# Hardware_Manual

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>Hardware_Manual | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | | July 18, 2024 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# Chapter 1

# Hardware_Manual

## 1.1 Amiga® Hardware Reference Manual: 5 Audio Hardware

This chapter shows you how to directly access the audio hardware to produce sounds. The major topics in this chapter are:

* A brief overview of how a computer produces sound.
* How to produce simple steady and changing sounds and more complex ones.
* How to use the audio channels for special effects, wiring them for stereo sound if desired, or using one channel to modulate another.
* How to produce quality sound within the system limitations.

A section at the end of the chapter gives you values to use for creating musical notes on the equal-tempered musical scale.

This chapter is not a tutorial on computer sound synthesis; a thorough description of creating sound on a computer would require a far longer document. The purpose here is to point the way and show you how to use the Amiga's features. Computer sound production is fun but complex, and it usually requires a great deal of trial and error on the part of the user -- you use the instructions to create some sound and play it back, readjust the parameters and play it again, and so on.

The following works are recommended for more information on creating music with computers:

* Wayne A. Bateman, Introduction to Computer Music (New York: John Wiley and Sons, 1980).
* Hal Chamberlain, Musical Applications of Microprocessors (Rochelle Park, New Jersey: Hayden, 1980).

Producing High-quality Sound The Audio State Machine

## 1.2 5 Audio Hardware / Introducing Sound Generation

Sound travels through air to your ear drums as a repeated cycle of air pressure variations, or sound waves. Sounds can be represented as graphs that model how the air pressure varies over time. The attributes of a sound, as you hear it, are related to the shape of the graph. If the waveform is regular and repetitive, it will sound like a tone with steady pitch (highness or lowness), such as a single musical note. Each repetition of a waveform is called a cycle of the sound. If the waveform is irregular, the sound will have little or no pitch, like a loud clash or rushing water. How often the waveform repeats (its frequency) has an effect upon its pitch; sounds with higher frequencies are higher in pitch. Humans can hear sounds that have a frequency of between 20 and 20,000 cycles per second. The amplitude of the waveform (highest point on the graph), is related to the perceived loudness of the sound. Finally, the general shape of the waveform determines its tone quality, or timbre. Figure 5-1 shows a particular kind of waveform, called a sine wave, that represents one cycle of a simple tone.

Figure 5-1: Sine Waveform

In electronic sound recording and output devices, the attributes of sounds are represented by the parameters of amplitude and frequency. Frequency is the number of cycles per second, and the most common unit of frequency is the Hertz (Hz), which is 1 cycle per second. Large values, or high frequencies, are measured in kilohertz (KHz) or megahertz (MHz).

Frequency is strongly related to the perceived pitch of a sound. When frequency increases, pitch rises. This relationship is exponential. An increase from 100 Hz to 200 Hz results in a large rise in pitch, but an increase from 1,000 Hz to 1,100 Hz is hardly noticeable. Musical pitch is represented in octaves. A tone that is one octave higher than another has a frequency twice as high as that of the first tone, and its perceived pitch is twice as high.

The second parameter that defines a waveform is its amplitude. In an electronic circuit, amplitude relates to the voltage or current in the circuit. When a signal is going to a speaker, the amplitude is expressed in watts. Perceived sound intensity is measured in decibels (db). Human hearing has a range of about 120 db; 1 db is the faintest audible sound. Roughly every 10 db corresponds to a doubling of sound, and 1 db is the smallest change in amplitude that is noticeable in a moderately loud sound. Volume, which is the amplitude of the sound signal which is output, corresponds logarithmically to decibel level.

The frequency and amplitude parameters of a sine wave are completely independent. When sound is heard, however, there is interaction between loudness and pitch. Lower-frequency sounds decrease in loudness much faster than high-frequency sounds.

The third attribute of a sound, timbre, depends on the presence or absence of overtones, or harmonics. Any complex waveform is actually a mixture of sine waves of different amplitudes, frequencies, and phases (the starting

point of the waveform on the time axis). These component sine waves are called harmonics. A square waveform, for example, has an infinite number of harmonics.

In summary, all steady sounds can be described by their frequency, overall amplitude, and relative harmonic amplitudes. The audible equivalents of these parameters are pitch, loudness, and timbre, respectively. Changing sound is a steady sound whose parameters change over time.

In electronic production of sound, an analog device, such as a tape recorder, records sound waveforms and their cycle frequencies as a continuously variable representation of air pressure. The tape recorder then plays back the sound by sending the waveforms to an amplifier where they are changed into analog voltage waveforms. The amplifier sends the voltage waveforms to a loudspeaker, which translates them into air pressure vibrations that the listener perceives as sound.

