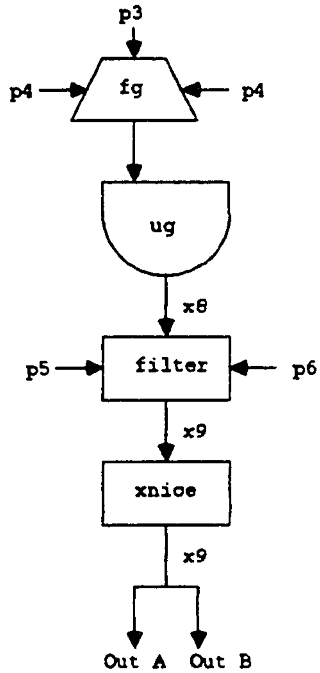


Illustration 5.1 Flowchart Diagrams for Subtractive Synthesis

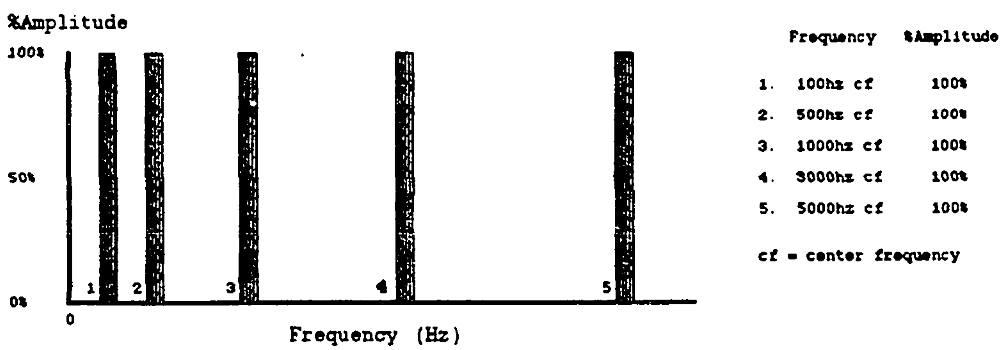


Illustration 5.2 Band-Pass Spectra (Ideal)

BAND-PASS FILTERING

The Orchestra

```
;       Band-pass Filtering Demonstration

orch

instr   1
local   <env,coefa,coefb,coefc>
kline   env,p4,p3,p4

rand    x8,env
ixrco   coefa,coefb,coefc,p5,p6
dreson  x9,x8,coefa,coefb,coefc

xnice   x9,x9
out     x9,3
endin
```

The Score

```
c     Band-pass Filtering Demonstration

c     p1     p2      p3     p4     p5     p6
c     ins#   start   time   ampl   cfrq   bwidth

x     1      .       .      dB     scps   scps
i     1      .       5      -20    100    20
r     .      .       1
i     .      .       5      .      500    .
r     .      .       1
i     .      .       5      .      1000   .
r     .      .       1
i     .      .       5      .      3000   .
r     .      .       1
i     .      .       5      .      5000   .
e
```

Illustration 5.3 MUSIC 1000 Code For Band-pass Filtering

The Orchestra section of the above example contains no *FNCTN* statement. This statement can be omitted because the noise generator is created by using the word RAND, and no *FNCTN* declaration is necessary. There are only a few important alterations existing between this subtractive synthesis example and the additive synthesis examples previously presented. These changes will be noted below.

<4> *local* *<env,coefa,coefb,coefc>*

The *LOCAL* statement is used to allocate storage for the variable *env*, which will perform the same task as it did in the example presented in the previous chapter. The new variable names are used to allocate storage for three filter coefficients which are required for the implementation of this particular filter; *coefa, coefb,* and *coefc.*

<6> *rand* *x9,env*

The MUSIC 1000 word *RAND* is used to generate pseudo-random noise. The format for *RAND* is:

RAND *XOUT,[xkvol]*

This format requires a register from which the noise generator may be output, and an optional amplitude value or envelope. The simple envelope *env* can be used and *x8* will serve as the *XOUT* register.

Note: Registers *x6* and *x7* can not be used with the *DRESON* filter component (described below) as these registers are reserved for filter implementation.

<7> *ixrco* *coefa,coefb,coefc,p5,p6*

IXRCO represents a mathematical process which is used to calculate filter coefficients for the band-pass filter. Parameter p5 is used to control the center frequency of the filter, while p6 is used for band width control. The numerical values to be used for these parameters are located in the Score section of this example. The bandwidth and center frequency of the *DRESON* filter are not *k-rate* units. Dynamic filtering is therefore not possible using *DRESON.*

<8> *dreson* *x8,x9,coefa,coefb,coefc*

DRESON is a double precision (32 bit), second-order recursive audio-rate band-pass filter. (A 16 bit filter, *RESON,* is no longer a part of the current version of MUSIC 1000). *DRESON* uses the filter coefficients which have been calculated by the word *IXRCO,* and generates a digital filter. Register *x9* represents an output register which is where the filtered signal will be placed. Register *x8* represents the input signal to the filter. The noise generated by the word *RAND* in line #6 is used as the input signal. The filter coefficients complete the *DRESON* format.

The Score

The Score used to demonstrate subtractive synthesis is very similar to the Score used for the additive synthesis examples. The first example generates a band-pass filter with a center frequency and band width controllable through the Score. A few modifications will allow the Score to generate either hi-pass or lo-pass filtering. Illustration 5.5 shows the necessary code for hi-pass filtering, and illustration 5.6 shows the ideal results for the filter. A listing of the code for band-pass filtering will first be presented for the purpose of comparison. Note the changes occurring at parameters p5 (cfrq), and p6 (bwidth).

```
<1>    c    Band-pass Filtering Demonstration

<2>    c    p1     p2     p3     p4     p5     p6
<3>    c    ins#   start  time   ampl   cfrq   bwidth

<4>    x    1      .      .      dB     scps   scps
<5>    i    1      .      5      -20    100    20
<6>    r    .      .      1
<7>    i    .      .      5      .      500    .
<8>    r    .      .      1
<9>    i    .      .      5      .      1000   .
<10>   r    .      .      1
<11>   i    .      .      5      .      3000   .
<12>   r    .      .      1
<13>   i    .      .      5      .      5000   .
<14>   e
```

Note: The dot character used in lines 5 through 13 allows the value from the previous line and same parameter number to be carried over to the next line. This eliminates the need to type the same value on each line of code.

HI-PASS FILTERING

The Orchestra

```
;        Hi-pass Filtering Demonstration

orch     fast

instr    1
local    <env,coefa,coefb,coefc>
kline    env,p4,p3,p4

rand     x8,env
ixrco    coefa,coefb,coefc,p5,p6
dreson   x9,x8,coefa,coefb,coefc

xnice    x9,x9
out      x9,3
endin
```

The Score

```
c    Hi-pass Filtering Demonstration

c    p1     p2      p3     p4     p5      p6
c    ins#   start   time   ampl   cfrq    bwidth

x    1      .       .      dB     fcps    fcps
i    1      .       5      -20    10500   19000
r    .      .       1
i    .      .       5      .      12500   15000
r    .      .       1
i    .      .       5      .      15000   10000
r    .      .       1
i    .      .       5      .      17500    5000
e
```

Illustration 5.4 MUSIC 1000 Code For Hi-pass Filtering

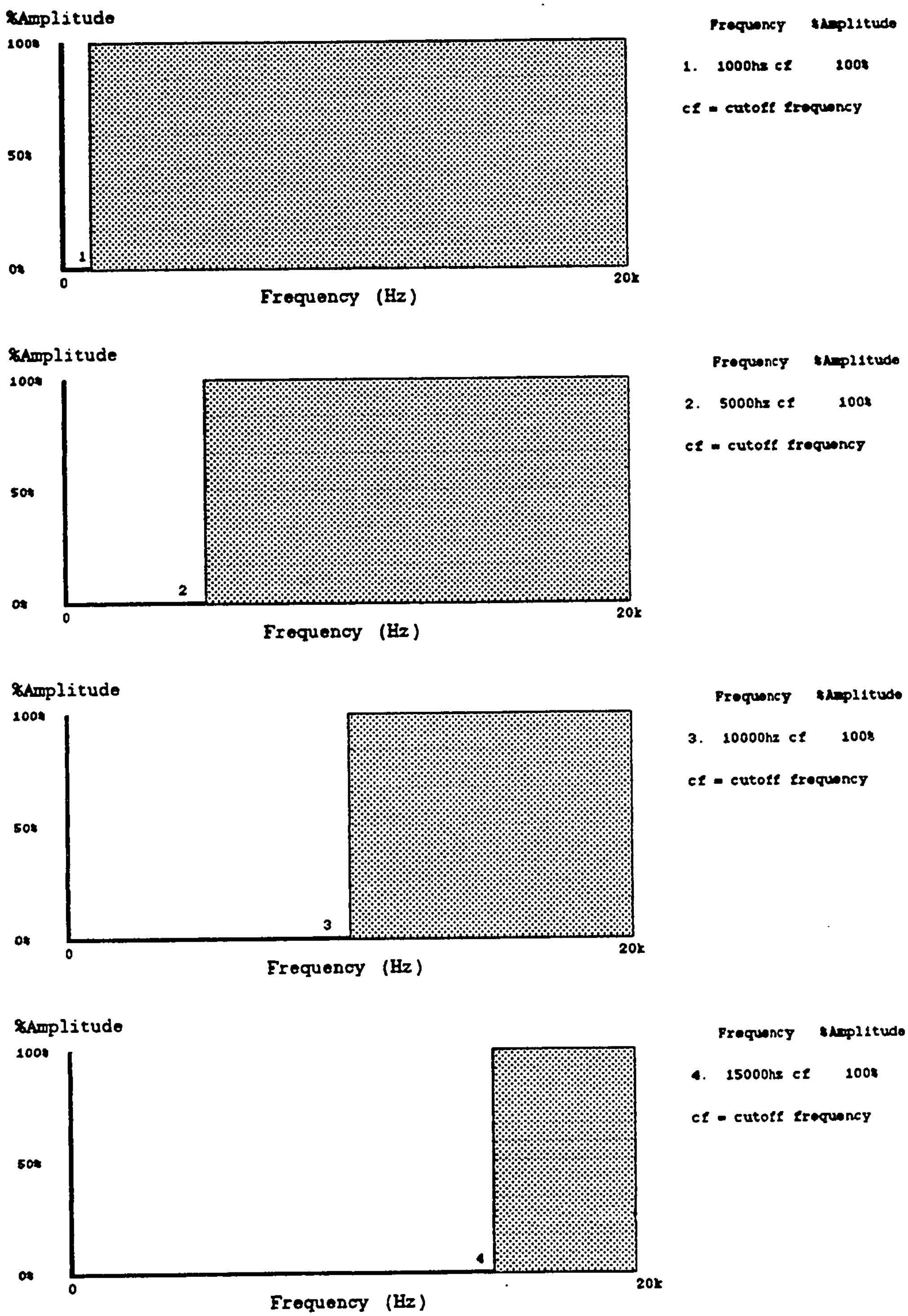


Illustration 5.5 Hi-Pass Spectra (Ideal)

In order to generate Hi-pass filtering, the band-width is used to determine the amount of signal which is to be passed. For example, the first of the four events presented above passes frequencies between 1 kHz and 20 kHz. This is achieved by specifying a band-width of 19 kHz, i.e., 20k - 1k = 19k. The center frequency must then be calculated in order to pass all frequencies from 1kHz to 20 kHz. The value is 10.5 kHz, and it allows one-half of the band-width (9.5 kHz) to pass both above it and below it. Mathematically, the filter passes the following:

10.5 kHz - 9.5 kHz = 1 kHz

10.5 kHz + 9.5 kHz = 20 kHz

Note: In order to specify frequencies above 10kHz, the scaling value *fcps* must be used, and, the Orchestra must contain the word *fast* following *orch*. (See above).

The remaining three events were calculated in the same manner and the results of the hi-pass example are as follows:

1. *1 kHz to 20 kHz*
2. *5 kHz to 20 kHz*
3. *10 kHz to 20 kHz*
4. *15 kHz to 20 kHz*

Lo-pass filtering is also possible with the band-pass filter *DRESON* by reversing the process described in the previous example. In this case, the frequencies to be passed will range from 0 Hz to the specified cutoff frequency. The MUSIC 1000 code necessary for lo-pass filtering is listed in illustration 5.6 and illustration 5.7 is a diagram of the ideal results from the filter.

