MAX LANGUAGE USER GUIDE

June 15, 1982

CONTENTS

USING THIS GUIDE

This Guide is an introduction to the MAX language. It assumes that you are familiar with the Synclavier II (R) II real-time system as well as with entering, editing and performing SCRIPT compositions or XPL programs on the computer terminal.

This Guide can also serve as a tutorial in the Scientific XPL language because it provides many simple programs for you to examine and run. But it does not attempt to be a complete guide to XPL. Turn to the _Scientific XPL Reference Manual_ for further information and review. MAX is designed for use with the XPL/4 version of the XPL operating system. Please refer to the _Scientific XPL/4 Documentation Update_ for additional information on the monitor, the Winchester disk and new catalog structure. The complete _Synclavier (R) II Instruction Manual_ and _Synclavier (R) II Setup and Installation Manual_ will also be helpful.

Fifteen sample MAX programs are stored on your MAX user diskette(s) under the filenames DEMO1 through DEMO15. It would be helpful if you read the following chapters with your computer on so that you can run these programs as you read about them. Also the comments in the sample programs offer a great deal of useful documentation.

PREFACE

The MAX Language offers complete control of the Synclavier (R) II Digital Synthesizer and access to all internal computer capabilities. It consists of a library of special XPL/4 procedures which you can use in the development of your own customized digital synthesis system. All New England Digital hardware is supported.

REAL-TIME INTERACTION

With MAX, you can create a system that allows a performer to interact with the Digital Synthesizer in new ways. You are not restricted to the Synclavier (R) II real-time system. The pedal could be used to control recording speed. Or, input from the keyboard, pedal, and knob could direct the computer to generate a particular set of pitches in a unique manner.

INSTRUCTIONAL APPLICATIONS

MAX is an ideal tool for musical instruction. For instance, an ear-training drill could be devised in which a note would sound and the user would respond by typing a pitch name. Since you have access to a powerful 16-bit minicomputer, you can create extremely sophisticated instructional drills.

FILM AND VIDEO

MAX can be used to design custom interfaces with film and video equipment or to automate such production tasks as the generating or reading of SMPTE time codes.

XPL LANGUAGE BASE

Scientific XPL is the language used in all Synclavier (R) II software. Its optimizing compiler is especially designed for real-time applications. MAX is a library of special Scientific XPL procedures which enhance the basic language with features needed for musical applications.

SPL COMPATIBILITY

Many users will be familiar with SPL, the music language developed for the Synclavier I Digital Synthesizer. While MAX is upwards compatible with SPL, it offers many new features and is easier to use. New features include several forms of pitch conversion, a simpler method of channel allocation, new waveform memory controls, the ability to accept input from and send output to any hardware interface, and a multiple task system for parallel processes.

1. MAX REQUIREMENTS

HARDWARE

To use the MAX language, you need a Synclavier (R) II Digital Synthesizer with at least 40K memory, a computer terminal, and one of the following storage device configurations:

a. two single density 5 1/4 inch drives
b. one double density 5 1/4 inch drive and
   one single density 5 1/4 inch drive
c. two double density 5 1/4 inch drives
d. two 8 inch drives
e. one Winchester disk and any of the above floppy drives

A single Winchester of any size will vastly expand the storage capacity of the system and will speed the compilation of programs. Additional Winchester disks and 5 1/4 inch drives can be added to any system.

MAX programs can be written and run without the Synclavier (R) II keyboard unit. However, the addition of this unit enables the user to enter real-time data into the program through the knob, buttons, and keys. One or two Morley pedals offer another means of entering real-time data.

And if you install the special PRINTER/MODEM port on the Digital Synthesizer, a printer and modem can be added as well. MAX programs can be listed on the DECwriter, LA-34, Printronix, PRISM80, and Diablo printers.

SOFTWARE

A SCRIPT-MAX-XPL software license is required for MAX.

The MAX software consists of various files that you INSERT into XPL programs to perform musical functions. This software is provided on one or several user diskettes. It is used with the standard Scientific XPL/4 operating system and monitor, loaded either from a system diskette or from the Winchester disk. <u>Repeat:</u> The MAX software will not work with earlier versions <u>of XPL.</u>

All MAX users, except those with drives for 5 1/4 inch single density diskettes, will receive two copies of the following diskettes:

Scientific XPL/4 System Diskette
MAX User Diskette (contains the six files of the MAX library and fifteen demo files)
Scientific XPL/4 Utility Program Diskette

MAX users with dual 5 1/4 inch single density drives will receive two copies each of the following diskettes:

Scientific XPL/4 System Diskette
MAX User Diskette #1 (contains MAXSYN, MAXIO, Demo1 through Demo11)
MAX User Diskette #2 (contains MAXSYN, MAXIO, MAXTASK, Demo11-Demo14)
MAX User Diskette #3 (contains MAXSYN, MAXIO, MAXTASK, Demo15)
MAX Source Diskette (MAXSYN1, MAXIO1, MAXTASK1)

Users of Winchester-based systems will also receive two copies each of the Winchester Installation and Winchester Bootload Diskettes. Instructions on their use may be found in the Scientific XPL/4 Documentation Update.

2. THE ORGANIZATION OF THE HARDWARE

The Synclavier (R) II Digital Synthesizer is a complex combination of 16-bit minicomputer and digital synthesizers. When you use the Synclavier (R) II real-time system software to operate the Digital Synthesizer, you can be somewhat fuzzy about which functions are performed by the computer and which are performed by the synthesizers. You interact with the Synclavier (R) II keyboard unit and pedal as a musician and performer rather than as a computer programmer.

To learn to program in MAX, however, you must understand clearly the functional distinctions between computer and synthesizer.

The computer is the director of the system. It determines the property of each sound, its harmonic content, volume envelope, duration, pitch, etc. and transfers all this information in the form of a compact set of digital code to the synthesizers. They in turn slavishly perform the repetitive task of synthesizing the digital waveforms. This highly efficient system can produce very precise numbers at a very high sampling rate, allowing glitch-free portamento and vibrato and preventing alias distortion.

There are eight channels, or voices, in each synthesizer. For each channel there are two Sample Rate Generators, one for the carrier and one for the modulator. These two generators are paired together to produce sounds with varying overtone content by frequency modulation.

A separate set of parameters, or control code, from the computer controls each channel. The control code for each channel includes: a 24-bit frequency descriptor, an 8-bit volume, 16-bit rates and 8-bit limits for the volume and index interpolators, a 2-bit index shift count, and a pointer to one of 32 waveform memories which can be loaded with any 256-point waveform. The figure on the following page indicates the organization of one synthesizer channel.

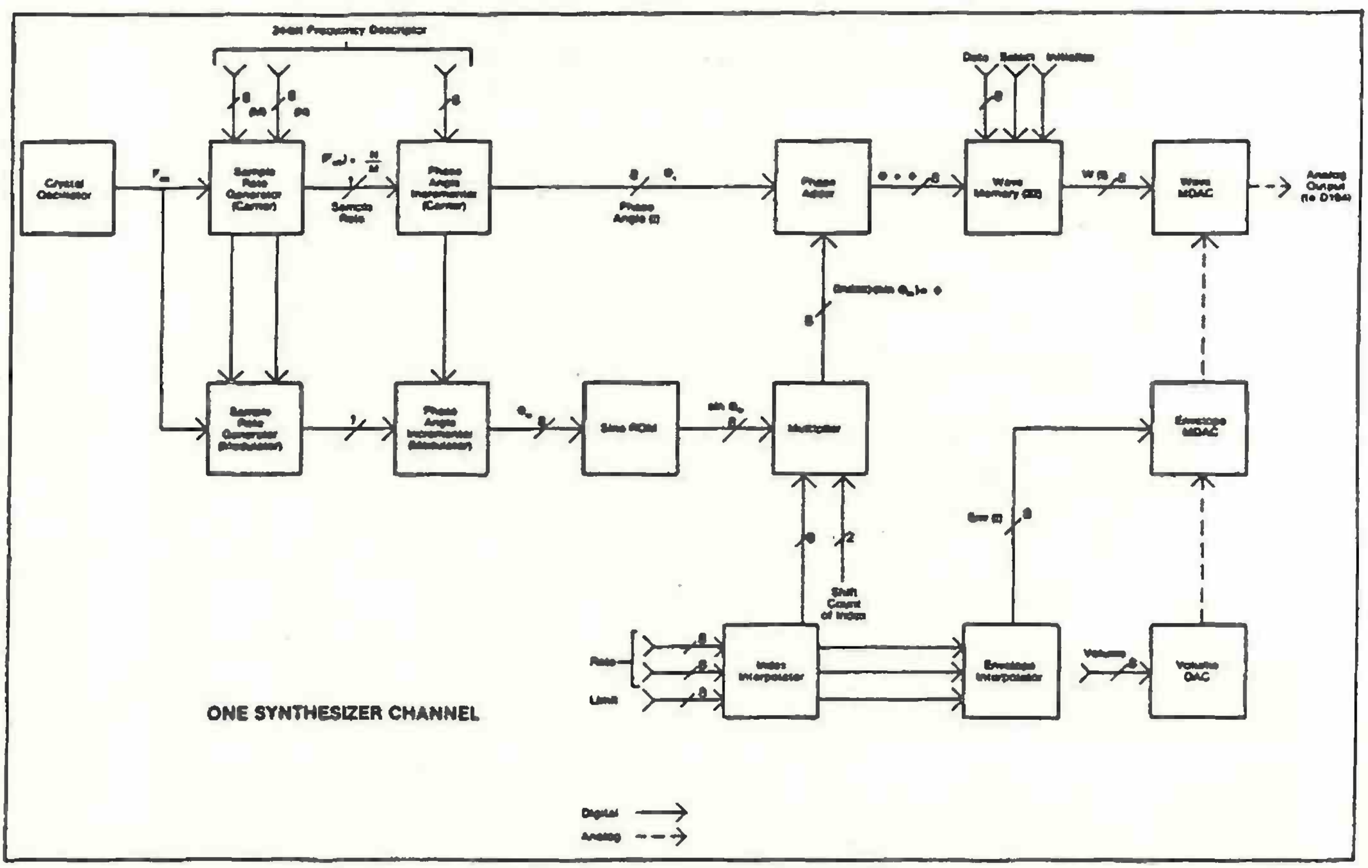
To send this control code, you do not have to resort to assembler language. Instead you use high-level, easy-to-learn MAX procedures such as SETFRQ to set the frequency for a channel or SETVOL to set the volume. In addition, you do not need to know the digital codes, but can use familiar units such as frequencies in hertz.

On the other hand, you are not limited to the note-processing data structure of the Synclavier (R) II real-time system. With MAX, sounds can become "events" of any form or length, interacting with other events or the performer in many different ways. You can create arbitrarily complex envelopes, combine channels, add multiple chorus oscillators, or design any type of vibrato or frequency modulation.

The synthesizers are not directly connected to the buttons and keys of the keyboard unit in any way. All input from and output to the Synclavier (R) II keyboard unit, or any other input or output device, are transmitted to and interpreted by the computer. The computer registers every key pressing, every turn of the knob, every push of a button. It displays messages in the window and lights the buttons.

This means that MAX programs can add new features or completely new capabilities to the Digital Synthesizer, or to the keyboard unit, pedal, ribbon controller, etc. New input or output devices can be added to the system and additional synthesizer boards can be plugged in at any time.

The modularity of both the software and the hardware is what allows New England Digital to continue to add new features to the Synclavier (R) II. And it is what allows you to design your own MAX system.



3. THE MAX LIBRARY

A Scientific XPL procedure is a group of computer instructions which accomplish a particular purpose. MAX is a library of proven XPL procedures, such as SETFRQ and SETVOL, which you may use, or CALL, in your XPL programs to do the low-level communication with the synthesizers and to control the I/O devices connected with the computer.

In other words, you write a simple functional statement. The MAX procedure will specify the necessary device addresses and registers for you.

Before you proceed with this manual, you should understand the basic concepts relating to procedures. If you are a new XPL programmer, refer to the Scientific XPL Reference Manual. There are several different types of procedures. Some are passed one or more parameters. Others are called with no parameters. And some procedures return a value or values. These last procedures are classified in this manual as functions.

There are three sections in the MAX library. Each is completely independent of the others, although they are designed to work together as well. Each section of the library is a file on the MAX user diskette which you INSERT into your XPL/4 program. You may insert one, two or all three in your programs. If you do not require the procedures of any of the sections, you need not waste computer memory by inserting unnecessary code in your programs.

The first section, or file, is called MAXSYN. It contains the basic procedures for setting frequencies, waveforms, and volumes for synthesizer channels. To use this section, include the statement

```
100 INSERT 'MAXSYN';
```

at the start of your program. The line number, shown here as 100, can be any number, as long as the statement appears before the first executable statement in the program. You do not need the Synclavier (R) II keyboard unit to use MAXSYN. MAXSYN is explained in Chapter 4.

The second section is called MAXIO. It contains procedures which accept real-time data from the Synclavier (R) II keyboard unit, pedals, and knob, as well as output data to the digital display window, the button lights, and the control voltage jacks on the back of the keyboard unit. To use this section, include the statement

```
110 INSERT 'MAXIO';
```

at the start of your program. The initialization code in MAXIO requires that a Synclavier (R) II keyboard unit be connected to the system. If you do not have a keyboard unit, you will not want to insert MAXIO. MAXIO is explained in Chapter 5 of this manual.

The third section is called MAXTASK. It is a multi-task executive, enabling you to perform complex events with parallel processes. To use this section, include the statement

```
120 INSERT 'MAXTASK';
```

at the start of your program. You may wish to use MAXTASK for applications which have nothing to do with digital synthesis, for instance, for real-time laboratory data analysis with multiple channels and events. In such an application, you would insert only the MAXTASK section into your program. The MAXTASK procedures are complicated and require some careful thought before use. Chapter 6 describes MAXTASK in detail.

New sections will be added to the MAX library in the future. Although these first three sections are independent of each other, future sections will most likely be "higher-level" and will rely upon the present three for carrying out certain functions.

There are two versions of each MAX file: the compacted version and the source version. In the compacted files, that is, MAXSYN, MAXIO, and MAXTASK, comments and line numbers have been removed for faster compilation. These files cannot be listed. In the source files, which can be listed, you will find many helpful comments. You can also make changes in the source files if desired. The source files are named MAXSYN1, MAXIO1, and MAXTASK1. Both versions of the files may be found on the 8-inch or double density 5 1/4-inch MAX user diskettes. On the single density 5 1/4 inch diskettes, the compacted files are found on User Diskette #1, #2, and #3. The source files are found on the Source Diskette.

The MAX files, in either form, may be saved on a different diskette from the diskette on which you develop your programs. If so, the XPL operating system will ask you to insert the proper diskette into the right-hand drive at certain points during compilation of a MAX program. This means that you do not have to keep a copy of the MAX library on all of your diskettes. Space is valuable if you are using minidiskettes.

Winchester disk users will want to store the MAX library files, as well as their own MAX programs, on the Winchester. You can use the Winchester Installation diskette for this purpose.

Although the three sections of the MAX library are independent, this manual covers them in an order which progresses from the simplest to the most difficult. And all our sample programs build on concepts explained in prior ones. Therefore, it is recommended that the chapters be read in consecutive order.

4. MAXSYN

MAXSYN is the basic set of procedures used to control the digital synthesizers. Each procedure call applies to a particular channel of the synthesizer. For a channel to sound, it must be assigned a frequency for the modulator and carrier, and an above zero value for its volume register and an above zero limit for its volume envelope interpolator. It may be given complicated volume envelope and harmonic envelopes by setting a series of rates and limits for the volume and harmonic interpolators. In addition, each channel can be linked to a waveform memory that the MAX program loads with a complex waveform.

After presenting a simple demonstration program, we will explain in detail the procedures for setting these parameters.

DEMO1 - A SIMPLE SOUND

First load the XPL/4 operating system and then place the MAX user diskette in the right-hand drive.

DEMO1 on the user diskette is a simple MAX program that plays 25 randomly selected pitches. Examine the program by typing OLD DEMO1 and then LIST on your terminal. Then type RUN to run the program. It will take about 30 seconds to compile the program and set up the channel before the pitches will be played.

Without explaining any of the statements in detail, the simple program works like this: It first obtains one of the synthesizer channels (the ALLOCATE function), opens up the volume (the SETVOL procedure) and sets the volume envelope limit to the maximum (the SETELIM procedure).

Then the program goes through a loop 25 times. Each time through, the program selects a random number between 2200 and 8800 (the RND function), uses the random number for a frequency number (the HERTZ function), sets the frequency for the channel (the SETFRQ procedure), and then waits for 500 seconds (the WAIT procedure) while the note plays.

After the last time through the loop, the synthesizer is shut down to zero volume (the ZEROSYN procedure).

ALLOCATING A CHANNEL

Since each voice, or channel, of the synthesizer has two oscillators, or Sample Rate Generators, channels are numbered by two's starting from zero. Thus, in an eight-voice Synclavier (R) II system, the channel numbers are 0, 2, 4, 6, 8, 10, 12, and 14.

You can let MAXSYN do the work of numbering and assigning channels through two convenient procedures: ALLOCATE and FREECHAN. ALLOCATE is a function which selects a channel from the list of free channels and returns the selected channel number. FREECHAN is a procedure which is passed a channel number; it places that channel back on the free channel list. These two procedures also keep track of the number of channels in the system, so that conflicts in channel allocation will not occur. Examine the following:

```
100 DCL CHAN FIXED;       /* CHAN IS FIXED POINT VARIABLE
                             USED TO HOLD CHANNEL NUMBER */
110 CHAN=ALLOCATE;        /* ALLOCATE SELECTS FREE CHANNEL
                             AND PASSES ITS NUMBER TO CHAN */
       ...
       ...                /* USE CHANNEL FOR SOUND EVENT */
       ...
       ...

160 CALL FREECHAN(CHAN); /* FREECHAN RETURNS CHANNEL NUMBER
                             TO LIST OF FREE CHANNELS */
```

SETTING THE PARAMETERS FOR A CHANNEL

We will now explain in detail the procedures used to set parameters for a channel. You will find that the names of more frequently used procedures and variables tend to be short, while those less frequently used are longer and contain periods. You will also note that there are several procedures that start with the letters SET. This usually means that the procedure is actually a call to two, or more, procedures. The first of the procedures usually, but not always, performs a calculation and the second emits data to the synthesizers. Sometimes there is no calculation necessary and the SET statement simply emits data to the synthesizers.

SPECIFYING A FREQUENCY

There are two frequencies to specify for each channel: the carrier and the modulator. There are also two procedures with which to do this. In most cases, you would use a call to SETFRQ, where the frequency of the carrier is specified and the modulator is determined by a specified modulator-to-carrier FM ratio, as in the Synclavier (R) II real-time system. At other times (e.g., for a constant modulator frequency), you may wish to call the SETFRQ2 procedure, where both frequencies are specified.

The call to SETFRQ takes the following form:

<ln> CALL SETFRQ (CHAN,FREQ.NUM,RATIO);

SETFREQ is the procedure which calculates and emits a frequency descriptor to the channel. CHAN is the channel number previously set by the ALLOCATE function. FREQ.NUM is the internal frequency code output by one of four conversion functions HERTZ, PITCH, KEY, or PCH to be described below. RATIO is the FM ratio multiplied by 1000. (i.e., use 1000 for an FM ratio of 1.000, 2000 for 2.000, 500 for 0.500, etc.) If the harmonic envelope is zero or unspecified (see page 23), there will be no FM. When this is the case, use a ratio code of 1000, in order to avoid unnecessary computation time.

The call to SETFREQ2 takes the following form (the "2" indicates that two frequencies are specified):

<ln> CALL SETFRQ2 (CHAN,C.FREQ.NUM,M.FREQ.NUM);

C.FREQ.NUM is the internal frequency code for the carrier frequency. M.FREQ.NUM is the internal frequency code for the modulator frequency. Any of the conversion functions described below may be used for either argument.

The Frequency Conversion Functions

Different users have different preferences in the way they refer to frequency. A researcher in psychoacoustics might want to specify frequency in hertz, while a composer might want to use pitch letters such as C, D, or F#. On the other hand, a composer of computer music might prefer the form of pitch specification known as octave-point pitch-class. And others may wish to use the simplest approach of all, key numbers. MAX supports all four methods.

HERTZ is a conversion function which accepts an integer representing a frequency in hertz multiplied by 10. That is, use 4400 for 440.0 hertz. Thus, the statement

```
100 CALL SETFRQ (CHAN,HERTZ(4400),1000);
```

would calculate and emit to the synthesizer channel a frequency descriptor based on a frequency of 440.0 hertz for the carrier as well as for the modulator.

PITCH is a conversion function which accepts the pitch letters used in the SCRIPT language. The letters are enclosed in apostrophe marks. Thus, the statement

```
100 CALL SETFRQ (CHAN,PITCH('C#3'),2000);
```

would calculate and emit to the channel a frequency descriptor based on middle C# with an FM ratio of 2.000. As in SCRIPT, a # indicates a sharp and an F indicates a flat. The number right after the pitch indicates the octave. This octave number will be used in the subsequent frequency specifications until a different octave is specified. (The initialization octave number is 3, the octave that begins with middle C.) Sharps or flats, however, apply only to the one call to SETFRQ.

KEY is a conversion function which accepts a key number, from 0, for the lowest C on the keyboard, to 60, for the highest C. This coding corresponds to codes returned by the keyboard scanning procedure in MAXIO. Thus, the statement

```
100 CALL SETFRQ (CHAN,KEY(24),1000);
```

would calculate and emit to the channel a frequency descriptor based on middle C and an FM ratio of 1.000.

PCH is a conversion function which accepts an octave-point pitch-class number, such as those used in MUSIC-11 and MUSIC4BF. This is a floating point value. The integer portion of the number indicates the octave in which the pitch lies, with octave 8 being the octave that begins with middle C. The two digits to the right of the decimal indicate pitch within the octave in semitones above C, from 00 for C and 11 for B. The subsequent digits represent additional tenths, hundredths, etc., of a semitone. Thus,

```
100 CALL SETFRQ (CHAN,PCH(8.0325),4000);
```

would calculate and emit to the channel a frequency descriptor based on one quarter of a semitone above D# in the middle octave with an FM ratio of 4.000.