A computer cannot store analog waveform information. In computer production of sound, a waveform has to be represented as a finite string of numbers. This transformation is made by dividing the time axis of the graph of a single waveform into equal segments, each of which represents a short enough time so the waveform does not change a great deal. Each of the resulting points is called a sample. These samples are stored in memory, and you can play them back at a frequency that you determine. The computer feeds the samples to a digital-to-analog converter (DAC), which changes them into an analog voltage waveform. To produce the sound, the analog waveforms are sent first to an amplifier, then to a loudspeaker.

Figure 5-2 shows an example of a sine wave, a square wave, and a triangle wave, along with a table of samples for each.

Figure 5-2: Digitized Amplitude Values

| TIME | SINE | SQUARE | TRIANGLE |
|---|---|---|---|
| 0 | 0 | 100 | 0 |
| 1 | 39 | 100 | 20 |
| 2 | 75 | 100 | 40 |
| 3 | 103 | 100 | 60 |
| 4 | 121 | 100 | 80 |
| 5 | 127 | 100 | 100 |
| 6 | 121 | 100 | 80 |
| 7 | 103 | 100 | 60 |
| 8 | 75 | 100 | 40 |
| 9 | 39 | 100 | 20 |
| 10 | 0 | -100 | 0 |
| 11 | -39 | -100 | -20 |
| 12 | -75 | -100 | -40 |
| 13 | -103 | -100 | -60 |
| 14 | -121 | -100 | -80 |
| 15 | -127 | -100 | -100 |
| 16 | -121 | -100 | -80 |
| 17 | -103 | -100 | -60 |
| 18 | -75 | -100 | -40 |
| 19 | -39 | -100 | -20 |

Note:
-----

The illustrations are not to scale and there are fewer dots in the wave forms than there are samples in the table. The amplitude axis values 127 and -128 represent the high and low limits on relative amplitude.



## 1.3 5 / Introducing Sound Generation / The Amiga Sound Hardware

The Amiga has four hardware sound channels. You can independently program each of the channels to produce complex sound effects. You can also attach channels so that one channel modulates the sound of another or combine two channels for stereo effects.

Each audio channel includes an eight-bit digital-to-analog converter driven by a direct memory access (DMA) channel. The audio DMA can retrieve two data samples during each horizontal video scan line. For simple, steady tones, the DMA can automatically play a waveform repeatedly; you can also program all kinds of complex sound effects.

There are two methods of basic sound production on the Amiga -- automatic (DMA) sound generation and direct (non-DMA) sound generation. When you use automatic sound generation, the system retrieves data automatically by direct memory access.

## 1.4 5 Audio Hardware / Forming and Playing a Sound

This section shows you how to create a simple, steady sound and play it. Many basic concepts that apply to all sound generation on the Amiga are introduced in this section.

To produce a steady tone, follow these basic steps:

1. Decide which channel to use.
2. Define the waveform and create the sample table in memory.
3. Set registers telling the system where to find the data and the length of the data.
4. Select the volume at which the tone is to be played.
5. Select the sampling period, or output rate of the data.
6. Select an audio channel and start up the DMA.



## 1.5 5 / Forming and Playing a Sound / Deciding Which Channel to Use

The Amiga has four audio channels. Channels 1 and 2 are connected to the left-side stereo output jack. Channels 0 and 3 are connected to the right-side output jack. Select a channel on the side from which the output is to appear.

## 1.6 5 / Forming and Playing a Sound / Creating the Waveform Data

The waveform used as an example in this section is a simple sine wave, which produces a pure tone. To conserve memory, you normally define only one full cycle of a waveform in memory. For a steady, unchanging sound, the values at the waveform's beginning and ending points and the trend or slope of the data at the beginning and end should be closely related. This ensures that a continuous repetition of the waveform sounds like a continuous stream of sound.

Sound data is organized as a set of eight-bit data items; each item is a sample from the waveform. Each data word retrieved for the audio channel consists of two samples. Sample values can range from -128 to +127.

As an example, the data set shown below produces a close approximation to a sine wave.

```
About the sample data.
----------------------
The data is stored in byte address order with the first digitized
 amplitude  value at the lowest byte address, the second at the next
byte address, and so on. Also, note that the first byte of data must
start at a word-address boundary.  This is because the audio DMA
retrieves one word (16 bits) at a time and uses the sample it reads
as two bytes of data.
```

To use audio channel 0, write the address of "audiodata" into AUD0LC, where the audio data is organized as shown below. For simplicity, "AUDxLC" in the table below stands for the combination of the two actual location registers ( AUDxLCH and AUDxLCL ). For the audio DMA channels to be able to retrieve the data, the data address to which AUD0LC points must be somewhere in chip RAM.

