Chapter Five
The Sound Generator

One of the attractions of the BBC Micro as a machine to have fun with is the presence of a sound generator chip with one noise channel and three tone channels. Just this hardware alone would lead you to expect to be able to produce three note chords and a range of simple sound effects. However, the software that is built into the MOS and the BASIC to handle it makes it a lot more powerful than the hardware specification might lead you to believe. By the clever use of interrupts and a system of queues the BBC Micro can make sounds and move things around the screen, etc. at the same time! In addition, the ENVELOPE command gives the BASIC programmer an amazingly high degree of control over the nature of the sound produced. Once again, the combination of good hardware enhanced by well thought out software makes the BBC Micro remarkable!
In this chapter we will take a closer look at the sound generator and the sound generating software inside the BBC Micro. Some of the discussion will be about the sound generator hardware itself and this will be of particular interest to the assembly language programmer. However, the first part of the chapter deals with the software - the SOUND and ENVELOPE commands - how they work and what they can be used for.

An overview

Before becoming too deeply involved in the details of using the sound generator it is worth taking an overview of the facilities it provides. There are three tone generators that can be used to produce either single notes or up to three-note chords. There is, in addition, a single noise channel that can produce eight different effects. This fairly simple hardware is controlled using two extensions to BASIC - SOUND and ENVELOPE. The SOUND command is the only one of the pair that actually causes anything to come out of the tiny speaker

just above the keyboard. Among other things, it controls the pitch, amplitude and duration of the notes produced. The ENVELOPE command is used to change the characteristics of the notes produced by the SOUND command. Used without the ENVELOPE command, SOUND produces a more or less pure tone with a given frequency, which is fine for most applications, e.g. beeps during games or playing simple tunes. However, if you want to try to produce more complicated sounds then you have to use the ENVELOPE command to alter the basic sound produced. There are two general reasons for wanting to produce more complex noises - either you are interested in music and making your BBC Micro sound like a piano, a flute, an organ, a guitar . . . or you want to make especially convincing sound effects such as a police siren, a gun shot, etc.

The study of the BBC Micro's sound capabilities, therefore, falls into these two categories - music and sound effects.

There are three levels of difficulty involved in making music with the BBC Micro;

1. Playing simple tunes.
2. Playing music with three-part harmony.
3. *Synthesising* the sound of other instruments.

The first two involve the use of only the SOUND command but the last one also needs a mastery of the ENVELOPE command. To get very far with any of the three you also need a reasonable understanding of music but if you feel a little unsure about this then programming sound is a very enjoyable way to learn.

The subject of sound effects is much more limited because all that we are trying to do is to compile a catalogue of 'recipes' to make a few standard noises. However, there are two ways of approaching sound effects. You can either use the SOUND command to control the noise channel or you can use the ENVELOPE command to define basic sounds. With the latter you can produce quite remarkable effects but there's still a great deal of scope for producing a wide variety of noises using the SOUND command, and it can fill the requirements of most games playing applications alone.

The SOUND command

The SOUND command has the general form

SOUND C,A,P,D

where C controls which channel - 0 (the noise channel), 1, 2 or 3 - produces the sound; A controls the volume and ranges from 0 (silence) to -15 (loudest); P controls the pitch of the note and ranges from 0 (lowest pitch) to 255 (highest); and D controls the duration of the note and ranges from 1 to 255 in twentieths of a second. (However, it is worth noticing that if D is set to -1 then the note produced will continue to sound until you take steps to stop it!) There are various extra meanings associated with the parameters C and A. Positive values of A in the range 1 to 4 cause the pitch and volume of the note to be controlled by the parameters of an ENVELOPE command. The channel parameter C is in fact quite complicated and is best thought of as a four-digit hexadecimal number

&HSFN

where each of the letters stands for a digit that controls a different aspect of sound production. What exactly each of them does is better left until later except to say that N is the channel number as described earlier.