In this last example using SETFRQ2,

```
100 CALL SETFRQ2 (CHAN,HERTZ(4400),HERTZ(5500));
```

a frequency descriptor based on a carrier frequency of 440.0 hertz and a modulator frequency of 550.0.

Special Tunings

The overall tuning base for the PITCH and KEY functions is the standard A-440. A new tuning may be established by a call to SET.TUNING.BASE. The argument specifying the new tuning base is a fixed point value representing a frequency in hertz multiplied by 10. Thus, the default is as if the statement

```
100 CALL SET.TUNING.BASE(4400);
```

had been executed. The statement

```
100 CALL SET.TUNING.BASE (8800);
```

would tune every pitch generated by the PITCH and KEY functions up an octave.

The octave ratio also affects the PITCH and KEY functions in the same way as the OCTAVE RATIO affects pitches in the Synclavier (R) II real-time system. A new octave ratio is established in an assignment statement which sets the value of the variable OCTAVE.RATIO equal to the octave ratio multiplied by 1000. Thus, the default assignment statement is

```
100 OCTAVE.RATIO = 1000;
```

whereas the following statement would establish an octave ratio of 2.000:

```
100 OCTAVE.RATIO = 2000;
```

The HERTZ and PCH functions are considered to be absolute, and will not be affected by changes in the tuning base or octave ratio.

The sample programs on your MAX User Diskette demonstrate many different MAX features. But each one also uses a different frequency specification function. As you have seen, DEMO1 uses the HERTZ function. DEMO2 uses the KEY function, DEMO3 uses the PCH function with data lists of floating point numbers placed in arrays, and DEMO4 uses the PITCH function.

Timing Considerations

The time required to compute the frequencies is minimal and can usually be ignored. If there is a great deal of parameter modification in your program, however, you may wish to break the SETFRQ procedure into its separate components. The CALCFREQ function can be called first to compute the digital code for the carrier and then for the modulator. It passed a frequency number from one of the four conversion functions or, in the case of the modulator, a number determined by the FM ratio. Using this frequency number, it computes three digital codes and places them in the global variables NOTEADD, NOTEINC, and NOTEDIV. The EMITEFRQ (for the carrier) and EMITIFRQ (for the modulator) procedures are passed the precomputed digital codes and emit them to the synthesizer. The EMIT step is effectively instantaneous, while the CALC step does take some time. Thus a program may be speeded up in a crucial place by precomputing and storing the NOTEADD, NOTEINC, and NOTEDIV values for each carrier and modulator frequency that will be needed. When each frequency is required, the EMIT procedures can transmit them rapidly.

SETTING THE VOLUME

The overall volume for a channel is set by a call to the SETVOL procedure. This procedure places an 8-bit number in the channel's volume register. The value ranges from 0 (for zero volume) to 255 (for maximum volume). The CALL statement in DEMO1 sets the volume for the allocated channel at the maximum:

```
11 CALL SETVOL(CHAN,255);
```

The overall level of various channels in a polyphonic composition are adjusted in calls to SETVOL at the beginning of the program in much the same way as a choral director assigns different numbers of singers to soprano, alto, tenor, and bass sections to achieve a balanced sound.

USING THE INTERPOLATORS

Each channel has two interpolators that assist in the generation of sounds with time-varying volumes and spectra. Let us say that you wished to generate a sound that started at zero and climbed to its peak level over a two millisecond period. To perform such an attack without the aid of an interpolator, the computer would have to change the volume register every 8 microseconds. Such a data rate is not possible when many channels are in simultaneous use.

The interpolator circuits in the synthesizer allow the generation of precise envelopes while reducing the computational load on the computer to a tolerable minimum. At the start of a change in volume or FM index, the program specifies a rate of increase (or decrease) and a limit, or new volume or harmonic index level. The interpolator increases or decreases the volume or FM index at the specified rate until the specified limit is reached.

The Volume Envelope

The rate for a channel's volume envelope interpolator is set by a call to the SETERATE procedure, where the arguments are the channel number and a time in milliseconds ranging from 0 to 9999. This time represents the time it will take to make a full scale (from 0 to maximum) change in volume. The limit for the volume envelope interpolator is set by a call to SETELIM, where the arguments are the channel number and a limit value between 0 and 255.

A channel's volume register and its volume envelope limit are two distinct values which combine to provide sixteen bits of dynamic range. SETVOL statements control the balance between different channels. SETELIM statements set the amount of the overall volume that will be reached at a given time.

You should always use the full 256-point range of the envelope interpolators when constructing attacks and decays. Doing so will eliminate any granularity associated with the discrete steps of the volume envelope interpolator.

Before any sound will be heard, the SETELIM procedure must be called and passed an above zero limit for the envelope interpolator. DEMO1 simply sets the envelope limit to maximum throughout each note:

```
12 CALL SETELIM(CHAN,255);
```

There is no call to SETERATE. The sound is instantly at maximum.

Volume envelopes, however, can be infinitely complex and completely arbitrary.

DEMO2 - A SIMPLE ENVELOPE

Now recall and list program DEMO2 on the MAX user diskette. Here the SETERATE and SETELIM procedures are used to create a simple two-segment event with an attack and a decay. For each segment of the sound, the rate and limit are set, followed by a WAIT while the segment occurs. Note that since the decay has a limit of zero, a call to ZEROSYN is not needed here to shut off the synthesizer channel.

You will also note the call to WAIT after each segment is begun. The WAIT procedure stops all processing except for the interpolators for the specified time.

DEMO3 - TWO CHANNELS

Now examine the DEMO3 program. Here two channels are allocated, one for alto (CHAN1) and one for bass (CHAN2). The two SETVOL statements

```
15   CALL SETVOL(CHAN1,255);
16   CALL SETVOL(CHAN2,180);
```

set the alto channel at full volume and the bass at about 70 percent loudness. But, since the envelope limits and rates are the same for both channels, they will both reach peak at the same time. Note that because there is no decay to zero in the envelopes, a ZEROSYN statement is required at the end to shut down the synthesizer channels.

Time-Varying Spectra

The SETIRATE and SETILIM procedures are used in the same way to change the FM index or depth of modulation. (For explanation of index, see John Chowning, "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation", J. of Audio Engineering Society, Vol. 21, No. 7, 1973, pp. 526-534.)

The SETILIM values correspond to the Synclavier (R) II HARMONIC ENVELOPE PEAK and SUSTAIN values which have a range of 0 to 1000. In MAX, however, this limit value is coded in the range from 0 to 255, along with a multiplier called a <u>shift count</u>. When the shift count is zero, the index of <u>modulation</u> limit will be as indicated in the SETILIM statement. When the shift count is one, the index limit will be twice the value indicated in the SETILIM statement. Shift counts of two and three will multiply the index limit by four and eight respectively. The procedures SETISHC and EMITISHC are used to load shift counts into the synthesizer. See the MAX Reference Manual for more details.

Combining and Dividing Procedures

It is possible to combine some of these interpolator procedures. The procedure SETE first calls SETERATE and then SETELIM, a frequently used sequence of calls. The call to SETE passes the channel number, the time (for the rate), and then the limit. The order of these arguments is easy to remember. The new rate must be set before the new limit, so that the interpolator does not start moving toward the limit at the wrong rate. The SETI procedure combines SETIRATE and SETILIM in the same way.

It is also possible to break SETERATE and SETIRATE down into calls to CALCRATE, which computes the numbers and stores them in the global variables INTADD and INTDIV, and calls to EMITERATE and EMITIRATE, which emit the numbers rapidly. You would use this method to save computation time in the middle of a composition.

DEMO4 - CHANGING SPECTRA

The program DEMO4 uses the SETIRATE and SETILIM procedures to create a changing index of modulation for the notes. You will also note the use of the combined SET procedures.

```
29   CALL SETE(CHAN,1000,0);
30   CALL SETI(CHAN,1500,0);
```

Line 29 sets a one-second decay to zero in the volume envelope; line 30 sets a one-and-a-half second decay to zero for the index limit.

CREATING A COMPLEX WAVEFORM

The waveform memories in the synthesizer hold complex, user-defined waveforms which are used to produce sounds with arbitrary harmonic content. There are 32 of these memories. Each one holds a 256-point waveform and can be used by any channel or shared among several channels. Each point of the waveform is represented by an eight-bit number having a value between 0 and 255.

In the MAXSYN initialization procedures, a sine wave is placed in one of the waveform memories. All channels are instructed to use this memory unless other instructions appear in the MAX program. Thus, the sounds produced in DEMO1, 2, 3, and 4, were based on a sine wave.

To produce a sound using a complex, i.e., nonsinusoidal, waveform, one of the waveform memories can be loaded with the values that will produce the desired periodic waveform.

There are several procedures involved. The first procedure, called CALCWAVE, calculates the waveform. The second, called EMITWAVE, loads the waveform into a waveform memory. The third, called SETWSEL, connects the channel with the waveform memory. There is a fourth more general procedure called SETWAVE that you may find very useful. Not surprisingly, this procedure combines the CALCWAVE and EMITWAVE procedures. Now we will describe these procedures in detail.

CALCWAVE computes the digital waveform that corresponds to a specified harmonic spectrum. It is passed an array specifying harmonic coefficients. These are fixed point values in the range from 0 to 1000 indicating the relative strengths of the various harmonics. (These numbers correspond to the DIGITAL TONE GENERATOR settings of 0.0 to 100.0 in the Synclavier (R) II real-time system.) The zeroeth element of the passed array indicates the number of harmonic coefficients. Using these coefficients, the computer calculates the waveform and places the numbers in a 256-point array called WAVEBUF. For example, the statement

```
100 CALL CALCWAVE(4,1000,500,333,250);
```

would compute a complex waveform with the fundamental at full volume, the octave at half volume, the fifth at one-third volume and the fourth at one-fourth volume.

CALCWAVE is useful for creating waveforms with entirely harmonic components. However, you may also set up arbitrary waveforms, by filling any 256-point array with specified values between 0 and 255.

The second procedure EMITWAVE loads the waveform into the waveform memory. It is passed a waveform memory number and the waveform array. The statement

```
120 CALL EMITWAVE(0,WAVEBUF);
```

places the WAVEBUF array in waveform memory 0 in the synthesizer.

The calculation of the settings for the waveform memory is the most time-consuming activity associated with the Digital Synthesizer. It can take as much as a tenth of a second, depending on the number of coefficients. For this reason, it is standard practice to calculate and load all of the waveform memories needed for a piece at the beginning of the program. If more than 32 different waveforms are needed, they may be calculated in advance and stored in any fixed point array. The actual loading of the waveform memory, with EMITWAVE, takes only several milliseconds.

The most complete waveform memory procedure is SETWAVE because it combines CALCWAVE and EMITWAVE. This procedure is passed a list of harmonic coefficients (like those used for CALCWAVE). It calculates the waveform and looks for a waveform memory in which to store it. The advantage of SETWAVE is that it checks to see if the specified waveform is already stored in any of the waveform memories before it performs the calculation. Thus, it prevents duplication of waveforms. Its disadvantage is that it may take time if a calculation is necessary. Therefore, it should be called at the beginning of a program, not repeatedly throughout a program. SETWAVE returns the number of the waveform memory used or -1 if none are free. It also keeps a count of the number of current uses of each waveform memory.

Correspondingly, the procedure FREEWAVE may be called to indicate that one usage of a memory has been completed. FREEWAVE is passed the waveform memory number. (Since memories can be shared, a memory is not considered available for a new waveform until all current usages of it have been terminated.)

The final procedure, SETWSEL, is used to select the desired waveform memory for a channel. It is passed a channel number and a waveform memory number.

DEMO5 and DEMO6 - USING COMPLEX WAVEFORMS

The programs DEMO5 and DEMO6 use the waveform memory procedures along with many of the procedures mentioned in previous sections. Compare the DEMO5 program and its sound to DEMO2. The two programs are similar except in the waveform used. In DEMO5, the complex waveform is calculated and loaded into a waveform memory and the memory is linked to the allocated channel <u>before</u> the composition begins.

DEMO6 shows how the SETWSEL procedure may be used to change the waveform used, by alternating the memory pointer between two precomputed waveforms. The waveform memory is changed at the start of each note, depending on whether the note number is even or odd, as below:

```
19   IF I MOD 2=0 THEN CALL SETWSEL(CHAN,WAVE1);
20   ELSE              CALL SETWSEL(CHAN,WAVE2);
```

SOME UTILITY FUNCTIONS

The procedures RND, WAIT, and ZEROSYN have been called in demos that you have already seen, but here they are described completely, along with a few others.

The RND function is a simple uniform random number generator. It produces an integer output in the range between its minimum value (the first argument) and one less than its maximum value (the second argument). Thus RND(0,6) would produce integers between 0 and 5. RND will always produce the same sequence of outputs during each run of a program.

The WAIT procedure, which has been used in every sample program so far, provides a simple delay for a specified number of milliseconds. No processing activities will occur during this delay, although the interpolators will continue to operate. The time values passed to WAIT will be reduced to the corresponding number of ticks of the system clock. The default clock rate is set at 200 hertz (5 milliseconds per tick), as is standard in all Synclavier (R) II systems. Thus, times should be in multiples of 5 milliseconds.

The CLEANUP procedure is used to silence a particular channel. It is passed a channel number as in

```
100 CALL CLEANUP (CHAN);
```

The ZEROSYN procedure repeatedly calls CLEANUP until all channels are silenced. Thus, you may use these two procedures to selectively or completely reset the synthesizer channels to zero.

Note that there is a call to ZEROSYN among the initialization procedures of the MAXSYN section so that all synthesizer channels are always set at zero prior to the execution of any MAXSYN program. This assures that results will be repeated each time a program is run.

The ALLOCATEX procedure is an extended version of the ALLOCATE procedure, and is useful in custom systems in which the channels have been wired to produce stereo or quad output placement. ALLOCATEX is passed a list of channels from which to select a channel. The list is an array in which the zeroeth element is the number of channels in the list. The channel numbers are the even numbers as described above.

As an example, suppose that channels 0, 2, 4, and 6 were connected to the left output, and channels 8, 10, 12, and 14 were connected to the right output. Then we could allocate a stereo pair as follows:

```
100 DCL LEFT.CHANNELS  DATA (4,0,2,4,6);
110 DCL RIGHT.CHANNELS DATA (4,8,10,12,14);
120 DCL (CHANL,CHANR) FIXED;
130
140 CHANL=ALLOCATEX(LEFT.CHANNELS);  /* GET A LEFT CHAN */
150 CHANR=ALLOCATEX(RIGHT.CHANNELS); /* AND A RIGHT CHAN */
    ...
    ...                              /* USE THE PAIR */
    ...
190 CALL FREECHAN(CHANL);            /* GIVE THEM BACK */
200 CALL FREECHAN(CHANR);
```

ALLOCATEX uses the same channel usage table as does ALLOCATE, so that both procedures may be used in the same program, in conjunction with FREECHAN.

DEMO7 - POLYPHONIC SOUNDS

DEMO7 shows how several channels can be active in overlapping stages to create polyphonic sounds. In this program, each pass through the loop allocates a new channel, and starts a decaying envelope event. Each new pass begins before the prior event has finished. In Chapter 6, you will learn how the MAXTASK procedures can create parallel processes to produce more sophisticated overlapping.

5. MAXIO

The MAXIO section enables the user to enter into MAX programs data from the buttons, control knob and keys of the Synclavier (R) II keyboard unit, as well as from the pedals or other devices. It also sends output to the digital display window and lights the buttons.

With the insertion of the MAXIO section, as well as MAXSYN, a simple program might wait for a key on the keyboard to be pressed, and then initiate an event using the pitch that corresponds to that key. Input from the pedal could determine the volume of the event. And the position of the knob could set a value such as FM index in a compositional algorithm.

USING SCANDATA

The MAXIO procedure SCANDATA is the main communications link to and from the peripheral devices. This procedure samples all of the input devices at the time of each call and places data in a number of global variables. You can use these variables for any purpose.

For synchronization purposes, the call to SCANDATA should almost always be placed within a loop that contains a WAIT call. This enables you to set up a determinable scanning rate, as you will see in the sample programs.

For the sake of consistency, all variables which contain the current position of a performance input device have names ending in .POS. All variables which contain the neutral position of those devices which have a neutral position have names ending in .BASE.

Input from the Pedals

The simplest function of SCANDATA is to write the current positions of the real-time effects and volume pedals into the variables RTEPEDAL.POS and VOLPEDAL.POS. The positions are coded so that a zero is written when the pedal is all the way up (or plugged in but turned off), and 225 is written when the pedal is all the way down (or not plugged in at all).

You might like to enter and run the following simple program. Connect the pedals to the jacks labeled OVERALL VOLUME and REAL TIME EFFECTS on the back of the keyboard unit. Then, as the program runs, push the pedals up and down.

```
100 INSERT 'MAXSYN';
110 INSERT 'MAXIO';
120 DCL I FIXED; /* LOOP INDEX */
130 DO I = 1 TO 20; /* DO 20 SCANS */
140    CALL SCANDATA;
150    PRINT 'THE VOLUME PEDAL POSITION IS ',VOLPEDAL.POS;
160    PRINT 'THE RTE    PEDAL POSITION IS ',RTEPEDAL.POS;
170    CALL WAIT(500); /* WAIT A HALF SECOND */
170 END;
```

This program would produce twenty scans, each half a second apart. Each scan would read the current positions of the two pedals, place the values in the variables VOLPEDAL.POS and RTEPEDAL.POS, and then display the new values on the terminal screen. Note that the computer time required for the SCANDATA procedure and the PRINT statement is insignificant compared to the length of the WAIT. Thus, it is the length of the WAIT that determines the timing.

Input from the Control Knob

The SCANDATA procedure also accepts input from the control knob and places it in three variables based on the current position of the control knob. In the first variable, KNOB.POS, is placed the current position of the control knob. The range is from around 100 (when the knob is all the way to the left) to 160 (when the knob is all the way to the right).

In the second variable, KNOB.BASE, is placed the neutral, or centered, position of the knob, typically 130. You can produce changing values for any parameter of a sound with the knob if you subtract KNOB.BASE from changing KNOB.POS values (KNOB.POS-KNOB.BASE). The result will be -30 when the knob is all the way left and +30 when the knob all the way right.

If you have used the control knob to change a parameter in the Synclavier (R) II real-time system, you know that it produces a smooth and gradual change in any selected parameters. This is the result of a filtering and smoothing function performed on KNOB.POS values. The same function is now available for your own use. SCANDATA compares the changing values in KNOB.POS with KNOB.BASE for you, filters the result, and stores the smoothed value in the third variable, KNOB.CHANGE.

To use this function, simply add KNOB.CHANGE to the variable to be changed. For example, the following statement would smooth the new ratio setting.

```
50 RATIO=RATIO+KNOB.CHANGE;
```

When using KNOB.CHANGE, it is recommended that you use it in a loop which calls SCANDATA at 200 hertz, which means including a WAIT of 5 milliseconds. If not, the knob will respond more slowly than in the real-time system. It is also helpful to display each new value (using the DISPLAY procedure described below).

KNOB.CHANGE can be added to several variables after each call to SCANDATA. Or, program logic can be devised to indicate which of the several variables is to be changed. (That, of course, is how the Synclavier (R) II real-time system works!!).

Input from the Buttons

SCANDATA also reads the control panel to determine if any button has been pressed. It sets up an array called PANSW, where each of eight values contains a 16-bit number indicating which buttons are currently pushed in each of the eight button panels on the control panel. The eight panels are numbered as follows:

0	2	4	6
1	3	5	7

Thus PANSW(0) holds the 16-bit word for the ENVELOPE buttons, PANSW(3) holds the word for the TRACKS buttons, and PANSW(6) holds the word for the buttons labeled TIMBRE BANK and TIMBRE ENTRY.

Each bit of the 16-bit word represents one button. The buttons in each panel are numbered in the same order as the TRACKS buttons, that is, from upper left to lower right. Thus, the least significant bit represents the upper leftmost button (button 1), and the most significant bit represents the lower right button (button 16). A "1" indicates the button is pressed and a "0" indicates the button is unpressed. When a button is pressed, it will automatically be lit by this procedure.

Writing to the Buttons

To write to the buttons, i.e., turn one or more "on" without pressing it, you set up the array DISPLAYSW in much the same form as for PANSW. Thus, setting DISPLAYSW(1) to "002000" (octal) would light button 7 under DIGITAL TONE GENERATOR and setting DISPLAYSW(2) to "000005" would light both START and RECORD buttons under RECORDER RECALL. You can also use whole integers instead of octal numbers. See Demo14 and DEMO15 where the MASK data list is used to light, and unlight, various buttons. Set up the DISPLAYSW array before the call to SCANDATA.

DEMO8 -- A RANDOM WALK

Now for a simple working example. Note that both MAXSYN and MAXIO are inserted at the start of DEMO8. In this program, the main loop will continue until the user presses a button in panel 0 (the ENVELOPE buttons).

```
19 DO WHILE (PANSW(0)=0);
```

Examine the second WAIT statement within that loop. This WAIT determines the speed of the program loop and its length is dependent upon the variable RTEPEDAL.POS.

```
24 CALL WAIT(SHL(RTEPEDAL.POS,1)+50);
```

The SHL function multiplies the pedal position by two; 50 is added so that the loop never comes to a complete halt. When the pedal is down, the WAIT will be long, and when the pedal is up, the WAIT will be short.

The loop makes a random step up or down the keyboard by incrementing or decrementing the variable NOTE and using that variable in a KEY frequency function. The variable THRESH is determined by the direction in which the knob is turned. If the knob is turned to the left, THRESH will be less than 50; if the knob is turned to the right, THRESH will be greater than 50. A random number from 0 to 100 is produced by the RND function. If this number is less than THRESH, the walk is biased in the upscale direction. If this number is more than THRESH, the walk is biased in the downscale direction. And if THRESH is 50 (when the knob is at center), the probabilities will be just about equal in either direction. This is called a "random walk". We begin our random walk at key 24, which is middle C.

