CHAPTER 4

PROGRAMMING SOUND AND MUSIC ON YOUR COMMODORE 64

INTRODUCTION

Your Commodore computer is equipped with one of the most sophisticated electronic music synthesizers available on any computer. It comes complete with three voices, totally addressable, **ATTACK/DECAY/SUSTAIN/RELEASE (ADSR)**, filtering, modulation, and "white noise." All of these capabilities are directly available for you through a few easy to use BASIC and/or assembly language statements and functions. This means that you can make very complex sounds and songs using programs that are relatively simple to design.

This section of your Programmer's Reference Guide has been created to help you explore all the capabilities of the 6581 "SID" chip, the sound and music synthesizer inside your Commodore computer. We'll explain both the theory behind musical ideas and the practical aspects of turning those ideas into real finished songs on your Commodore computer.

You need not be an experienced programmer nor a music expert to achieve exciting results from the music synthesizer. This section is full of programming examples with complete explanations to get you started.

You get to the sound generator by POKEing into specified memory locations. A full list of the locations used is provided in Appendix O. We will go through each concept, step by step. By the end you should be able to create an almost infinite variety of sounds, and be ready to perform experiments with sound on your own.

Each section of this chapter begins by giving you an example and a full line-by-line description of each program, which will show you how to use the characteristic being discussed. The technical explanation is for you to read whenever you are curious about what is actually going on.

The workhorse of your sound programs is the POKE statement. POKE sets the indicated memory location (MEM) equal to a specified value (NUM).

```
POKE MEM,NUM
```

The memory locations (MEM) used for music synthesis start at 54272 ($D400) in the Commodore 64. The memory locations 54272 to 54296 inclusive are the POKE locations you need to remember when you're using the 6581 (SID) chip register map. Another way to use the locations above is to remember only location 54272 and then add a number from 0 through 24 to it. By doing this you can POKE all the locations from 54272 to 54296 that you need from the SID chip. The numbers (NUM) that you use in your POKE statement must be between 0 and 255, inclusive.

When you've had a little more practice with making music, then you can get a little more involved, by using the PEEK function. PEEK is a function that is equal to the value currently in the indicated memory location.

X=PEEK(MEM)

The value of the variable X is set equal to the current contents of memory location MEM.

Of course, your programs include other BASIC commands, but for a full explanation of them, refer to the BASIC Statements section of this manual.

Let's jump right in and try a simple program using only one of the three voices. Computer ready? Type NEW, then type in this program, and save it on your Commodore DATASSETTE™ or disk. Then, RUN it.

EXAMPLE PROGRAM 1:

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT:REM CLEAR SOUND CHIP
20 POKES+5,9:POKES+6,0
30 POKES+24,15                :REM SET VOLUME TO
MAXIMUM
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Here's a line-by-line description of the program you've just typed in. Refer to it whenever you feel the need to investigate parts of the program that you don't understand completely.

LINE-BY-LINE EXPLANATION OF EXAMPLE PROGRAM 1:

Line(s)	Description
5	Set S to start of sound chip.
10	Clear all sound chip registers.
20	Set Attack/Decay for voice 1 (A=0,D=9).
	Set Sustain/Release for voice 1 (S=0,R=0).
30	Set volume at maximum.
40	Read high frequency, low frequency, duration of note.
50	When high frequency less than zero, song is over.
60	Poke high and low frequency of voice 1.
70	Gate sawtooth waveform for voice 1.
80	Timing loop for duration of note.
90	Release sawtooth waveform for voice 1.
100	Return for next note.
110–180	Data for song: high frequency, low frequency, duration (number of counts) for each note.
190	Last note of song and negative 1s signaling end of song.

VOLUME CONTROL

Chip register 24 contains the overall volume control. The volume can be set anywhere between 0 and 15. The other four bits are used for purposes we'll get into later. For now it is enough to know volume is 0 to 15. Look at line 30 to see how it's set in Example Program 1.

FREQUENCIES OF SOUND WAVES

Sound is created by the movement of air in waves. Think of throwing a stone into a pool and seeing the waves radiate outward. When similar waves are created in air, we hear it. If we measure the time between one peak of a wave and the next, we find the number of seconds for one cycle of the wave (n = number of seconds). The reciprocal of this number (1/n) gives you the cycles per second. Cycles per second are more commonly known as the frequency. The highness or lowness of a sound (pitch) is determined by the frequency of the sound waves produced.