Programming tunes is simply a matter of converting notes into numbers. This is easy once you know that middle C corresponds to a value of 53 and going up or down by a whole tone corresponds to adding or subtracting 8. The only thing that you have to be careful to remember is that there isn't always a whole tone between two notes. For example, between the notes of C and D there is a whole tone but between E and F there is only a semi-tone. The pattern of tones and semi-tones from C to C an octave above is

```
C - D - E - F - G - A - B - C
  T   T   S   T   T   T   S
```

which is easy to remember because it's the same as the pattern of white and black notes on the piano. Obviously, sharps and flats can be produced by adding or subtracting 4. So you can produce the full chromatic scale by

```
10 FOR P=53 TO 97 STEP 4
20 SOUND 1,-15,P,10
30 NEXT P
```

This short program can also be used to demonstrate a unique feature of the BBC Micro. If you add line 15:

```
15 PRINT P
```

you will discover that the numbers are printed on the screen and even though the program finishes, the sound keeps on coming. The reason for this remarkable behaviour is that the BBC Micro maintains a queue of sounds that are produced one after the other as soon as the current sound is completed. The sound queue is processed independently of any BASIC program that is running and each SOUND statement simply adds a note to the end of the queue. This means that a BASIC program isn' held up for the duration of each note. The only time that this fails is when the queue becomes full and a SOUND statement tries to add another note to it. The result is that the program then has to wait until the end of the currently sounding note when the queue is reduced by one and the SOUND statement can add its note. There is a separate queue for each channel and each can hold up to four notes.

Programming tunes

To make a tune recognisable, not only must it have each note at the right pitcn, each note must also last for the correct time, The normal system of musical notation is based on repeatedly dividing a time interval by two to obtain shorter notes so it is a good idea to include a variable in all music programs that sets the length of the fundamental unit of time. As an example of programming a simple tune consider the first few notes of *Hearts of Oak* (see Figure 5.1). Translating each note to its pitch and duration value for the SOUND statement gives the two rows of numbers under the music in Figure 5.1. The best way to convert these numbers to sound is to use a DATA statement thus:

```
 5 C=5
10 DATA 69,1,89,1,89,.75,89,.25,89,1,105,.75,
    97,.25,89,1,85.75,77.25,69,.75,99,99
20 READ P,D
30 IF P=99 THEN STOP
40 SOUND 1,-15,P,D*C
50 SOUND 1,1,P,2
60 GOTO 20
```

Line 50 has the effect of leaving short silences between each of

the notes. Without this line all the notes run together. Try deleting it and re-running the program to appreciate the effect - it is one that you' d want to use to ' slurhotes. You can program any tunes that you have music for in the same way.

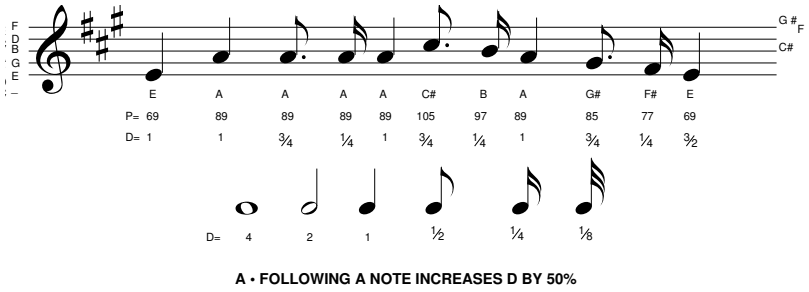


Fig. 5.1. The first few notes of *Hearts of Oak* and their digital values for the SOUND command. (Reprinted by permission of *Computing Today*.)

Three note chords

Most home computers with a sound generator could manage the simple tune given in the last section. What is special about the BBC Micro is that it is possible to generate three notes at the same time. To see how this sounds, try the following:

```
10 DIM N(13)
20 DATA 53,61,69,73,81,89,88,101,109,117,
      121,129,137
30 FOR I=1 TO 13
40 READ N(I)
50 NEXT I
60 A$=INKEY$(0)
70 IF A$="" THEN GOTO 60
80 A=VAL(A$)
90 SOUND 1,-15,N(A),20
100 SOUND 2,-15,N(A+2),20
110 SOUND 3,-15,N(A+4),20
120 GOTO 60
```

If you RUN this program, by pressing each of the keys l to 8 you will be able to hear the eight chords produced by adding a third and a fifth to each of the notes of the scale of C. (A third is a musical interval corresponding to playing a note two notes higher up the scale and a

fifth corresponds to playing a note four notes higher up.) This is the simplest kind of chord, called a *triad*, and is very pleasing to the ear. Typing in almost any combination of the number keys 1 to 9 will produce something tuneful and it is easy to sit at your BBC Micro and produce music. For example, if you want to hear a snatch of tune that is almost recognisable try typing in the following sequence.

```
5 5 6 6 4 5 7 7 8 7 6 5
```

No prizes for guessing this one! The array N is used to hold the pitch values for the notes of the scale of C and enough notes higher up to form the triad on B. You can write a program to play a piece of music with up to three-note chords using the same method as given for the single melody in the last section.