Try running the program, but first make sure that the pedal is plugged into the jack labeled REAL TIME EFFECTS on the back of the keyboard unit, that it is turned on, and that it is pushed all the way down. Once the program is going, push the pedal up and the walk will speed up. Turn the knob to change the direction of the walk. If the pitch nears either end of the keyboard, the program will start the walk over again at middle C. Try this out.

To stop the program, push any button in the ENVELOPE panel.

USING SCAN.KEYBOARD AND SCAN.RELEASE

The SCAN.KEYBOARD procedure may be called after SCANDATA to check if any new keys have been pressed on the keyboard. It returns the number of new keys pressed since the last call to SCAN.KEYBOARD as well as a list of the key numbers of the new keys. This procedure uses the same coding as that used in the KEY frequency function.

Similarly, SCAN.RELEASE may be called after SCANDATA, and either before or after SCAN.KEYBOARD, to test for any keys that have been released since the last call. It returns the number of new releases, and a list of the key numbers of the released keys.

Both SCAN.KEYBOARD and SCAN.RELEASE are passed an array in which to return the new key numbers and a number which gives the size of the array. For example, the following code sets up a 4-note keyboard scan,

```
100 DCL  NEWKEYS(3)  FIXED;
110 DCL  RELKEYS(3)  FIXED;
120 DCL  (NUMNEW,NUMREL)  FIXED;
130
140 CALL SCANDATA;
150 NUMNEW = SCAN.KEYBOARD(NEWKEYS,3);
160 NUMREL = SCAN.RELEASE(RELKEYS,3);
```

If there are, in fact, four new keys pressed, NUMNEW will be 4, and the key numbers will be returned in NEWKEYS(0) to NEWKEYS(3). They will be arranged from lowest pitch to highest. If there is only one new key, NUMNEW will be 1, and the key number will be in NEWKEYS(0). Obviously, if there are no new keys, NUMNEW will be zero.

The same interpretation holds for the NUMREL and RELKEYS outputs of SCAN.RELEASE.

DEMO9 - A MONOPHONIC SYNTHESIZER

DEMO9 uses three procedures, SCANDATA, SCAN.KEYBOARD, and KEY to create a monophonic synthesizer.

DEMO9 looks for just one key at a time, and produces only one event at a time. The key number of the key that is pressed is used to set the frequency for the event. For a polyphonic synthesizer, you could ask for several keys, and initiate separate events for each one. The MAXTASK multitask procedures described in the next chapter can be used in the scheduling of the multiple simultaneous events.

WRITING TO THE DIGITAL DISPLAY WINDOW

You may use the DISPLAY procedure to display a value in the digital display window on the keyboard unit. Three arguments are passed to this procedure: the value to be displayed, the decimal point position, and a units light code. The decimal position indicates the number of digits to the right of the decimal point from 4 for .0000 to 0 for 0000. The units code indicates which light to the right of the display is to be lit: 1 indicates the light labeled MILLISECONDS, 2 indicates the light labeled HERTZ, 4 indicates the light labeled ARBITRARY, and 8 indicates the light labeled DECIBELS.

The call to DISPLAY sets up the data. At the next call to SCANDATA the correct value will appear in the display window. For example, the statement below, followed by a call to SCANDATA, would display 100.0 in the window and turn on the ARBITRARY light:

```
100 CALL DISPLAY(1000,1,4);
```

The DISPLAY.ERROR procedure can be called to display error messages. The argument is a number from 0 to 9 which will select an error message from "Err0" through "Err9". Again, the error message will appear after the next call to SCANDATA. For example, the following statement would display Err7.

```
100 CALL DISPLAY(7);
```

Of course, the program will designate the situation which will result in this call.

DEMO10 - ANOTHER MONOPHONIC SYNTHESIZER

DEMO10 is a monophonic synthesizer similar to DEMO9 except that it provides the user with means to change the overtone content of the sound during performance with the pedal and the control knob.

When the user turns the knob, a variable called RATIO is changed by KNOB.CHANGE (the filtered result of KNOB.POS-KNOB.BASE).

```
22 RATIO=RATIO+KNOB.CHANGE;
```

The changing value of RATIO is used in the call to SETFRQ as the argument specifying the FM ratio.

```
27 CALL SETFRQ(CHAN,KEY(KEYLIST(0)),RATIO);
```

The value of RATIO is also displayed each time through the loop.

```
31 CALL DISPLAY(RATIO,3,4);
```

The index limit is set by the position of the pedal.

```
28 CALL SETILIM(CHAN,SHR(RTEPEDAL.POS,1));
```

Thus, the user of the program will use the knob to change the FM ratio, the pedal to control the depth of modulation, and the keyboard to instigate new notes.

INPUT FROM THE RIBBON CONTROLLER

If a ribbon controller is installed on your keyboard control unit, the variables RIB.POS, RIB.BASE, and RIB.ACTIVE will be set by a call to SCANDATA.

First of all, when the ribbon is activated (is being depressed), the value of RIB.ACTIVE will be 1. When it is inactive, the value of RIB.ACTIVE will be 0.
If RIB.ACTIVE is 1, RIB.BASE will be set at the base value indicated by the place on the ribbon where it was first pressed.

Finally, RIB.POS will be set at the place currently being pressed. The value associated with RIB.BASE may be subtracted from that associated with RIB.POS to produce changing values.

These three variables have no intrinsic meaning or use. You define their usage in your MAX program.

PITCH BEND INPUT

Input from the jack labeled PITCH BEND on the back of the keyboard control unit is returned in the variable PBI.POS. The neutral value in PBI.BASE is the value of the input at initialization time.

Note that this "PITCH BEND" input does not intrinsically control pitch. You must assign it a purpose in your MAX program, just as with the ribbon controller input.

SWITCH INPUTS AND FILTER CONTROL OUTPUTS

There are a number of variables associated with the switch inputs and control voltage outputs on the back of the Synclavier (R) II keyboard unit. Once again, it is emphasized that these variables have no intrinsic meaning and that their names only associate them with particular jacks. In a MAX program, the function is assigned by the user.

Inputs

During a call to SCANDATA, input is sensed from the six SWITCH jacks on the back of the keyboard control unit and values are placed in six global variables. The variable names are HOLD.SWITCH (from the HOLD input jack), REP.SWITCH (from the REPEAT input jack), GLIDE.SWITCH (from the PORTAMENTO input jack), SUST.SWITCH (from the SUSTAIN input jack), ARP.SWITCH (from the ARPEGGIATE input jack), and PUNCH.SWITCH (from the PUNCH IN/OUT input jack). The values are zero (or false) if nothing is connected or if the switch is not closed, and one (or true) if a switch is connected and closed.

Outputs

There are eight output digital-to-analog channels which are used to output control voltages. These eight-bit DACs are on the back panel. In MAX, you can use these control voltages for any purpose, such as controlling filters, analog synthesizers, or other equipment, by writing a digital value or values to one or more of the eight output variables.

A value of zero will output zero volts, while 255 (the maximum value) will produce 10 volts.

The output variable names are LPFILT.OUT (for the LOW PASS output jack), HPFILT.OUT (for the HIGH PASS output jack), BPFILT.OUT (for the BANDPASS output jack), BANDWIDTH.OUT (for the BANDWIDTH output jack), CV.OUT (for the KEYBOARD CV output jack), GATE.OUT (for the KEYBOARD GATE output jack), TRIGGER.OUT (for the KEYBOARD TRIGGER output jack), and RIBBON.OUT (for the RIBBON output jack). The voltages will be outputted from the appropriate jack(s) on the back of the keyboard control unit after the next call to SCANDATA.

One application for these inputs and outputs might be to produce an interactive audio/visual environment. In this scheme, input would come from the movements of an audience across light beams with photocells latched into the switch inputs. The output voltages would control sets of lights or trigger slide projectors. The changing audience input, along with any other user input, could thus control both the sound coming from the synthesizer and the lighting.

6. MAXTASK

The MAXTASK section of the MAX library allows you to create several "simultaneous" events where the processor seems to be performing several procedures, or tasks, at once. In reality, the processor is "time-sharing" between the tasks.

With no single program and single sequence of instructions, execution can be very difficult to follow. There are also a number of ways in which subtle errors or major conflicts can be introduced. Therefore, to prevent errors, read these instructions carefully.

WHAT IS A TASK?

A task is a procedure which has some special features. First, it cannot, as a rule, be passed any values; it generally uses global values instead. Second, it is invoked by a START statement rather than a CALL statement. This statement takes the form

<ln> START <taskname> TASK;

When a task is started, it proceeds to execute on its own. It is terminated by a TERMINATE statement rather than a RETURN statement. This statement takes the form

<ln> TERMINATE TASK;

Fourth, a task can terminate other tasks by a KILL statement. This statement takes the form

<ln> KILL <taskname> TASK;

THE MAIN TASK

The multiple tasks are performed within an overall MAIN task. After the INSERT statements, you include the procedure name statement

150 MAIN: PROCEDURE;

At the end of the program you include a procedure END statement

250 END MAIN;

In the MAXTASK initialization code, there is a START MAIN

TASK statement, which invokes your MAIN task. All your other tasks are invoked from within this MAIN task.

THE TIME SHARING ALGORITHM

The algorithm used to divide the processor's time between the multiple tasks is as follows: The processor executes one task until it goes into an idle period. Then it starts to execute the oldest task that is not in an idle period.

Each task must go into idle periods frequently for the process to work. There is no direct way that a task can be pre-empted. An idle period is entered by a call to SUSPEND in which the argument specifies a length of time in milliseconds. This is similar to a call to WAIT, except that SUSPEND creates a delay only in the one task while WAIT stops all processing in the entire program. For synchronization purposes, you should not call SUSPEND with a time value that is less than one clock tick (five milliseconds).

DEMO11 - AN IDIOT'S DELIGHT

In this program, the procedures MAIN, IDIOT1, and IDIOT2 are the tasks. The MAIN task is invoked by the MAXTASK initialization code. Tasks IDIOT1 and IDIOT2 are invoked by the START statements in lines 26 and 27. Both call SUSPEND repeatedly. The SUSPEND in IDIOT1 creates idle periods of one second, and the SUSPEND in IDIOT2 creates idle periods of three seconds.

Run DEMO11 and watch the printout on your terminal screen. The nature of a multiple task program will be simply and clearly demonstrated.

Both of the tasks will be running, and they will print out their messages at the different intervals specified. IDIOT1's message will be displayed every second, and IDIOT2's message will be displayed every three seconds. Since both of the tasks are in infinite loops, you will have to press the LOAD button on the computer to kill this program.

In the example programs in the MAXIO chapter, you saw the use of SCANDATA combined with repeated five-millisecond WAITs. The WAITs keep things correctly synchronized. By using SUSPEND instead of WAIT, you can synchronize several simultaneous events, as well as satisfy the requirement of frequent idle periods.

It is easy to write a task that waits for an external event to happen but does not tie up the system while waiting. First of all, determine a realistic scanning rate required by the situation. Then write a loop that checks for the input, and iteratively calls SUSPEND until the necessary input conditions are satisfied. Examine the following:

THE WRONG WAY to wait for a pulse

```
100 DO WHILE (INPUT.SIGNAL<100); /* WAIT FOR PULSE */
110 END;                         /* CANNOT USE THIS STRUCTURE
                                    IN INTERRUPT ENV. */
```

THE RIGHT WAY to wait for a pulse

```
100 DO WHILE (INPUT.SIGNAL<100); /* WAIT FOR PULSE */
110    CALL SUSPEND(50);         /* GIVE OTHER TASKS A CHANCE TO RUN */
120 END;
```

Each task may consist of any legal MAX or XPL statements, including START and CALL statements. You should not call WAIT in a task, because it monopolizes the processor for the specified length of time and does not allow other tasks to proceed. (Of course, there may be a situation where this is the desired effect.) You should also avoid the use of INPUT and LINPUT statements in a task, since these statements put the whole processor into a idle state until the input is made. Similarly, PRINT statements dedicate large amounts of time for typing the characters, and can skew the execution scheduler.

DEMO12 - SIMPLE POLYRHYTHMS

Now examine and run DEMO12. It is similar in structure to DEMO11, with two tasks within the MAIN task. These tasks, however, include calls to MAXSYN procedures which play a set of pitches randomly chosen from a specified list. The lower melody line (LINE1) maintains a constant tempo of one pitch per second, while the other (LINE2) changes tempo several times.

MAXTASK ERRORS

As you may have seen when you ran DEMO12, the final result was MAXTASK Error 4, which indicates that all tasks are terminated. Error 4 is often a normal termination result.

The complete set of MAXTASK errors is as follows:

Error 1: Too many tasks active. The default limit is 18, but this may be expanded by increasing the value in the NUM.TASKS declaration in the MAXTASK code.

The default limit of 18 tasks has been chosen so that there may be a task for each of 16 voices, plus a main task, and one extra task.

Error 2: Error in starting of task. The START statement has an incorrect format.

Error 3: Stack length exceeded. Correct this problem by increasing the value in the LEN.STACK declaration in the MAXTASK1 source code. Refer to the section of the Scientific XPL/4 Reference Manual under PDL for more information on push down stack requirements.

Error 4: All tasks terminated. This error is generated when all tasks have been terminated. This may be a normal result in some instances.

Error 5: Error in killing of task. The KILL statement has an incorrect format.

AUTOMATIC LOCAL VARIABLES

A local variable is one that is known only within its own procedure or task. In DEMO11 and DEMO12, local variables are declared within the tasks. This method works fine when there is only one copy of a task active at a time. If, however, there is more than one copy active, then the local variables must be made distinct for each of the copies, or they will modify each other. To accomplish this, the local variables are declared AUTOMATIC.

The term AUTOMATIC comes from PL/1, and means that these variables are AUTOMATICally allocated into new locations when each new copy of the task is invoked. An AUTOMATIC type declaration takes the form

<ln> DCL <variable> AUTOMATIC<n>;

where <n> stands for a number from 1 to 6, representing each of the six variables. Thus, you may use up to six AUTOMATIC variables per task. Each variable is a fixed point scalar. This limit of six is not too severe, as only one or two channel numbers, and one or two pitch codes need be declared AUTOMATIC in most cases. Event tasks will also generally share a large number of global variables.

DEMO13 - ONE TASK, MANY COPIES

Examine the program DEMO13. This program has a task called DO.AN.EVENT, which performs a single event. The MAIN task creates several copies of DO.AN.EVENT in order to play several events at the same time. Thus the variable CHAN must be declared AUTOMATIC,

19 DCL CHAN AUTOMATIC1;

to make it local to each copy of the task.

Push the RTE pedal all the way down and run DEMO13. It will take about 55 seconds to compile the program and set it up for execution, since all three library sections are included.

This program plays complex overlapping events. The pitch for each event is randomly selected from a list of pitches. The RND function is also used in the selection of the waveform, FM ratio, volume and harmonic envelopes, and suspend times for each event.

The pedal position controls the speed of the events. When the pedal is down, the events are long and infrequent. As the pedal is raised, the events become shorter and more frequent.

To stop the program, hold down any button in the ENVELOPE panel.

PASSING VALUES TO A TASK

It is sometimes necessary to send values to a task. As stated above, an argument list cannot be used for this purpose. However, you can use the SET clause in the START statement to set initial values for the automatic variables of a task. The statement takes the form

```
<ln> START <taskname> SET(<v1>,<v2>,<v3>,<v4>,<v5>,<v6>) TASK;
```

where the expressions <v1> through <v6> stand for values to be loaded into the variables AUTOMATIC1 through AUTOMATIC6 for each copy of the task. You must specify six values even if the invoked task does not have six automatic variables.

DEMO14 - A POLYPHONIC SYNTHESIZER

DEMO14 is a powerful demonstration of the MAX potential, because it shows how to construct a polyphonic synthesizer producing several simultaneous events with complex multi-segment envelopes. It also shows how to use the SET clause.

The loop in the MAIN task has a real-time scanning rate of 5 milliseconds:

```
49     CALL SUSPEND(5);
```

It scans the keyboard and checks for up to ten pressed keys. If any of the ten keys is in the bottom octave of the keyboard, its number will be used to select between preset timbres, mimicking the classic Hammond B-3.

The key number determines the timbre number, and a number from 1 to 12 is then displayed in the digital display window.

```
57      IF KEYLIST (J)<12 THEN DO;
58         TIMBRE#=KEYLIST(J)
59         CALL DISPLAY(TIMBRE#+1,0,0);
```

The variable TIMBRE# is then used to select between a number of stored complex envelopes, as well as a set of ratios. The complex multi-segment envelopes are created by stepping through parts of a data list which specifies a rate and a limit for each of six independent segments. (We selected six segments for this example. Any number could have been chosen.)

TIMBRE# values 3, 6, 9, and 12 produce a complex envelope with a "re-iterate" sound, or echo delay, when you play a quick run on the keyboard. TIMBRE# values 2, 5, 8, and 11 produce a long attack followed by a sudden change in the FM index. TIMBRE# values 1, 4, 7, and 10 produce a volume envelope with a second (smaller) peak.

The pressing of any key above the bottom octave will start a new task which will be played with the current timbre. The new key number is passed to the DO.AN.EVENT task in a SET clause:

```
61  ELSE START DO.AN.EVENT SET(KEYLIST(J),0,0,0,0,0) TASK;
```

The value of KEYLIST (J) will be placed in variable AUTOMATIC1, which is named KEY# in the event task in line 26.

After an event task has started and a channel has been allocated for it, the program lights a button in the TRACKS button panel. When the event is finished and the channel has been freed, the program turns out the light. Thus, the TRACKS lights give a visual indication of the number of multiple task events in progress, as well as the number of channels in use.

An important detail to note is that due to the way AUTOMATIC variables are implemented, they cannot be used as indices of DO loops. Thus, the loop through the segments of the envelope is written as a direct IF-THEN-GOTO loop.

Terminate DEMO14 by pressing any button in the ENVELOPE button panel.

7. A COMPREHENSIVE EXAMPLE

DEMO15 provides a good complete demonstration of many of the features of MAX. The basic form of this program resembles that of DEMO13 and DEMO14: A loop in the MAIN task starts multiple copies of an event task.

The control knob, digital display window, and the buttons in the ENVELOPE panel are used to change timbre parameters, in a simplified version of the Synclavier (R) II real-time system. Only four buttons are used - VE ATTACK, VE FINAL DECAY, HE ATTACK, and HE PEAK. The scanning loop in the MAIN task watches for one of these buttons to be pressed, to select which value is to be changed. As the KNOB.CHANGE input from the control knob is added to the selected value, the result is displayed in the digital display window.

The pedal controls the rate at which copies of the event task are started, and the notes from the keyboard are placed in an eight-note first-in first-out queue.

The event task first selects a channel. Then it selects a pitch from the FIFO stack, to which it adds, or subtracts, an interval randomly chosen from a list of approximately consonant intervals. It then performs the event, using the current values for the parameters being changed by the knob, along with some random choices.

As in DEMO14, the TRACKS buttons are used to indicate the number of events active. The envelope rates and the SUSPEND delays have been chosen so that generally no more than twelve events are happening at once. (If you have an 8-voice system, only eight events will occur.) Since the default maximum number of tasks is eighteen, exceeding this limit will cause MAXTASK Error 1. Owners of 24-voice or 32-voice systems may wish to modify this default in the source code as described above under MAXTASK Error 1.

This example is provided as a baseline for experimentation with interactive compositional systems. For example, the FIFO queue of pitches is very interesting to experiment with. Play one key eight times, and the queue will contain only that pitch. The events will be played with random consonant pitches around it. Play an eight-note scale, and the next eight events will be based on eight separate pitches. Change the volume attack from 300 ms (the initial value) to 0 for percussive sounds. Change the harmonic peak. Alter the rate of event initiation by pushing the pedal up and down.

To stop the program, press the STOP button under RECORDER CONTROL. The events will decay to zero.

This example incorporates most of the features of MAX, but it still represents only a small portion of the capabilities of MAX. As you develop your own MAX system, you will discover a wide range of ways to interact with the synthesizers and a large number of applications.

APPENDIX ONE: SPL COMPATIBILITY

MAX is upwards compatible with SPL, the Sound Producing Language developed for the Synclavier I synthesizer (1977-79), which has been frequently used in academic environments for research and computer aided instruction, as well as for composition. MAX improves on SPL in several important ways.

First, MAXIO and MAXTASK are completely new. SPL provided only the procedures for controlling the synthesizer. Since there was no built-in capability for multiple tasks in SPL, overlapping or interrelated events were difficult, though not entirely impossible, to program.

In MAX, frequency specification has been made easier and more flexible. In SPL, the carrier and modulating frequencies were on entirely different channels. An integer frequency had to be directly specified for each. In MAX, the FM pairs are on the same channel and the frequency of the modulator can be specified either directly or by ratio. Also, there are the four new frequency conversion functions.

In SPL, the WAIT procedure was passed a time in centiseconds, while the RATE procedure accepted times in milliseconds. In MAX, all times are specified in milliseconds for consistency.

This manual completely supersedes the <u>SPL Reference Manual</u>.