LO-PASS FILTERING

The Orchestra

```
;        Lo-pass Filtering Demonstration

orch     fast

instr    1
local    <env,coefa,coefb,coefc>
kline    env,p4,p3,p4

rand     x8,env
ixrco    coefa,coefb,coefc,p5,p6
dreson   x9,x8,coefa,coefb,coefc

xnice    x9,x9
out      x9,3
endin
```

The Score

```
c     Lo-pass Filtering Demonstration

c     p1     p2      p3     p4     p5     p6
c     ins#   start   time   ampl   cfrq   bwidth

x     1      .       .      dB     fcps   fcps
i     1      .       5      -20    500    1000
r     .      .       1
i     .      .       5      .      2500   5000
r     .      .       1
i     .      .       5      .      5000   10000
r     .      .       1
i     .      .       5      .      7500   15000
e
```

Illustration 5.6 MUSIC 1000 Code For Lo-pass Filtering

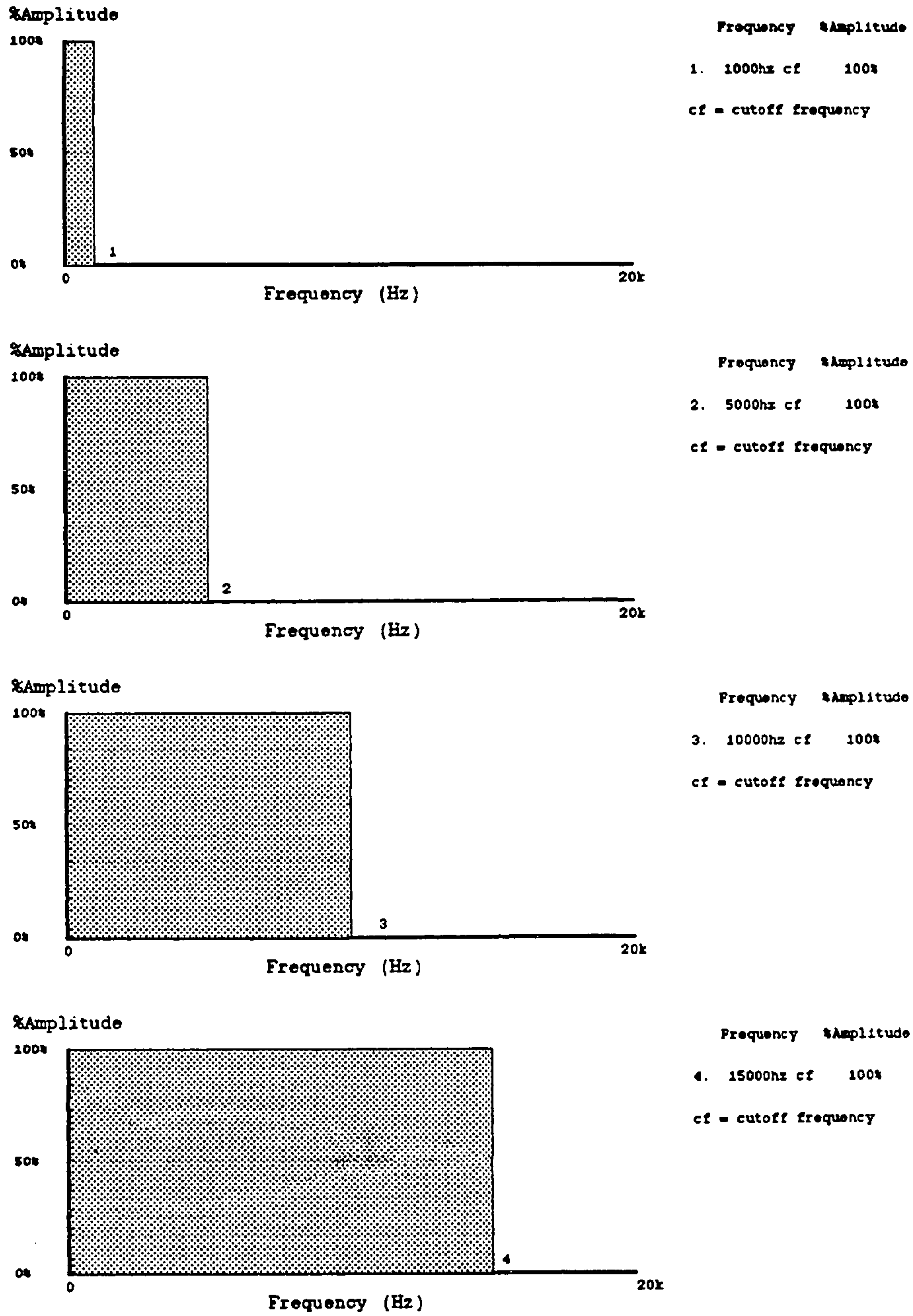


Illustration 5.7 Lo-Pass Spectra (Ideal)

The results of the above example are the following:

1. *0 Hz to 1 kHz*
2. *0 Hz to 5 kHz*
3. *0 Hz to 10 kHz*
4. *0 Hz to 15 kHz*

CHAPTER 6

MODULATION SYNTHESIS

The demonstrations of frequency modulation are located on disk #5, titled ***Frequency Modulation Examples***. The examples of amplitude modulation are located on disk #6, titled ***Amplitude Modulation Examples***.

Modulation synthesis is achieved by changing or modulating one signal with another signal. Sub-audio rate modulation can be used to create effects such as vibrato and tremolo. Audio rate modulation can be used to build timbral spectrums which may be easily altered through dynamic processes. Both frequency modulation (FM) and amplitude modulation (AM) are possible on the DMX-1000, and examples of each will be included in this section.

FREQUENCY MODULATION

Frequency modulation synthesis was extensively explored by John Chowning (see bibliography) at Stanford's CCRMA. Chowning proved that frequency modulation can be used to generate realistic instrument timbres by varying the modulation index over time. The DMX-1000 is capable of similar complex modulation configurations. MUSIC 1000 includes the word *FRQMOD* which is used for linear frequency modulation. *FRQMOD* is capable of producing either audio-rate synthesis or sub-audio modulation. Illustration 6.1 shows the flowchart patches for an FM voice. The FFT results and waveforms are shown in illustration 6.2. Illustration 6.3 is an example of the MUSIC 1000 code which uses the word *FRQMOD* to generate a signal with a modulation frequency that is changed via software every 5 seconds.

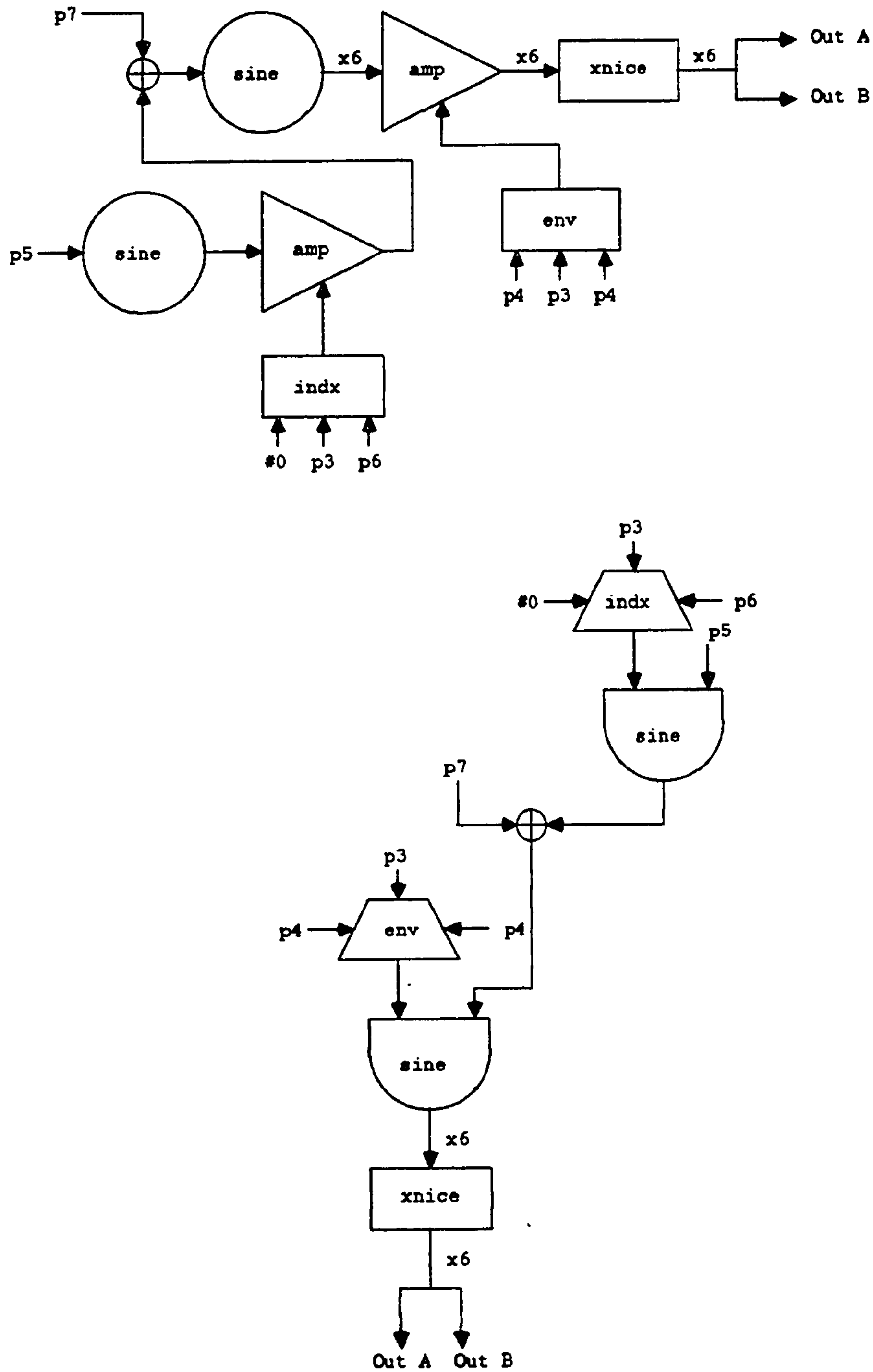
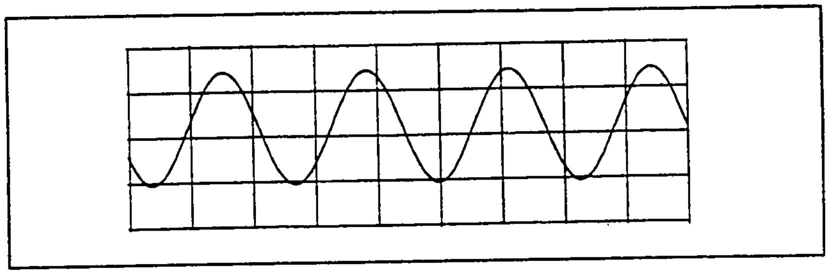
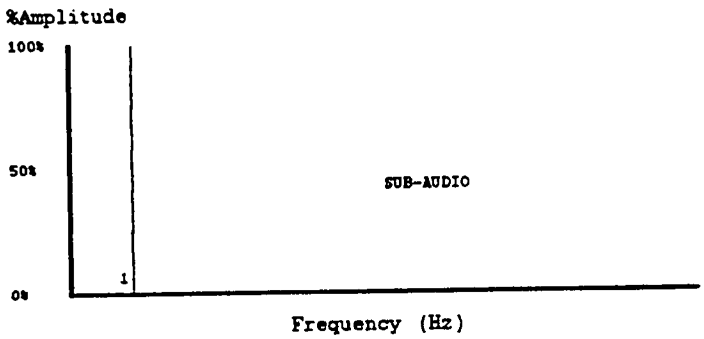
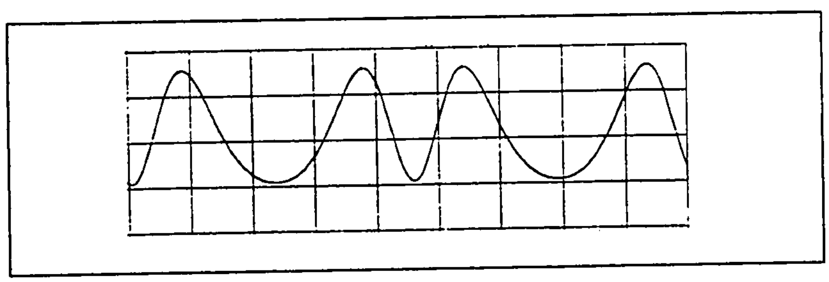


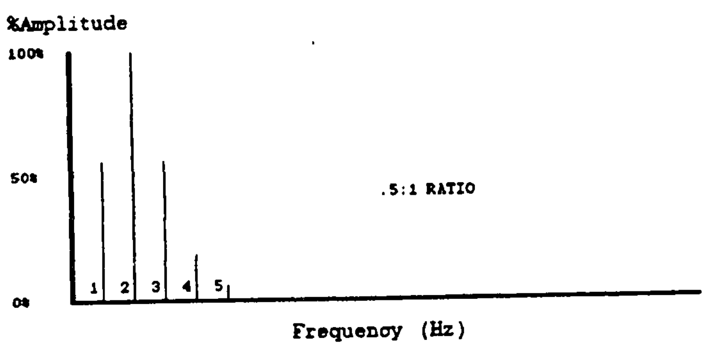
Illustration 6.1 Flowchart Diagrams for Frequency Modulation