There is one thing wrong with the previous program and that is that each note of the chord starts at a slightly different time. In other words, each of the SOUND commands starts off its note in the chord as soon as it is reached. As they are executed one after another, the note on channel 1 starts a little before that on channel 2, which starts a little before that on channel 3. The solution to this problem would be to tell the sound generator to wait for two other notes after the one initiated by line 90 before making any noise at all. This is the purpose of the S part of the channel parameter introduced in the section about the form of the SOUND command. If you use a non-zero value for S, the sound generator will wait for other notes before it starts playing. The number of notes that it waits for is given by the value of S and the SOUND commands that produce them must also use the same value of S. For example, in the case of the triads played by the previous program the SOUND commands would be replaced by

```
 90 SOUND &0201,-15,N(A),20
100 SOUND &0202,-15,N(A),20
110 SOUND &0203,-15,N(A+4),20
```

The first SOUND command has a value of S equal to 2 so the sound generator waits for two more SOUND commands with S set to 2 before producing a chord made up of all three notes.

The other parts of the channel parameter are also concerned with the timing of notes. The H part of the parameter can either be a 0 or a l. If it is a 1, it adds a dummy note to the sound queue that allows any previous notes to continue without being cut short by another note.

This really only makes any sense when used with the ENVELOPE command. The F part can be either 0 or 1 and if it is 1 it causes any notes stored in the channel's queue to be removed or 'flushed' and the note specified by the current SOUND command to be produced immediately. This is useful for cutting short sound effects and starting new ones, synchronised with external effects. For example, in a graphics game you might want to stop the noise of a fire gun and replace it by an explosion.

Simple sound effects

The only sound channel that we haven't discussed as yet is the noise channel - Channel 0. The noise produced by this channel depends on the value of the pitch parameter P in the SOUND command:

Value of P	*Noise*
0	High frequency periodic.
1	Medium frequency periodic.
2	Low frequency periodic.
3	Periodic of a frequency set by channel 1.
4	High frequency 'white' noise.
5	Medium frequency 'white' noise.
6	Low frequency 'white' noise.
7	Noise of frequency set by channel 1.

The first three noises (P=0 to 2) are rasping noises that come in very handy for 'losing' noises in games! Values of P between 4 and 6 produce hissing noises of various frequencies. White noise is a special sort of hissing noise that is made up by mixing a note of every pitch in much the same way that white light is made up by mixing light of every colour.

There isn't very much that you can do to change the nature of the sounds produced when P has a value of 0,1,2,4,5 or 6 apart from altering the volume and duration. However, by changing only these two parameters and combining noises you can still produce a useful range of effects. For example, if you make any noise very short it begins to sound *percussive* (like something being hit) and if you

combine a very short burst of white noise with a very short high pitched tone you produce a noise like a metallic click. Try:

```
10 SOUND 0,-15,4,1:SOUND 1,-15,200,1
```

Similarly, mixing two noise-like sounds produces new effects. So, for example:

```
10 SOUND 0,-15,4,1:SOUND 0,-15,3,1
20 GOTO 10
```

produces a sound like a machine gun. Notice that as this example uses the same channel twice, the two sounds follow each other to give a rhythmical pulsing sound. Using this idea with two different pitches of ' white' noise produces a sound very like a helicopter:

```
10 SOUND 0,-15,4,2
20 SOUND 0,-15,5,1
30 GOTO 10
```

Notice that one of the sounds has to be twice as long to give the pulsating beat of a helicopter' s rotor blades. You can go on experimenting like this indefinitely! The range of sounds that can be produced using channel 0 alone is so great that discovering new sounds is easy. Putting a name to them is,quite a different problem!

The pitch values 3 and 7 are special because they produce noises on channel 0 that are controlled by the pitch on channel 1. This opens the door to sound effects that involve noises that change in pitch. For example.

```
10 SOUND 0,-15,7,55
20 FOR I=200 TO 255
30 SOUND 1,0,I,1
40 NEXT I
```

produces a noise like a space ship taking off. The pitch of the noise on channel 0 started by line 10 is continuously changed by line 30. Notice that using a volume of 0 means that the notes produced by line 30 are silent! Finally, try:

```
10 SOUND 0,-15,7,55
20 SOUND 1,0,200,1
30 SOUND 1,0,255,1
40 GOTO 20
```

which produces a sound like a car engine being started (or rather

failing to start!)

The ENVELOPE command

The ENVELOPE command is a very sophisticated way of controlling the output of the sound generator. Without it there would be no way of producing really complicated sounds without resorting to assembly language. Part of the trouble with under-standing the way an ENVELOPE command produces a sound is that it is always used in conjunction with a SOUND command and it is these two commands together that determine the actual sound produced.