DEMO1 -1-

```
/* MAX DEMO 1     26 JANUARY 1982 */

INSERT 'MAXSYN';                                /* LOAD THE MAX1 SECTION OF THE LIBRARY */

DCL I      FIXED;                               /* LOOP INDEX */
DCL CHAN   FIXED;                               /* CHANNEL NUMBER */
DCL FREQ   FIXED;                               /* PITCH IN HERTZ TIMES 10 */

CHAN=ALLOCATE;                                  /* GET A CHANNEL */

CALL SETVOL(CHAN,255);                          /* OPEN UP VOLUME */
CALL SETELIM(CHAN,255);                         /* SET ENVELOPE LIMIT TO MAXIMUM */

DO I=1 TO 25;                                   /* PLAY 25 RANDOM PITCHES */
   FREQ=RND(2200,8800);                         /* GET A RANDOM NUMBER */
   CALL SETFRQ(CHAN,HERTZ(FREQ),1000);          /* PUT THIS PITCH ON FIRST CHAN */
   CALL WAIT(500);                              /* WAIT HALF A SECOND */
END;
CALL ZEROSYN;                                   /* SHUT EVERYTHING DOWN */
```

DEMO2 -1-

```
/* MAX DEMO 2     30 JANUARY 1982 */

INSERT 'MAXSYN';                                /* GET MAX SYNTH ROUTINES */

DCL I        FIXED;                             /* LOOP INDEX */
DCL CHAN     FIXED;                             /* CHANNEL NUMBER */
DCL NOTE     FIXED;                             /* CLAVIER KEY NUMBER FOR NOTE */

CHAN=ALLOCATE;                                  /* ALLOCATE A CHANNEL */
CALL SETVOL(CHAN,255);                          /* OPEN THE VOLUME FOR THE CHANNEL */

DO I=1 TO 25;                                   /* PLAY 25 NOTES */
   NOTE=RND(12,48);                             /* GET A NUMBER BETWEEN 12 AND 48 */
   CALL SETFRQ(CHAN,KEY(NOTE),1000);
   CALL SETERATE(CHAN,50); CALL SETELIM(CHAN,255);    /* 50 MS ATTACK */
   CALL WAIT(50);                               /* WAIT FOR ATTACK */
   CALL SETERATE(CHAN,1000); CALL SETELIM(CHAN,0);    /* 1 SEC DECAY */
   CALL WAIT(1000);                             /* WAIT FOR DECAY */
END;
```

DEMO3 -1-

```
/* MAX DEMO 3     2 FEBRUARY 1982 */

INSERT 'MAXSYN';                                /* GET MAX SYNTH ROUTINES */

DCL I        FIXED;                             /* LOOP INDEX */
DCL CHAN1    FIXED;                             /* ALTO CHANNEL */
DCL CHAN2    FIXED;                             /* BASS CHANNEL */
DCL ALTO.NOTES FLOATING DATA
(8.00, 8.04, 8.09, 9.00, 8.09, 8.00, 8.04, 9.00, 8.00, 8.09, 9.00);
DCL BASS.NOTES FLOATING DATA
(6.07, 7.00, 6.04, 7.07, 6.04, 6.07, 7.00, 6.07, 6.07, 6.04, 6.07);

CHAN1=ALLOCATE;                                 /* ALLOCATE A CHANNEL FOR ALTO */
CHAN2=ALLOCATE;                                 /* ALLOCATE A CHANNEL FOR BASS */
CALL SETVOL(CHAN1,255);                         /* FULL VOLUME FOR ALTO'S */
CALL SETVOL(CHAN2,180);                         /* LOWER VOLUME FOR BASS  */

CALL SETERATE(CHAN1,2000);                      /* FADE IN OVER 2 SECONDS */
CALL SETERATE(CHAN2,2000);
CALL SETELIM(CHAN1,255);                        /* TO MAX ENVELOPE LIMIT */
CALL SETELIM(CHAN2,255);

DO I=0 TO 10;                                   /* LOOP OVER NOTES */
   CALL SETFRQ(CHAN1,PCH(ALTO.NOTES(I)),1000);  /* SET FOR THE I'TH NOTE */
   CALL SETFRQ(CHAN2,PCH(BASS.NOTES(I)),1000);
   CALL WAIT(500);                              /* PLAY FOR HALF A SECOND */
END;
CALL ZEROSYN;                                   /* SHUT DOWN */
```

DEMO4 -1-

```
/* MAX DEMO 4    30 JANUARY 1982 */

INSERT 'MAXSYN';                                      /* GET MAX SYNTH ROUTINES */

DCL I        FIXED;                                   /* LOOP INDEX */
DCL CHAN     FIXED;                                   /* CHANNEL NUMBER */
DCL FREQ.LIST(9) FIXED;                               /* FREQUENCIES */
DCL TIME.LIST(9) FIXED;                               /* DURATIONS */

/* CREATE THE NOTE LIST */
FREQ.LIST(1)=PITCH('E4');  TIME.LIST(1)=500;
FREQ.LIST(2)=PITCH('D#');  TIME.LIST(2)=500;
FREQ.LIST(3)=PITCH('E');   TIME.LIST(3)=500;
FREQ.LIST(4)=PITCH('D#');  TIME.LIST(4)=600;
FREQ.LIST(5)=PITCH('E');   TIME.LIST(5)=700;
FREQ.LIST(6)=PITCH('B3');  TIME.LIST(6)=800;
FREQ.LIST(7)=PITCH('D4');  TIME.LIST(7)=900;
FREQ.LIST(8)=PITCH('C');   TIME.LIST(8)=1000;
FREQ.LIST(9)=PITCH('A3');  TIME.LIST(9)=2000;

CHAN=ALLOCATE;                                        /* ALLOCATE A CHANNEL */
CALL SETVOL(CHAN,255);                                /* OPEN THE VOLUME FOR THE CHANNEL */

DO I=1 TO 9;                                          /* WE HAVE 9 NOTES TO PLAY */
   CALL SETFRQ(CHAN,FREQ.LIST(I),1000);               /* SET FOR THE I'TH NOTE */
   CALL SETERATE(CHAN,50);  CALL SETELIM(CHAN,255);   /* 50 MS ATTACK */
   CALL SETIRATE(CHAN,50);  CALL SETILIM(CHAN,30);    /* INDEX ENVELOPE */
   CALL WAIT(50);                                     /* WAIT FOR ATTACK */
   CALL SETE(CHAN,1000,0);                            /* 1 SEC DECAY */
   CALL SETI(CHAN,1500,0);                            /* INDEX ENVELOPE */
   CALL WAIT(TIME.LIST(I));                           /* WAIT FOR DECAY */
END;
```

DEMO5 -1-

```
/* MAX DEMO 5   6 FEBRUARY 1982 */

INSERT 'MAXSYN';                                      /* GET MAX SYNTH ROUTINES */

DCL I      FIXED;                                     /* LOOP INDEX */
DCL CHAN   FIXED;                                     /* CHANNEL NUMBER */
DCL WAVE   FIXED;                                     /* WAVE MEMORY NUMBER */
DCL HARMS DATA (4,1000,500,333,250);                  /* HARMONIC DATA */

WAVE=SETWAVE(HARMS);                                  /* GET A WAVE MEMORY */
CHAN=ALLOCATE;                                        /* ALLOCATE A CHANNEL */
CALL SETWSEL(CHAN,WAVE);                              /* LINK CHANNEL AND WAVE MEMORY */

CALL SETVOL(CHAN,255);                                /* OPEN THE VOLUME FOR THE CHANNEL *

DO I=1 TO 25;                                         /* PLAY 25 RANDOM NOTES */
   CALL SETFRQ(CHAN,KEY(RND(12,48)),1000);            /* SET THE FREQ */
   CALL SETE(CHAN,50,255);                            /* 50 MS ATTACK */
   CALL WAIT(50);                                     /* WAIT FOR ATTACK */
   CALL SETE(CHAN,1000,0);                            /* 1 SEC DECAY */
   CALL WAIT(1000);                                   /* WAIT FOR DECAY */
END;
```

DEMO6 -1-

```
/* MAX DEMO 6   6 FEBRUARY 1982 */

INSERT 'MAXSYN';                                      /* GET MAX SYNTH ROUTINES */

DCL I                 FIXED;                          /* LOOP INDEX */
DCL CHAN              FIXED;                          /* CHANNEL NUMBER */
DCL (WAVE1,WAVE2)     FIXED;                          /* WAVE MEMORY NUMBERS */
DCL HARMS1 DATA (4,1000,500,333,250);                 /* WAVEFORM1 */
DCL HARMS2 DATA (5,100,500,1000,0,500);               /* WAVEFORM2 */

WAVE1=SETWAVE(HARMS1);                                /* GET A WAVE MEMORY */
WAVE2=SETWAVE(HARMS2);                                /* AND ANOTHER */
CHAN=ALLOCATE;                                        /* ALLOCATE A CHANNEL */

CALL SETVOL(CHAN,255);                                /* OPEN THE VOLUME FOR THE CHANNEL */

DO I=24 TO 48;                                        /* PLAY 2 OCTAVES OF KEYS */
   CALL SETFRQ(CHAN,KEY(I),1000);                     /* SET THE FREQ */
   IF I MOD 2 =0 THEN CALL SETWSEL(CHAN,WAVE1);       /* PICK A WAVE MEM */
   ELSE               CALL SETWSEL(CHAN,WAVE2);
   CALL SETE(CHAN,50,255);                            /* 50 MS ATTACK */
   CALL WAIT(50);                                     /* WAIT FOR ATTACK */
   CALL SETE(CHAN,700,0);                             /* 0.7 SEC DECAY */
   CALL WAIT(700);                                    /* WAIT FOR DECAY */
END;
```

DEMO7 -1-

```
/* MAX DEMO 7    8 FEBRUARY 1982 */
/* OVERLAPPING NOTES BY USE OF 'ALLOCATE' AND 'FREECHAN' */

INSERT 'MAXSYN';                                        /* GET MAX SYNTH ROUTINES */

DCL I     FIXED;                                        /* LOOP INDEX */
DCL CHAN  FIXED;                                        /* CHANNEL NUMBER */
DCL NOTES FLOATING DATA
(8.00, 8.02, 8.04, 8.06, 8.08, 8.10, 9.00);             /* A SIMPLE SCALE */

DO I=0 TO 6;                                            /* LOOP OVER NOTES */
   CHAN=ALLOCATE;                                       /* GET A NEW CHANNEL */
   CALL SETVOL(CHAN,255);                               /* OPEN THE NEW CHANNEL */

   CALL SETFRQ(CHAN,PCH(NOTES(I)),4000);                /* SET FOR THE I'TH NOTE */
   CALL SETE(CHAN,20,255);                              /* 20 MS ATTACK */
   CALL SETI(CHAN,20,100);                              /* INDEX ENVELOPE */
   CALL WAIT(50);                                       /* WAIT FOR ATTACK */

   CALL SETE(CHAN,4000,0);                              /* 4 SEC DECAY */
   CALL SETI(CHAN,4000,0);                              /* INDEX ENVELOPE */
   CALL WAIT(600);                                      /* WAIT UNTIL NEXT NOTE, BUT LET THIS ONE CONTINUE */
END;
```

DEMO8 -1-

```
/* MAX DEMO 8        31 JANUARY 1982 */
/* EXAMPLE OF 'RANDOM WALK' COMPOSITION */

INSERT 'MAXSYN';                                        /* GET MAX SYNTH ROUTINES */
INSERT 'MAXIO';                                         /* GET CONTROL AND I/O FUNCTIONS */

DCL (WAVE1,CHAN) FIXED;                                 /* WAVE AND CHANNEL NUMBERS */
DCL NOTE         FIXED;                                 /* NOTE NUMBER */
DCL THRESH       FIXED;                                 /* DECISION THRESHOLD */
DCL HARM DATA (6,1000,500,333,250,200,167);             /* HARMONIC CONTENTS */

WAVE1=SETWAVE(HARM);                                    /* SET UP WAVEFORM */
CHAN=ALLOCATE; CALL SETVOL(CHAN,255);                   /* ALLOCATE CHANNEL AND SET VOLUME */
CALL EMITWSEL(CHAN,WAVE1);                              /* LINK CHANNEL TO WAVE MEM */

NOTE=24;                                                /* START THE RANDOM WALK ON MIDDLE C */
CALL SCANDATA;                                          /* DO INITIAL SCAN */

DO WHILE (PANSW(0)=0);                                  /* REPEAT UNTIL BUTTON IN PANEL 0 IS PUSHED */
   CALL SETFRQ(CHAN,KEY(NOTE),1000);                    /* CONVERT KEY NUM TO FREQ AND SET */
   CALL SETE(CHAN,0,255);                               /* FASTEST ATTACK */
   CALL WAIT(5);                                        /* WAIT FOR ATTACK */
   CALL SETE(CHAN,500,0);                               /* HALF SEC DECAY */
   CALL WAIT(SHL(RTEPEDAL.POS,1)+50);                   /* USE PEDAL POS TO FIND WAIT TIME */

   THRESH = 50+(KNOB.POS-KNOB.BASE);                    /* COMPUTE DECISION THRESHOLD */
   IF RND(1,100)<THRESH THEN NOTE=NOTE+1;               /* IF NUMBER < THRESH GO UP */
   ELSE                      NOTE=NOTE-1;               /* ELSE GO DOWN */
   IF NOTE<06 THEN NOTE=24;                             /* LIMIT RANGE OF WALK */
   IF NOTE>55 THEN NOTE=24;                             /* LIMIT TOP END */

   CALL SCANDATA;                                       /* SCAN INPUTS AGAIN */
END;
```

DEMO9 -1-

```
/* MAX DEMO 9    31 JANUARY 1982 */
/* EXAMPLE OF MONOPHONIC SYNTHESIZER EMULATION */

INSERT 'MAXSYN';                                        /* GET MAX SYNTH ROUTINES */
INSERT 'MAXIO';                                         /* GET CONTROL AND I/O FUNCTIONS */

DCL I            FIXED;                                 /* NUMBER OF NEW KEYS */
DCL (WAVE1,CHAN) FIXED;
DCL HARM DATA (6,1000,500,333,250,200,167);
DCL KEYLIST(0)   FIXED;                                 /* HOLD RETURNED KEY# */

WAVE1=SETWAVE(HARM);                                    /* STANDARD CHANNEL SETUP */
CHAN=ALLOCATE; CALL SETVOL(CHAN,255);
CALL EMITWSEL(CHAN,WAVE1);

CALL SCANDATA;                                          /* DO INTIAL SCAN */
DO WHILE (PANSW(0)=0);                                  /* REPEAT UNTIL PANEL 0 BUTTON IS PUSHED */
   I=SCAN.KEYBOARD(KEYLIST,0);                          /* TEST FOR A NEW KEY ON CLAVIER */
   IF I>0 THEN DO;                                      /* TEST FOR SOMETHING TO DO */
      CALL SETFRQ(CHAN,KEY(KEYLIST(0)),1000);           /* SET THE FREQUENCY */
      CALL SETE(CHAN,0,255);                            /* FAST ATTACK */
      CALL WAIT(5);                                     /* WAIT FOR ATTACK */
      CALL SETE(CHAN,1000,0);                           /* 1 SEC DECAY */
   END;
   CALL WAIT(5);                                        /* WAIT ONE CLOCK PULSE BEFORE SCANNING AGAIN */
   CALL SCANDATA;                                       /* SCAN AGAIN */
END;                                                    /* OF REPEAT LOOP */
```

DEMO10 -1-

```
/* MAX DEMO 10    10 FEBRUARY 1982 */
/* EXAMPLE OF MONOPHONIC SYNTHESIZER EMULATION */
/* WITH REAL TIME EFFECTS */

INSERT 'MAXSYN';                                   /* GET MAX SYNTH */
INSERT 'MAXIO';                                    /* GET CONTROL AND I/O FUNCTIONS */

DCL I                FIXED;                        /* HOLDS KEYBOARD SCAN OUTPUT */
DCL (WAVE1,CHAN) FIXED;
DCL HARM DATA (6,1000,500,333,250,200,167);
DCL RATIO            FIXED;                        /* THE FM RATIO */
DCL KEYLIST(0)       FIXED;                        /* NEW KEYS LIST */

WAVE1=SETWAVE(HARM);                               /* STANDARD CHANNEL SETUP */
CHAN=ALLOCATE; CALL SETVOL(CHAN,255);
CALL EMITWSEL(CHAN,WAVE1);
RATIO=1000;                                        /* INITIAL RATIO IS 1.000 */

CALL DISPLAY(RATIO,3,4);                           /* DISPLAY THE RATIO */
CALL SCANDATA;                                     /* DO INTIAL SCAN */
DO WHILE (PANSW(0)=0);                             /* REPEAT UNTIL PANEL 0 BUTTON IS PUSHED */
   RATIO=RATIO+KNOB.CHANGE;                        /* UPDATE RATIO */
   IF RATIO<0 THEN RATIO=0;                        /* LIMIT TO ZERO */

   I=SCAN.KEYBOARD(KEYLIST,0);                     /* TEST FOR ANY NEW KEYS ON CLAVIER */
   IF I>0 THEN DO;                                 /* TEST FOR SOMETHING TO DO */
      CALL SETFRQ(CHAN,KEY(KEYLIST(0)),RATIO);     /* SET THE FREQUENCY */
      CALL SETILIM(CHAN,SHR(RTEPEDAL.POS,1));      /* SET HARMONIC LEVEL */
      CALL SETE(CHAN,0,255);                       /* FAST ATTACK */
      CALL WAIT(5);                                /* WAIT FOR ATTACK */
      CALL SETE(CHAN,1000,0);                      /* 1 SEC DECAY */
   END;

   CALL WAIT(5);                                   /* WAIT ONE CLOCK PULSE BEFORE SCANNING AGAIN */
   CALL DISPLAY(RATIO,3,4);                        /* SHOW NEW RATIO */
   CALL SCANDATA;                                  /* SCAN AGAIN */
END;                                               /* OF REPEAT LOOP */
```