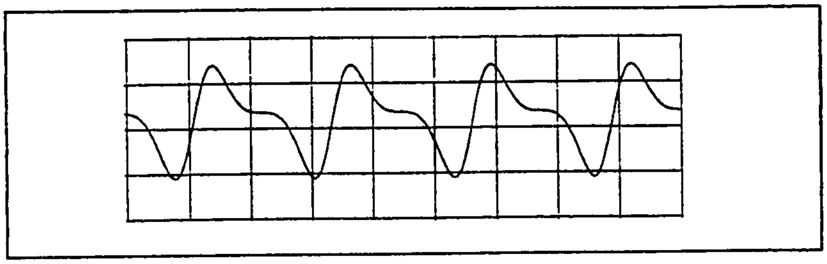


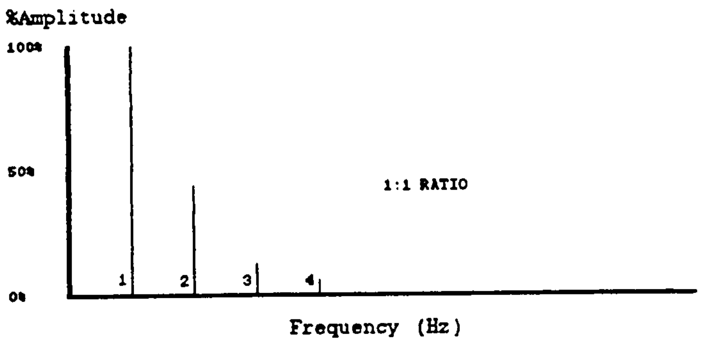
	Frequency	%Amplitude
1.	219.88hz	100%



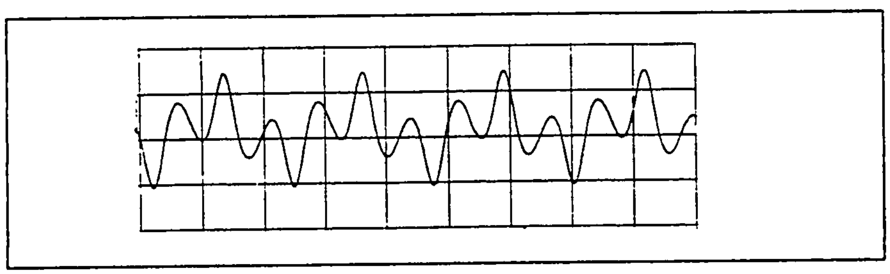


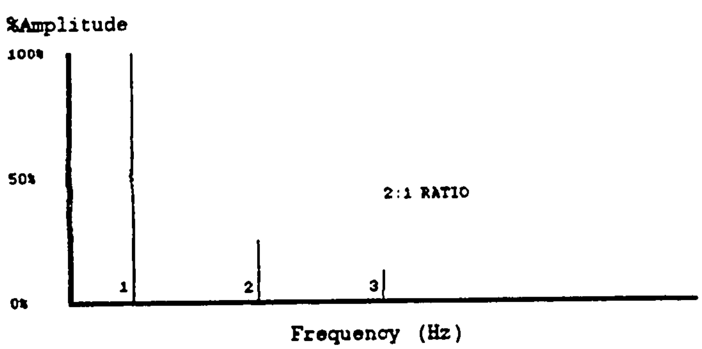
	Frequency	%Amplitude
1.	109.94hz	56.3%
2.	219.88hz	100%
3.	329.83hz	58.1%
4.	439.77hz	15.3%
5.	549.72hz	2.4%



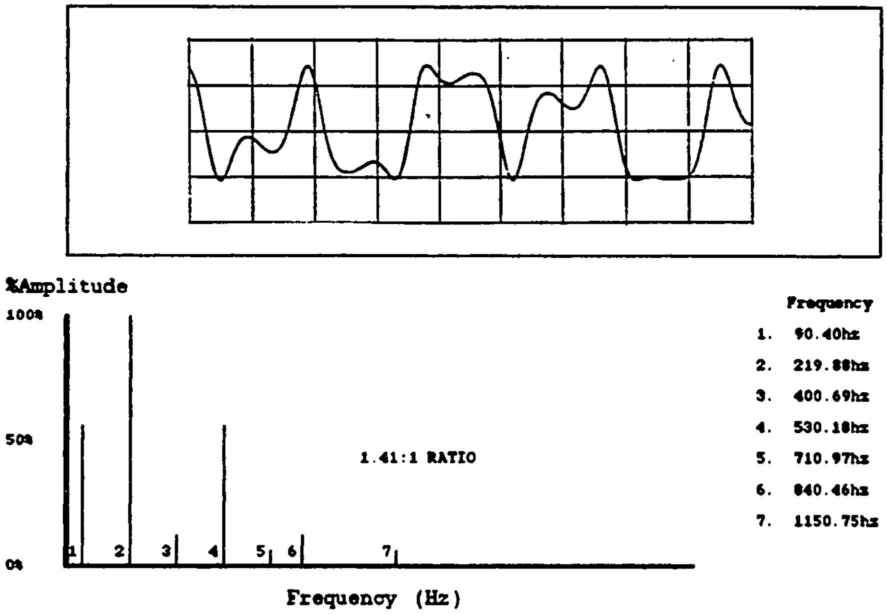


	Frequency	%Amplitude
1.	219.88hz	100%
2.	439.77hz	41.8%
3.	660.88hz	11.1%
4.	880.77hz	2.4%





	Frequency	%Amplitude
1.	219.88hz	100%
2.	660.88hz	25.7%
3.	1100.66hz	12.2%



	Frequency	%Amplitude
1.	90.40hz	57.2%
2.	219.88hz	100%
3.	400.69hz	15.0%
4.	530.18hz	57.9%
5.	710.97hz	2.3%
6.	840.46hz	15.2%
7.	1150.75hz	2.6%

Illustration 6.2 FFT Results And Waveforms

The Orchestra

```
;       Frequency Modulation Demonstration

orch

fnctn   sine,1024,fourier
        1,32767

instr   1
local   <env,indx>
kline   env,p4,p3,p4
kline   indx,#0,p3,p6

frqmod  x6,#sine,p5,indx,p7,env

xnice   x6,x6
out     x6,3
endin
```

The Score

```
c   Frequency Modulation Demonstration

c   p1    p2     p3    p4    p5    p6    p7
c   ins#  start  time  ampl  mfrq  indx  cfrq

x   1     .      .     dB    scps  scps  scps
i   1     .      5     -20   5     5     220
r   .     .      1
i   .     .      5     .     110   110   .
r   .     .      1
i   .     .      5     .     220   220   .
r   .     .      1
i   .     .      5     .     440   440   .
r   .     .      1
i   .     .      5     .     310   310   .
e
```

Illustration 6.3 MUSIC 1000 Code for Frequency Modulation

Analysis of the FM instrument patch utilizes many MUSIC 1000 words which have been used in the previous examples. This FM instrument uses a sine wave for both the modulating signal and the carrier signal. Five different modulating frequencies are used in order to illustrate both sub-audio and audio-rate modulation.

The Orchestra

```
<1>          ;       Frequency Modulation Demonstration

<2>          orch
```

The Orchestra begins with the word *ORCH* located at line #2.

```
<3>          fnctn     sine,1024,fourier
<4>                    1,32767
```

Lines 3 and 4 begin the function declaration. This FM example uses a sine wave for both the modulating signal and the carrier signal. The function statement therefore will be identical to the sine wave generated in the additive synthesis example.

```
<5>          instr     1
```

The instrument number is assigned in line 5 of the Orchestra. Like the previous examples, the instrument reference number 1 is used.

```
<6>          local     <env,indx>
```

The *LOCAL* statement is used to allocate storage space for two variables, *env*, and *indx*. These variables will be used to create envelopes for amplitude and modulation index.

```
<7>          kline     env,p4,p3,p4
```

The amplitude envelope *env* is generated in line 7 and it uses parameters p3 and p4 from the Score. Parameter p3 determines the duration of the event while parameter p4, specified in dB, determines the output of the instrument.

```
<8>          kline     indx,#0,p3,p6
```

KLINE is used to generate a linear slope to be applied to the modulation index. This will create a modulation index which gradually increases over the five second duration of each event. The #0 represents a constant value of zero. All constants must be preceded by the # symbol. This index envelope begins at zero and increases over the time specified by p3 to the value located at p6 in the Score. Parameter p6 is specified in *scps*, or cycles per second.

<9> ***FRQMOD*** ***x6,#sine,p5,indx,p7,env***

FRQMOD is a MUSIC 1000 word which allows for the generation of linear frequency modulation. *FRQMOD* requires that a modulation frequency (mfrq), an index (indx), a carrier frequency (cfrq), and an amplitude value or envelope be specified. The proper syntax for the word *FRQMOD* is as follows:

FRQMOD ***XOUT,IFUNC,XKMSI,XKNDX,XKCSI,XKAMP***

This frequency modulation component creates an FM oscillator pair. Modulation occurs as the output of the modulation oscillator is added to the frequency input of the carrier oscillator. Both oscillators function in a way which is similar to the *OSCIL* unit. The following chart defines the format for *FRQMOD*:

XOUT = The register or location of the signal
IFUNC = The function to be used for synthesis
XKMSI = The modulation frequency
XKNDX = The modulation index
XKCSI = The carrier frequency
XKAMP = The amplitude of the signal

An equivalent version of *FRQMOD* could be constructed in MUSIC 1000 code in the following fashion:

```
oscil     x6,#sine,p5,indx
xadd      x6,p4
oscil     x7,#sine,x6,env
```

The results from the above would be identical to the use of *FRQMOD*. The output of the first oscillator (the modulator), which is located at x6, is added to the carrier frequency parameter, p4. This signal is then used to control the frequency of the carrier oscillator.

The Score

```
<1> c     Frequency Modulation Demonstration

<2> c     p1     p2      p3     p4     p5     p6     p7
<3> c     ins#   start   time   ampl   mfrq   indx   cfrq
```

Lines 1 through 3 of the Score are comment statements which clearly identify the various parameters and their applications for this specific example. The words below the parameter number refer to its application as illustrated here:

p1 = ins# (instrument number)
p2 = start (starting time in beats)
p3 = time (overall time or duration of an event)
p4 = ampl (the amplitude of an event)
p5 = mfrq (the modulation frequency)
p6 = indx (the modulation index)
p7 = cfrq (the carrier frequency)

```
<opcode  p1    p2    p3    p4    p5     p6     p7>
<4> x    1     .     .     dB    scps   scps   scps
```

Line 4 is a scaling statement which provides multipliers for various operations. The instrument number is identified first by the value 1. The two dots which follow are characters which fill the locations for p2 and p3 since the *x* statement ignores these parameters. The scaling value *dB* is used for output amplitude as in the previous examples allowing for the instrument's loudness to be specified in decibels. The *scps* scaling factor is used for the next three parameters which include the modulation frequency, the carrier frequency, and the modulation index. Specification of the modulation index in Hz is appropriate as various indices can be easily achieved.

The audio result of this example will consist of five FM events with different modulating frequencies for each event. The five *i* or event lines provide the numerical values necessary for this process. A rest of one second separates the events as determined by p3 in each *r* statement. The modulation index is specified in hertz and its value equals the modulation frequency, therefore, producing an index of 1. The Score, once again, is the following:

```
<4>  x   1    .    .    dB    scps   scps   scps
<5>  i   1    .    5   -20      5      5     220
<6>  r   .    .    1
<7>  i   .    .    5    .     110    110      .
<8>  r   .    .    1
<9>  i   .    .    5    .     220    220      .
<10> r   .    .    1
<11> i   .    .    5    .     440    440      .
<12> r   .    .    1
<13> i   .    .    5    .     310    310      .
<14> e
```

The audio result of this example can be listed as follows:

cfrq = 220hz	*mfrq = 5hz*	=	*sub-audio*
cfrq = 220hz	*mfrq = 110hz*	=	*.5 to 1 ratio*
cfrq = 220hz	*mfrq = 220hz*	=	*1 to 1 ratio*
cfrq = 220hz	*mfrq = 440hz*	=	*2 to 1 ratio*
cfrq = 220hz	*mfrq = 310hz*	=	*1.41 to 1 ratio*

AMPLITUDE MODULATION

Amplitude modulation is achieved by altering the instantaneous amplitude of a carrier signal with a modulating signal. The result can be used to generate sidebands as well as create effects such as tremolo (using a sub-audio modulating wave), and ring or balanced modulation. AM is similar to frequency modulation except that AM can only be used to generate one set of sidebands, and not multiple sidebands as with FM.

MUSIC 1000 does not include a word which creates AM. This voice must be configured according to the elements necessary for amplitude modulation. Illustration 6.4 shows the AM patch with flowchart diagrams. Illustration 6.5 shows the FFT results and waveforms. This patch is shown in MUSIC 1000 code in illustration 6.6. The example generates a tone with a modulation frequency that changes every five seconds.