The sound generator in your Commodore computer uses two locations to determine the frequency. Appendix E gives you the frequency values you need to reproduce a full eight octaves of musical notes. To create a frequency other than the ones listed in the note table use "F_{cut}" (frequency output) and the following formula to represent the frequency (F_n) of the sound you want to create. Remember that each note requires both a high and a low frequency number.

$$F_n = F_{out}/.06097$$

Once you've figured out what F_n is for your "new" note the next step is to create the high and low frequency values for that note. To do this you must first round off F_n so that any numbers to the right of the decimal point are left off. You are now left with an integer value. Now you can set the high frequency location (F_{hi}) by using the formula $F_{hi}=INT(F_n/256)$ and the low frequency location (F_{lo}) should be $F_{lo}=F_n-(256^*F_{hi})$.

At this point you have already played with one voice of your computer. If you wanted to stop here you could find a copy of your favorite tune and become the maestro conducting your own computer orchestra in your "at home" concert hall.

USING MULTIPLE VOICES

Your Commodore computer has three independently controlled voices (oscillators). Our first example program used only one of them. Later on, you'll learn how to change the quality of the sound made by the voices. But right now, let's get all three voices singing.

This example program shows you one way to translate sheet music for your computer orchestra. Try typing it in, and then SAVE it on your DATASSETTE™ or disk. Don't forget to type NEW before typing in this program.

EXAMPLE PROGRAM 2:

```
10 S=54272:FORL=STOS+24:POKEL,0:NEXT
20 DIMH(2,200),L(2,200),C(2,200)
30 DIMFQ(11)
40 V(0)=17:V(1)=65:V(2)=33
50 POKES+10,8:POKES+22,128:POKES+23,244
60 FORI=0TO11:READFQ(I):NEXT
100 FORK=0TO2
110 I=0
120 READNM
130 IFNM=0THEN250
140 WA=V(K):WB=WA-1:IFNM<0THENNM=-NM:WA=0:WB=0
150 DR%=NM/128:OC%=(NM-128*DR%)/16
160 NT=NM-128*DR%-16*OC%
170 FR=FQ(NT)
```

```
180 IFOC%=7THEN200
190 FORJ=6TOOC%STEP-1:FR=FR/2:NEXT
200 HF%=FR/256:LF%=FR-256*HF%
210 IFDR%=1THENH(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA:
I=I+1:GOTO120
220 FORJ=1TODR%-1:H(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA:
I=I+1:NEXT
230 H(K,I)=HF%:L(K,I)=LF%:C(K,I)=WB
240 I=I+1:GOTO120
250 IFI>IMTHENIM=I
260 NEXT
500 POKES+5,0:POKES+6,240
510 POKES+12,85:POKES+13,133
520 POKES+19,10:POKES+20,197
530 POKES+24,31
540 FORI=0TOIM
550 POKES,L(0,I):POKES+7,L(1,I):POKES+14,L(2,I)
560 POKES+1,H(0,I):POKES+8,H(1,I):POKES+15,H(2,I)
570 POKES+4,C(0,I):POKES+11,C(1,I):POKES+18,C(2,I)
580 FORT=1TO80:NEXT:NEXT
590 FORT=1TO200:NEXT:POKES+24,0
600 DATA34334,36376,38539,40830
610 DATA43258,45830,48556,51443
620 DATA54502,57743,61176,64814
1000 DATA594,594,594,596,596
1010 DATA1618,587,592,587,585,331,336
1020 DATA1097,583,585,585,585,587,587
1030 DATA1609,585,331,337,594,594,593
1040 DATA1618,594,596,594,592,587
1050 DATA1616,587,585,331,336,841,327
1060 DATA1607
1999 DATA0
2000 DATA583,585,583,583,327,329
2010 DATA1611,583,585,578,578,578
2020 DATA196,198,583,326,578
2030 DATA326,327,329,327,329,326,578,583
2040 DATA1606,582,322,324,582,587
2050 DATA329,327,1606,583
2060 DATA327,329,587,331,329
2070 DATA329,328,1605,578,834
2080 DATA324,322,327,585,1602
2999 DATA0
3000 DATA567,566,567,304,306,308,310
3010 DATA1591,567,311,310,567
3020 DATA305,304,299,308
3030 DATA304,171,176,306,291,551,306,308
3040 DATA310,308,313,306,295,297,299,304
3050 DATA1586,562,567,310,315,311
3060 DATA308,313,297
3070 DATA1586,567,560,311,309
3080 DATA308,309,306,308
3090 DATA1577,299,295,306,310,311,304
3100 DATA562,546,1575
3999 DATA0
```

Here is a line-by-line explanation of Example Program 2. For now, we are interested in how the three voices are controlled.