DEMO11 -1-

```
/* DEMO 11        11 FEBRUARY 82 */
/* ILLUSTRATION OF MULTIPLE TASKS */

INSERT 'MAXTASK';

MAIN: PROCEDURE;                                   /* THIS IS THE MAIN TASK */

   IDIOT1:  PROCEDURE;                             /* THIS IS SUBTASK 1 */
      DCL (I,J) FIXED;
      I=0; J=0;
      DO WHILE 1;                                  /* REPEAT FOREVER */
         PRINT 'This is task 1';
         CALL SUSPEND(1000);
      END;
   END IDIOT1;

   IDIOT2:  PROCEDURE;                             /* THIS IS SUBTASK 2 */
      DCL (I,J) FIXED;
      I=0; J=0;
      DO WHILE 1;
         PRINT 'This is task 2';
         CALL SUSPEND(3000);
      END;
   END IDIOT2;

   START IDIOT1 TASK;                              /* START THE IDIOTS YAKKING */
   START IDIOT2 TASK;

   TERMINATE TASK;                                 /* DONE WITH MAIN */
END MAIN;
```

```
/* MAX DEMO 12      3 MARCH 82 */
/* ILLUSTRATION OF PARALLEL MUSICAL LINES */

INSERT 'MAXSYN';
INSERT 'MAXTASK';

DCL KEYLIST1 DATA (12,16,18,19,21,24);                       /* PITCHES FOR BASS */
DCL KEYLIST2 DATA (24,28,31,32,36,40,43,45,48);              /* MELODY PITCHES */

MAIN: PROCEDURE;                                             /* THIS IS THE MAIN TASK */

   LINE1:  PROCEDURE;                                        /* LOWER LINE, 1 PITCH PER SECOND */
      DCL (CHAN, I) FIXED;
      CHAN=ALLOCATE;
      CALL SETVOL(CHAN,255);  CALL SETE(CHAN,500,255);
      DO I=1 TO 20;
         CALL SETFRQ(CHAN,KEY(KEYLIST1(RND(0,5))),1000);
         CALL SUSPEND(1000);
      END;
      CALL SETE(CHAN,300,0);
      CALL SUSPEND(300);
      CALL CLEANUP(CHAN);
      TERMINATE TASK;
   END LINE1;

   LINE2:  PROCEDURE;                                        /* UPPER LINE: 1, 2, 4 PITCHES/SEC */
      DCL (CHAN, I) FIXED;
      CHAN=ALLOCATE;
      CALL SETVOL(CHAN,255);  CALL SETE(CHAN,500,255);
      DO I=0 TO 43;
         CALL SETFRQ(CHAN,KEY(KEYLIST2(RND(0,8))),1000);
         DO CASE (I/4);
            CALL SUSPEND(1000);
            CALL SUSPEND(500);
            CALL SUSPEND(500);
            CALL SUSPEND(250);
            CALL SUSPEND(250);
            CALL SUSPEND(500);
            CALL SUSPEND(500);
            CALL SUSPEND(250);
            CALL SUSPEND(250);
            CALL SUSPEND(500);
            CALL SUSPEND(500);
         END;
      END;
      CALL SETE(CHAN,300,0);
      CALL SUSPEND(300);
      CALL CLEANUP(CHAN);
      TERMINATE TASK;
   END LINE2;

   START LINE1 TASK;                                         /* START EACH LINE */
   START LINE2 TASK;

   TERMINATE TASK;                                           /* DONE WITH MAIN */
END MAIN;
```

```
/* MAX DEMO 13        5 MARCH 82 */
/* COMPOSITION WITH OVERLAPPING EVENTS.  RTEPED CONTROLS SPEED. */
/* PEDAL DOWN=> INFREQUENT LONG EVENTS  UP=> RAPID SHORT EVENTS */

INSERT 'MAXSYN';
INSERT 'MAXIO';
INSERT 'MAXTASK';

MAIN: PROCEDURE;                                            /* THIS IS THE MAIN TASK */

   DCL WAVE(2) FIXED;                                       /* HOLDS WAVEFORM MEM NUMBERS */
   DCL HARMO DATA (1,1000);                                 /* WAVE 0 IS A SINE WAVE */
   DCL HARM1 DATA (3,1000,0,300);                           /* WAVE 1 HAS 3RD HARM */
   DCL HARM2 DATA (5,1000,200,100,50,300);                  /* WAVE 2 IS COMPLEX */
   DCL RATIOS DATA (1000,1500,2000,1000,1000,1002,0998);    /* PICK ONE */
   DCL PITCHES FLOATING DATA (8.00,8.07,9.00,7.09,8.01,8.05,9.05,9.04);

   DO.AN.EVENT: PROCEDURE;                                  /* THIS TASK WILL PROCESS AN EVENT */
      DCL CHAN AUTOMATIC1;

      CHAN=ALLOCATE;                                        /* GET US A CHANNEL */
      IF CHAN>=0 THEN DO;
         CALL SETWSEL(CHAN,WAVE(RND(0,3)));                 /* PICK A WAVEFORM MEM AT RANDOM */
         CALL SETVOL(CHAN,255);                             /* OPEN UP OUR VOLUME */
         CALL SETERATE(CHAN,100+SHL(RTEPEDAL.POS,2));
         CALL SETIRATE(CHAN,100+SHL(RTEPEDAL.POS,2));

         CALL SETFRQ(CHAN,PCH(PITCHES(RND(0,8))),RATIOS(RND(0,7)));
         CALL SETELIM(CHAN,RND(150,250));                   /* PICK ENV PEAK */
         CALL SETILIM(CHAN,RND(0,30));                      /* PICK ENV PEAK */
         CALL SUSPEND(250+RTEPEDAL.POS+RTEPEDAL.POS*RND(0,150)/10);  /* WAIT .5 TO 3.8 SECONDS */

         CALL SETELIM(CHAN,0);                              /* START TO SHUT OFF OUR CHANNEL */
         CALL SETILIM(CHAN,RND(0,5));
         CALL SUSPEND(200+RTEPEDAL.POS*RND(0,25)/10);       /* WAIT .2 TO 1 SECONDS */
         CALL FREECHAN(CHAN);                               /* GIVE UP OUR CHANNEL */
      END;
      TERMINATE TASK;                                       /* THIS EVENT IS DONE */
   END DO.AN.EVENT;

   WAVE(0)=SETWAVE(HARMO);                                  /* SET UP WAVEFORMS */
   WAVE(1)=SETWAVE(HARM1);
   WAVE(2)=SETWAVE(HARM2);

   CALL SCANDATA;
   DO WHILE (PANSW(0)=0);                                   /* EVENT INITIATING LOOP */
      START DO.AN.EVENT TASK;
      CALL SUSPEND(100+SHL(RTEPEDAL.POS,3));                /* WAIT A BIT */
      CALL SCANDATA;
   END;

   TERMINATE TASK;                                          /* DONE WITH MAIN */
END MAIN;
```

```
/* MAX DEMO 14       24 MARCH 82 */
/* polyphonic synthesizer with multi-segment envelopes */
/* bottom octave of keyboard selects timbre, like on Hammond B-3 */

INSERT 'MAXSYN';
INSERT 'MAXIO';
INSERT 'MAXTASK';

MAIN: PROCEDURE;                                            /* MAIN TASK */

   DCL RATIOS DATA (1000,995,2000,0501);
   DCL ENVE   DATA (10,255, 500,100, 500,100, 100,200, 500,50, 100,0,
   2000,255, 1000,150, 300,255, 100,100, 100,50, 100,0,
   10,255, 500,0, 10,195, 500,0, 10,105, 500,0);

   DCL ENVI   DATA (10,0, 500,12, 500,8, 500,3, 100,15, 500,10, 100,0,
   2000,240, 1000,100, 300,200, 100,150, 100,70, 100,0,
   10,100, 500,10, 10,75, 500,6, 10,50, 500,0);
   DCL TIMBRE#    FIXED;
   DCL MASK       DATA (1,2,4,8,16,32,64,128,256,512,1024,2048,
   4096,8192,16384,32768);
   DCL KEYLIST(9) FIXED;                                    /* TEN FINGERS */
   DCL (I,J)      FIXED;

   DO.AN.EVENT: PROCEDURE;                                  /* THIS TASK WILL PROCESS AN EVENT */
      DCL KEY# AUTOMATIC1;
      DCL CHAN AUTOMATIC2;
      DCL I    AUTOMATIC3;
      DCL J    AUTOMATIC4;

      CHAN=ALLOCATE;                                        /* GET US A CHANNEL */
      IF CHAN>=0 THEN DO;
         IF CHAN<32 THEN DISPLAYSW(3)=DISPLAYSW(3)\MASK(CHAN/2);
         ELSE            DISPLAYSW(5)=DISPLAYSW(5)\MASK((CHAN-32)/2);
         CALL SETVOL(CHAN,255);                             /* OPEN UP OUR VOLUME */
         CALL SETFRQ(CHAN,KEY(KEY#),RATIOS(TIMBRE#/3));
         J=(TIMBRE# MOD 3)*12;
         I=0; LOOPSTART:
         CALL SETE(CHAN,ENVE(J),ENVE(J+1));
         CALL SETI(CHAN,ENVI(J),ENVI(J+1));
         CALL SUSPEND(ENVE(J));                             /* WAIT FOR STAGE */
         J=J+2;
         I=I+1; IF I<=5 THEN GOTO LOOPSTART;
         CALL FREECHAN(CHAN);                               /* GIVE UP OUR CHANNEL */
         IF CHAN<32 THEN DISPLAYSW(3)=DISPLAYSW(3)&NOT(MASK(CHAN/2));
         ELSE            DISPLAYSW(5)=DISPLAYSW(5)&NOT(MASK((CHAN-32)/2));
      END;
      TERMINATE TASK;                                       /* THIS EVENT IS DONE */
   END DO.AN.EVENT;

   TIMBRE#=0;                                               /* SET INITIAL TIMBRE */
   CALL DISPLAY(TIMBRE#+1,0,0);                             /* DISPLAY NUMBER 1 TO 12 */
   CALL SCANDATA;
   DO WHILE (PANSW(0)=0);
      I=SCAN.KEYBOARD(KEYLIST,9);
      DO J=0 TO I-1;                                        /* CHECK NEW KEYS */
         IF KEYLIST(J)<12 THEN DO;                          /* CHANGE TIMBRE */
            TIMBRE#=KEYLIST(J);                             /* NEW TIMBRE */
            CALL DISPLAY(TIMBRE#+1,0,0);
         END;
         ELSE START DO.AN.EVENT SET (KEYLIST(J),0,0,0,0,0) TASK;
      END;
      CALL SUSPEND(5);
      CALL SCANDATA;
   END;
   TERMINATE TASK;
END MAIN;
```

```
/* MAX DEMO 15       11 MARCH 82 */
/* comprehensive example */

INSERT 'MAXSYN';
INSERT 'MAXIO';
INSERT 'MAXTASK';

MAIN: PROCEDURE;                                          /* MAIN TASK */

   DCL WAVE(2) FIXED;                                     /* HOLDS WAVEFORM MEM NUMBERS */
   DCL HARM0   DATA (3,1000,1000,200);                    /* WAVE 0 */
   DCL HARM1   DATA (3,1000,0,300);                       /* WAVE 1 */
   DCL HARM2   DATA (5,1000,100,100,50,400);              /* WAVE 2 */
   DCL RATIOS DATA (1000,1001,2000,0500,995,1000);
   DCL MASK    DATA (1,2,4,8,16,32,64,128,256,512,1024,2048,
   4096,8192,16384,32768);

   DCL PITCHSET.SIZE              LIT '8';
   DCL PITCHSET(PITCHSET.SIZE) FIXED;                     /* FIFO STACK OF PITCHES TO BE USED */
   DCL PITCHSET.PTR               FIXED;                  /* STACK POINTER */

   DCL INTERVAL      DATA (0,4,7,9,12);                   /* INTERVALS TO USE */
   DCL VALUE(3)      FIXED;                               /* DATA FOR TIMBRE */
   DCL UN DATA (1,1,1,4);                                 /* UNITS CODE */
   DCL SW DATA ("10","40","4000","40000");                /* SWITCH BITS */

   DO.AN.EVENT: PROCEDURE;                                /* THIS TASK WILL PROCESS AN EVENT */
      DCL CHAN AUTOMATIC1;
      DCL I    AUTOMATIC2;

      CHAN=ALLOCATE;                                      /* GET US A CHANNEL */
      IF CHAN>=0 THEN DO;
         I=PITCHSET(RND(0,PITCHSET.SIZE+1));              /* BASE PITCH */
         IF RND(0,10)<5 THEN I=I+INTERVAL(RND(0,5));      /* PICK INTERVAL */
         ELSE                I=I-INTERVAL(RND(0,5));
         IF CHAN<32 THEN DISPLAYSW(3)=DISPLAYSW(3)\MASK(CHAN/2);
         CALL SETWSEL(CHAN,WAVE(RND(0,3)));               /* PICK A WAVEFORM MEM */
         CALL SETVOL(CHAN,255);                           /* OPEN UP OUR VOLUME */

         CALL SETFRQ(CHAN,KEY(I),RATIOS(RND(0,6)));
         CALL SETE(CHAN,VALUE(0),RND(150,250));           /* PICK ENV PEAK */
         CALL SETI(CHAN,VALUE(2),VALUE(3));               /* SET IND PEAK */
         CALL SUSPEND(250+SHL(RTEPEDAL.POS,2));           /* WAIT .25 TO 1.3 SECONDS */

         CALL SETE(CHAN,VALUE(1),0);                      /* FINAL DECAY */
         CALL SETI(CHAN,1000,RND(0,5));
         CALL SUSPEND(VALUE(1));                          /* WAIT FOR DECAY */
         CALL FREECHAN(CHAN);                             /* GIVE UP OUR CHANNEL */
         IF CHAN<32 THEN DISPLAYSW(3)=DISPLAYSW(3)&NOT(MASK(CHAN/2));
      END;
      TERMINATE TASK;                                     /* THIS EVENT IS DONE */
   END DO.AN.EVENT;

   DCL (I,J,K)     FIXED;
   DCL TICK.COUNT FIXED;
   DCL KEYLIST(9) FIXED;                                  /* HUMANS HAVE TEN FINGERS */

   WAVE(0)=SETWAVE(HARM0);
   WAVE(1)=SETWAVE(HARM1);
   WAVE(2)=SETWAVE(HARM2);

   VALUE(0)=300; VALUE(1)=1000; VALUE(2)=1000; VALUE(3)=20;
   K=0;                                                   /* INITIAL CHANGE IS E ATK */
   DISPLAYSW(0)=SW(K);                                    /* SET LIGHT */
   PITCHSET.PTR=0;                                        /* SET UP PITCH STACK */
   DO I=0 TO PITCHSET.SIZE; PITCHSET(I)=24; END;          /* INIT TO MID C */

   CALL SCANDATA;
   TICK.COUNT=0;                                          /* TICKS UNTIL NEXT NEW EVENT */
   DO WHILE (PANSW(2)<>"02");                             /* LOOP UNTIL STOP BUTTON */
      IF TICK.COUNT=0 THEN DO;
         START DO.AN.EVENT TASK;
         TICK.COUNT=25+RTEPEDAL.POS/2+RND(0,35);          /* NEXT EVENT TIME */
      END;
      CALL SUSPEND(5);                                    /* WAIT A TICK */
      TICK.COUNT=TICK.COUNT-1;                            /* COUNT A TICK */
      CALL SCANDATA;
```

```
         I=SCAN.KEYBOARD(KEYLIST,9);                                 /* CHECK KEYS */
         DO J=0 TO I-1;                                              /* UPDATE PITCHSET */
            PITCHSET(PITCHSET.PTR)=KEYLIST(J);
            PITCHSET.PTR=PITCHSET.PTR+1;                             /* UPDATE POINTER */
            IF PITCHSET.PTR>PITCHSET.SIZE THEN PITCHSET.PTR=0;      /* WRAP AROUND */
         END;
         DO I=0 TO 3;
            IF PANSW(0)=SW(I) THEN DO; K=I; DISPLAYSW(0)=SW(I); END;
         END;
         VALUE(K)=VALUE(K)+KNOB.CHANGE;                              /* ADD IN ADJUST */
         IF VALUE(K)<0 THEN VALUE(K)=0;                              /* LIMIT */
         IF VALUE(K)>9999 THEN VALUE(K)=9999;
         CALL DISPLAY(VALUE(K),0,UN(K));                             /* DISPLAY VALUE */
      END;

      TERMINATE TASK;
   FND MAIN;
```

```
/*  MAX SYNTHESIZER ROUTINES   MODIFICATION LEVEL 2  23 MAR 82  */
/*  PRINCIPAL DEVELOPER:  JEFFREY S. RISBERG                    */
/*  COPYRIGHT 1982 NEW ENGLAND DIGITAL CORPORATION              */

DCL LOAD  LIT 'WRITE(5)=';  DCL MUL LIT 'WRITE(6)=';      /* DEFINE HARDWARE   */
DCL DIV   LIT 'WRITE(7)=';  DCL RES LIT 'READ(5)';        /* MUL/DIV INSTRUCS.  */

DCL STAB(256)     FIXED;                                  /* SINE TABLE FOR CREATING WAVEFORMS */
DCL LOGTAB(1000)  FIXED;                                  /* LOGARITHMS TABLE FOR FREQ CALCULATIONS */
DCL FTAB(1024)    FIXED;                                  /* FREQUENCY LOOK-UP TABLE */

DCL MAX.CHAN LIT '63';                                    /* UP TO 32 VOICE SYSTEM */
DCL CHANNELS.IN.USE(MAX.CHAN) FIXED;
DCL CHANNELS.LIST DATA (32,0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,
36,38,40,42,44,46,48,50,52,54,56,58,60,62);

/* GLOBAL VALUES FOR MAX (RETURNED BY VARIOUS ROUTINES) */

DCL (NOTEINC,NOTENUM,NOTEADD) FIXED;                      /* RETURNED BY CALCFRQ */
DCL (INTADD,INTDIV,TMC) FIXED;                            /* RETURNED BY CALCRATE (INTERPOLATORS) */
DCL WAVEBUF(255) FIXED;                                   /* RETURNED BY CALCWAVE (COMPUTED WAVE SHAPE) */
```

```
/*   */

/* WAVESHAPE MEMORY ROUTINES:
   --------- ------ --------

   'SETWAVE' IS THE MOST COMPREHENSIVE ROUTINE, AND TAKES A LIST OF
   HARMONIC COEFFICIENTS, ALLOCATES OR RE-USES A WAVEMEMORY, CALLS
   'CALCWAVE' TO COMPUTE THE WAVEFORM, AND CALLS 'EMITWAVE' TO SCALE
   THE DATA AND LOAD THE WAVEFORM MEMORY.  'CALCWAVE' AND 'EMITWAVE'
   MAY ALSO BE CALLED DIRECTLY.  'FREEWAVE' IS CALLED TO FREE UP A
   USAGE OF A WAVEMEMORY. */

DCL NUM.MEMORIES LIT '32';                              /* NUMBER OF WAVE MEMORIES */
DCL WAVEMEM.CONTENTS(24*NUM.MEMORIES) FIXED;            /* STORE HARMONIC DATA */
DCL WAVEMEM.USEDBY(NUM.MEMORIES) FIXED;                 /* STORE COUNTS OF USAGE */
DCL SINE.MEM FIXED;                                     /* WILL POINT TO MEM WITH SINE WAVE, SET UP IN INIT */
DCL CH1.MEM  FIXED;                                     /* MEM# FOR CHANNEL 1, FOR RESTORE IN EMITWAVE */

/* THE FOLLOWING SUBROUTINES ARE CALLED FROM 'CALCWAVE' TO
   ADD AND SUBTRACT HARMONICS FROM THE GLOBAL ARRAY 'WAVEBUF'.

   'ADDCOEF' AND 'SUBCOEF' BOTH TAKE TWO ARGUMENTS:
   THE COEFFICIENT NUMBER (1=FUNDAMENTAL, 2=SECOND HARMONIC, ETC),
   AND A RELATIVE AMPLITUDE (RANGE 0-1000, WHICH IS 0 TO 100.0 ON
   THE REAL-TIME SYSTEM. */

ADDCOEF:PROC(NUM,COEF);                                 /* PASS HARMONIC NUMBER AND VALUE (0-1000) */
   DCL (I,J,K,NUM,COEF) FIXED;
   IF COEF=0 THEN RETURN;                               /* NOTHING TO ADD IN */
   J=0;
   DO I=0 TO 255;                                       /* CYCLE THROUGH */
      LOAD COEF; MUL STAB(J); K=RES; DIV 2000;
      WAVEBUF(I)=WAVEBUF(I)+RES;
      J=(J+NUM)&255;
   END;
FND ADDCOEF;

SUBCOEF:PROC(NUM,COEF);                                 /* PASS HARMONIC NUMBER AND VALUE (0-1000) */
   DCL (I,J,K,NUM,COEF) FIXED;
   IF COEF=0 THEN RETURN;                               /* NOTHING TO SUBTRACT OUT */
   J=0;
   DO I=0 TO 255;
      LOAD COEF; MUL STAB(J); K=RES; DIV 2000;
      WAVEBUF(I)=WAVEBUF(I)-RES;
      J=(J+NUM)&255;
   END;
END SUBCOEF;

/* THE ROUTINE 'CALCWAVE' IS USED TO CALCULATE AN ENTIRE WAVE SHAPE.
   IT IS PASSED A FIXED POINT ARRAY CONTAINING A LIST OF
   HARMONIC COEFFICIENTS (LOCATION ZERO OF THE ARRAY CONTAINS THE
   NUMBER OF COEFFS IN THE ARRAY).

   THE TIME-DOMAIN TABULAR FUNCTION OF THE ARRAY IS COMPUTED AND
   STORED IN THE GLOBAL ARRAY 'WAVEBUF'. */

CALCWAVE:PROCEDURE(COEFLIST);                           /* PASS ARRAY CONTAINING COEFFICIENTS */
   DCL COEFLIST ARRAY;                                  /* PASS DATA */
   DCL I FIXED;
   DO I=0 TO 255; WAVEBUF(I)=0; END;                    /* ZERO OUT WAVEBUF */
   DO I=1 TO COEFLIST(0);                               /* LOOP OVER PARAMETER LIST */
      CALL ADDCOEF(I,COEFLIST(I));                      /* ADD IT IN */
   END;
END CALCWAVE;

/* A ROUTINE 'EMITWAVE' IS CALLED TO SCALE A WAVEFORM DATA ARRAY, AND
   LOAD A SYNTHESIZER WAVE MEMORY WITH THE WAVEFORM.

   THE PROCEDURE 'EMITWAVE' IS PASSED TWO ARGUMENTS:
   THE FIRST IS THE WAVE MEMORY NUMBER (0-31), AND THE SECOND IS
   A 256 LOCATION (0-255) FIXED POINT ARRAY.  IT IS NORMALLY THE
   GLOBAL ARRAY 'WAVEBUF', ALTHOUGH IT MAY BE ANY OTHER FIXED
   POINT ARRAY DECLARED IN THE USER PROGRAM. */

EMITWAVE:PROCEDURE(MEM#,WAVEBUF);                       /* CAN PASS ANY FIXED POINT ARRAY */
   DCL MEM# FIXED;
   DCL WAVEBUF ARRAY;                                   /* IS USUALLY GLOBAL ARRAY 'WAVEBUF' */
```

```
      DCL (I,J,K) FIXED;                                          /* AND TEMPS */
      DCL (MIN,MAX) FIXED;

      /* SCALE THE WAVESHAPE - BRING IT WITHIN BOUNDS 0 TO 255 */

      MIN=WAVEBUF(0); MAX=WAVEBUF(0);                             /* FIND MIN AND MAX */
      DO I=1 TO 255;
         IF WAVEBUF(I) < MIN THEN MIN=WAVEBUF(I);
         IF WAVEBUF(I) > MAX THEN MAX=WAVEBUF(I);
      END;
      DO I=0 TO 255; WAVEBUF(I)=WAVEBUF(I)-MIN; END;              /* SUBTRACT MIN */
      MAX=MAX-MIN;                                                /* FIND NEW MAX */
      IF MAX=0 THEN MAX=1;                                        /* AVOID DIVISION ERROR */
      I=MAX/512;                                                  /* TRY TO ROUND IN FOLLOWING ROUTINE */
      DO J=0 TO 255;
         LOAD WAVEBUF(J)+I; MUL 255; K=RES; DIV MAX;
         WAVEBUF(J)=RES;                                          /* COMPUTE SCALED ANSWER */
      END;

      /* NOW WE ARE READY TO EMIT IT TO THE SYNTHESIZER */

      DO J=0 TO MAX.CHAN BY 16;                                   /* FOR EACH SYNTH */
         IF CHANNELS.IN.USE(J)>=0 THEN DO;                        /* CHECK IF IT EXISTS */
            WRITE("160")=J+1; WRITE("161")="06"; WRITE("162")=0;  /* FIRST ZERO OUT MAR */
            WRITE("161")="04";                                    /* 'TIM' FUNCTION */
            WRITE("162")=MEM#+32;                                 /* SELECT MEMORY TO LOAD */
            WRITE("161")="07";                                    /* 'MEM' FUNCTION */
            DO I=0 TO 255; WRITE("162")=WAVEBUF(I); END;          /* LOAD MEMORY */
         END;
      END;                                                        /* SYNTHS */
      /* RESTORE MEM# FOR CHANNEL PAIR 0/1 */
      WRITE("160")=1; WRITE("161")="04"; WRITE("162")=CH1.MEM;
   END EMITWAVE;

   /* SETWAVE ROUTINE:
      FOR USER'S CONVENIENCE, SETWAVE ALLOCATES A WAVE MEMORY OR
      REUSES ONE IF IT ALREADY CONTAINS THE SAME WAVEFORM; THEN
      CALCULATES THE WAVEFORM AND LOADS IT INTO THE MEMORY.  RETURNS
      MEMORY NUMBER (0-32) CHOSEN, OR -1 IF NONE ARE FREE. */

   SETWAVE: PROCEDURE(COEFLIST);
      DCL COEFLIST ARRAY;                                         /* PASS DATA */
      DCL (I,J,K) FIXED;
      DCL (STARTING.POINT) FIXED;                                 /* USED IN ROTARY SEARCH FOR FREE MEMORY */
      DCL MEM# FIXED;                                             /* WAVE MEMORY NUMBER SELECTED */
      K=COEFLIST(0);                                              /* GET NUMBER OF COEFS */
      IF K>24 THEN K=24;                                          /* ONLY THE FIRST 24 HARMONIC COEFS ARE COMPARED
      DO I=0 TO NUM.MEMORIES-1;                                   /* SEARCH ALL MEMORIES */
         DO J=1 TO K;                                             /* COMPARE GIVEN COEFFS */
            IF COEFLIST(J)<>WAVEMEM.CONTENTS(I*24+J-1) THEN
            GOTO CANNOT.USE.THIS.MEMORY;                          /* CANNOT USE IF DIFFERENT */
         END;
         DO J=K+1 TO 24;                                          /* COMPARE FOLLOWING COEFFS */
            IF WAVEMEM.CONTENTS(I*24+J-1)<>0 THEN                 /* SHOULD BE ZERO */
            GOTO CANNOT.USE.THIS.MEMORY;
         END;
         WAVEMEM.USEDBY(I)=WAVEMEM.USEDBY(I)+1;                   /* MARK NEW USE */
         RETURN I;                                                /* SAME ONE FOUND, REUSE */
         CANNOT.USE.THIS.MEMORY:
      END;
      I=0;                                                        /* LOOK FOR FREE MEMORY */
      DO WHILE (I<NUM.MEMORIES) & (WAVEMEM.USEDBY(STARTING.POINT)<>0);
         I=I+1; STARTING.POINT=STARTING.POINT+1;
         IF STARTING.POINT=NUM.MEMORIES THEN STARTING.POINT=0;    /* WRAP AROUND */
      END;
      IF I=NUM.MEMORIES THEN DO;                                  /* NONE AVAILABLE */
         RETURN -1;                                               /* SIGNAL ERROR */
      END;
      MEM#=STARTING.POINT;                                        /* USE THIS ONE */
      WAVEMEM.USEDBY(MEM#)=1;                                     /* MARK ITS USE */
      STARTING.POINT=STARTING.POINT+1;                            /* START WITH NEXT MEMORY NEXT TIME */
      IF STARTING.POINT=NUM.MEMORIES THEN STARTING.POINT=0;       /* WRAP AROUND */

      CALL CALCWAVE(COEFLIST);                                    /* COMPUTE THE WAVEFORM */
      CALL EMITWAVE(MEM#,WAVEBUF);                                /* SCALE AND EMIT IT */
      DO I=1 TO K;                                                /* SAVE THE HARMONIC CONTENTS */
         WAVEMEM.CONTENTS(MEM# 24+I-1)=COEFLIST(I);
```

```
    END;
    DO I=K+1 TO 24; WAVEMEM.CONTENTS(MEM#*24+I-1)=0; END;      /* ADD FINAL ZEROES */
    RETURN MEM#;                                                /* RETURN THE WAVE MEMORY NUMBER USED */
END SETWAVE;

/* ROUTINE 'FREEWAVE' FREES UP A USAGE OF A GIVEN WAVEFORM MEMORY */

FREEWAVE: PROCEDURE(MEM#);                                      /* PASS MEMORY NUMBER */
    DCL MEM# FIXED;
    WAVEMEM.USEDBY(MEM#)=WAVEMEM.USEDBY(MEM#)-1;                /* FREE ONE USAGE */
    IF WAVEMEM.USEDBY(MEM#)<0 THEN WAVEMEM.USEDBY(MEM#)=0;      /* AVOID EXTRA FREES */
END FREEWAVE;
```

```
/*   */

/* INTERPOLATOR ROUTINES:

   .  THE ROUTINE 'CALCRATE' IS USED TO CALCULATE THE NECESSARY NUMBERS
   .  THAT ARE USED TO CONTROL THE INTERPOLATOR RATES IN THE DIGITAL
   .  SYNTHESIZER.   IT IS PASSED ONE ARGUMENT REPRESENTING THE NUMBER
   .  OF MILLISECONDS (0-5000) THAT A FULL SCALE (0-255 OR 255-0)
   .  PARAMETER CHANGE SHOULD TAKE.

   .  TWO NUMBERS ARE RETURNED IN THE GLOBAL VARIABLES 'INTADD' AND 'INTDIV'.
   .  THESE NUMBERS WILL REPRESENT THE INTERPOLATOR ADDER AND DIVISOR
   .  RESPECTIVELY.    THEY ARE WRITTEN OUT TO THE SYNTHESIZER BY
   .  THE 'EMITERATE' AND 'EMITIRATE' ROUTINES. */

CALCRATE: PROC(MS);                                 /* CALCULATE INTERPOLATOR NUMBERS */
   DCL MS FIXED;                                    /* THE TIME IN MILLISECONDS */
   INTDIV=MS-SHR(MS,2);
   IF INTDIV=0 THEN INTDIV=1;
   INTADD=256;
   DO WHILE INTDIV IGT 256;                         /* LIMIT TO 256 */
      INTDIV=SHR(INTDIV,1); INTADD=SHR(INTADD,1);
   END;
   INTDIV=256-INTDIV; INTADD=INTADD-1;              /* CORRECT FOR SYNTH ALGORITHM */
   IF MS<0 THEN MS=30000;                           /* USE 30 SECOND TIME CONSTANT IF LONGER DECAY */
   LOAD MS; DIV 25; MS=RES;                         /* COMPUTE MS/25 */