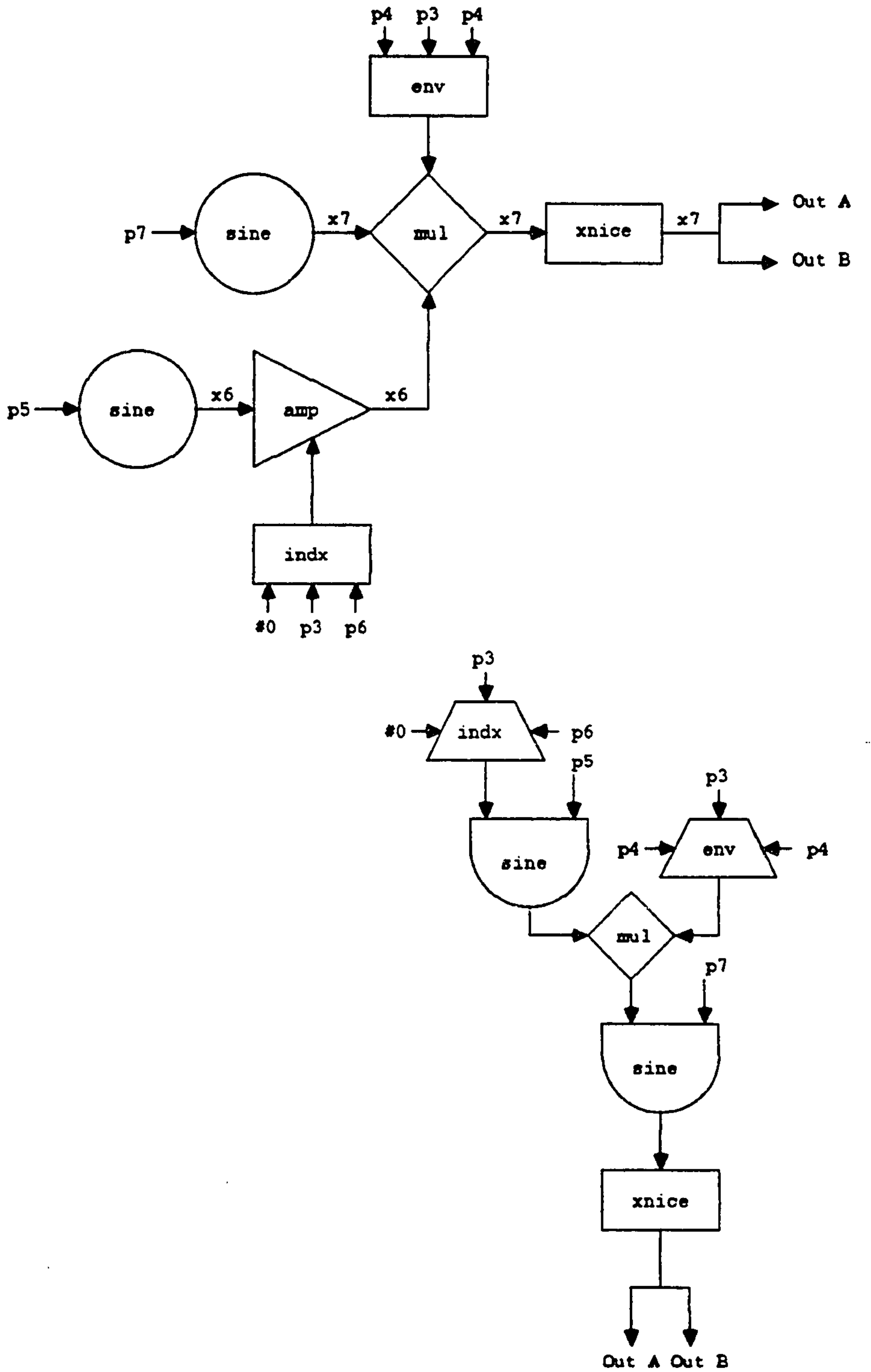
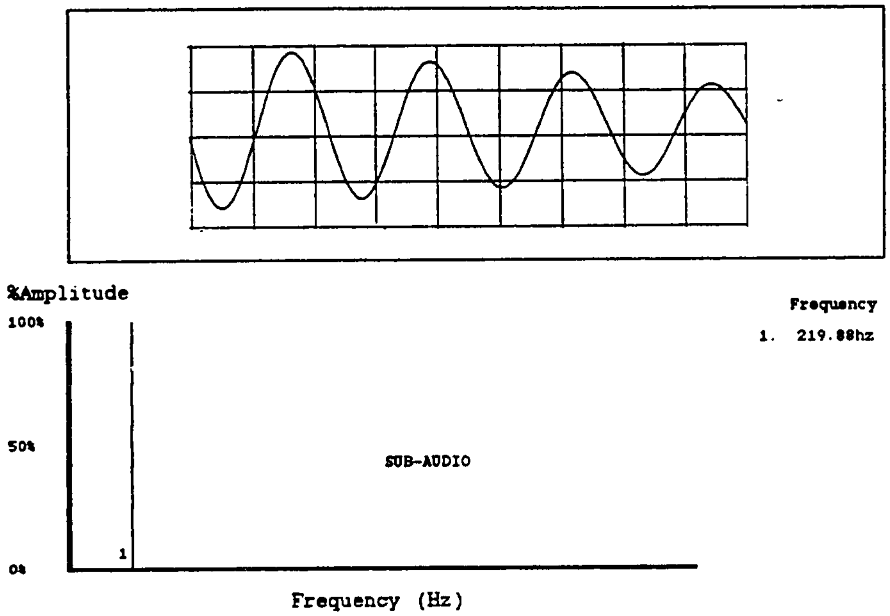
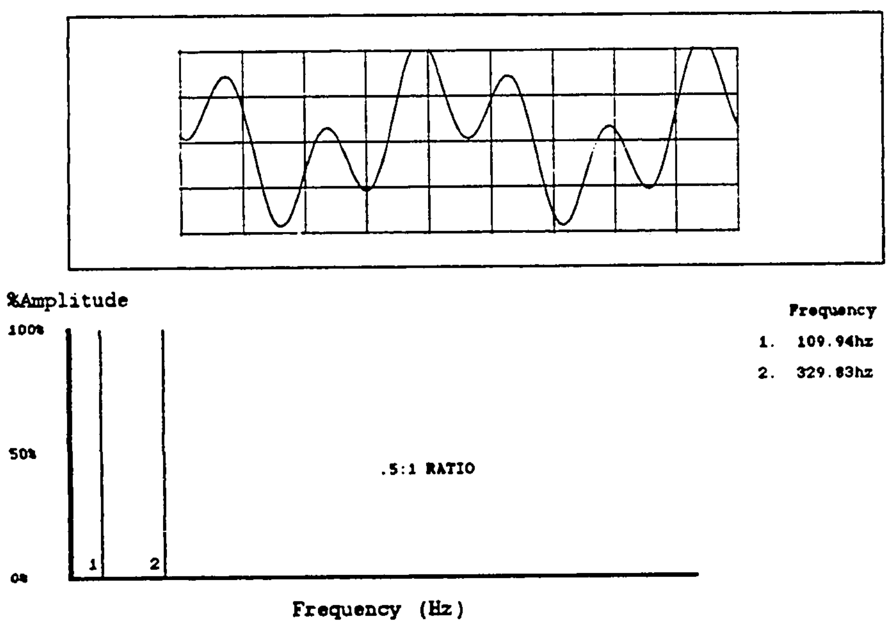


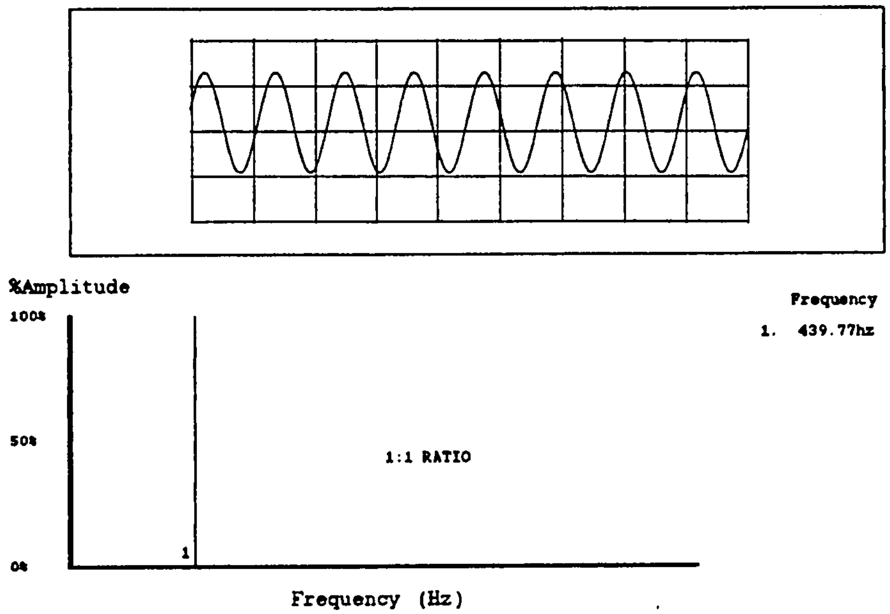
Illustration 6.4 Flowchart Diagrams for Amplitude Modulation



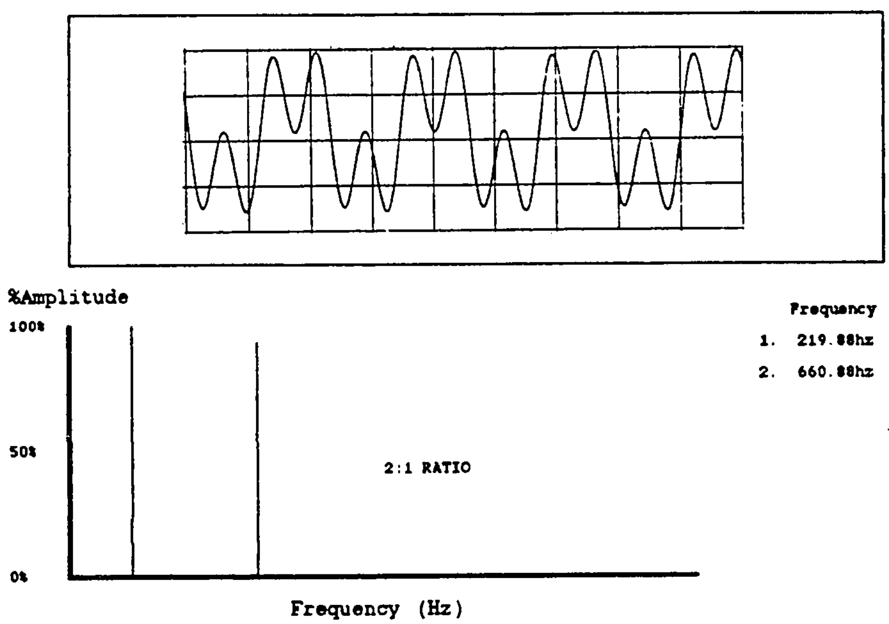
	Frequency	%Amplitude
1.	219.88hz	100%



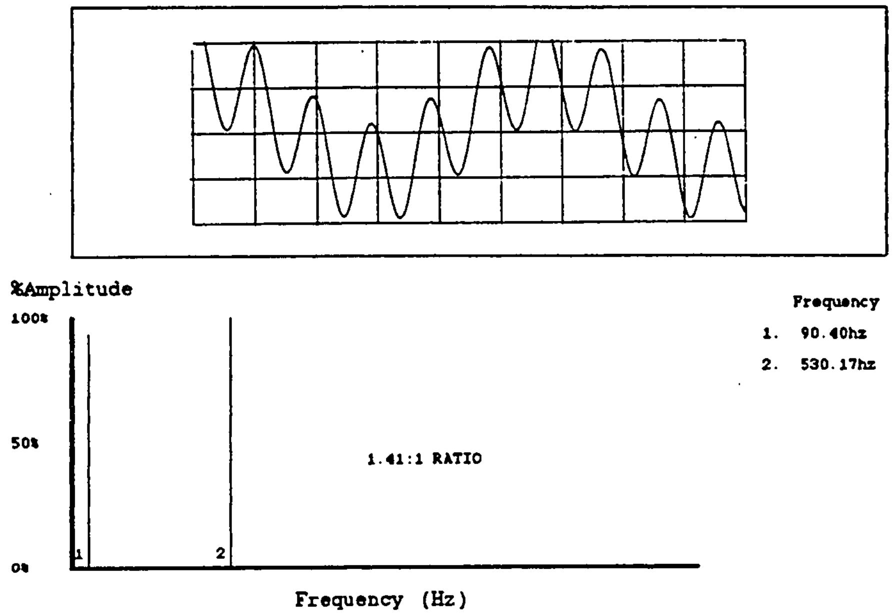
	Frequency	%Amplitude
1.	109.94hz	99.7%
2.	329.83hz	100%



	Frequency	%Amplitude
1.	439.77hz	100%



	Frequency	%Amplitude
1.	219.88hz	100%
2.	660.88hz	99.6%



	Frequency	%Amplitude
1.	90.40hz	98.77%
2.	530.17hz	100%

Illustration 6.5 FFT Results And Waveforms

The Orchestra

```
;       Amplitude Modulation Demonstration

orch

fnctn   sine,1024,fourier
        1,32767

instr   1
local   <env,indx>
kline   env,p4,p3,p4
kline   indx,#0,p3,p6

oscil   x6,#sine,p5,indx
xmul    x6,env
oscil   x7,#sine,p7,x6

xnice   x7,x7
out     x7,3
endin
```

The Score

```
c       Amplitude Modulation Demonstration

c       p1      p2      p3      p4      p5      p6      p7
c       ins#    start   time    ampl    mfrq    indx    cfrq

x       1       .       .       dB      scps    dB      scps
i       1       .       10      -5         5    -10      220
r       .       .       1
i       .       .       10      .        110    .       .
r       .       .       1
i       .       .       10      .        220    .       .
r       .       .       1
i       .       .       10      .        440    .       .
r       .       .       1
i       .       .       10      .        310    .       .
e
```

Illustration 6.6 MUSIC 1000 Code For Amplitude Modulation

Since MUSIC 1000 provides no word for AM (such as frqmod for FM), the AM voice must be physically patched through software. This is the reason for the word *XMUL*, which allows for the multiplication of one signal by another signal. The AM voice is achieved through this process.

The Orchestra section which is used for this AM example should, at this point, be relatively familiar as it utilizes much of the same material required for the previous examples. There are only three lines, 10, 11, and 12, which may be somewhat unclear. These three lines are used to create AM. The modulation process begins at line 10.

```
<9>          oscil     x6,#sine,p5,indx
```

Line 9 is used to create a sine wave oscillator with its signal placed at register *x6*. Its amplitude is determined by the envelope *indx*.

```
<10>         xmul      x6,env
```

Line 10 of the Orchestra is a multiplier which takes the signal located at *x6* and multiplies it by the function *env*. This process essentially creates an oscillator with a simple envelope.

```
<11>         oscil     x7,#sine,p7,x6
```

This oscillator uses the signal at *x6* as an amplitude input which creates the amplitude modulation. It is important to note that the multiplication process which has been executed in line 10 will produce an output only if the values to be multiplied are greater than zero. If either value equals zero, no output will be the result. This is the reason for a gradual increase in amplitude as well as modulation as the index grows larger.

The Score section of this example is similar to the previous Score examples. Five events are created through the *i*, or event statements located in lines 4 through 8 of the Score. Each event lasts ten seconds as the modulation frequency (p5) varies with each event. A one second rest is also applied at the end of each event.

It has been shown that modulation synthesis is a useful tool for the generation of various timbral configurations with an output which can be dynamically controlled over time. Complex AM and FM algorithms are possible using MUSIC 1000 and the DMX-1000; however, the examples presented above are not intended to be complex in nature. The following chapter illustrates some aspects of the musical capabilities of MUSIC 1000 by demonstrating a musical example to be played by the DMX-1000.