The ENVELOPE command is fairly difficult to use because it has so many different parameters and because it is difficult to see how these parameters are used to produce any desired sound. The User Guide goes into some detail about what each of the parameters actually does but it is still worth pointing out the general principle behind the operation of the ENVELOPE command.

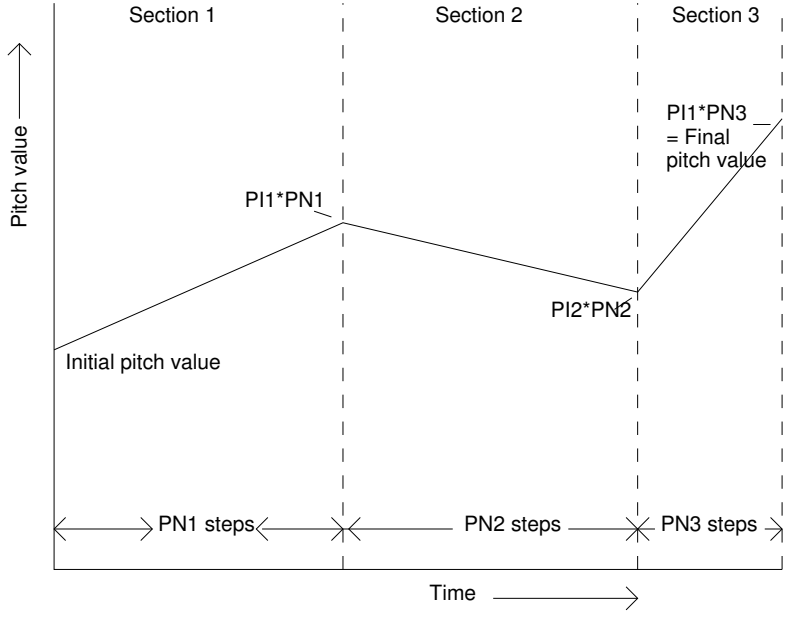


Fig 5.2. Pitch graph.

Using the notation defined on page 245 of the User Guide, the format of an ENVELOPE command is:

```
ENVELOPE  N, T,PII,PI2,PI3,PN1,PN2,PN3,
          AA,AD,AS,AR,ALA,ALD
```

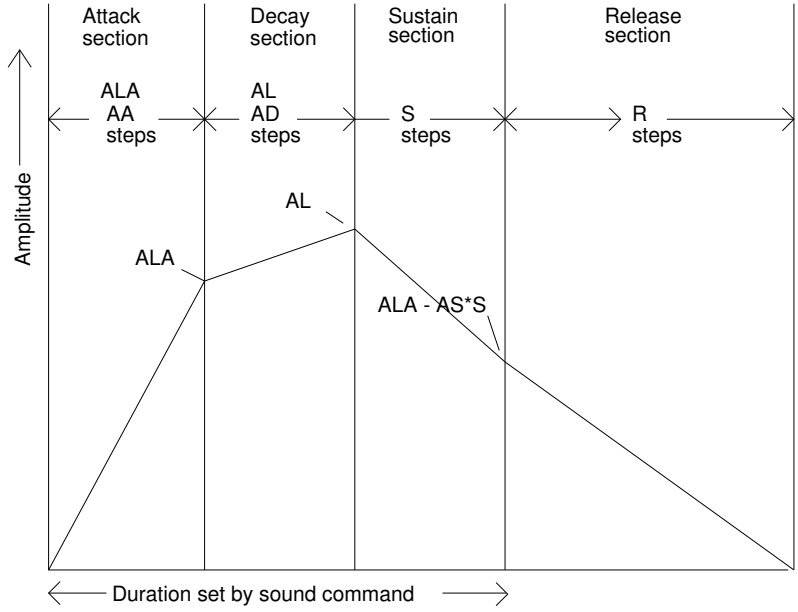
Broadly speaking, there are two types of parameter in an ENVELOPE command - pitch parameters and amplitude parameters. The pitch parameters control the variation in pitch (if any) that occur and the amplitude parameters control any variation in volume that occurs. You can think of these two sets of parameters as defining two graphs - a graph of pitch with time and a graph of amplitude with time. The way that the pitch parameters determine the pitch graph can be seen in Figure 5.2. The three parts of the pitch graph are each controlled by two parameters. PN1 to PN3 set the number of steps in each section and PI1 to PI3 set the change in the pitch value for each step in each of the sections. There are two pieces of information missing from this graph. We also need to know how long each step lasts for and the starting pitch value. The first requirement is met by the parameter T, which specifies the duration of each step in hundredths of a second. The initial pitch value is set by the pitch value in the SOUND command that refers to the envelope. It is possible that the total time specified for the pitch graph, i.e. (PN1+PN2+PN3)×step duration, is shorter than the time that the note sounds for. In this case one of two things can happen. If bit 7 of the T parameter is a one, then the pitch value remains at the final value specified by the graph for the rest of the sound. However, if bit 7 of T is zero then the pitch value returns to the initial value and the whole pitch graph is used repeatedly until the sound ends. Thus the parameter T conveys two pieces of information, the duration of a step and whether the pitch graph should ' auto repeat' .