   IF MS<6 THEN MS=6;                               /* MIN OF 6 PERIODS */
   LOAD 5000; DIV MS; TMC=10000-RES;                /* COMPUTE 10000-5000/MS */
   LOAD 0; WRITE(4)=TMC; DIV 10000; TMC=RES;        /* TMC FDIV 10000 PDS ONLY */
END CALCRATE;

EMITIRATE: PROC(CHAN,IADD,IDIV);                    /* PASS CHANNEL, ADDER, DIVISOR */
   DCL (CHAN,IADD,IDIV) FIXED;
   WRITE("160")=CHAN;                               /* SEL CHAN */
   WRITE("161")="11";WRITE("162")=IADD;             /* ADDER */
   WRITE("161")="10";WRITE("162")=IDIV;             /* DIVISOR */
END EMITIRATE;

EMITERATE: PROC(CHAN,IADD,IDIV);                    /* PASS CHANNEL, ADDER, DIVISOR */
   DCL (CHAN,IADD,IDIV) FIXED;
   WRITE("160")=CHAN+1;                             /* SEL CHAN */
   WRITE("161")="11";WRITE("162")=IADD;             /* ADDER */
   WRITE("161")="10";WRITE("162")=IDIV;             /* DIVISOR */
END EMITERATE;

EMITILIM: PROC(CHAN,LIMIT);                         /* PASS CHANNEL, INDEX LIMIT (0-255) */
   DCL (CHAN,LIMIT) FIXED;
   IF LIMIT=255 THEN LIMIT=254;
   WRITE("160")=CHAN;
   WRITE("161")="12";WRITE("162")=LIMIT;
END EMITILIM;

EMITELIM: PROC(CHAN,LIMIT);                         /* PASS CHANNEL, ENV LIMIT (0-255) */
   DCL (CHAN,LIMIT) FIXED;
   IF LIMIT=255 THEN LIMIT=254;
   WRITE("160")=CHAN+1;
   WRITE("161")="12";WRITE("162")=LIMIT;
END EMITELIM;

DCL SETILIM LITERALLY 'EMITILIM';
DCL SETELIM LITERALLY 'EMITELIM';

SETIRATE: PROC(CHAN,MILLIS);                        /* PASS CHANNEL, INDEX ATTACK TIME IN MS */
   DCL (CHAN,MILLIS) FIXED;
   CALL CALCRATE(MILLIS);                           /* COMPUTE INTERPOLATOR INFORMATION */
   CALL EMITIRATE(CHAN,INTADD,INTDIV);              /* AND WRITE OUT ADDER, DIVIDER */
END SETIRATE;

SETERATE: PROC(CHAN,MILLIS);                        /* PASS CHANNEL, ENV ATTACK TIME IN MS */
   DCL (CHAN,MILLIS) FIXED;
   CALL CALCRATE(MILLIS);                           /* CALCULATE INTERPOLATOR INFORMATION */
   CALL EMITERATE(CHAN,INTADD,INTDIV);              /* AND EMIT DATA TO SYNTH */
END SETERATE;

SETE: PROC(CHAN,MILLIS,LIMIT);                      /* SET ENV RATE AND LIMIT */
   DCL (CHAN,MILLIS,LIMIT) FIXED;
   CALL SETERATE(CHAN,MILLIS);
```

```
    CALL SETELIM(CHAN,LIMIT);
END SETE;

SETI: PROC(CHAN,MILLIS,LIMIT);                    /* SET INDEX RATE AND LIMIT */
    DCL (CHAN,MILLIS,LIMIT) FIXED;
    CALL SETIRATE(CHAN,MILLIS);
    CALL SETILIM(CHAN,LIMIT);
END SETI;
```

```
/*    */

/* FREQUENCY CALCULATION ROUTINES
   --------- ----------- --------

   LOG1000 PROCEDURE IS USED TO COMPUTE THE LOG OF A NUMBER WHERE 1000
   REPRESENTS 1.000.  FINDS LOG (BASE 2) TIMES 1024.  */

LOG1000: PROC(VAL);                                               /* VAL IS 1000 = 1.000 */
   DCL (VAL, I) FIXED;
   IF VAL=0 THEN RETURN -16384;                                   /* RATIO OF ZERO */
   I=0;
   DO WHILE VAL<1000;  I=I-1024;  VAL=SHL(VAL, 1);  END;          /* ADJUST VALUE */
   DO WHILE VAL>1999;  I=I+1024;  VAL=SHR(VAL, 1);  END;
   RETURN I+LOGTAB(VAL-1000);                                     /* LOOK UP FRACTIONAL PART FROM TABLE */
END LOG1000;

LOG4400: PROC(VAL);                                               /* SAME AS LOG1000 ONLY 4400=1.000 */
   DCL VAL FIXED;
   IF VAL<0 THEN VAL=-VAL;                                        /* ABSOLUTE VALUE */
   LOAD VAL;  MUL 1000;  VAL=RES;  DIV 4400;  VAL=RES;            /* SCALE BY 1000/1024 */
   RETURN LOG1000(VAL);
END LOG4400;

DCL WESTERN.SCALE DATA (0,85,171,256,341,427,512,597,683,768,853,939);
DCL BITS          DATA (1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768);
DCL PREVIOUS.OCTAVE FIXED;                                        /* STORAGE FOR OCTAVE FOLLOWING IN 'PITCH' */
DCL OCTAVE.RATIO    FIXED;                                        /* OCTAVE RATIO VALUE FOR FRQ ROUTINES (X 1000) */
DCL FREQ.BASE       FIXED;                                        /* LOG4400 OF TUNING.BASE */

SET.TUNING.BASE: PROC(VAL);                                       /* SET TUNING BASE FOR CALLS TO 'PITCH' AND 'KEY' */
   DCL VAL FIXED;                                                 /* 4400 = 440.0 = A-440 TUNING */
   FREQ.BASE=LOG4400(VAL);                                        /* GET THE LOG */
END SET.TUNING.BASE;

/* 'KEY' TAKES A CLAVIER KEY NUMBER (0-60) AND PRODUCES THE CORRESPONDING
   FREQUENCY NUMBER IN THE INTERNAL FORM */

KEY: PROCEDURE(NUMBER);                                           /* PASSED KEY NUMBER (0-60) */
   DCL (NUMBER, OCTAVE, PITCH, FNUM) FIXED;
   OCTAVE=NUMBER/12;                                              /* FIND THE OCTAVE */
   PITCH=NUMBER-OCTAVE*12;                                        /* FIND THE PITCH */
   FNUM=SHL(OCTAVE,10)+WESTERN.SCALE(PITCH)+FREQ.BASE-2816;
   FNUM=FNUM*OCTAVE.RATIO/1000;                                   /* OCT RAT ADJ */
   RETURN FNUM+2816;
END KEY;

/* 'PITCH' IS A CONVENIENT ROUTINE TO TAKE A PITCH IN CHARACTER FORM
   AS USED IN SCRIPT (EG., 'C#3', 'A4', 'EF3'), AND PRODUCE A FREQUENCY
   NUMBER IN THE INTERNAL FORM.  'PITCH' ALSO REMEMBERS THE OCTAVE OF
   THE PREVIOUS CALL, FOR OCTAVE FOLLOWING AS IN SCRIPT.  */

PITCH: PROCEDURE(NOTE);                                           /* NOTE IS A STRING SUCH AS 'C#3' */
   DCL NOTE     ARRAY;                                            /* THE INPUT */
   DCL SCALE    ARRAY DATA ('C D EF G A B');
   DCL DIGITS   ARRAY DATA (' 12345');
   DCL CHAR     FIXED;                                            /* CHARACTER FROM NOTE STRING */
   DCL KEY#     FIXED;                                            /* CLAVIER KEY NUMBER */
   DCL (I,J,K) FIXED;

   CHAR=BYTE(NOTE,0)&"177";                                       /* GET PITCH CLASS LETTER */
   IF (CHAR>="141")&(CHAR<="173") THEN CHAR=CHAR-"40";            /* UP THE CASE */
   KEY#=-1;
   DO I=0 TO 11;                                                  /* FIND MATCH IN SCALE */
      IF CHAR=("177"&BYTE(SCALE,I)) THEN KEY#=I;
   END;
   IF KEY#<0 THEN RETURN SHL(7,10);                               /* HIGH-PITCH SCREAM AS ERR MESSAGE */
   ELSE DO J=1 TO NOTE(0)-1;                                      /* LOOP OVER FOLLOWING CHARS */
      CHAR=BYTE(NOTE,J)&"177";                                    /* GET NEXT LETTER */
      IF CHAR="043" THEN KEY#=KEY#+1;                             /* CHECK FOR SHARP */
      IF (CHAR="106")\(CHAR="146") THEN KEY#=KEY#-1;              /* FLAT */
      DO K=1 TO 5;                                                /* CHECK FOR OCTAVE */
         IF CHAR=("177"&BYTE(DIGITS,K)) THEN PREVIOUS.OCTAVE=K;  /* SAVE IT */
      END;
   END;
   RETURN KEY(KEY#+12*(PREVIOUS.OCTAVE-1));
END PITCH;
```

```
/* 'HERTZ' TAKES A FREQUENCY AS AN INTEGER REPRESENTING HERTZ TIMES 10,
   (THUS 4400 MEANS A-440), AND PRODUCES THE CORRESPONDING FREQUENCY
   NUMBER IN THE INTERNAL FORM. */

HERTZ: PROCEDURE(NUMBER);                                  /* FREQUENCY TIMES 10 */
   DCL NUMBER FIXED;
   RETURN LOG4400(NUMBER)+2816;                            /* TAKE LOG PLUS A-440 OFFSET */
END HERTZ;

/* 'PCH' TAKES A OCTAVE.PITCHCLASS NUMBER, AS IN MUSIC-11 OR MUSIC4BF,
   AND PRODUCES THE INTERNAL FREQUENCY NUMBER.  NOTE THAT THIS IS THE
   ONLY ROUTINE WHICH TAKES A FLOATING ARGUMENT. */

PCH: PROCEDURE(NUMBER);                                    /* FLOATING OCT.PCLASS, EG., 8.00 = MIDDLE C */
   DCL NUMBER          FLOATING;                           /* 8.09 = A ABOVE MID C */
   DCL (OCT,OFFSET) FIXED;
   DCL FRAC            FLOATING;
   OCT=INT(NUMBER);                                        /* EXTRACT THE OCTAVE */
   FRAC=NUMBER-OCT;                                        /* EXTRACT THE FRACTION */
   OFFSET=INT(1024.*FRAC*100./12.+0.5);
   RETURN SHL(OCT-6,10)+OFFSET;
END PCH;

/* 'CALCFRQ' TAKES FREQUENCY NUMBER (AS PRODUCED BY 'PITCH', 'HERTZ',
   'PCH', OR 'KEY'), AND FINDS THE INCREMENT, NOTE NUMBER, AND OCTAVE
   ADDER FOR THE SYNTHESIZER. */

CALCFRQ: PROCEDURE(FREQ.NUM);                              /* GIVEN INTERNAL FORM FREQUENCY NUMBER */
   DCL (FREQ.NUM,I,J,K) FIXED;
   IF FREQ.NUM<=-4096 THEN DO;
      NOTEINC=5; NOTENUM=199; NOTEADD=0;
   END;
   ELSE DO;
      I=FREQ.NUM&1023;                                     /* EXTRACT FRACTIONAL OCTAVE PART */
      NOTENUM=FTAB(I);                                     /* LOOKUP FREQUENCY */
      NOTEINC=SHR(NOTENUM,8);                              /* EXTRACT INCREMENT */
      NOTENUM=NOTENUM&255;                                 /* AND EXTRACT DIVISOR */

      IF FREQ.NUM>=1024 THEN DO;                           /* WE CAN COMPUTE NOTEADDER */
         NOTEADD=SHR(FREQ.NUM,10)-1;
         IF NOTEADD>7 THEN NOTEADD=7;                      /* LIMIT IT */
      END;
      ELSE DO;                                             /* CORRECT INC/DIVISOR PAIRS FOR SUPER LOW FRE
         NOTEADD=0;                                        /* USE LOWEST */
         I=SHR((1024+1023)-FREQ.NUM,10);                   /* COMPUTE NUMBER OF OCTAVE TO BRING DOWN */
         DO WHILE (I<>0)&((NOTEINC&1)=0);                  /* EVEN INCREMENTS */
            I=I-1; NOTEINC=SHR(NOTEINC,1);
         END;
         DO WHILE (I<>0)&(NOTENUM<128);
            I=I-1; NOTENUM=SHL(NOTENUM,1);
         END;
         IF I<>0 THEN DO;                                  /* MORE TO DO */
            J=NOT(BITS(I)-1);                              /* COMPUTE MASK FOR BITS TO SAVE */
            NOTENUM=NOTENUM*(NOTEINC&J)/NOTEINC;           /* CORRECT DIVISOR FOR LOST BITS */
            DO WHILE I<>0; NOTEINC=SHR(NOTEINC,1); I=I-1; END;
         END;
      END;
   END;
   NOTEINC=NOTEINC-1;                                      /* ADJUST FOR SYNTH ALGORITHM */
   NOTENUM=256-NOTENUM+SHR(NOTEINC,3);
END CALCFRQ;

EMITIFRQ: PROC(CHAN,NUM,INC,IADD);                         /* PASS CHANNEL, NOTENUM, INC, ADDER */
   DCL (CHAN,INC,NUM,IADD) FIXED;
   DISABLE;                                                /* DON'T LET THIS GET INTERRUPTED */
   WRITE("160")=CHAN;                                      /* SELECT INDEX CHANNEL */
   WRITE("161")="02"; WRITE("162")=IADD;                   /* ADDER VALUE */
   WRITE("161")="01"; WRITE("162")=NUM;                    /* NOTE NUMBER */
   WRITE("161")="00"; WRITE("162")=INC;                    /* INCREMENT */
   ENABLE;                                                 /* DONE NOW */
END EMITIFRQ;

EMITEFRQ: PROC(CHAN,NUM,INC,IADD);                         /* PASS CHANNEL, NOTENUM, INC, ADDER */
   DCL (CHAN,INC,NUM,IADD) FIXED;
   DISABLE;
   WRITE("160")=CHAN+1;                                    /* SELECT ENV CHANNEL */
```

```
        WRITE("161")="02"; WRITE("162")=IADD;                    /* ADDER */
        WRITE("161")="01"; WRITE("162")=NUM;                     /* NUMBER */
        WRITE("161")="00"; WRITE("162")=INC;
        ENABLE;
    END EMITEFRQ;

    SETFRQ: PROC(CHAN, FREQ.NUM, RATIO);                         /* PASS CHANNEL, FREQ, RATIO */
        DCL (CHAN, FREQ.NUM, RATIO) FIXED;
        CALL CALCFRQ(FREQ.NUM);                                  /* COMPUTE NUMBERS FOR FREQUENCY */
        CALL EMITEFRQ(CHAN, NOTENUM, NOTEINC, NOTEADD);
        IF RATIO<>1000 THEN CALL CALCFRQ(FREQ.NUM+LOG1000(RATIO));
        CALL EMITIFRQ(CHAN, NOTENUM, NOTEINC, NOTEADD);
    END SETFRQ;

    SETFRQ2: PROC(CHAN, E.FREQ.NUM, I.FREQ.NUM);                 /* PASS CHANNEL, TWO FREQS */
        DCL (CHAN, I.FREQ.NUM, E.FREQ.NUM) FIXED;
        CALL CALCFRQ(I.FREQ.NUM);                                /* COMPUTE NUMBERS FOR INDEX FREQ */
        CALL EMITIFRQ(CHAN, NOTENUM, NOTEINC, NOTEADD);
        IF I.FREQ.NUM<>E.FREQ.NUM THEN CALL CALCFRQ(E.FREQ.NUM); /* ENV FREQ */
        CALL EMITEFRQ(CHAN, NOTENUM, NOTEINC, NOTEADD);
    END SETFRQ2;
```

```
/*    */

/* EMIT AND SET ROUTINES FOR VOLUME, WAVEFORM SELECT, AND SHIFT COUNT */

EMITVOL:PROC(CHAN,VOL);                                  /* PASS CHANNEL, VOLUME 0-255 */
   DCL (CHAN,VOL) FIXED;
   WRITE("160")=CHAN+1;                                  /* SELECT CHANNEL */
   WRITE("161")="13";WRITE("162")=VOL;                   /* 'VOL' FUNCTION */
END EMITVOL;

EMITWSEL:PROC(CHAN,MEM#);                                /* PASS CHANNEL, WAVE MEM # */
   DCL (CHAN,MEM#) FIXED;
   IF CHAN=0 THEN CH1.MEM=MEM#;                          /* SAVE CHAN PAIR 0/1 MEM# */
   WRITE("160")=CHAN+1;                                  /* SELECT CHANNEL */
   WRITE("161")="04";                                    /* FUNCTION CODE 'TIM' */
   WRITE("162")=MEM#+32;                                 /* INDICATE WAVE MEMORY NUMBER */
END EMITWSEL;

EMITISHC:PROC(CHAN,ISHC);                                /* PASS CHANNEL, INDEX SHIFT COUNT */
   DCL (CHAN,ISHC) FIXED;
   WRITE("160")=CHAN;
   WRITE("161")="13";WRITE("162")=ISHC;
END EMITISHC;

DCL SETVOL  LITERALLY 'EMITVOL';
DCL SETWSEL LITERALLY 'EMITWSEL';
DCL SETISHC LITERALLY 'EMITISHC';

/* CHANNEL CLEANUP ROUTINE */

CLEANUP: PROCEDURE(CHAN);                                /* CLEAN UP A CHANNEL PAIR, GIVEN EVEN CHAN # */
   DCL CHAN FIXED;                                       /* THE CHANNEL PAIR TO BE CLEANED */
   WRITE("160")=CHAN;                                    /* FIRST HIT EVEN CHANNEL */
   WRITE("161")="10";WRITE("162")=255;WRITE("161")="11";WRITE("162")=255;
   WRITE("161")="12";WRITE("162")=0;
   WRITE("161")="02";WRITE("162")=0;
   WRITE("161")="01";WRITE("162")=255;
   WRITE("161")="00";WRITE("162")=255;                   /* WRITE LARGE INC TO ZERO THETA */

   WRITE("160")=CHAN+1;                                  /* NOW HIT ODD CHANNEL */
   WRITE("161")="10";WRITE("162")=255;WRITE("161")="11";WRITE("162")=255;
   WRITE("161")="12";WRITE("162")=0;
   WRITE("161")="00";WRITE("162")=255;                   /* HARDWARE ALSO WRITES ONU=0;NNU=255 */
   WRITE("161")="13";WRITE("162")=0;                     /* ZERO OUT VOLUME REGISTER */
   CALL EMITWSEL(CHAN,SINE.MEM);                         /* LINK TO SINE ROM */
END CLEANUP;
```

```
/*   */

/* UTILITY ROUTINES FOR TIMING, RANDOM NUMBERS, CHANNEL ALLOCATION */

DCL MAXSYN.CLOCK.DIVISOR FIXED;                        /* MILLISECONDS PER TICK */

/* BASIC WAIT ROUTINE */

WAIT: PROC(NUM);                                       /* PASS NUMBER OF MILLISECONDS TO WAIT (.001 SEC) */
   DCL (NUM, I) FIXED;
   NUM=NUM/MAXSYN.CLOCK.DIVISOR;                       /* FIND NUMBER OF TICKS TO WAIT */
   DO I=1 TO NUM; WRITE(3)=0; END;                     /* WAIT */
END WAIT;

/* RANDOM NUMBER GENERATOR ROUTINE */

RND: PROC(MIN, MAX);
   DCL (MIN, MAX) FIXED;
   DCL SEED FIXED;
   IF SEED=0 THEN SEED="157632";
   SEED=ROT(SEED, 3)+(ROT(SEED, 5) XOR ROT(SEED, 9));
   RETURN MIN+(SEED MOD (MAX-MIN));
END;

/* CHANNEL ALLOCATION ROUTINES:   */

ALLOCATEX: PROC(CHAN.SET);                             /* ALLOCATE A CHANNEL FROM LIST */
   DCL CHAN.SET ARRAY;                                 /* LIST OF CHANNELS */
   DCL (CHAN, I) FIXED;
   DO I=1 TO CHAN.SET(0);                              /* LOOP OVER LIST */
      CHAN=CHAN.SET(I);
      IF CHANNELS.IN.USE(CHAN)=0 THEN DO;              /* TEST */
         CHANNELS.IN.USE(CHAN)=1;                      /* MARK IT AS IN USE */
         RETURN CHAN;                                  /* RETURN IT */
      END;
   END;
   RETURN -1;                                          /* NO CHANNELS ARE AVAILABLE */
END ALLOCATEX;

ALLOCATE: PROC;                                        /* ALLOCATE ANY FREE CHANNEL */
   RETURN ALLOCATEX(CHANNELS.LIST);                    /* USE COMPLETE CHANNELS LIST */
END ALLOCATE;

FREECHAN: PROCEDURE(CHAN);                             /* FREE THIS CHANNEL */
   DCL CHAN FIXED;                                     /* THE CHANNEL NUMBER */
   CHANNELS.IN.USE(CHAN)=0;
END FREECHAN;

ZEROSYN: PROCEDURE;                                    /* ZERO OUT AND FREE UP ALL CHANNELS */
   DCL CHAN FIXED;                                     /* CHANNEL NUMBER */
   DO CHAN=0 TO MAX.CHAN BY 2;
      IF CHANNELS.IN.USE(CHAN)<>-1 THEN DO;            /* TEST EXISTANCE */
         CALL CLEANUP(CHAN);
         CALL FREECHAN(CHAN);                          /* FREE IT */
      END;
   END;
END ZEROSYN;
```

```
/*    */

LOCATE.FILE: PROC(CAT.BUF,FILENAME);                              /* SEARCH FOR FILE */
   DCL (CAT.BUF,FILENAME) ARRAY;
   DCL CFN(4)          FIXED;
   DCL (I,J,K,POS)   FIXED;
   DO I=1 TO 4; CFN(I)=0; END;
   DO I=0 TO FILENAME(0)-1;                                        /* COPY FILENAME AND PAD WITH NULLS */
      J=BYTE(FILENAME,I)&"177";
      CALL PBYTE(CFN,I,J);                                         /* PUT INTO CORRECTED FILE NAME */
   END;
   DO I=1 TO 32;                                                   /* CHECK EACH FILE */
      POS=SHL(I-1,3);                                              /* GET CATALOG BLOCK PTR */
      K=1;                                                         /* SET FLAG */
      DO J=0 TO 3; IF CAT.BUF(POS+J)<>CFN(J+1) THEN K=0; END; /* CHECK NAME */
      IF (K) THEN RETURN CAT.BUF(POS+4);                           /* RETURN STARTING SECTOR IF FOUND */
   END;
   RETURN -1;                                                      /* NOT FOUND */
END LOCATE.FILE;

/* INITIALIZATION ROUTINE */

DCL UCAT LIT '10000';

MAXSYN.INITIALIZE: PROCEDURE;                                      /* SET UP MAX */
   DCL (I,J)              FIXED;
   DCL SINE.CONTENTS DATA (1,1000);                                /* DATA FOR SINE WAVE */
   DCL TABLE.SECTOR   FIXED;                                       /* STARTING SECTOR FOR DATA TABLES FILE */

   CALL DISKREAD(UCAT,WAVEBUF,255);                                /* READ USER CATALOG. USE WAVEBUF AS BUFFER
   TABLE.SECTOR=LOCATE.FILE(WAVEBUF,'.TABLES')+UCAT;
   IF TABLE.SECTOR<UCAT THEN TABLE.SECTOR=LOCATE.FILE(WAVEBUF,'.STAB-4')+UCAT;
   IF TABLE.SECTOR<UCAT THEN DO;
      CALL DISKREAD(0,WAVEBUF,255);                                /* READ SYSTEM CATALOG */
      TABLE.SECTOR=LOCATE.FILE(WAVEBUF,'.TABLES');
      IF TABLE.SECTOR<0 THEN TABLE.SECTOR=LOCATE.FILE(WAVEBUF,'.STAB-4');
   END;
   IF TABLE.SECTOR<0 THEN DO;
      PRINT '''.TABLES'' file is not present on your disk.';
      PRINT 'SAVE a copy on this user disk and run again.';
      CALL EXIT(-1);
   END;
   CALL DISKREAD(TABLE.SECTOR,   FTAB,1024);                       /* SET UP TABLES */
   CALL DISKREAD(TABLE.SECTOR+8,STAB,256);
   CALL DISKREAD(TABLE.SECTOR+9,LOGTAB,1000);

   DO I=0 TO MAX.CHAN;                                             /* BEGIN TEST OF CHANNELS IN SYNTHESIZER */
      WRITE("160")=I;
      IF READ("160") THEN CHANNELS.IN.USE(I)=0;                    /* TEST IF IT EXISTS */
      ELSE                CHANNELS.IN.USE(I)=-1;                   /* MARK AS MISSING */
   END;
   DO I=0 TO NUM.MEMORIES-1;                                       /* CLEAR OUT WAVE MEM USAGE DATA */
      WAVEMEM.USEDBY(I)=0;                                         /* FREE WAVE MEMS */
      DO J=0 TO 23; WAVEMEM.CONTENTS(24*I+J)=0; END;
   END;
   SINE.MEM=SETWAVE(SINE.CONTENTS);                                /* SET UP A SINE WAVE */
   /*IF SHR(CORE(CORE(1)+14),1)&1 THEN*/ MAXSYN.CLOCK.DIVISOR=5; /* SET CLOCK RATE */
   /*ELSE MAXSYN.CLOCK.DIVISOR=10;*/

   PREVIOUS.OCTAVE=3;                                              /* START OCTAVE FOLLOWING IN 'PITCH' AT MIDDLE
   OCTAVE.RATIO=1000;                                              /* SET UP NORMAL TUNING */
   CALL SET.TUNING.BASE(4400);                                     /* AND NORMAL BASE */

   CALL ZEROSYN;
END MAXSYN.INITIALIZE;

CALL MAXSYN.INITIALIZE;
```

```
/* MAX I/O ROUTINES    MODIFICATION LEVEL 2       23 MARCH 1982 */
/* PRINCIPAL DEVELOPER:  JEFFREY S. RISBERG                     */
/* COPYRIGHT 1982  NEW ENGLAND DIGITAL CORPORATION              */

/* THESE ROUTINES ENABLE READING OF INPUTS FROM THE KEYBOARD,
.  CONTROL PANEL, CONTROL KNOB, INPUT PEDALS, RIBBON CONTROLLER,
.  AND FOOTSWITCHES, AND OUTPUT DISPLAY OF NUMBERS, LIGHTS, AND
.  CONTROL VOLTAGES. */

/* IMPORTANT GLOBAL INFORMATION */

DCL NUM.OCTAVES       LIT '06';                    /* NUMBER OF OCTAVES TO SCAN FROM KEYBOARD */
DCL NUM.SW.PANS       LIT '08';                    /* NUMBER OF SWITCH PANELS TO SCAN */

DCL PANSW(NUM.SW.PANS-1)       FIXED;              /* INPUT: FROM PANEL SWITCHES */
DCL CLAVIER(NUM.OCTAVES-1)     FIXED;              /* INPUT: FROM CLAVIER */
DCL DIGDISPLAY(NUM.OCTAVES-1)  FIXED;              /* OUTPUT: DIGITS TO DISPLAY */
DCL DISPLAYSW(NUM.SW.PANS-1)   FIXED;              /* OUTPUT: BUTTONS TO LIGHT ON PANEL */

DCL KNOB.POS        FIXED;                         /* KNOB POSITION */
DCL KNOB.CHANGE     FIXED;                         /* AMOUNT TO CHANGE A PARAMETER BY */
DCL KNOB.BASE       FIXED;                         /* NEUTRAL KNOB POSITION */
DCL VOLPEDAL.POS    FIXED;                         /* VOLUME PEDAL POSITION */
DCL RTEPEDAL.POS    FIXED;                         /* REAL TIME EFFECTS PEDAL POSITION */
DCL PBI.POS         FIXED;                         /* PITCH BEND INPUT */
DCL PBI.BASE        FIXED;                         /* NEUTRAL PITCH BEND INPUT */
DCL RIB.ACTIVE      FIXED;                         /* TRUE IF RIBBON ACTIVE */
DCL RIB.BASE        FIXED;                         /* STARTING RIBBON POSITION */
DCL RIB.POS         FIXED;                         /* RIBBON VALUE */
DCL RTE.MAX    LIT '225';                          /* LARGEST RTE OR VOL PEDAL POSITION */

DCL SWITCHDATA(5) FIXED;                           /* HOLDS SWITCH INPUTS */
DCL HOLD.SWITCH    LIT  'SWITCHDATA(0)';           /* HOLD SWITCH */
DCL REP.SWITCH     LIT  'SWITCHDATA(1)';           /* REPEAT SWITCH */
DCL GLIDE.SWITCH   LIT  'SWITCHDATA(2)';           /* PORTAMENTO SWITCH */
DCL SUST.SWITCH    LIT  'SWITCHDATA(3)';           /* SUSTAIN SWITCH */
DCL ARP.SWITCH     LIT  'SWITCHDATA(4)';           /* ARPEGGIATE SWITCH */
DCL PUNCH.SWITCH   LIT  'SWITCHDATA(5)';           /* PUNCH IN/PUNCH OUT */

DCL OUT.DATA(7)    FIXED;                          /* HOLDS OUTPUT VOLTAGES */
DCL LPFILT.OUT     LIT  'OUT.DATA(0)';             /* LOW PASS FILT OUT */
DCL HPFILT.OUT     LIT  'OUT.DATA(1)';             /* HIGH PASS FILT OUT */
DCL BPFILT.OUT     LIT  'OUT.DATA(2)';             /* BAND PASS FILT OUT */
DCL BANDWIDTH.OUT  LIT  'OUT.DATA(3)';
DCL CV.OUT         LIT  'OUT.DATA(4)';             /* CONTROL VOLTAGE OUT */
DCL GATE.OUT       LIT  'OUT.DATA(5)';             /* KEYBOARD GATE OUT */
DCL TRIGGER.OUT    LIT  'OUT.DATA(6)';             /* KEYBOARD TRIGGER OUT */
DCL RIBBON.OUT     LIT  'OUT.DATA(7)';             /* RIBBON VOLTAGE OUT */

DCL DR.A LIT '"130"';                              /* SWITCH/CLAVIER INPUT DATA REGISTER ADDRESS */
DCL CR.A LIT '"131"';                              /* CONTROL REGISTER - USED ALSO TO READ KNOB */
DCL AD.A LIT '"163"';                              /* AD CONVERTER ADDRESS */
DCL SW.A LIT '"160"';                              /* SWITCH CONVERTER ADDRESS */

SCANDATA: PROCEDURE;                               /* SCAN ALL INPUT DATA */
   DCL (I,J) FIXED;
   DCL (BASE.ADJUST) FIXED;                        /* COUNTER FOR KNOB BASE ADJUSTMENT */
   DCL (DELTA,ACCUM) FIXED;                        /* KNOB ADJUSTMENT VALUES */

   /* READ DATA FROM D130 CLAVIER/CONTROL PANEL INTERFACE */

   WRITE(CR.A)="20";                               /* FIRST STOP THE WRITE IN PROGRESS */
   DO I=0 TO (NUM.OCTAVES-1);                      /* FIRST THE KEYBOARD */
      WRITE(CR.A)=I;                               /* SET UP ADDRESS */
      WRITE(DR.A)=DIGDISPLAY(I);                   /* AND NUMBER TO DISPLAY IN LEDS */
      WRITE(CR.A)=I\"40"; WRITE(CR.A)=I;           /* PULSE EXW FOR LATCH STROBE */
      WRITE(CR.A)=I\"100";                         /* NOW READ THE */
      CLAVIER(I)=NOT(READ(DR.A));                  /* GET KEYBOARD DATA */
      WRITE(CR.A)=I;                               /* AND OFF WITH READ */
   END;

   DO I=0 TO (NUM.SW.PANS-1);                      /* NEXT READ CONTROL PANEL */