CHAPTER 7

NON-LINEAR WAVESHAPING, INPUT, AND REVERB

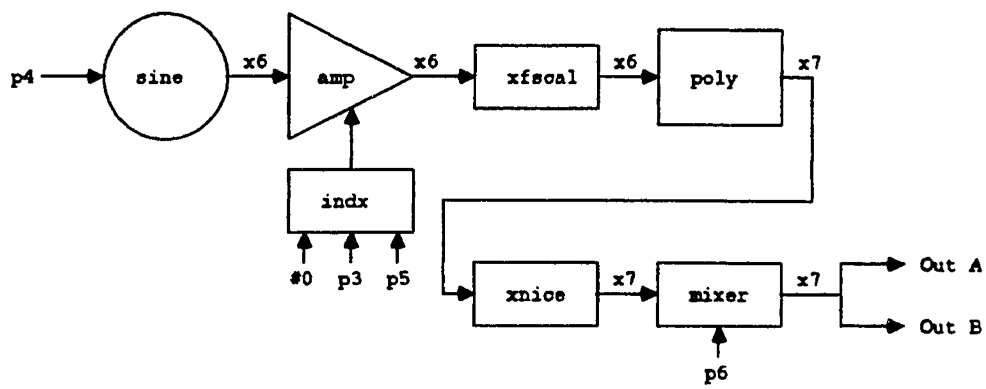
The demonstrations of non-linear waveshaping are located on disk #7, titled ***Non-Linear Waveshaping Examples***. The input and reverb demonstration is located on disk #8, titled ***Input and Reverb Example***.

NON-LINEAR WAVESHAPING

Non-linear waveshaping is a method of distorting a signal in order to alter the timbral quality of the sound. This technique is based upon a mathematical formula from which a table of weighted sums is generated by Chebycheff polynomials. These polynomials are created in MUSIC 1000 by using the ***Cheby*** function type. The spectrum resulting from this distortion process depends upon the number and amplitude of the various components entered into the Cheby table.

The process for this demonstration begins with the generation of a sine wave. The amplitude of the sine wave is used to determine the amount of distortion that takes place. As the amplitude of the sine wave increases, the spectrum changes as a result. MUSIC 1000 requires that the output of the sine wave is first placed into a scaling function, ***xfscal***, in order to scale the oscillator signal for the appropriate table size. This signal is then placed into the Cheby table where transformation takes place, and finally routed to an output location. The MUSIC 1000 word ***mixer*** is included to allow for final attenuation of the signal.

The following four examples are different only in the Orchestra section where the Cheby function table is created. Line #4 of each Orchestra is used to specify the content of the Cheby table. The first example begins with a sine wave at a frequency of 220 Hz which is gradually transformed into its third harmonic (approximately 660 Hz). The third harmonic is selected in the argument list on line #4 of the function named ***poly***. The value 100 is placed in the third location of the list and the first and second arguments are assigned the value 0. This allows 100% of the third harmonic to be generated while the first and second remain at zero amplitude. In a similar manner, the second example generates the fifth harmonic, the next generates the seventh harmonic, and the final example combines all of the above. The flowchart patches for non-linear waveshaping are shown in illustration 7.1 while illustration 7.2 shows the MUSIC 1000 code necessary for the process. Illustration 7.3 shows the FFT results and spectra of the above.



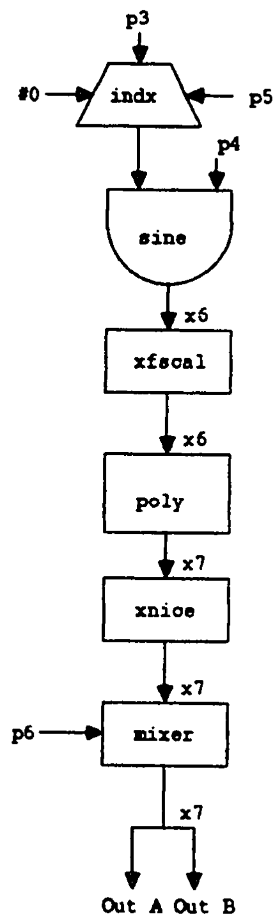


Illustration 7.1 Flowchart Diagrams for Non-Linear Waveshaping

NON-LINEAR WAVESHAPING

The Score

```
c     Non-Linear Waveshaping Demonstration

c     p1    p2     p3    p4    p5     p6
c     ins#  start  time  freq  indx   ampl

x     1     .      .     scps  1      dB

i     1     .      10    220   16000  -20
r     .     .      1
i     .     .      10    .     32767  .
e
```

The Orchestra

```
;        Non-Linear Waveshaping Demonstration

orch

fnctn    poly,2048,cheby,normal
         3,0,0,100

fnctn    sine,1024,fourier
         1,32767

instr    1
local    <indx>
kline    indx,#0,p3,p5
oscili   x6,#sine,p4,indx
xfscal   x6,x6,#poly,bipol
table    x7,#poly,x6
xnice    x7,x7
mixer    x7,<<x7,p6>>
out      x7,3
endin
```

Function for Example 2

```
<3>      fnctn      poly,2048,cheby,normal
<4>                 5,0,0,0,0,100
```

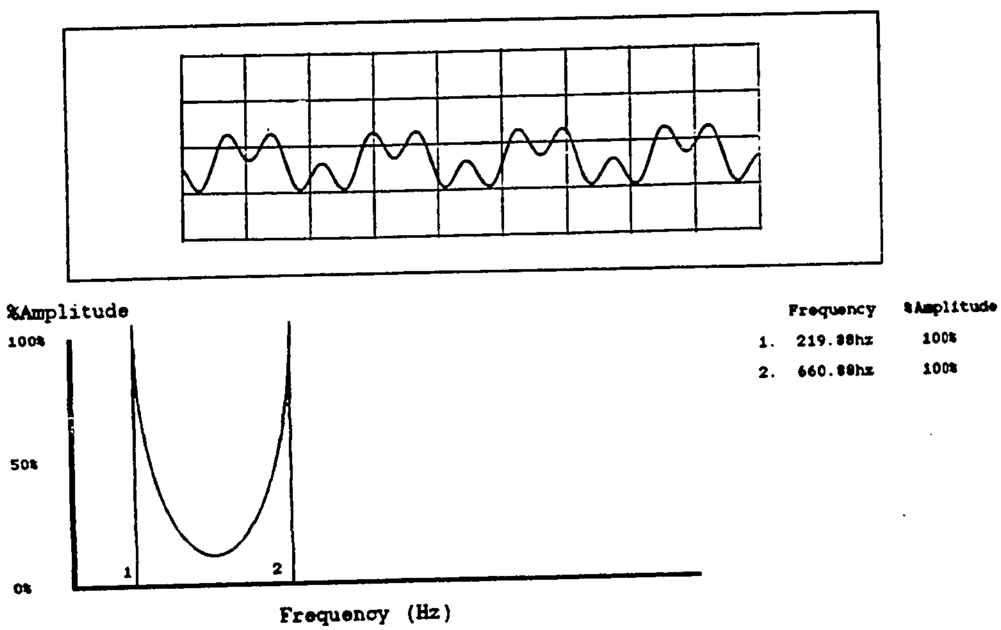
Function for Example 3

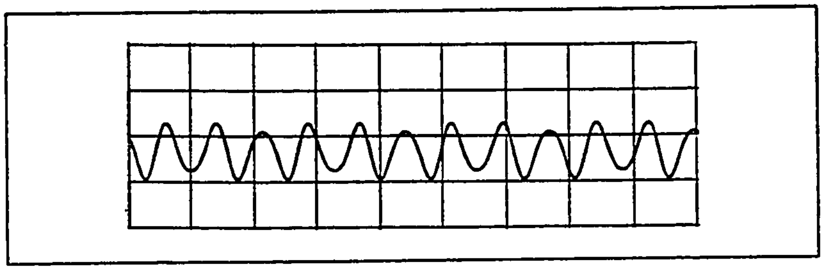
```
<3>      fnctn      poly,2048,cheby,normal
<4>                 7,0,0,0,0,0,0,100
```

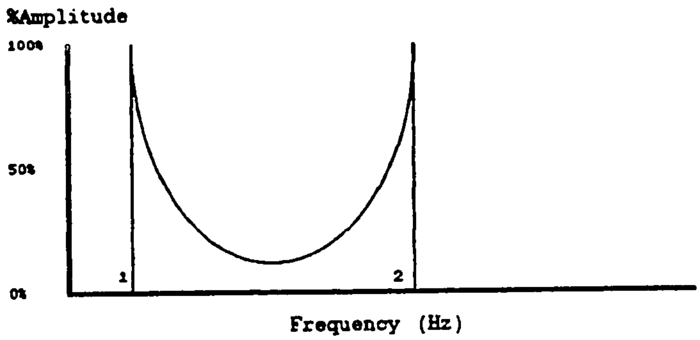
Function for Example 4

```
<3>      fnctn      poly,2048,cheby,normal
<4>                 7,0,0,100,0,100,0,100
```

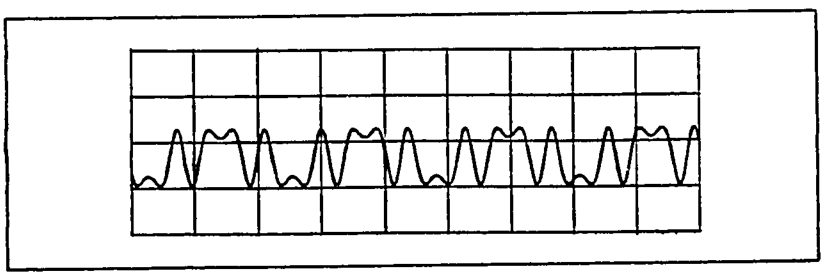
Illustration 7.2 MUSIC 1000 Code for Non-Linear Waveshaping

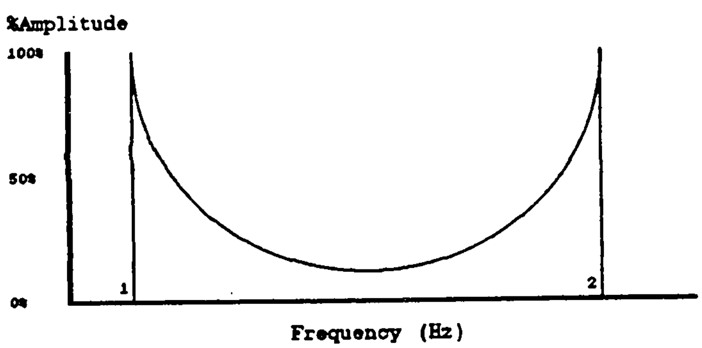




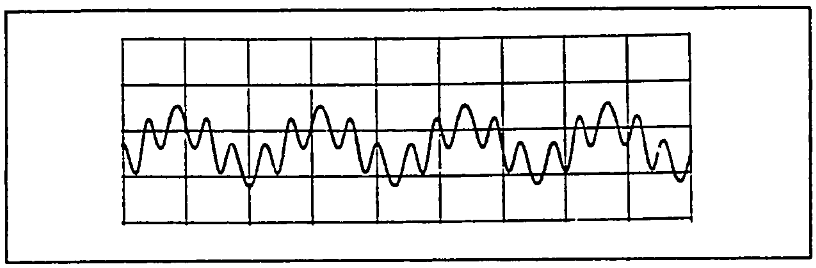


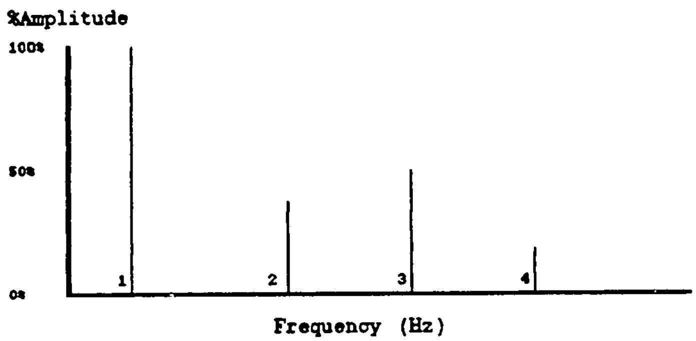
	Frequency	%Amplitude
1.	219.88hz	100%
2.	1100.66hz	100%





	Frequency	%Amplitude
1.	219.88hz	100%
2.	1541.66hz	100%





	Frequency	%Amplitude
1.	219.88hz	99.9%
2.	660.88hz	31.6%
3.	1100.66hz	51.7%
4.	1541.66hz	22.5%

Illustration 7.3 FFT Results and Spectra

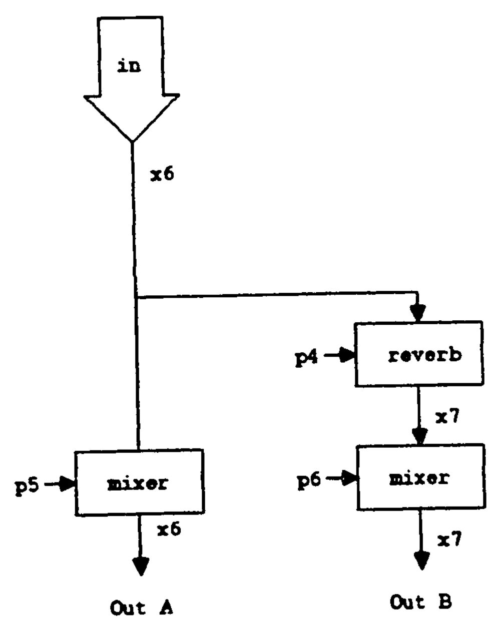
The above MUSIC 1000 code generates two ten second events for each of the 4 examples. Each event is followed by a one second rest. The index begins at zero and gradually increases to 16000 in the first of the two events. The value of 32767 is used in the second event, which creates both maximum index and amplitude. As the index approaches 32767, the fundamental frequency is lost and the specified harmonic(s) are generated.