The way that the amplitude parameters specify the amplitude graph can be seen in Figure 5.3. The amplitude graph is a little more difficult to follow than the pitch graph because the time that each section lasts isn' texplicitly stated. The attack section comes to an end when the amplitude reaches the specified attack level, ALA. During this period the amplitude, which always starts at zero unless continuing a previous note, increases or decreases by an amount given in AA at each step. The attack section is followed by the decay section. During this period the amplitude increases or decreases by an amount specified by AS at each step until it reaches the final decay level specified in ALD. Notice that the times of the attack and decay sections are not specified directly. Instead, each section lasts until the

amplitude reaches the specified final value. However, it is easy to work out how long each lasts:

Attack period = ALA/AA × step duration

Decay period = ALD/AD × step duration



Notes $S = D - \frac{ALA}{AA} - \frac{ALD}{AD}$ steps

$R = \frac{(ALA - AS*S)}{AR}$

R may be cut short by the start of another note.

Fig 5.3. Amplitude graph.

The time that the third section - the sustain section - lasts isn't set by the ENVELOPE command. The overall time that the note lasts is set by the duration specified in the SOUND command that makes use of the envelope. The sustain section lasts for however much time the note has left to sound after the attack and decay sections. During this period the amplitude decreases by an amount specified in AS at each step. The final section of the graph - the release section - is the strangest of all in that it happens after the 'official' end of the note as set by the duration

in the SOUND command. If the note isn't followed immediately by another then the amplitude continues to fall by an amount specified in AR at each step. The note finally terminates because the amplitude reaches zero or because another note starts.