      WRITE(CR.A)=I\"10";                          /* SET UP ADDRESS */
      WRITE(CR.A)=I\"110";                         /* NOW READ THEM FIRST */
      PANSW(I)=READ(DR.A);                         /* AND READ NEW SETTING */
      WRITE(CR.A)=I\"10";                          /* AND OFF WITH READ */
      WRITE(DR.A)=DISPLAYSW(I)\PANSW(I);           /* LIGHT THOSE PRESSED */
```

```
       END;
       WRITE(CR.A)="20";                                       /* SET UP EXTERNAL WRITE AGAIN */
       WRITE(CR.A)="260";                                      /* START CONVERSION, SET UP EXTERNAL WRITE */

       I=4;                                                    /* READ THE FOOT SWITCH INPUTS (HOLD, SUSTAIN, ETC.
       DO J=0 TO 5;                                            /* ALL SIX OF THEM */
          SWITCHDATA(J)=(READ(SW.A)&I)<>0;                     /* TRUE IF NONZERO */
          I=SHL(I,1);
       END;

       /* READ ANALOG INPUTS */

       WRITE(AD.A)=2; KNOB.POS=READ(AD.A);                     /* READ KNOB POSITION */
       BASE.ADJUST=BASE.ADJUST+1;                              /* COUNT TO ADJUST FOR DRIFTING BASE LINE */
       IF BASE.ADJUST=1000 THEN BEGIN;                         /* DO THE ADJUSTMENT */
          BASE.ADJUST=0;                                       /* CLEAR COUNTER */
          IF KNOB.POS<KNOB.BASE THEN KNOB.BASE=KNOB.BASE-1;    /* NUDGE IT ONE WAY */
          ELSE KNOB.BASE=KNOB.BASE+1;                          /* OR THE OTHER */
       END;

       KNOB.CHANGE=0;                                          /* START WITH ZERO */
       DELTA=KNOB.POS-KNOB.BASE;                               /* GET POSITION */
       IF DELTA<0 THEN DELTA=-DELTA;                           /* COMPUTE ABSOLUTE VALUE */
       IF DELTA>12 THEN DO;                                    /* HAVE A 12 WIDE DEAD BAND */
          DELTA=DELTA+DELTA-20;                                /* SCALE FOR MIN OF 4 */
          IF DELTA>31 THEN DELTA=31;                           /* MAX OF 31 TO PREVENT OVERFLOW */
          DELTA=SHL(4+(DELTA&3),SHR(DELTA,2)+5);
          ACCUM=ACCUM+DELTA;                                   /* ACCUMULATE MOD 16 BITS */
          IF ACCUM<0 THEN DO;                                  /* IF CARRY INTO SIGN */
             ACCUM=ACCUM-"100000";                             /* ALLOW REMAINDER */
             KNOB.CHANGE=SHR(DELTA,11)+1;                      /* CHANGE BY THIS AMOUNT */
             IF KNOB.POS<KNOB.BASE THEN KNOB.CHANGE=-KNOB.CHANGE;
          END;
       END;
       ELSE IF DELTA<8 THEN ACCUM="077777";                    /* SET UP HYSTERESIS */

       WRITE(AD.A)=2+16; RTEPEDAL.POS=240-READ(AD.A);          /* SCAN RTE PED POSITION */
       IF RTEPEDAL.POS<0 THEN RTEPEDAL.POS=0;                  /* LIMIT TO ZERO */
       IF RTEPEDAL.POS>RTE.MAX THEN RTEPEDAL.POS=RTE.MAX;      /* LIMIT TO MAX */

       WRITE(AD.A)=2+12; VOLPEDAL.POS=240-READ(AD.A);          /* SCAN VOL PED POSITION */
       IF VOLPEDAL.POS<0 THEN VOLPEDAL.POS=0;                  /* LIMIT TO ZERO */
       IF VOLPEDAL.POS>RTE.MAX THEN VOLPEDAL.POS=RTE.MAX;      /* LIMIT TO MAX */

       WRITE(AD.A)=2+8; PBI.POS=READ(AD.A);                    /* READ PITCH BEND INPUT */

       WRITE(AD.A)=2+4; I=READ(AD.A);                          /* READ ADC VALUE FOR RIBBON CONT */
       IF RIB.ACTIVE=0 THEN DO;                                /* CHECK FOR START OF RIBBON CONTROLLER */
          IF (RIB.POS>2)&(I>2) THEN RIB.ACTIVE=1;              /* START RIBBON CONT */
          RIB.BASE=I;                                          /* SAVE RIBBON CONTROLLER BASE */
       END;
       ELSE DO;                                                /* CHECK FOR END OF RIBBON CONTROLLER */
          IF I<3 THEN RIB.ACTIVE=0;                            /* RIBBON NO LONGER ACTIVE */
       END;
       RIB.POS=I;                                              /* SAVE CURRENT READING, AS OUTPUT AND FOR NEXT TIME

       WRITE(AD.A)=3;                                          /* INITIATE NEXT ADC CONVERSION */

       DO I=0 TO 7;                                            /* PUT CONTROL VOLT VALUES ONTO D164 DACS */
          WRITE("164")=I;                                      /* SELECT */
          WRITE("165")=OUT.DATA(I);                            /* PUT VALUE */
       END;
    END SCANDATA;
```

```
/*    */

/* DISPLAY PROCEDURE

.   'DISPLAY' IS CALLED TO CALCULATE THE DIGITS TO DISPLAY IN THE
    4 DIGIT LED DISPLAY ON THE CONTROL PANEL.

    'DISPLAY' IS PASSED THREE ARGUMENTS:
    WORD - A BINARY NUMBER WHOSE DECIMAL EQUIVALENT IS TO BE DISPLAYED
.   DP   - A NUMBER THAT INDICATES THE DECIMAL POINT SETTING
    UNS  - A NUMBER THAT INDICATES THE UNITS TO DISPLAY */

DCL DIGITS DATA ("077", "006", "133", "117", "146",
"155", "174", "047", "177", "147");                        /* DIGIT CODES */

DISPLAY: PROCEDURE(WORD, DP, UNS);                          /* PASS NUMBER, DP PS, UNITS */
   DCL (WORD, DP, UNS) FIXED;
   DCL POWERS DATA (1000,100,10,1);
   DCL SIGN  FIXED;                                         /* THE SIGN CONTROL BIT */
   DCL (I,J) FIXED;
   DCL LIST(3) FIXED;                                       /* STORE THE FOUR DIGITS HERE */

   SIGN=0;                                                  /* ASSUME POSITIVE NUMBER */
   IF WORD<0 THEN DO;
      WORD=-WORD; SIGN="100";                               /* SET SIGN TO - FOR DISPLAY */
      DP=1;                                                 /* ALWAYS USE DECIMAL POINT OF ONE FOR NEGATIVE NUMBE
      DO WHILE WORD>=1000; WORD=WORD/10; DP=DP-1; END;      /* ADJUST DP */
   END;
   ELSE DO;
      DO WHILE WORD>=10000; WORD=WORD/10; DP=DP-1; END;     /* ADJUST DP */
   END;
   DO I=0 TO 3;                                             /* COMPUTE 4 DIGITS */
      LIST(I)=WORD/POWERS(I);                               /* MOST SIG */
      WORD=READ(4);                                         /* READ MUL/DIV UNIT TO GET MODULO */
   END;
   J=0;                                                     /* FIND FIRST NON-ZERO DIGIT */
   DO WHILE (J<3)&(LIST(J)=0)&(J<4-DP); J=J+1; END;         /* FIND FIRST TO DISPLAY */
   DO I=J TO 3;                                             /* LOOK UP BCD'S AS REQUIRED */
      LIST(I)=DIGITS(LIST(I));
      IF I=4-DP THEN LIST(I)=LIST(I)\"200";                 /* LIGHT DECIMAL POINT IF REQUIRED */
   END;
   IF J<>0 THEN LIST(J-1)=SIGN;                             /* STORE SIGN */
   DIGDISPLAY(1)=SHL(LIST(0),8)+LIST(1);                    /* FORM OUTPUTS */
   DIGDISPLAY(0)=SHL(LIST(2),8)+LIST(3);
   DIGDISPLAY(2)=UNS;                                       /* AND SET UP UNITS BIT WORD ALSO */
END DISPLAY;

/* 'DISPLAY.ERROR' PUTS 'Err0' TO 'Err9' IN THE DISPLAY */

DISPLAY.ERROR: PROCEDURE(NUM);                              /* CALLED WITH VALUE 0 TO 9 */
   DCL NUM FIXED;
   DIGDISPLAY(0)="050000"+DIGITS(NUM);                      /* FORM THE DISPLAY */
   DIGDISPLAY(1)="074520";                                  /* 'ERR' */
   DIGDISPLAY(2)=0;                                         /* NO UNITS */
END DISPLAY.ERROR;

/*    */

/* KEYBOARD SCAN ROUTINE:  CHECKS FOR NEW KEYS */

DCL OLD.CLAVIER.K(NUM.OCTAVES-1) FIXED;                     /* OLD KEYS DOWN FOR SCAN.K */
DCL OLD.CLAVIER.R(NUM.OCTAVES-1) FIXED;                     /* OLD KEYS DOWN FOR SCAN.R */

SCAN.OCTAVE: PROC(NEWKEYS,OCT,LIST,LIST.PTR,LIST.SIZE);
   DCL LIST      ARRAY;
   DCL (NEWKEYS,OCT,LIST.PTR,LIST.SIZE) FIXED;
   DCL (I,MASK) FIXED;
   MASK=1;                                                  /* SET UP MASK */
   DO I=0 TO 11;                                            /* KEYS IN OCTAVE */
      IF (NEWKEYS&MASK)<>0 THEN DO;                         /* GOT NEW KEY */
         IF LIST.PTR>LIST.SIZE THEN RETURN LIST.PTR;        /* LIST FULL? */
         LIST(LIST.PTR)=OCT*12+I;                           /* PUT IN LIST */
         LIST.PTR=LIST.PTR+1;                               /* COUNT IT */
      END;
      MASK=SHL(MASK,1);                                     /* MOVE ON TO NEXT BIT */
   END;
   RETURN LIST.PTR;                                         /* RETURN NEW PTR */
```

```
END SCAN.OCTAVE;

SCAN.KEYBOARD:  PROCEDURE(LIST,LIST.SIZE);                 /* PASS LIST AND SIZE OF LIST */
   DCL LIST                  ARRAY;                        /* RETURNED LIST OF NEW KEYS */
   DCL (LIST.PTR,LIST.SIZE) FIXED;
   DCL (I,NEWKEYS)           FIXED;
   LIST.PTR=0;                                             /* NO NEW KEYS */
   DO I=0 TO NUM.OCTAVES-1;                                /* LOOP OVER ALL OCTAVES */
      NEWKEYS=CLAVIER(I) & (NOT(OLD.CLAVIER.K(I)));
      OLD.CLAVIER.K(I)=CLAVIER(I);                         /* SAVE PREVIOUS */
      IF NEWKEYS<>0 THEN
      LIST.PTR=SCAN.OCTAVE(NEWKEYS,I,LIST,LIST.PTR,LIST.SIZE);
   END;                                                    /* OF LOOP ON OCTAVES */
   RETURN LIST.PTR;                                        /* NUMBER OF NEW KEYS */
END SCAN.KEYBOARD;

SCAN.RELEASE:  PROCEDURE(LIST,LIST.SIZE);                  /* PASS LIST AND SIZE OF LIST */
   DCL LIST                  ARRAY;                        /* RETURNED LIST OF RELEASED KEYS */
   DCL (LIST.PTR,LIST.SIZE) FIXED;
   DCL (I,NEWRELS)           FIXED;
   LIST.PTR=0;
   DO I=0 TO NUM.OCTAVES-1;                                /* LOOP OVER ALL OCTAVES */
      NEWRELS=(NOT(CLAVIER(I))) & OLD.CLAVIER.R(I);
      OLD.CLAVIER.R(I)=CLAVIER(I);                         /* SAVE PREVIOUS */
      IF NEWRELS<>0 THEN
      LIST.PTR=SCAN.OCTAVE(NEWRELS,I,LIST,LIST.PTR,LIST.SIZE);
   END;                                                    /* OF LOOP ON OCTAVES */
   RETURN LIST.PTR;                                        /* NUMBER OF RELEASED KEYS */
END SCAN.RELEASE;

/* INITIALIZATION ROUTINE */

MAXIO.INITIALIZE:  PROCEDURE;
   DCL I FIXED;
   DO I=0 TO NUM.SW.PANS-1;  DISPLAYSW(I)=0;  END;        /* CLEAR THE LIGHTS */
   DO I=0 TO NUM.OCTAVES-1;  DIGDISPLAY(I)=0;  END;       /* CLEAR DISPLAY */
   CALL SCANDATA;                                          /* SET UP THE SCANNING */
   DO I=0 TO NUM.OCTAVES-1;
      OLD.CLAVIER.K(I)=CLAVIER(I);  OLD.CLAVIER.R(I)=CLAVIER(I);
   END;
   WRITE(AD.A)=3;  WRITE(AD.A)=2;                          /* START CONVERSION */
   KNOB.POS=READ(AD.A);                                    /* READ INITIAL KNOB POSITION */
   KNOB.BASE=KNOB.POS;                                     /* SAVE THIS AS NEUTRAL POSITION */
   WRITE(AD.A)=2+8;  PBI.BASE=READ(AD.A);                  /* GET NEUTRAL PBI */
   RIB.ACTIVE=0;                                           /* RIBBON IS NOT ACTIVE */
END MAXIO.INITIALIZE;

CALL MAXIO.INITIALIZE;
```

```
/* MAX MULTI-TASKING EXEC   MODIFICATION LEVEL 2   25 MAR 82  */
/* PRINCIPAL DEVELOPER:  CAMERON JONES                        */
/* COPYRIGHT 1982  NEW ENGLAND DIGITAL CORPORATION            */

/* Users are requested to read carefully the explanations and pitfalls
listed in the User's Guide in the Chapter for 'MAXTASK'.           */

/* Global Variables:

The following literal declarations are used to allocate tables and
storage areas within MAXTASK. */

dcl num.tasks  lit '18';                      /* space reserved for this many tasks */
dcl len.stack  lit '64';                      /* push down stack length for each task */
dcl n.automat  lit '6';                       /* number of automatic vars for each task */
dcl stack.reg  lit '"315"';                   /* Scientific XPL's Push Down Stack Register */
dcl stack.mem  lit '"355"';                   /* Top-most word on Scientific XPL's stack */
dcl stack.minc lit '"375"';                   /* Stack top with increment */
dcl pc.reg     lit '"317"';                   /* Scientific XPL's Program Counter Register */
dcl call.instr lit '"736"';                   /* Scientific XPL's CALL instruction */
dcl store.regs lit '"150230"';                /* Sub call to store registers */
dcl alltsk.err lit '4';                       /* error type for all tasks terminated */

/* interrupt handler for clock ticks */
dcl cur.time  fixed;                          /* used to count clock units */
when d03int then cur.time=cur.time+1;         /* keep track of time here */

/* standard error message routine (assumes that a terminal is present) */
ERROR: PROC(NUMBER);                          /* PRINT ON TERMINAL */
   dcl number fixed;
   disable;                                   /* stop interrupts */
   print; print 'MAXTASK Error',;
   if number=alltsk.err then
   print '/Warning Type ',number,':  All Tasks Terminated.';
   else print ' Type ',number,' Encountered.   Program Halted.';
   call exit(-1);
END ERROR;
```

```
/*    */

/* Storage Management:

A linked list is used to organize the task list.   The following
definitions are provided: */

dcl mt.fptr  (num.tasks) fixed;                                 /* holds forward  pointer  */
dcl mt.time  (num.tasks) fixed;                                 /* holds start    time     */
dcl mt.strt  (num.tasks) fixed;                                 /* holds starting location */
dcl mt.stkp  (num.tasks) fixed;                                 /* push  stack    pointer  */
dcl mt.stkt  (num.tasks) fixed;                                 /* push  stack    top      */
dcl mt.stak  (num.tasks*(len.stack+n.automat+1)) fixed;         /* holds stack area */
dcl (block.ptr,free.ptr) fixed;                                 /* pointers */

dcl tlist.ptr fixed;                                            /* currently executing block */

/* automatic vars are just after stack area in currently exec block */
dcl automatic1  lit 'lit ''core(mt.stkt(tlist.ptr)+1)'' ';
dcl automatic2  lit 'lit ''core(mt.stkt(tlist.ptr)+2)'' ';
dcl automatic3  lit 'lit ''core(mt.stkt(tlist.ptr)+3)'' ';
dcl automatic4  lit 'lit ''core(mt.stkt(tlist.ptr)+4)'' ';
dcl automatic5  lit 'lit ''core(mt.stkt(tlist.ptr)+5)'' ';
dcl automatic6  lit 'lit ''core(mt.stkt(tlist.ptr)+6)'' ';

GET.BLOCK: proc;                                                /* procedure to get a block.  returns global ptr
   if free.ptr=0 then call error(1);                            /* too many tasks */
   block.ptr=free.ptr;
   free.ptr=mt.fptr(block.ptr);
END GET.BLOCK;

REL.BLOCK: proc;                                                /* procedure to release 'block.ptr' */
   mt.fptr(block.ptr)=free.ptr;
   free.ptr=block.ptr;
END REL.BLOCK;
```

```
/*    */

/* THE FOLLOWING LITERALS ARE USED TO EFFECTIVELY ADD NEW STATEMENTS
TO XPL WHICH ENABLE A PROCEDURE TO BE GIVEN THE NAME OF ANOTHER
PROCEDURE.  FOR EXAMPLE, A STATEMENT 'START taskname TASK;' BECOMES:

DO; CALL MAXTASK.START; CALL taskname; END;

IN WHICH THE MAXTASK.START ROUTINE CAN PICK UP THE TASK'S ADDRESS
FROM THE SECOND CALL STATEMENT AND THEN SKIP OVER THE CALL.  THE
'DO' AND 'END' SERVE TO BRACKET THE TWO CALLS.  SEE THE MANUAL FOR
USAGE OF THE 'SET' OPTION IN A START STATEMENT.  */

DCL START     LIT 'DO; CALL MAXTASK.START; CALL ';
DCL KILL      LIT 'DO; CALL MAXTASK.KILL; CALL ';
DCL TERMINATE LIT 'DO; CALL MAXTASK.TERMINATE';
DCL SET       LIT '; CALL MAXTASK.SET';
DCL TASK      LIT '; END';

/* TASK START ROUTINE:

TASKS ARE STARTED BY THE STATEMENT:    START taskname TASK;     */

MAXTASK.START: PROC;                                        /* procedure to start a task */
   dcl (i,j) fixed;
   i=read(stack.mem)-1;                                     /* pick up pointer to our return */
   j=core(i)+1;                                             /* pick up pointer to next instructions */
   if core(j)<>call.instr then call error(2);               /* incorrect call to start */
   j=core(j+1);                                             /* pick up pointer to procedure to initiate */
   core(i)=core(i)+2;                                       /* skip over that call on our return */
   call get.block;                                          /* get a block of storage */

   if tlist.ptr=0 then do;                                  /* if nothing in list, put ourselves there */
      mt.fptr(block.ptr)=0;tlist.ptr=block.ptr;
   end;
   else do;                                                 /* link ourselves after current block */
      mt.fptr(block.ptr)=mt.fptr(tlist.ptr);                /* set up our forward pointer */
      mt.fptr(tlist.ptr)=block.ptr;                         /* link ourselves on immediately after current task *
   end;

   mt.time(block.ptr)=cur.time;                             /* indicate our time */
   mt.strt(block.ptr)=j;                                    /* indicate where in memory procedure is */
   mt.stkp(block.ptr)=addr(mt.stak((block.ptr-1)*(len.stack+n.automat+1))); /* compute stack location */
   mt.stkt(block.ptr)=mt.stkp(block.ptr)+len.stack;         /* compute stack top for error checking */
   core(mt.stkp(block.ptr))=0;                              /* set return location to zero to start procedure */
END MAXTASK.START;

MAXTASK.SET: PROC(V1,V2,V3,V4,V5,V6);                       /* PRESET AUTOMATIC VARS */
   dcl (v1,v2,v3,v4,v5,v6) fixed;
   core(mt.stkt(block.ptr)+1)=v1;
   core(mt.stkt(block.ptr)+2)=v2;
   core(mt.stkt(block.ptr)+3)=v3;
   core(mt.stkt(block.ptr)+4)=v4;
   core(mt.stkt(block.ptr)+5)=v5;
   core(mt.stkt(block.ptr)+6)=v6;
END MAXTASK.SET;
```

```
/*   */

/* SUSPEND is called from a task to suspend execution of that task.
The argument specifies the time in milliseconds for which the task
is to be suspended. i.e., CALL SUSPEND(500); means half a second. */

dcl exec.stack               fixed;                          /* Exec's stack pointer */
dcl maxtask.clock.divisor fixed;                             /* clock rate */

SUSPEND: PROC(TIME);                                         /* suspend current task */
   dcl time  fixed;                                          /* the time in milliseconds */
   dcl ticks fixed;                                          /* the time in clock ticks */
   dcl backp fixed;                                          /* holds back pointer */
   ticks=time/maxtask.clock.divisor;                         /* find number of ticks */

   if tlist.ptr=0 then tlist.ptr=block.ptr;                  /* tlist=0 when other task was terminated */
   else do;                                                  /* suspend current task */
      block.ptr=tlist.ptr;                                   /* get pointer to currently executing block */
      mt.stkp(block.ptr)=read(stack.reg);                    /* read current stack pointer & store it */
      if mt.stkp(block.ptr)>mt.stkt(block.ptr) then call error(3); /* stack length exceeded */
      mt.time(block.ptr)=mt.time(block.ptr)+ticks;           /* compute next starting time */
      backp=addr(tlist.ptr);                                 /* initialize back pointer */
      tlist.ptr=mt.fptr(block.ptr);                          /* put next block on front of que, unlinking ourselv
      do while (core(backp)<>0)&(mt.time(core(backp))<mt.time(block.ptr)); /* position next block */
         backp=addr(mt.fptr(core(backp)));
      end;
      mt.fptr(block.ptr)=core(backp);                        /* link our front */
      core(backp)=block.ptr;                                 /* position our block on que */
   end;

   /* now wait for time */

   do while cur.time<mt.time(tlist.ptr);                     /* wait for time to occur */
   end;

   /* start next task */

   write(stack.reg)=mt.stkp(tlist.ptr);                      /* set up stack pointer */
   if read(stack.mem)=0 then do;                             /* indicates start of a procedure */
      write(stack.minc)=core(exec.stack+1);                  /* get exec return location */
      write(stack.mem)=read(stack.reg)-2;                    /* back pointer */
      if core(mt.strt(tlist.ptr))=store.regs then do;        /* must put extra stack frame */
         core(read(stack.reg)+7)=read(stack.reg)             /* set up back pointer */
         write(stack.reg)=read(stack.reg)+7;                 /* update stack pointer */
         write(pc.reg)=mt.strt(tlist.ptr)+1;                 /* and enter procedure */
      end;
      else write(pc.reg)=mt.strt(tlist.ptr);                 /* start from top of procedure */
   end;
   return;                                                   /* else just return to continue with procedure */
END SUSPEND;
```

```
/*    */

/* Two facilties are provided for terminating the execution of a task.
To terminate the current task, use the statement:  TERMINATE TASK;
To terminate another task, use the statement:  KILL taskname TASK;
These literals will call upon the appropriate routine below.  */

MAXTASK.TERMINATE: PROC;                                         /* procedure to terminate a task */
   block.ptr=tlist.ptr; tlist.ptr=mt.fptr(block.ptr);            /* unlink ourselves from tlist */
   call rel.block;                                               /* release the block of storage */
   block.ptr=tlist.ptr; tlist.ptr=0;                             /* set up for special call to wait */
   if block.ptr=0 then call error(alltsk.err);                   /* all tasks terminate - we are done! */
   call suspend(0);                                              /* and we are done */
END MAXTASK.TERMINATE;

MAXTASK.KILL: PROC;                                              /* kill another task */
   dcl (i,j) fixed;
   i=read(stack.mem)-1;                                          /* pick up pointer to our return */
   j=core(i)+1;                                                  /* get pointer to our return */
   if core(j)<>call.instr then call error(5);                    /* error */
   j=core(j+1);                                                  /* pick up pointer to procedure */
   core(i)=core(i)+2;                                            /* increment our return */
   i=addr(mt.fptr(tlist.ptr));                                   /* use i for back pointer */
   block.ptr=mt.fptr(tlist.ptr);                                 /* and start with block after ourselves */
   do while block.ptr<>0;                                        /* scan each block */
      if mt.strt(block.ptr)=j then do;                           /* match - kill it */
         core(i)=mt.fptr(block.ptr);                             /* save forward pointer */
         call rel.block;                                         /* release block */
      end;
      else i=addr(mt.fptr(block.ptr));                           /* move back pointer up to our address */
      block.ptr=core(i);                                         /* get next forward pointer */
   end;
   if mt.strt(tlist.ptr)=j then call maxtask.terminate;          /* we killed ourselves */
   else return;                                                  /* otherwise return to continue processing */
END MAXTASK.KILL;
```

```
/*   */

/* initialization code */

dcl main procedure;                                          /* forward reference to user's main task */

   free.ptr=1;                                               /* first free block is block #1 */
   do block.ptr=1 to num.tasks-1;                            /* set up list */
      mt.fptr(block.ptr)=block.ptr+1;
   end;

   /*if shr(core(core(1)+14),1)&1 then*/ maxtask.clock.divisor=5; /* set rate */
   /*else maxtask.clock.divisor=10;*/

   start main task;                                          /* start the user's main task */
   enable;                                                   /* start the ball game rolling */
   exec.stack=read(stack.reg);                               /* save pointer to exec's push down stack */
   block.ptr=tlist.ptr; tlist.ptr=0; call suspend(0);        /* terminate ourselves */
   terminate task;                                           /* tasks return here if they user 'return'. TERMINATE
                                                                                                          THEM
```