The sine wave used above is created by the oscillator, ***oscili***, which is an interpolating oscillator capable of smoothing the function for a more accurate waveform. This oscillator component also allows for a smaller function size to be used if memory shortage becomes a problem. The output of the oscillator is routed into the scaling component, ***xfscal***, by means of register *x6*. The appropriate table size is determined by ***xfscal*** as it reduces the size of the input signal to equal that of the assigned function, ***poly***. This allows for the scaled input signal to act as a pointer to values located in the function table. The optional word, ***bipol***, is included if the input signal is both positive and negative, as with an oscillator. Input signals which are only positive or negative in nature do not require the word *bipol*.

The word *table* is where the actual waveshaping process occurs. The scaled input signal is the pointer to the function table and its amplitude determines how much of the table will be accessed. The deeper the pointer is placed into the function table (by increasing the amplitude of the signal), the greater the amount of transformation. In the above MUSIC 1000 code, *x7* represents the output of the table, *poly* refers to the function name, and *x6* is the pointer. The result is a non-linear waveshaping instrument with a timbral quality which is dependent upon the amplitude of the sine wave.

INPUT AND REVERB

In order to use the input and reverb example on diskette #8, a line level signal must be routed into the DMX-1000 at channel 1 with an appropriate cable. This demonstration allows for the original signal to be heard through channel A, while the same signal with reverb is realized at channel B. The reverb time is variable as it is defined within the Score as a parameter. Illustration 7.4 shows the flowchart patches for this example and illustration 7.5 is a listing of the MUSIC 1000 code.



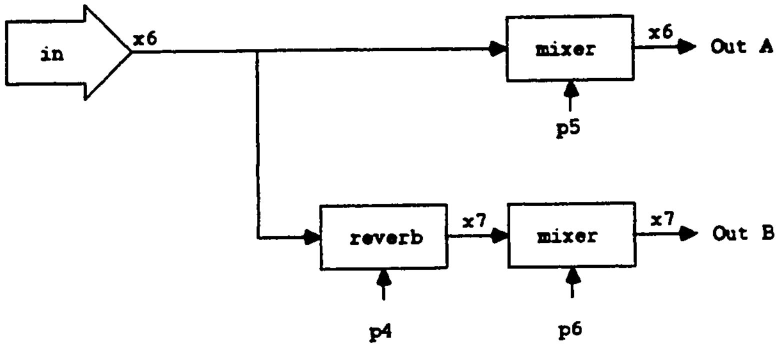


Illustration 7.4 Flowchart Diagrams for Input and Reverb

INPUT AND REVERB

The Score

```
<1>     c     Input and Reverb Demonstration

<2>     c     p1     p2    p3    p4     p5     p6
<3>     c    ins#  start time reverb ampl1  ampl2

<4>     x     1      .     .     1      dB     dB

<5>     i     1      .     1    4000   -20     -5
<6>     e
```

The Orchestra

```
<1>     ;     Input and Reverb Demonstration

<2>     orch

<3>     instr      1,noauto

<4>     in         x6,1
<5>     mixer      x6,<<x6,p5>>
<6>     out        x6,1

<7>     reverb     x7,x6,p4
<8>     mixer      x7,<<x7,p6>>
<9>     out        x7,2

<10>    endin
```

Illustration 7.5 MUSIC 1000 Code for Input and Reverb

The Score above uses a reverb time of 4000 which is located at p4. Parameter p5 is used to attenuate the loudness of the original signal which is output at channel A. The component ***mixer***, found in the Orchestra, allows for this attenuation to occur. In the same manner, p6 is used to attenuate the reverberated portion of the signal which is output at channel B.

The MUSIC 1000 word ***noauto***, found in the ***instr*** line of the Orchestra, is used to keep the instrument active until a ***koff*** command is executed. Once the Orchestra begins running it will continue to do so until ***koff*** occurs. The word ***noauto*** makes this possible.

The signal is brought into the DMX-1000 through the word ***in***, which accepts a signal at either input channel 1 or input channel 2. The above example uses channel 1 as its input channel and the signal is placed at register *x6*. The signal is then attenuated through the ***mixer*** component and output to channel A. The MUSIC 1000 word ***in*** has the following format:

IN   *XIN[,ACHNL]*

Reverb occurs as the signal located at *x6* is placed into the reverberation component and output to location *x7*. Parameter p4 is used to control the reverb time. The reverberated signal is then placed into a ***mixer*** for attenuation and then output through channel B. ***Reverb*** is used in the following way:

REVERB   *XOUT,XIN,REV.TIME*

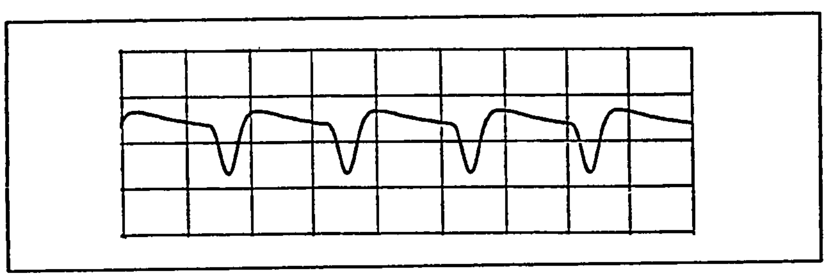
The above example for input and reverb demonstrates a process with numerous possibilities. In the following chapter, some of the musical potential of the MUSIC 1000 language will be explored by the generation of a musical example.

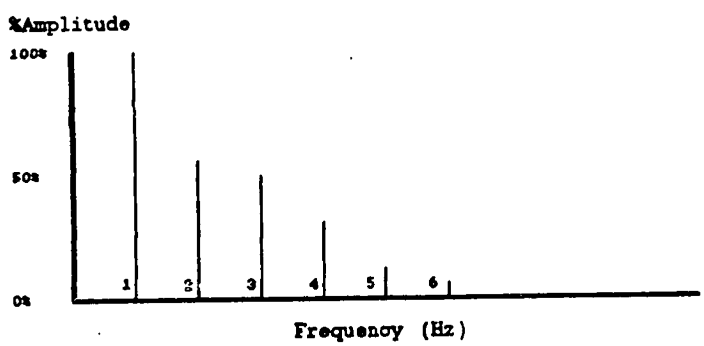
CHAPTER 8

A MUSICAL EXAMPLE

The purpose of this chapter is to illustrate the simulation of an organ tone as well as its control by generating a Score which plays *Menuet* by J.S. Bach. The spectrum of a Rodgers electric organ was obtained by a Fast Fourier Transform (FFT) on the instruments audio signal. This spectrum was then entered into the DMX-1000 for fairly accurate reproduction. Four drawbars are included in the organ simulation and the amplitude of each of the four can be varied from one note to the next through software. Timbral evolution is therefore possible by altering the loudness of the drawbars as this example illustrates.

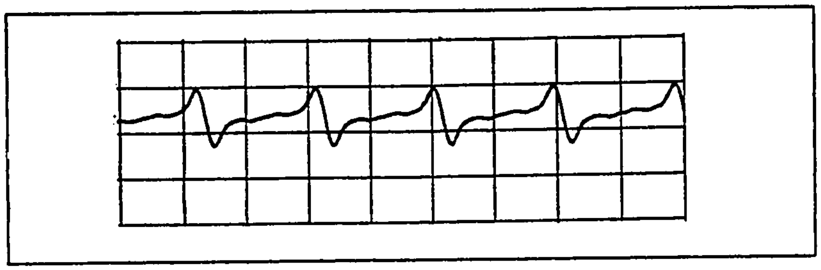
The following spectrum illustrations were taken from both the Rodgers organ and the DMX-1000. The spectrums were sampled at a rate of 200us and a frequency of approximately 261Hz (middle C). Illustration 8.1 is the spectrum obtained from the Rodgers organ while illustration 8.2 was obtained from the DMX-1000. Slight differences are most likely the result of phasing.

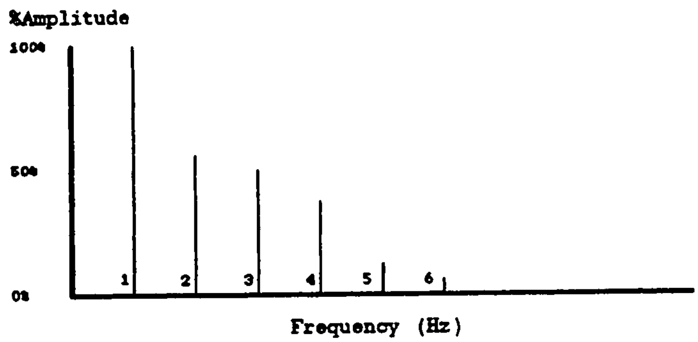




	Frequency	%Amplitude
1.	262.65hz	100%
2.	524.07hz	58.9%
3.	786.71hz	50.4%
4.	1049.36hz	29.7%
5.	1312.01hz	11.6%
6.	1573.42hz	3.9%

Illustration 8.1 Waveform and FFT Results, Rodgers Organ

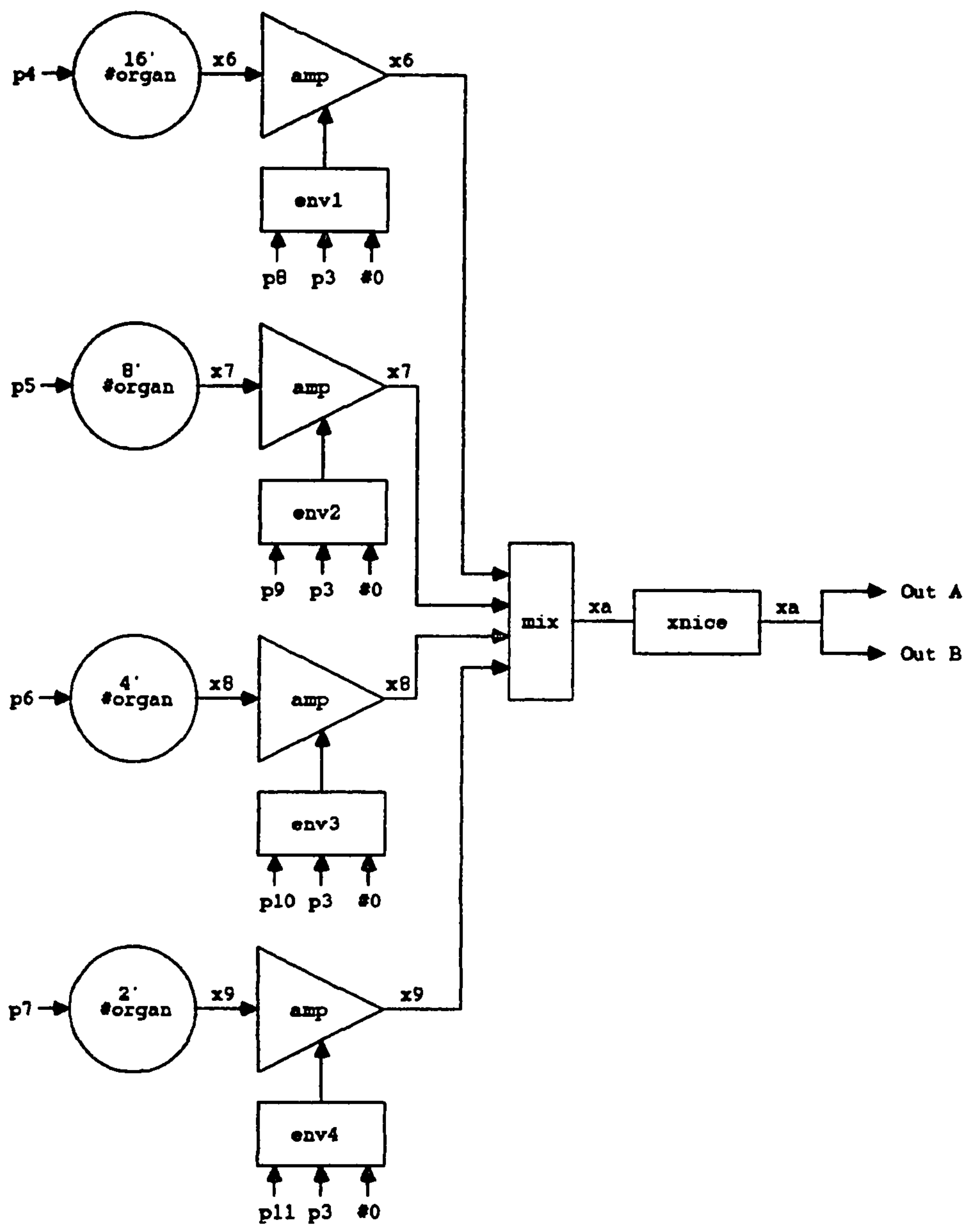




	Frequency	%Amplitude
1.	261.42hz	100%
2.	522.85hz	52.6%
3.	785.49hz	51.2%
4.	1046.91hz	32.2%
5.	1308.33hz	16.5%
6.	1569.76hz	6.2%

Illustration 8.2 Waveform and FFT Results, DMX-1000

The Orchestra for the organ simulation is relatively simple as the spectrum is not dynamic and may be built by using Fourier (additive) synthesis. The percentage of amplitude values extracted from the Rodgers organ sample are used as an argument list for harmonic amplitudes. Two identical voices are used for this example in order to play two separate parts. Illustration 8.3 shows the flowchart patches for the organ voice while illustration 8.4 shows the MUSIC 1000 Orchestra code.