LINE-BY-LINE EXPLANATION OF EXAMPLE PROGRAM 2:

Line(s)	Description
10	Set S equal to start of sound chip and clear all sound chip registers.
20	Dimension arrays to contain activity of song, 1/16th of a measure per location.
30	Dimension array to contain base frequency for each note.
40	Store waveform control byte for each voice.
50	Set high pulse width for voice 2. Set high frequency for filter cutoff. Set resonance for filter and filter voice 3.
60	Read in base frequency for each note.
100	Begin decoding loop for each voice.
110	Initialize pointer to activity array.
120	Read coded note.
130	If coded note is zero, then next voice.
140	Set waveform controls to proper voice. If silence, set waveform controls to 0.
150	Decode duration and octave.
160	Decode note.
170	Get base frequency for this note.
180	If highest octave, skip division loop.
190	Divide base frequency by 2 appropriate number of times.
200	Get high and low frequency bytes.
210	If sixteenth note, set activity array: high frequency, low frequency, and waveform control (voice on).
220	For all but last beat of note, set activity array: high frequency, low frequency, waveform control (voice on).
230	For last beat of note, set activity array: high frequency, low frequency, waveform control (voice off).
240	Increment pointer to activity array. Get next note.
250	If longer than before, reset number of activities.
260	Go back for next voice.
500	Set Attack/Decay for voice 1 (A=0, D=0). Set Sustain/Release for voice 1 (S=15, R=0).

Line(s)	Description
510	Set Attack/Decay for voice 2 (A=5, D=5). Set Sustain/Release for voice 2 (S=8, R=5).
520	Set Attack/Decay for voice 3 (A=0, D=10). Set Sustain/Release for voice 3 (S=12, R=5).
530	Set volume 15, low-pass filtering.
540	Start loop for every 1/16th of a measure.
550	POKE low frequency from activity array for all voices.
560	POKE high frequency from activity array for all voices.
570	POKE waveform control from activity array for all voices.
580	Timing loop for 1/16th of a measure and back for next 1/16th measure.
590	Pause, then turn off volume.
600–620	Base frequency data.
1000–1999	Voice 1 data.
2000–2999	Voice 2 data.
3000–3999	Voice 3 data.

The values used in the data statements were found by using the note table in Appendix E and the chart below:

NOTE TYPE	DURATION
1/16	128
1/8	256
DOTTED 1/8	384
1/4	512
1/4+1/16	640
DOTTED 1/4	768
1/2	1024
1/2+1/16	1152
1/2+1/8	1280
DOTTED 1/2	1536
WHOLE	2048

The note number from the note table is added to the duration above. Then each note can be entered using only one number which is decoded by your program. This is only one method of coding note values. You may be able to come up with one with which you are more comfortable. The formula used here for encoding a note is as follows:

1) The duration (number of 1/16ths of a measure) is multiplied by 8.
2) The result of step 1 is added to the octave you've chosen (0–7).
3) The result of step 2 is then multiplied by 16.
4) Add your note choice (0–11) to the result of the operation in step 3.

In other words:

$$((((D*8)+O)*16)+N)$$

Where D = duration, O = octave, and N = note

A silence is obtained by using the negative of the duration number (number of 1/16ths of a measure * 128).

CONTROLLING MULTIPLE VOICES

Once you have gotten used to using more than one voice, you will find that the timing of the three voices needs to be coordinated. This is accomplished in this program by:

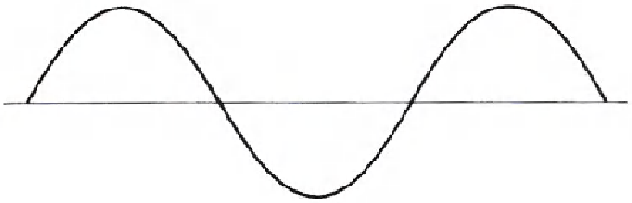
1) Divide each musical measure into 16 parts.
2) Store the events that occur in each 1/16th measure interval in three separate arrays.

The high and low frequency bytes are calculated by dividing the frequencies of the highest octave by two (lines 180 and 190). The waveform control byte is a start signal for beginning a note or continuing a note that is already playing. It is a stop signal to end a note. The waveform choice is made once for each voice in line 40.