Experimenting with ENVELOPE

The above description of how the ENVELOPE command works is all very well but how do you specify the values of the parameters to produce a sound of your choice? There is no easy answer to this question. Sounds have to be constructed by trial and error. To aid in this process the following program allows the parameters of an ENVELOPE command to be changed one at a time and the result heard.

```
 10 MODE 4
 20 PROCINIT
 30 PROCPRINT
 40 PROCCHANGE
 50 GOTO 40
 60 STOP

 70 DEF PROCPRINT
 80 CLS
 90 @%=&00020205:PRINT TAB(10);
    "(T) Time unit=";T/100;" s"
100 PRINT:PRINT TAB(10);"Frequency Section"
110 PRINT:@%=10
120 PRINT "(P1) Repeat =";R$
130 PRINT "(P2) Change in pitch section 1
     =";PI1
140 PRINT "(P3) Number of steps in section 1
     =";PN1
150 PRINT "(P4) Change in pitch section 2
     =";PI2
160 PRINT "(P5) Number of steps in section 2
     =";PN2
170 PRINT "(P6) Change in pitch section 3
     =";PI3
180 PRINT "(P7) Number of steps in section 3
     =";PN3
190 PRINT "(P8) Initial Pitch ";P
200 PRINT
210 PRINT TAB(10);"Amplitude Section"
220 PRINT
230 PRINT "(A1) Attack rate of change = ";AA;
    "per step"
240 PRINT "(A2) Attack target level   = ";ALA
250 PRINT "(A3) Decay rate of change  = ";AD;
    "per step"
260 PRINT "(A4) Decay target level    = ";ALD
```

```
270 PRINT "(A5) Sustain rate of change=";AS;
    "per step"
280 PRINT "(A6) Release rate of change=";AR
290 PRINT
300 PRINT "(D) Total Duration =";D/20;
    "s (=";D/20/T*100;" steps)"

310 PRINT TAB(5,28);"Press S to hear sound"

320 ENDPROC

330 DEF PROCINIT
340 T=1
350 R$="OFF":R=1
360 PI1=0
370 PI2=0
380 PI3=0
390 PN1=0
400 PN2=0
410 PN3=0
420 P=128
430 AA=10
440 AD=-10
450 AS=-20
460 AR=-10
470 ALA=100
480 ALD=10
490 D=5
500 ENDPROC

510 DEF PROCSOUND
520 ENVELOPE 1,T+128*R,PI1,PI2,PI3,PN1,PN2,
    PN3,AA,AD,AS,AR,ALA,ALD
530 SOUND 1,1,P,D
540 ENDPROC

550 DEF PROCCHANGE
560 A$=INKEY$(0)
570 IF A$="" THEN GOTO 560
580 IF A$="S" THEN PROCSOUND:GOTO 560
590 IF A$="T" THEN PRINT TAB(2,30);
    "Time unit (in secs)= ";:INPUT T:T=T*100
600 IF A$="D" THEN PRINT TAB(2,30);
    "Total Duration (in secs)= ";:INPUT D:D=D*20
610 IF A$="P" THEN PROCPITCH:GOTO 560
620 IF A$="A" THEN PROCAMP:GOTO 560
630 PROCPRINT
640 GOTO 560

650 DEF PROCPITCH
660 A$=INKEY$(0)
670 IF A$<"0" OR A$>"9" THEN GOTO 660
680 A=EVAL(A$)
690 IF A=1 THEN GOTO 790
700 PRINT TAB(2,30);"Pitch parameter ";A;
    " = ";:INPUT PP
710 IF A=2 THEN PI1=PP
720 IF A=3 THEN PN1=PP
730 IF A=4 THEN PI2=PP
740 IF A=5 THEN PN2=PP
```

```
  750 IF A=6 THEN PI3=PP
  760 IF A=7 THEN PN3=PP
  770 IF A=8 THEN P=PP
  780 GOTO 810
  790 PRINT TAB(2,30);"Repeat ON or OFF";:INPUT R$
  800 IF R$="ON" THEN R=0 ELSE R=1
  810 PROCPRINT
  820 ENDPROC

  830 DEF PROCAMP
  840 A$=INKEY$(0)
  850 IF A$<"0" OR A$>"9" THEN GOTO 840
  860 A=EVAL(A$)
  870 PRINT TAB(2,30);"Amplitude parameter ";A;" = ";:INPUT
PP
  880 IF A=1 THEN AA=PP
  890 IF A=2 THEN ALA=PP
  900 IF A=3 THEN AD=PP
  910 IF A=4 THEN ALD=PP
  920 IF A=5 THEN AS=PP
  930 IF A=6 THEN AR=PP
  940 PROCPRINT
  950 ENDPROC
```

The procedure PROCINIT sets initial values to all the ENVELOPE and SOUND parameters. PROCPRINT prints a list of all the parameters, their meaning and their current value. PROCCHANGE can be used to change the value of any of the parameters by typing the parameter's code (written in brackets on the left by PROCPRINT). To hear the sound, simply press S. Using this program you can construct and adjust sound effects very easily. Once you have the sound that you want, write down the final values for all the parameters and use them in ENVELOPE and SOUND commands in your own programs.

The sound generator hardware

The sound generator chip used in the BBC Micro is a SN 76489 bus-controlled sound generator. This chip was designed to be used directly with microprocessors. However, if you recall the discussion in Chapter One of the way that VIA-A is used as a slow data bus to various peripherals including the sound generator chip you will realise that it is not interfaced directly to the 6502's data or address bus. The SN 76489 has eight data inputs, two control inputs and one control output. The eight data lines are directly connected to the A side of VIA-A. Only one of the control lines is actually used and this is the WE (write enable) line. This is connected to output 0 of the 74LS259 addressable latch. The WE line must be low (i.e. logic zero) before any data is

transferred to the sound generator. Thus the sequence to write a single byte of information to the chip is:

1. Set the A side of VIA-A to outputs and store the data in the data register,
2. Set the WE line to logic zero, As WE is connected to output zero of the addressable latch this is achieved by setting bits 0 to 3 of the B side of VIA-A to zero, taking care not to alter the state of any of the other bits in the B side data register.
3. After 32 clock pulses the data will have been read in to the chip and the WE line must be returned to one. This is done by setting bit 4 of the B side of VIA-A to one. After this, the sound generator chip is ready for the next byte.