16'
#organ
8'
#organ
4'
#organ
2'
#organ
p4
p5
p6
p7
x6
x7
x8
x9
amp
amp
amp
amp
env1
env2
env3
env4
p8 p3 #0
p9 p3 #0
p10 p3 #0
p11 p3 #0
mix
xa
xnice
xa
Out A
Out B

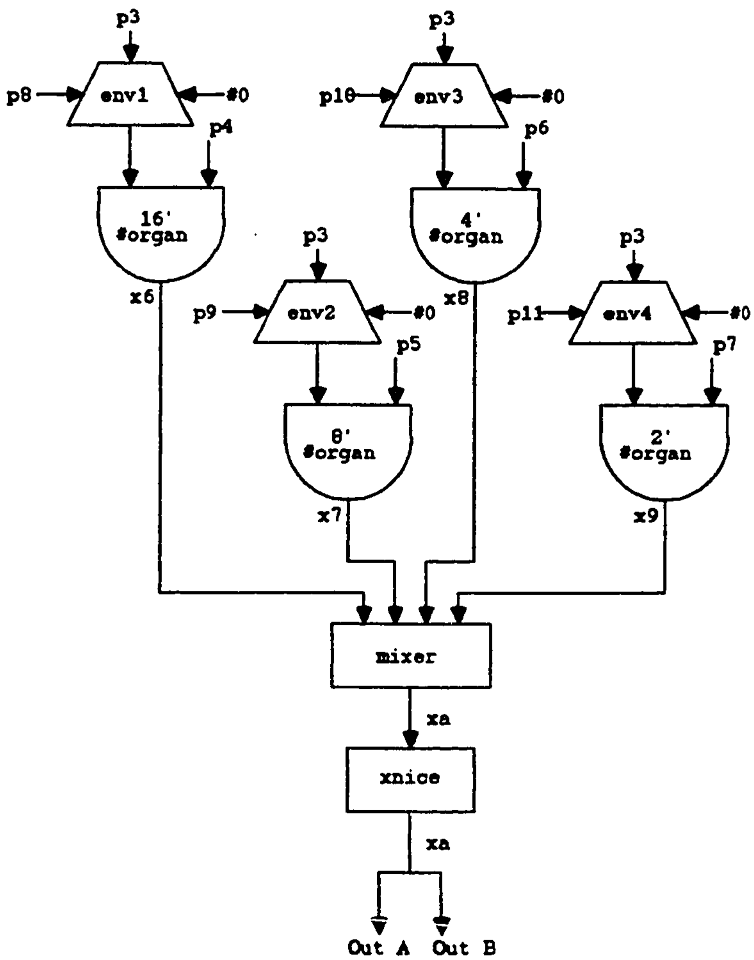


Illustration 8.3 Flowchart Diagrams for Organ Simulation

The Orchestra

```
;       Organ Simulation

orch

fnctn   organ,512,fourier,normal
        6,100,589,504,297,116,39

instr   1
local   <decay,sustn,env1,env2,env3,env4>
idiv    decay,p3,#50
isub    sustn,p3,decay
klnseg  env1,<p8,sustn,p8,decay,#0>    ; 16'
klnseg  env2,<p9,sustn,p9,decay,#0>    ; 8'
klnseg  env3,<p10,sustn,p10,decay,#0>  ; 4'
klnseg  env4,<p11,sustn,p11,decay,#0>  ; 2'

oscil   x6,#organ,p4,env1              ; 16'
oscil   x7,#organ,p5,env2              ; 8'
oscil   x8,#organ,p6,env3              ; 4'
oscil   x9,#organ,p7,env4              ; 2'

mixer   xa,<<x6,8000>,<x7,8000>,<x8,8000>,<x9,8000>>
xnice   xa,xa
out     xa,3
endin


instr   2
local   <decay,sustn,env1,env2,env3,env4>
idiv    decay,p3,#50
isub    sustn,p3,decay
klnseg  env1,<p8,sustn,p8,decay,#0>    ; 16'
klnseg  env2,<p9,sustn,p9,decay,#0>    ; 8'
klnseg  env3,<p10,sustn,p10,decay,#0>  ; 4'
klnseg  env4,<p11,sustn,p11,decay,#0>  ; 2'

oscil   x6,#organ,p4,env1              ; 16'
oscil   x7,#organ,p5,env2              ; 8'
oscil   x8,#organ,p6,env3              ; 4'
oscil   x9,#organ,p7,env4              ; 2'

mixer   xa,<<x6,8000>,<x7,8000>,<x8,8000>,<x9,8000>>
xnice   xa,xa
out     xa,3
endin
```

Illustration 8.4 MUSIC 1000 Orchestra for Organ Simulation

The Score

The Score section of this example is divided into measures, and two identical instruments are used. The MUSIC 1000 word *SNOTE* is used to control the frequency of the oscillators. There are four different pitch values for each instrument which create the drawbar effect. Each of the drawbars also have separate amplitude values which allow for the loudness of each to be varied from one note to the next. Illustration 8.5 shows the musical notation of Bach's Menuet while illustration 8.6 is the MUSIC 1000 Score which is used to generate the composition.

Illustration 8.5 Menuet by J.S.Bach

The Score

```
c     Organ Simulation
c     Menuet  ---  J.S. Bach

t     0  120  43  120  48  80

c     p1   p2   p3   p4   p5   p6   p7   p8   p9   p10  p11
c    ins# strt time fc/2  fc   fc*2 fc*4 env1 env2 env3 env4

c        drawbars = 16'  8'   4'   2'
c          levels =                     16'  8'   4'   2'

c     instrument 1 --------------------------------------

x     1    .    .   snote snote snote snote dB   dB   dB   dB

c     measure 1      16'  8'   4'   2'   16'  8'   4'   2'

i     1    .    1    d7   d8   d9   d10 -20  -20  -20  -20
i     .    .   .5    g6   g7   g8   g9   .    .    .    .
i     .    .   .5    a6   a7   a8   a9   .    .    .    .
i     .    .   .5    b6   b7   b8   b9   .    .    .    .
i     .    .   .5    c7   c8   c9   c10  .    .    .    .

c     measure 2      16'  8'   4'   2'   16'  8'   4'   2'

i     .    .    1    d7   d8   d9   d10  .    .    .    .
i     .    .    1    g6   g7   g8   g9   .    .    .    .
i     .    .    1    g6   g7   g8   g9   .    .    .    .

c     measure 3      16'  8'   4'   2'   16'  8'   4'   2'

i     .    .    1    e7   e8   e9   e10 -25  -20  -15  -15
i     .    .   .5    c7   c8   c9   c10  .    .    .    .
i     .    .   .5    d7   d8   d9   d10  .    .    .    .
i     .    .   .5    e7   e8   e9   e10  .    .    .    .
i     .    .   .5   fs7  fs8  fs9  fs10  .    .    .    .

c     measure 4      16'  8'   4'   2'   16'  8'   4'   2'

i     .    .    1    g7   g8   g9   g10 -15  -20  -25  -25
i     .    .    1    g6   g7   g8   g9   .    .    .    .
i     .    .    1    g6   g7   g8   g9   .    .    .    .
```

```
c    measure 5     16'   8'    4'    2'    16'   8'    4'    2'

i    .   .   1     c7    c8    c9    c10   .     .     -20   -20
i    .   .   .5    d7    d8    d9    d10   -20   .     -15   .
i    .   .   .5    c7    c8    c9    c10   .     .     .     .
i    .   .   .5    b6    b7    b8    b9    .     .     .     .
i    .   .   .5    a6    a7    a8    a9    .     .     .     .

c    measure 6     16'   8'    4'    2'    16'   8'    4'    2'

i    .   .   1     b6    b7    b8    b9    .     .     .     .
i    .   .   .5    c7    c8    c9    c10   .     .     .     .
i    .   .   .5    b6    b7    b8    b9    .     .     .     .
i    .   .   .5    a6    a7    a8    a9    .     .     .     .
i    .   .   .5    g6    g7    g8    g9    .     .     .     .

c    measure 7     16'   8'    4'    2'    16'   8'    4'    2'

i    .   .   1     fs6   fs7   fs8   fs9   -15   -15   .     .
i    .   .   .5    g6    g7    g8    g9    -20   -20   -15   .
i    .   .   .5    a6    a7    a8    a9    .     .     .     .
i    .   .   .5    b6    b7    b8    b9    .     .     .     .
i    .   .   .5    g6    g7    g8    g9    .     .     .     .

c    measure 8     16'   8'    4'    2'    16'   8'    4'    2'

i    .   .   1     b6    b7    b8    b9    -17   -17   -17   -20
i    .   .   2     a6    a7    a8    a9    .     .     .     .

c    measure 9     16'   '8    4'    2'    16'   8'    4'    2'

i    .   .   1     d7    d8    d9    d10   -20   -20   -20   -20
i    .   .   .5    g6    g7    g8    g9    .     .     .     .
i    .   .   .5    a6    a7    a8    a9    .     .     .     .
i    .   .   .5    b6    b7    b8    b9    .     .     .     .
i    .   .   .5    c7    c8    c9    c10   .     .     .     .

c    measure 10    16'   8'    4'    2'    16'   8'    4'    2'

i    .   .   1     d7    d8    d9    d10   .     -15   -15   .
i    .   .   1     g6    g7    g8    g9    .     .     .     .
i    .   .   1     g6    g7    g8    g9    .     .     .     .

c    measure 11    16'   8'    4'    2'    16'   8'    4'    2'

i    .   .   1     e7    e8    e9    e10   -20   -20   -20   -20
i    .   .   .5    c7    c8    c9    c10   .     .     .     .
i    .   .   .5    d7    d8    d9    d10   .     .     .     .
i    .   .   .5    e7    e8    e9    e10   .     .     .     .
i    .   .   .5    fs7   fs8   fs9   fs10  .     .     .     .
```

```
c    measure 12    16'   8'    4'    2'    16'   8'    4'    2'

i    .   .   1     g7    g8    g9    g10   -15   -15   .     .
i    .   .   1     g6    g7    g8    g9    -20   -20   .     .
i    .   .   1     g6    g7    g8    g9    .     .     .     .

c    measure 13    16'   8'    4'    2'    16'   8'    4'    2'

i    .   .   1     c7    c8    c9    c10   .     .     .     .
i    .   .  .5     d7    d8    d9    d10   .     .     -15   -15
i    .   .  .5     c7    c8    c9    c10   .     .     .     .
i    .   .  .5     b6    b7    b8    b9    .     .     .     .
i    .   .  .5     a6    a7    a8    a9    .     .     .     .

c    measure 14    16'   8'    4'    2'    16'   8'    4'    2'

i    .   .   1     b6    b7    b8    b9    -20   -20   -20   -20
i    .   .  .5     c7    c8    c9    c10   .     .     .     .
i    .   .  .5     b6    b7    b8    b9    .     .     .     .
i    .   .  .5     a6    a7    a8    a9    .     .     .     .
i    .   .  .5     g6    g7    g8    g9    .     .     .     .

c    measure 15    16'   8'    4'    2'    16'   8'    4'    2'

i    .   .   1     a6    a7    a8    a9    -15   -15   .     .
i    .   .  .5     b6    b7    b8    b9    .     .     .     .
i    .   .  .5     a6    a7    a8    a9    .     .     .     .
i    .   .  .5     g6    g7    g8    g9    .     .     .     .
i    .   .  .5     fs6   fs7   fs8   fs9   .     .     .     .

c    measure 16    16'   8'    4'    2'    16'   8'    4'    2'

i    .   .   3     g6    g7    g8    g9    -20   -20   -20   -20


c    instrument 2  ------------------------------------

x    2   .   .  snote snote snote snote dB    dB    dB    dB

c    measure 1     16'   8'    4'    2'    16'   8'    4'    2'

i    2   .   2     g5    g6    g7    g8    -20   -20   -20   -20
i    .   .   1     a5    a6    a7    a8    .     .     .     .

c    measure 2     16'   8'    4'    2'    16'   8'    4'    2'

i    .   .   3     b5    b6    b7    b8    .     .     .     .
```

```
c   measure 3    16'  8'  4'  2'  16'  8'  4'  2'
i   .  .  3      c6   c7  c8  c9  -15  -20 -25 -25
c   measure 4    16'  8'  4'  2'  16'  8'  4'  2'
i   .  .  3      b5   b6  b7  b8  -25  -20 -15 -15
c   measure 5    16'  8'  4'  2'  16'  8'  4'  2'
i   .  .  3      a5   a6  a7  a8  -20  -20 -20 -20
c   measure 6    16'  8'  4'  2'  16'  8'  4'  2'
i   .  .  3      g5   g6  g7  g8  .    .   .   .
c   measure 7    16'  8'  4'  2'  16'  8'  4'  2'
i   .  .  1      d6   d7  d8  d9  -15  -15 -15 -20
i   .  .  1      b5   b6  b7  b8  .    .   .   .
i   .  .  1      g5   g6  g7  g8  .    .   .   .
c   measure 8    16'  8'  4'  2'  16'  8'  4'  2'
i   .  .  2      d6   d7  d8  d9  -20  -20 -20 .
i   .  .  1      c6   c7  c8  c9  .    .   .   .
c   measure 9    16'  8'  4'  2'  16'  8'  4'  2'
i   .  .  2      g5   g6  g7  g8  .    .   .   .
i   .  .  1      a5   a6  a7  a8  .    .   .   .
c   measure 10   16'  8'  4'  2'  16'  8'  4'  2'
i   .  .  3      b5   b6  b7  b8  .    -15 -15 .
c   measure 11   16'  8'  4'  2'  16'  8'  4'  2'
i   .  .  3      c6   c7  c8  c9  -20  -20 -20 -20
c   measure 12   16'  8'  4'  2'  16'  8'  4'  2'
i   .  .  3      b5   b6  b7  b8  -15  -15 .   .
c   measure 13   16'  8'  4'  2'  16'  8'  4'  2'
i   .  .  3      a5   a6  a7  a8  -20  -20 -15 -15
```

```
c    measure 14   16'  8'   4'   2'   16'  8'   4'   2'

i    .    .   3   g5   g6   g7   g8   -20  -20  -20  -20

c    measure 15   16'  8'   4'   2'   16'  8'   4'   2'

i    .    .   1   c6   c7   c8   c9   -15  -15  .    .
i    .    .   1   d6   d7   d8   d9   .    .    .    .
i    .    .   1   d5   d6   d7   d8   .    .    .    .

c    measure 16   16'  8'   4'   2'   16'  8'   4'   2'

i    .    .   3   g5   g6   g7   g8   -20  -20  -20  -20

e
```

Illustration 8.6 MUSIC 1000 Score for Menuet

The Score above contains a *TEMPO* statement which is represented by the opcode *T*, and it controls the tempo of the entire score. A slight deceleration of tempo is executed at the end of the example and it is done with the *T* statement. The *TEMPO* statement has the following format:

```
t    p1  p2  p3  ...  p50
```

p1	*ignored*
p2	*initial tempo in beats per minute*
p3,p5,p7 ... p49	*referenced times in beats*
p4,p6,p8 ... p50	*tempi for the referenced times*

Example:

```
t    1   120   43   120   48   80
```

The initial tempo is 120 beats per minute. The example has a total of 48 beats, and the deceleration is to begin at beat 43. The tempo will gradually change from 120 to 80 between beat 43 and beat 48. The process works like this:

Tempo at beat 1 = 120
Tempo at beat 43 = 120
Tempo at beat 48 = 80

The tempo slows from 120 to 80 beats per minute between beat 43 and beat 48.