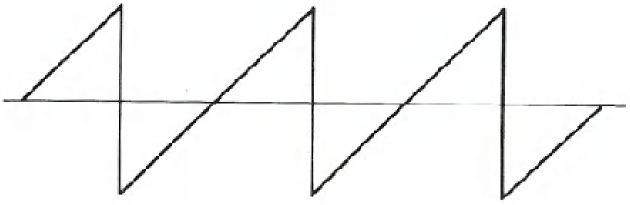
Again, this is only one way to control multiple voices. You may come up with your own methods. However, you should now be able to take any piece of sheet music and figure out the notes for all three voices.

CHANGING WAVEFORMS

The tonal quality of a sound is called the *timbre*. The timbre of a sound is determined primarily by its "waveform." If you remember the example of throwing a pebble into the water you know that the waves ripple evenly across the pond. These waves almost look like the first sound wave we're going to talk about, the sinusoidal wave, or sine wave for short (shown below).



To make what we're talking about a bit more practical, let's go back to the first example program to investigate different waveforms. The reason for this is that you can hear the changes more easily using only one voice. LOAD the first music program that you typed in earlier, from your DATASSETTE^{TM} or disk, and RUN it again. That program is using the sawtooth waveform (shown here)



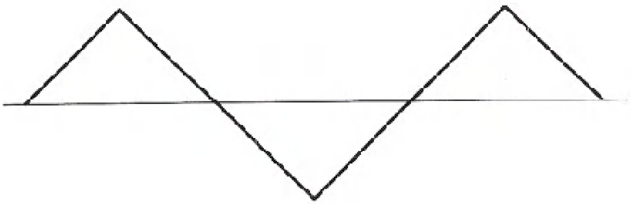
from the 6581 SID chip's sound generating device. Try changing the note start number in line 70 from 33 to 17 and the note stop number in line 90 from 32 to 16. Your program should now look like this:

EXAMPLE PROGRAM 3 (EXAMPLE 1 MODIFIED):

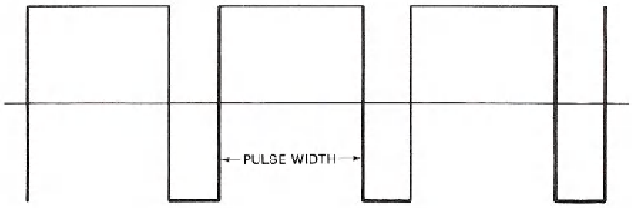
```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
20 POKES+5,9:POKES+6,0
30 POKES+24,15
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,17
80 FORT=1TODR:NEXT
90 POKES+4,16:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Now RUN the program.

Notice how the sound quality is different, less twangy, more hollow. That's because we changed the sawtooth waveform into a triangular waveform (show below).



The third musical waveform is called a variable pulse wave (shown below).



It is a rectangular wave and you determine the length of the pulse cycle by defining the proportion of the wave which will be high. This is accomplished for voice 1 by using registers 2 and 3: Register 2 is the low byte of the pulse width (L_{pw} = 0 through 255). Register 3 is the high 4 bits (H_{pw} = 0 through 15).

Together these registers specify a 12-bit number for your pulse width, which you can determine by using the following formula:

$$PW_n = H_{pw}*256 + L_{pw}$$

The pulse width is determined by the following equation:

$$PW_{out} = (PW_n/40.95)\ \%$$

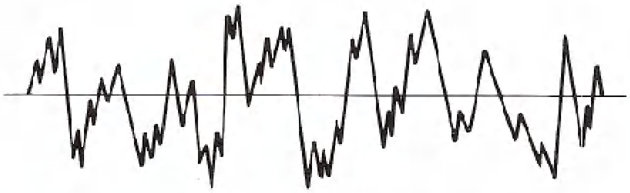
When PW_n has a value of 2048, it will give you a square wave. That means that register 2 (L_{pw}) = 0 and register 3 (H_{pw}) = 8.

Now try adding this line to your program:

```
15 POKES+3,8:POKES+2,0
```

Then change the start number in line 70 to 65 and the stop number in line 90 to 64, and RUN the program. Now change the high pulse width (register 3 in line 15) from an 8 to a 1. Notice how dramatic the difference in sound quality is?

The last waveform available to you is white noise (shown here).

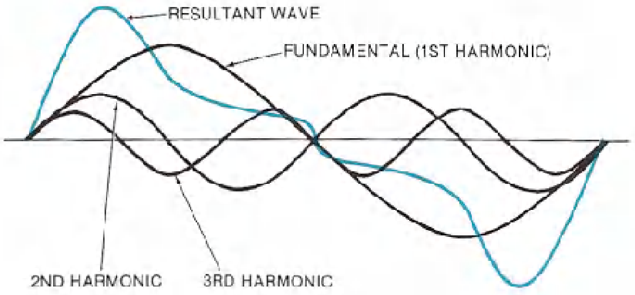


It is used mostly for sound effects and such. To hear how it sounds, try changing the start number in line 70 to 129 and the stop number in line 90 to 128.

UNDERSTANDING WAVEFORMS

When a note is played, it consists of a sine wave oscillating at the fundamental frequency and the harmonics of that wave.

The fundamental frequency defines the overall pitch of the note. Harmonics are sine waves having frequencies which are integer multiples of the fundamental frequency. A sound wave is the fundamental frequency and all of the harmonics it takes to make up that sound.



In musical theory let's say that the fundamental frequency is harmonic number 1. The second harmonic has a frequency twice the fundamental frequency, the third harmonic is three times the fundamental frequency, and so on. The amounts of each harmonic present in a note give it its timbre.

An acoustic instrument, like a guitar or a violin, has a very complicated harmonic structure. In fact, the harmonic structure may vary as a single note is played. You have already played with the waveforms available in your Commodore music synthesizer. Now let's talk about how the harmonics work with the triangular, sawtooth, and rectangular waves.

A triangular wave contains only odd harmonics. The amount of each harmonic present is proportional to the reciprocal of the square of the harmonic number. In other words harmonic number 3 is 1/9 quieter than harmonic number 1, because the harmonic 3 squared is 9 (3 × 3) and the reciprocal of 9 is 1/9.

As you can see, there is a similarity in shape of a triangular wave to a sine wave oscillating at the fundamental frequency.

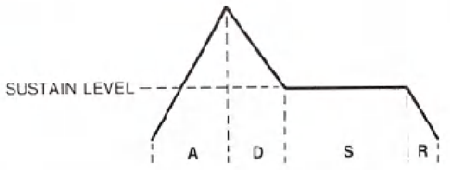
Sawtooth waves contain all the harmonics. The amount of each harmonic present is proportional to the reciprocal of the harmonic number. For example, harmonic number 2 is 1/2 as loud as harmonic number 1.

The square wave contains odd harmonics in proportion to the reciprocal of the harmonic number. Other rectangular waves have varying harmonic content. By changing the pulse width, the timbre of the sound of a rectangular wave can be varied tremendously.

By choosing carefully the waveform used, you can start with a harmonic structure that looks somewhat like the sound you want. To refine the sound, you can add another aspect of sound quality available on your Commodore 64 called *filtering,* which we'll discuss later in this section.

THE ENVELOPE GENERATOR

The volume of a musical tone changes from the moment you first hear it, all the way through until it dies out and you can't hear it anymore. When a note is first struck, it rises from zero volume to its peak volume. The rate at which this happens is called the **ATTACK**. Then, it falls from the peak to some middle-ranged volume. The rate at which the fall of the note occurs is called the **DECAY**. The mid-ranged volume itself is called the **SUSTAIN** level. And finally, when the note stops playing, it falls from the **SUSTAIN** level to zero volume. The rate at which it falls is called the **RELEASE**. Here is a sketch of the four phases of a note:



Each of the items mentioned above give certain qualities and restrictions to a note. The bounds are called parameters.

The parameters **ATTACK/DECAY/SUSTAIN/RELEASE** and collectively called **ADSR**, can be controlled by your use of another set of locations in the sound generator chip. LOAD your first example program again. RUN it again and remember how it sounds. Then, try changing line 20 so the program is like this:

EXAMPLE PROGRAM 4 (EXAMPLE 1 MODIFIED):

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
20 POKES+5,88:POKES+6,195
30 POKES+24,15
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Registers 5 and 6 define the ADSR for voice 1. The **ATTACK** is the high nybble of register 5. Nybble is half a byte, in other words the lower 4 or higher 4 on/off locations (bits) in each register. **DECAY** is the low nybble. You can pick any number 0 through 15 for **ATTACK**, multiply it by 16 and add to any number 0 through 15 for **DECAY**. The values that correspond to these numbers are listed below.

SUSTAIN level is the high nybble of register 6. It can be 0 through 15. It defines the proportion of the peak volume that the **SUSTAIN** level will be. **RELEASE** rate is the low nybble of register 6.

Here are the meanings of the values for ATTACK, DECAY, and RELEASE:

VALUE	ATTACK RATE (TIME/CYCLE)	DECAY/RELEASE RATE (TIME/CYCLE)
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

Here are a few sample settings to try in your example program. Try these and a few of your own. The variety of sounds you can produce is astounding! For a violin type sound, try changing line 20 to read:

```
20 POKES+5,88:POKES+6,89:REM A=5;D=8;S=5;R=9
```

Change the waveform to triangle and get a xylophone type sound by using these lines:

```
20 POKES+5,9:POKES+6,9:REM A=0;D=9;S=0;R=9
70 POKES+4,17
90 POKES+4,16: FORT=1TO50:NEXT
```

Change the waveform to square and try a piano type sound with these lines:

```
15 POKES+3,8:POKES+2,0
20 POKES+5,9:POKES+6,0: REM A=0;D=9;S=0;R=0
70 POKES+4,65
90 POKES+4,64:FORT=1TO50:NEXT
```

The most exciting sounds are those unique to the music synthesizer itself, ones that do not attempt to mimic acoustic instruments. For example try:

```
20 POKES+5,144:POKES+6,243:REM A=9;D=0; S=15;R=3
```

FILTERING

The harmonic content of a waveform can be changed by using a filter. The SID chip is equipped with *three* types of filtering. They can be used separately or in combination with one another. Let's go back to the sample program you've been using to play with a simple example that uses a filter. There are several filter controls to set.

You add line 15 in the program to set the *cutoff frequency* of the filter. The cutoff frequency is the reference point for the filter. You SET the high and low frequency cutoff points in registers 21 and 22. To turn ON the filter for voice 1, POKE register 23.

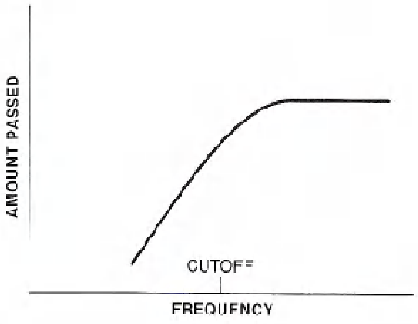
Next change line 30 to show that a high-pass filter will be used (see the SID register map).

EXAMPLE PROGRAM 5 (EXAMPLE 1 MODIFIED):

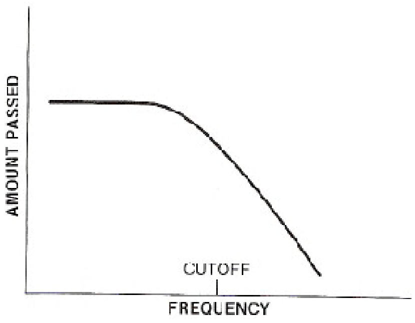
```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
15 POKES+22,128:POKES+21,0:POKES+23,1
20 POKES+5,9:POKES+6,0
30 POKES+24,79
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Try RUNning the program now. Notice the lower tones have had their volume cut down. It makes the overall quality of the note sound tinny. This is because you are using a high-pass filter which attenuates (cuts down the level of) frequencies below the specified cutoff frequency.

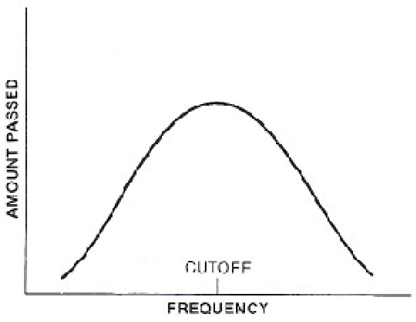
There are three types of filters in your Commodore computer's SID chip. We have been using the *high-pass filter*. It will pass all the frequencies at or above the cutoff, while attenuating the frequencies below the cutoff.



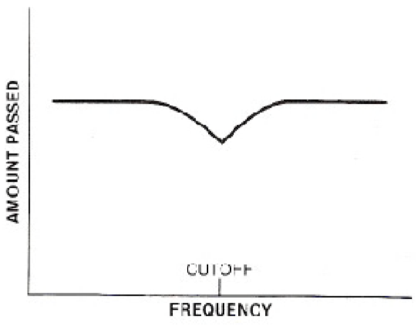
The SID chip also has a *low-pass filter*. As its name implies, this filter will pass the frequencies below cutoff and attenuate those above.



Finally, the chip is equipped with a *bandpass* filter, which passes a narrow band of frequencies around the cutoff, and attenuates all others.



The high- and low-pass filters can be combined to form a *notch reject filter* which passes frequencies away from the cutoff while attenuating at the cutoff frequency.