Click Track and External Clock Synchronization.

Three procedures are provided in MAXSYN to allow the user to produce a click track, and read from and write to the external clock. This allows you to utilize the external clock input and output in arbitrary ways, such as for specialized timing and synchronization applications.

Generating a Click Track.

The EMIT.CLICK procedure will produce a click on the click track output when it is called. Thus, a click track could be produced by the following statements

```
130  DO WHILE 1; /* REPEAT FOREVER */
140     CALL EMIT.CLICK; /* PUT OUT A CLICK */
150     CALL WAIT(1000); /* WAIT A SECOND BEFORE NEXT CLICK */
160  END;
```

This would produce a click every second. Clearly, more complex patterns of clicks could be produced by a more complex timing cycle.

Writing to the external clock.

The external clock output may be toggled by the user by calling the TOGGLE.EXT.CLOCK procedure. This means that the external clock output will switch from "high" to "low", or "low" to "high". Since most external devices reading the clock output will detect the edges rather than the levels, this enables the user to produce an edge. The "high" and "low" levels are approximately 1.5 volts peak to peak, bipolar.

For example, the following routine would produce a 15 ms pulse every 500 ms

```
170  DO I=0 to 9999; /* TICK COUNTER */
180     IF (I MOD 100)=0 THEN CALL TOGGLE.EXT.CLOCK; /* EDGE UP */
190     IF (I MOD 100)=3 THEN CALL TOGGLE.EXT.CLOCK; /* EDGE DOWN */
200     CALL WAIT(5); /* WAIT 5 MS */
210  END;
```

The clock is toggled at 0,3,100,103,200,203, and so on, with 5 ms between each tick. The initial value is "low", so this will cause a change to "high" at 0 ms, followed by a toggle back to "low" 3 ticks later, or 15 ms.

Reading from the external clock.

The current state of the external clock input may be obtained from the EXT.CLOCK.IN function. This may be placed in a variable as in the statement

```
220  X = EXT.CLOCK.IN; /* STORE THE CURRENT CLOCK INPUT IN 'X' */
```

The external clock input is expecting an approximately square bipolar wave about 1.5 volts peak to peak. The value obtained will be 1 when the clock input is "high", and 0 when the input is "low". Since the external clock input is not latched in the input circuitry, the input may float after each change in the input pulse. Thus the software should latch the input when it changes, and save it to detect further edges.

If the External Clock Out and In jacks are connected together on a system, AC coupling between the two I/O drivers will cause the External Clock In to float to "high" within 10 ms after the pulse. Thus testing of programs in thus manner may cause spurious results.