The drawbar effect is achieved by using four different *SNOTE* values as well as four separate *dB* values. Each corresponds to one of the drawbars; 16', 8', 4', or 2'. By altering the *SNOTE* values to change pitch, and the *dB* values to change loudness, the organ simulation is possible. The note lengths are determined by p3. The value 1 represents a quarter note, .5 equals an eighth note etc.

The MUSIC 1000 word *SNOTE* allows for the specification of oscillator frequency by the assignment of a pitch letter and an octave number. For example, middle C is represented by the code *c8*, the octave above it would be *c9*, and so forth. Sharps and flats are represented by the characters *s* and *f*, therefore, *cs8* is the C sharp a semitone above middle C.

The process of translating a score from traditional notation into MUSIC 1000 code is unusual, but not difficult. The Score for this example is an illustration of the musical potential of the MUSIC 1000 language. Although the required method of notation is different than more traditional types of musical notation, MUSIC 1000 does provide the user with an acceptable format for musical generation. There are many possible ways to generate sonic information with MUSIC 1000 and the DMX-1000, and this example represents one method for entering musical data under the rules of MUSIC 1000.

CHAPTER 9

FILE MANIPULATION

The basic operation of the Terak computer system will be addressed including the creation of a new work disk, the basic operations of *ASOTE,* and the manipulation of files. The following instructions can be used to create Score and Orchestra files, and edit them as necessary.

The Creation of a Work Disk

1. Insert a MUSIC 1000 System Disk into the A (left) drive unit.

2. Place a write enable sticker on a new 8" diskette and place it in the B (right) drive unit.

3. Turn the computer system on.

4. Type the DATE as prompted, e.g., ***01-jan-89***

5. Type ***@A:NEWDSK*** followed by ***<return>***.

6. The computer will now prompt for disk type;
   Type *1*

   This corresponds with single sided/double density diskettes.

7. The computer will now prompt for drive unit;
   Type *1*

   This corresponds with drive unit B (or 1).

8. When the format is complete
   Type *S*

   This corresponds with Stop.

9. The disk will be complete and ready for use after the diskette has been formatted and the necessary files are copied to it. The process is complete when the dot (.) prompt appears.

File Conventions

There are four necessary files for the execution of any MUSIC 1000 program. These files include *SCORE.USR* and *ORCH.USR*. Both of these files are in ASCII and are created by the user. Once compiled, the additional files named *SCORE.SRT* and *ORCH.SAV* are generated by the compiling process. *ORCH.SAV* is an executable file which reads *SCORE.SRT*. In order to execute *ORCH.SAV* the command ***RUN ORCH*** is entered by the user. The name of the program may also be altered as in example 4.1 when *ORCH.SAV* has been changed to *SINE.SAV*, and ***RUN SINE*** is used to execute the program.

The *SCORE.SRT* file can not be changed and is unique to each diskette. This creates a problem when more than one score is desirable. While numerous *XXX.SAV* programs may exist on one diskette, only one *SCORE.SRT* program may exist. It is not possible to use more then one *SCORE* on any one diskette.

Compiling Times

The compiling process requires a *SCORE.USR* file and an *ORCH.USR* file on the same diskette. In order to compile both the Score and the Orchestra the following command must be typed:

*@NEWSO followed by a **<return>***

The compiling process may last from 5 to 15 minutes.

It is possible to compile either the Score or the Orchestra independently of the other. This will save a notable amount of time if the Score requires modification rather than the Orchestra. The Score compiles rather quickly depending upon its length. Typical times range from 1 minute to 4 or 5 minutes. The Orchestra sections will require a minimum of five minutes to compile. In order to compile the Score only type:

*@NEWS followed by a **<return>***

In order to compile the Orchestra only type:

*@NEWO followed by a **<return>***

File Manipulations

A few easy to use file manipulations will provide great assistance during the course of program development. These processes include *RENAME*, *DEL*, and *COPY*.

RENAME Used to change the name of files.

To rename files type the following:

RENAME filename.old filename.new <return>

For example:

RENAME ORCH.SAV SINE.SAV <return>

ORCH.SAV would now be titled *SINE.SAV* on the diskette.

DEL Used to delete files.

To delete a file type the following:

DEL filename.ext <return>

For example:

DEL ORCH.SAV <return>

The file *ORCH.SAV* will be erased from the diskette. **BE CAREFUL**, once files are deleted they are gone. Exercise extreme caution when using *DEL*.

COPY Used to copy files, and backup diskettes.

To copy a file from the A drive unit to the B drive unit do the following making sure that the diskette to be copied onto is a formatted work disk:

COPY/WAI A:filename.ext B:*.* <return>

The system will then prompt the user to mount the input volume (the diskette with the files to be copied), in drive A (drive 0). Remove the MUSIC 1000 System Disk from drive A and replace it with the diskette containing the files to be copied. Press *Y* and then ***<return>***. The system will next prompt for an output volume in drive B (drive 1). Place the destination diskette into drive B, press *Y* and then ***<return>***. The file(s) will then be copied to the diskette in the B drive unit. Once the process is complete, the system will prompt the user to mount the system volume in drive A (0). Remove the disk from drive A and replace the MUSIC 1000 System Disk. Press *Y* and *<return>* to finish the procedure. The * symbols represent wildcards which can be used in place of file names. In order to backup an entire diskette by copying every available file type:

COPY/WAI   A:.*   B:*.*   <return>*

ASOTE Basics

The available text editor *ASOTE* (A Screen Oriented Text Editor), is not a difficult package to learn. Appendix A includes complete documentation concerning *ASOTE* and it should be referred to as necessary. The following procedure is a format for the creation of a text-file using *ASOTE*.

1. Boot the system with a ***MUSIC 1000 Systems Disk*** placed in the A drive unit.

2. Place a formatted work diskette in the B drive unit (making sure that a write enable sticker is on the diskette).

3. Type the date as prompted.

4. Type *R ASOTE*
   Press *<return>*

5. ASOTE will then ask for a file name. To create a new file such as SCORE.USR type the name, ***SCORE.USR*** as prompted and press ***<return>***.

6. The ASOTE menu will then appear on the top of the screen. IMPORTANT... Type an *I* before typing text. *I* allows the INSERT mode to be used.

7. Type the file. NOTE... In order to type lower case characters the DC2 key must be pressed. It is located on the lower right side of the keyboard just above the ETX key.

8. When the text-file is complete the ETX (accepts) key must be pressed to exit the

insert mode.

9. Press *Q* to QUIT the edit mode.

10. Press *U* to UPDATE the file and save it, or press *B* to save the file and create a backup.

11. Press *X* to exit ASOTE. The file is now saved to the B diskette.

Modifying Existing Files

The following instructions allow for the copying of a file from the A drive unit to the B drive unit and for its modification using *ASOTE*. The file to be used in this example will be the *SCORE.USR* file on the ***Subtractive Synthesis Examples*** Diskette #2. Any file may be modified in the same manner.

1. Boot the system in the usual fashion.

2. Copy the files onto a work diskette by doing the following:

   Type *COPY/WAI A:*.* B:*.**
   Press *<return>*

   Note: It is not necessary to copy files in order to modify them, it is done here to preserve the original diskette.

3. When the process is complete and the ***MUSIC 1000 Systems Disk*** is returned in the A drive:

   Type ***R ASOTE***
   Press *<return>*

4. Type *SCORE.USR* when prompted.

5. The SCORE.USR file will then be listed on the screen.

6. Using the cursor keys (arrow keys), move the cursor to the first *i* statement. Place the cursor directly on top of the 2 in the value 20 located at the end of the line. This value controls the filter bandwidth.

7. Type ***X*** (exchange text).

8. Type ***1*** followed by pressing the ***SPACE BAR***.

9. Press ***ETX*** (accept text). The 20 should now be changed to 1.

10. Press ***Q*** to QUIT the edit mode followed by ***U*** to UPDATE the file and an ***X*** to EXIT ASOTE.

11. Type ***RENAME BAND.USR ORCH.USR <return>***

12. To re-compile the Score type ***@NEWS <return>***.

13. Type ***RENAME ORCH.USR BAND.USR <return>***

14. Once the compiling process is complete the results may be heard by running the Orchestra (***RUN.BAND***). The filter now has a bandwidth of 1hz (maximum Q).

CONCLUSION

This interactive tutorial is designed to illustrate the various applications of the DMX-1000 and its control under the language MUSIC 1000. By demonstrating a number of synthesis techniques, it has been shown that MUSIC 1000 does support various forms of synthesis. While this manual is not intended to be an exhaustive list of synthesis schemes, the demonstrations inculded in this tutorial prove the flexibility of MUSIC 1000 and the ability to generate numerous synthesis formats on the DMX-1000.

The musical applications of the language MUSIC 1000 have been explored only briefly due to the aesthetic implications involved with the generation of "music". The focus of this tutorial is placed on the creation and processing of audio signals, while the musical applications of the sonic information are left up to the user.

The purpose of this manual is to demonstrate that various synthesis techniques are possible on the DMX-1000, and the examples which support this intention are not meant to be interesting in a musical sense. The examples presented in this tutorial are demonstrations of a few basic building blocks which are important to the field of electro-acoustics.

PLEASE NOTE:

Copyrighted materials in this document have not been filmed at the request of the author. They are available for consultation, however, in the author's university library.

These consist of pages:

92-98, Appendix A

99-104, Appendix B

U·M·I

APPENDIX C

Audio Tape Contents

Side A

1. Audio Demonstration Number One

   Additive Synthesis
   Example One: Sine Wave Synthesis
   Example Two: Square Wave Synthesis
   Example Three: Sawtooth Wave Synthesis
   Example Four: Inverted Spectrum Synthesis

2. Audio Demonstration Number Two

   Subtractive Synthesis
   Example One: Band Pass Filtering
   Example Two: High Pass Filtering
   Example Three: Low Pass Filtering

3. Audio Demonstration Number Three

   Modulation Synthesis
   Example One: Frequency Modulation Synthesis
   Example Two: Amplitude Modulation Synthesis

4. Audio Demonstration Number Four

   Nonlinear Waveshaping Synthesis
   Example One: Third Harmonic
   Example Two: Fifth Harmonic
   Example Three: Seventh Harmonic
   Example Four: Third, Fifth, and Seventh Harmonics

5. Audio Demonstration Number Five

   Input and Reverb
   Example One: Input and Reverb

6. Audio Demonstration Number Six

   A Musical Example
   Example One: *Menuet* by J.S. Bach

Side B

1. *cps Reflections*

   Eric Gatzert
   Copyright 1987

BIBLIOGRAPHY

Texts

Dodge, C., and Jerse, T.A. *Computer Music Synthesis, Composition, and Performance.* Schirmer, New York, 1977.

Howe, H.S. *Electronic Music Synthesis.* Norton, New York, 1975.

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

Roads, C., and Strawn, J. eds. *Foundations of Computer Music.* The MIT Press: Cambridge, Mass., 1985.

Roads, C. eds. *The Music Machine: Selected Readings from Computer Music Journal.* The MIT Press: Cambridge, Mass., 1988.

Strange, J.A. *Electronic Music Systems, Techniques, and Controls.* Wm. C. Brown Company Publishers: Dubuque, Iowa., 1983.

Articles

Arfib, D. "Digital synthesis of complex spectra by means of multiplication of nonlinear distorted sine waves." *Journal of the Audio Engineering Society* 27(10): 757-768, 1979.

Chowning, J. "Computer synthesis of the singing voice." In Johan Sundberg (ed.), *Sound Generation in Winds, Strings, and Computers.* Royal Swedish Acamedy of Music: Stockholm, 1980.

LeBrun, M. "Digital waveshaping synthesis." *Journal of the Audio Engineering Society* 27(4):250-266, 1979.

Roads, C. "A tutorial on nonlinear waveshaping synthesis." *Computer Music Journal* 3(2):29-34, 1979.

Manuals

Wallraff, D. *MUSIC-1000 Manual*. Digital Music Systems Inc.: Boston, Mass., 1983.

Wallraff, D. *DMX-1000 Hardware Manual*. Digital Music Systems Inc.: Boston, Mass., 1983.

Wallraff, D., and Marshal, P. *Computer Music Catalog*. Digital Music Systems Inc.: Boston, Mass., 1983.