Register 24 determines which type filter you want to use. This is in addition to register 24's function as the overall volume control. Bit 6 controls the high-pass filter (0 = off, 1 = on), bit 5 is the bandpass filter, and bit 4 is the low-pass filter. The low 3 bits of the cutoff frequency are determined by register 21 (L_{cf}) (L_{cf} = 0 through 7). While the 8 bits of the high cutoff frequency are determined by register 22 (H_{cf}) (H_{cf} = 0 through 255).

Through careful use of filtering, you can change the harmonic structure of any waveform to get just the sound you want. In addition, changing the filtering of a sound as it goes through the ADSR phases of its life can produce interesting effects.

ADVANCED TECHNIQUES

The SID chip's parameters can be changed dynamically during a note or sound to create many interesting and fun effects. In order to make this easy to do, digitized outputs from *oscillator three* and *envelope generator three* are available for you in registers 27 and 28, respectively.

The output of oscillator 3 (register 27) is directly related to the waveform selected. If you choose the sawtooth waveform of oscillator 3, this register will present a series of numbers incremented (increased step by step) from 0 to 255 at a rate determined by the frequency of oscillator 3. If you choose the triangle waveform, the output will increment from 0 up to 255, then decrement (decrease step by step) back down to 0. If you choose the pulse wave, the output will jump back-and-forth between 0 and 255. Finally, choosing the noise waveform will give you a series of random numbers. When oscillator 3 is used for modulation, you usually do NOT want to hear its output. Setting bit 7 of register 24 turns the audio output of voice 3 off. Register 27 always reflects the changing output of the oscillator and is not affected in any way by the envelope (ADSR) generator.

Register 25 gives you access to the output of the envelope generator of oscillator 3. It functions in much the same fashion that the output of oscillator 3 does. The oscillator must be turned on to produce any output from this register.

Vibrato (a rapid variation in frequency) can be achieved by adding the output of oscillator 3 to the frequency of another oscillator. Example Program 6 illustrates this idea.

EXAMPLE PROGRAM 6:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+3,8
40 POKES+5,41:POKES+6,89
50 POKES+14,117
60 POKES+18,16
70 POKES+24,143
80 READFR,DR
90 IFFR=0THENEND
100 POKES+4,65
110 FORT=1TODR*2
120 FQ=FR+PEEK(S+27)/2
130 HF=INT(FQ/256):LF=FQAND255
140 POKES+0,LF:POKES+1,HF
150 NEXT
160 POKES+4,64
170 GOTO80
500 DATA4817,2,5103,2,5407,2
510 DATA8583,4,5407,2,8583,4
520 DATA5407,4,8583,12,9634,2
530 DATA10207,2,10814,2,8583,2
540 DATA9634,4,10814,2,8583,2
550 DATA9634,4,8583,12
560 DATA0,0
```

Here is a line-by-line explanation of Example Program 6:

LINE-BY-LINE EXPLANATION OF EXAMPLE PROGRAM 6:

Lines(s)	Description
10	Set S to beginning of sound chip.
20	Clear all sound chip locations.
30	Set high pulse width for voice 1.
40	Set Attack/Decay for voice 1 (A=2, D=9).
	Set Sustain/Release for voice 1 (S=5, R=9).
50	Set low frequency for voice 3.
60	Set triangle waveform for voice 3.
70	Set volume 15, turn off audio output of voice 3.
80	Read frequency and duration of note.
90	If frequency equals zero, stop.
100	POKE start pulse waveform control voice 1.
110	Start timing loop for duration.
120	Get new frequency using oscillator 3 output.
130	Get high and low frequency.
140	POKE high and low frequency for voice 1.
150	End of timing loop.
160	POKE stop pulse waveform control voice 1.
170	Go back for next note.
500–550	Frequencies and durations for song.
560	Zeros signal end of song.

A wide variety of sound effects can also be achieved using dynamic effects. For example, the following siren program dynamically changes the frequency output of oscillator 1 when it's based on the output of oscillator 3's triangular wave:

EXAMPLE PROGRAM 7:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+14,5
40 POKES+18,16
50 POKES+3,1
60 POKES+24,143
70 POKES+6,240
80 POKES+4,65
90 FR=5389
100 FORT=1TO200
110 FQ=FR+PEEK(S+27)*3.5
120 HF=INT(FQ/256):LF=FQ-HF*256
130 POKES+0,LF:POKES+1,HF
140 NEXT
150 POKES+24,0
```

Here is a line-by-line explanation of Example Program 7:

LINE-BY-LINE EXPLANATION OF EXAMPLE PROGRAM 7:

Line(s)	Description
10	Set S to start of sound chip.
20	Clear sound chip registers.
30	Set low frequency of voice 3.
40	Set triangular waveform voice 3.
50	Set high pulse width for voice 1.
60	Set volume 15, turn off audio output of voice 3.
70	Set Sustain/Release for voice 1 (S=15, R=0).
80	POKE start pulse waveform control voice 1.
90	Set lowest frequency for siren.
100	Begin timing loop.
110	Get new frequency using output of oscillator 3.
120	Get high and low frequencies.
130	POKE high and low frequencies for voice 1.
140	End timing loop.
150	Turn off volume.