This procedure may seem complicated but it is very easy to write a BASIC procedure to transfer bytes to the sound generator:

```
1000 DEF PROCSOUND(BYTE%)
1010 LOCAL VIA%,TEMP%
1020 VIA%=&FE40
1030 VIA%?3=&FF
1040 VIA%?&F=BYTE%
1050 TEMP%=?VIA%
1060 ?VIA%=(TEMP% AND &F0)
1090 ?VIA%=(TEMP% AND &F8)
1100 ?VIA%=TEMP%
1110 ENDPROC
```

Line 1020 sets VIA% to the address of VIA-A. Line 1030 sets VIA-A' s A side to outputs (see Chapter Six for more explanation of the VIA' s control registers). Line 1040 stores the data in the data register and lines 1060 and 1 090 change the WE line to zero and then back to one. Notice the use of line 1050 and the variable TEMP% to avoid altering the state of any other of the outputs. The only part of the procedure that is left to chance is the time between setting the WE line to zero and then back again to one. As BASIC is slow compared to the 4 MHz clock rate fed to the sound generator chip it is safe to assume that at least 32 clock pulses occur between line 1060 and 1090. If you convert this procedure into an assembly language routine then it would be necessary to add a pause while the 32 clock pulses happened.

The only extra information necessary to control the sound generator is the format of the data bytes. This is easier to understand after looking a little more at how the sound generator chip works.

Each of the tone generators takes the form of a 10-bit counter connected to a four-stage attenuator. The output of each attenuator is summed together to produce the audio signal(see Figure 5.4). The

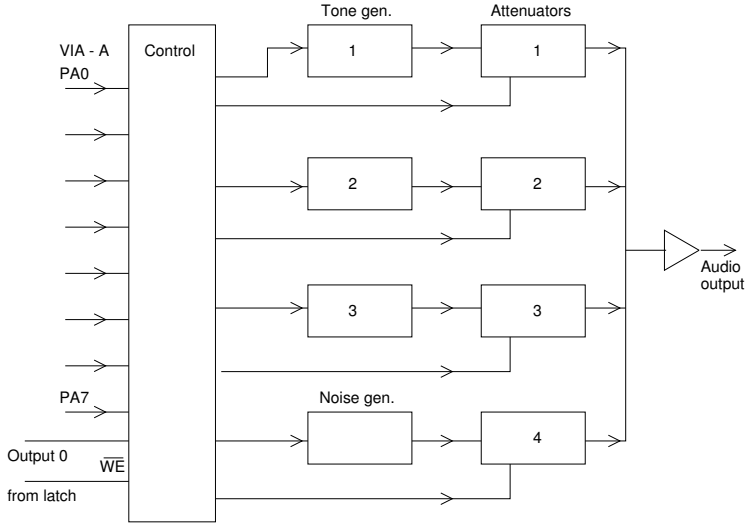


Fig. 5.4. Block diagram of SN76489AN sound generator

counter is decremented once every 16 clock pulses and whenever it reaches zero its output to the attenuator changes state, i.e. if it was a high it will go low and vice versa. To set the frequency of the output signal the counter can be loaded with a 10-bit binary number which is reloaded automatically each time the counter reaches zero. Thus, the frequency of the square wave that is fed to the attenuator is given by:

$$f = N/32n$$

where N is the input clock frequency in Hz and n is the 10-bit binary number. (On the BBC Micro, N=4 MHz.) Each stage of the four-stage attenuator can be switched on and off individually. The stages give 2dB, 4dB, 8dB and 16dB attenuation and selecting all four stages turns the output of the tone generator off. Obviously, specifying the attenuation requires four bits, one for each stage. The noise generator is also connected to the audio output via a four-stage attenuator. The

noise generator itself takes the form of a shift register with a feedback loop that can either be configured to produce a roughly periodic signal or a sequence of pseudo-random bits. The pseudo-random sequence when fed into the audio signal is a good approximation to white noise - a sound that should contain all frequencies at the same time! In addition the rate at which the shift register is clocked can be set to one of three preset frequencies or the output of tone generator three can be used.

The sound generator chip has eight internal registers which control the three tone generators and their associated attenuators and the noise generator and its attenuator. Each byte of information sent to the sound generator chip contains a three-bit address used to select which register the information is destined for. The only complication is the frequency information for the tone generators. Obviously, to send a 10-bit number to a tone generator will need two bytes and hence two calls to the procedure PROCSOUND given above. The first byte contains the register address and four bits of the 10-bit number. The