The noise waveform can be used to provide a wide range of sound effects. This example mimics a hand clap using a filtered noise waveform:

EXAMPLE PROGRAM 8:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+0,240:POKES+1,33
40 POKES+5,8
50 POKES+22,104
60 POKES+23,1
70 POKES+24,79
80 FORN=1TO15
90 POKES+4,129
100 FORT=1TO250:NEXT:POKES+4,128
110 FORT=1TO30:NEXT:NEXT
120 POKES+24,0
```

Here is a line-by-line explanation of Example Program 8:

LINE-BY-LINE EXPLANATION OF EXAMPLE PROGRAM 8:

Line(s)	Description
10	Set S to start of sound chip.
20	Clear all sound chip registers.
30	Set high and low frequencies for voice 1.
40	Set Attack/Decay for voice 1 (A=0, D=8).
50	Set high cutoff frequency for filter.
60	Turn on filter for voice 1.
70	Set volume 15, high-pass filter.
80	Count 15 claps.
90	Set start noise waveform control.
100	Wait, then set stop noise waveform control.
110	Wait, then start next clap.
120	Turn off volume.

SYNCHRONIZATION AND RING MODULATION

The 6581 SID chip lets you create more complex harmonic structures through synchronization or ring modulation of two voices.

The process of synchronization is basically a logical ANDing of two wave forms. When either is zero, the output is zero. The following example uses this process to create an imitation of a mosquito:

EXAMPLE PROGRAM 9:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+1,100
40 POKES+5,219
50 POKES+15,28
60 POKES+24,15
70 POKES+4,19
80 FORT=1TO5000:NEXT
90 POKES+4,18
100 FORT=1TO1000:NEXT:POKES+24,0
```

Here is a line-by-line explanation of Example Program 9:

LINE-BY-LINE EXPLANATION OF EXAMPLE PROGRAM 9:

Line(s)	Description
10	Set S to start of sound chip.
20	Clear sound chip registers.
30	Set high frequency voice 1.
40	Set Attack/Decay for voice 1 (A=13, D=11).
50	Set high frequency voice 3.
60	Set volume 15.
70	Set start triangle, sync waveform control for voice 1.
80	Timing loop.
90	Set stop triangle, sync waveform control for voice 1.
100	Wait, then turn off volume.

The synchronization feature is enabled (turned on) in line 70, where bits 0,1, and 4 of register 4 are set. Bit 1 enables the syncing function between voice 1 and voice 3. Bits 0 and 4 have their usual functions of gating voice 1 and setting the triangular waveform.

Ring modulation (accomplished for voice 1 by setting bit 3 of register 4 in line 70 of the program below) replaces the triangular output of oscillator 1 with a "ring modulated" combination of oscillators 1 and 3. This produces non-harmonic overtone structures for use in mimicking bell or gong sounds. This program produces a clock chime imitation:

EXAMPLE PROGRAM 10:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+1,130
40 POKES+5,9
50 POKES+15,30
60 POKES+24,15
70 FORL=1TO12:POKES+4,21
80 FORT=1TO1000:NEXT:POKES+4,20
90 FORT=1TO1000:NEXT:NEXT
```

Here is a line-by-line explanation of Example Program 10:

LINE-BY-LINE EXPLANATION OF EXAMPLE PROGRAM 10:

Line(s)	Description
10	Set S to start of sound chip.
20	Clear sound chip registers.
30	Set high frequency for voice 1.
40	Set Attack/Decay for voice 1 (A=0, D=9).
50	Set high frequency for voice 3.
60	Set volume 15.
70	Count number of dings, set start triangle, ring mod waveform control voice 1.
80	Timing loop, set stop triangle, ring mod.
90	Timing loop, next ding.

The effects available through the use of the parameters of your Commodore 64's SID chip are numerous and varied. Only through experimentation on your own will you fully appreciate the capabilities of your machine. The examples in this section of the Programmer's Reference Guide merely scratch the surface.

Watch for the book **MAKING MUSIC ON YOUR COMMODORE COMPUTER** for everything from simple fun and games to professional-type musical instruction.