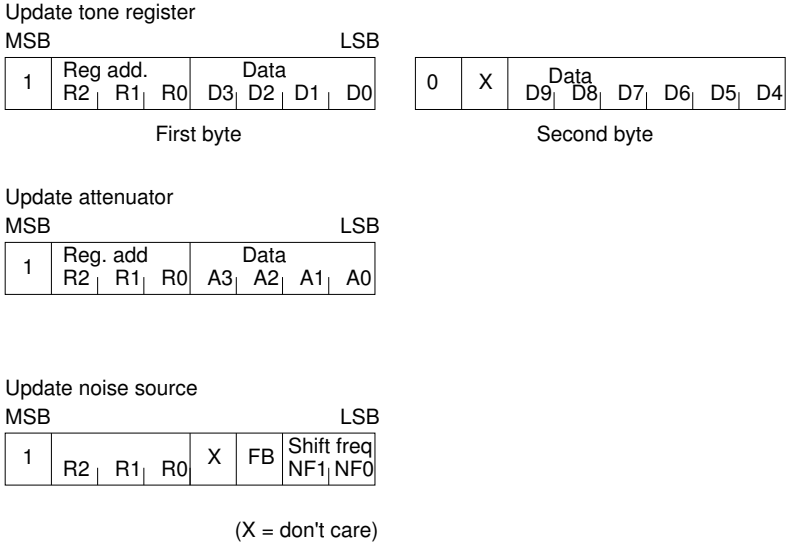


Fig. 5.5. Data formats for sound generator.

second byte carries the remaining six bits of the number and is distinguished from all the others by having its most significant bit set to zero. The data formats used to write to each type of register can be seen in Figure 5.5. The appropriate register address bits select the register as follows:

R2	R1	R0	Register
0	0	0	Tone 1 frequency.
0	0	1	Tone 1 attenuation.
0	1	0	Tone 2 frequency.
0	1	1	Tone 2 attenuation.
1	0	0	Tone 3 frequency.
1	0	1	Tone 3 attenuation.
1	1	0	Noise control.
1	1	1	Noise attenuation.

If an attenuation register is selected then the format of the attenuation bits is:

A3	A2	A1	A0	Attenuation
0	0	0	0	0dB
0	0	0	1	2dB
0	0	1	0	4dB
0	1	0	1	8dB
1	0	0	0	16dB
1	1	1	1	OFF

If the noise control register is selected, then, if FB=0, the result is periodic noise and, if FB=1, the result is white noise. The two bits NFO and NF1 control the frequency of the shift register' sclock as indicated below:

NF1	NF0	Shift rate
0	0	N/512
0	1	N/1024
1	0	N/2048
1	1	Tone generator 3' s output

As an example, consider the problem of using PROCSOUND to produce a white noise at full volume. This implies sending two bytes to the sound generator - the first setting the noise attenuator to 0dB and the second setting the noise control register to white noise. Thus the

format of the first byte is 11110000. The first l is always present before a register address, the next three bits form the register address and the last four bits form the desired attenuation. Changing this to hex gives &F0 as the first byte to be sent using PROCSOUND. By similar reasoning the second byte works out to 11100100 or &E4. Thus the final program is:

```
10 PROCSOUND(&F0)
20 PROCSOUND(&E4)
30 STOP
```

(To which should be added PROCSOUND, of course.) As a final example, consider the following program:

```
10 PROCSOUND(&90)
20 PROCSOUND(&80)
30 PROCSOUND(RND AND &DF)
40 GOTO 30
```

The first byte sets the attenuation on tone generator 1 to OdB. The second byte sets the first four bits of the 10-bit number sent to tone generator 1 to zero. The third byte sent by line 30 supplies the final six bits of the number at random. (RND AND 8r.DF) generates a six-bit number. Lines 30 and 40 form a loop that repeatedly sends new random values for the six bits. It is a feature of the sound generator chip that any values that begin with zero will update the six bits of the last tone register selected. The result of this is that short random tones are generated until you press ESCAPE.

This description of the sound generator hardware within the BBC Micro should convince you how clever and convenient the SOUND and ENVELOPE commands are. For example, the sound generator chip has no facility for specifying the duration of a note. A note continues to sound until software turns it off using the attenuator setting. The BBC Micro uses the regular timer interrupt as an opportunity to see if the duration of a note specified in a SOUND command has been completed. If it has not the sound generator is left alone; if it has then the tone is switched off. By using interrupts in this way the BBC Micro can appear to be getting on with something else while the sound generator produces sounds! The action of the ENVELOPE command is also based on interrupts. Each envelope specifies a time unit after which the sound generator is updated. For example, if an envelope specifies a *step size* of one hundredth of a

second then the attenuation and frequency of the selected tone generator is updated at each timer interrupt.

Conclusion

There is enough material concerning the BBC Micro' s sound generator and sound generating software for a book dealing with nothing else! In this brief introduction there should be sufficient information to suggest many interesting and enjoyable uses of this remarkable facility. Whether you are interested in making music or programming impressive sound effects you will find enough scope for experimentation.