Table 5-1: Sample Audio Data Set for Channel 0

| | | | |
|---|---|---|---|
| audiodata ---> | AUD0LC * | 100 | 98 |
| | AUD0LC + 2 ** | 92 | 83 |
| | AUD0LC + 4 | 71 | 56 |
| | AUD0LC + 6 | 38 | 20 |
| | AUD0LC + 8 | 0 | -20 |
| | AUD0LC + 10 | -38 | -56 |
| | AUD0LC + 12 | -71 | -83 |
| | AUD0LC + 14 | -92 | -83 |
| | AUD0LC + 16 | -100 | -98 |
| | AUD0LC + 18 | -92 | -83 |
| | AUD0LC + 20 | -71 | -56 |
| | AUD0LC + 22 | -38 | -20 |

```
                    AUD0LC + 24       0      20
                    AUD0LC + 26      38      56
                    AUD0LC + 28      71      83
                    AUD0LC + 30      92      98
     Notes:
     ------
     * Audio data is located on a word-address boundary.
     ** AUD0LC stands for  AUD0LCL and AUD0LCH .
```

## 1.7 5 / Forming and Playing a Sound / Telling the System About the Data

In order to retrieve the sound data for the audio channel, the system needs to know where the data is located and how long (in words) the data is.

The location registers AUDxLCH and AUDxLCL contain the high three bits and the low fifteen bits, respectively, of the starting address of the audio data. Since these two register addresses are contiguous, writing a long word into AUDxLCH moves the audio data address into both locations. The "x" in the register names stands for the number of the audio channel where the output will occur. The channels are numbered 0, 1, 2, and 3.

These registers are location registers, as distinguished from pointer registers. You need to specify the contents of these registers only once; no resetting is necessary when you wish the audio channel to keep on repeating the same waveform. Each time the system retrieves the last audio word from the data area, it uses the contents of these location registers to again find the start of the data. Assuming the first word of data starts at location "audiodata" and you are using channel 0, here is how to set the location registers:

```
WHERE0DATA:
        LEA     CUSTOM,a0       ; Base chip address...
        LEA     AUDIODATA,a1
        MOVE.L  a1,AUD0LCH(a0)  ;Put address (32 bits)
                                ;  into location register.
```

The length of the data is the number of samples in your waveform divided by 2, or the number of words in the data set. Using the sample data set above, the length of the data is 16 words. You write this length into the audio data length register for this channel. The length register is called AUDxLEN, where "x" refers to the channel number. You set the length register AUD0LEN to 16 as shown below.

```
SETAUD0LENGTH:
        LEA     CUSTOM,a0       ; Base chip address
        MOVE.W  #16,AUD0LEN(a0) ; Store the length...
```

## 1.8 5 / Forming and Playing a Sound / Selecting the Volume

The volume you set here is the overall volume of all the sound coming from the audio channel. The relative loudness of sounds, which will concern you when you combine notes, is determined by the amplitude of the wave form.

There is a six-bit volume register for each audio channel. To control the volume of sound that will be output through the selected audio channel, you write the desired value into the register AUDxVOL, where "x" is replaced by the channel number. You can specify values from 64 to 0. These volume values correspond to decibel levels. At the end of this chapter is a table showing the decibel value for each of the 65 volume levels.

For a typical output at volume 64, with maximum data values of -128 to 127, the voltage output is between +.4 volts and -.4 volts. Some volume levels and the corresponding decibel values are shown in Table 5-2.

Table 5-2: Volume Values

| Volume | Decibel Value | |
|---|---|---|
| 64 | 0 | (maximum volume) |
| 48 | -2.5 | |
| 32 | -6.0 | |
| 16 | -12.0 | (12 db down from the volume at maximum level) |

For any volume setting from 64 to 0, you write the value into bits 5-0 of AUD0VOL. For example:

```
SETAUD0VOLUME:
        LEA     CUSTOM,a0
        MOVE.W  #48,AUD0VOL(a0)
```

The decibels are shown as negative values from a maximum of 0 because this is the way a recording device, such as a tape recorder, shows the recording level. Usually, the recorder has a dial showing 0 as the optimum recording level. Anything less than the optimum value is shown as a minus quantity.

## 1.9 5 / Forming and Playing a Sound / Selecting the Data Output Rate

The pitch of the sound produced by the waveform depends upon its frequency. To tell the system what frequency to use, you need to specify the sampling period. The sampling period specifies the number of system clock ticks , or timing intervals, that should elapse between each sample (byte of audio data) fed to the digital-to-analog converter in the audio channel. There is a period register for each audio channel. The value of the period register is used for count-down purposes; each time the register counts down to 0, another sample is retrieved from the waveform data set for output. In units, the period value represents clock ticks per sample. The minimum period value you should use is 124 ticks per sample NTSC (123 PAL) and the maximum is 65535. These limits apply to both PAL and NTSC machines. For high-quality sound, there are other constraints on the sampling period (see the section called Producing High-quality Sound ).

The period is inversely proportional to the frequency.
---

A low period value corresponds to a higher frequency sound and a high period value corresponds to a lower frequency sound.

Limitations on Selection of Sampling Period
Specifying the Period Value

## 1.10 5 / / Data Output Rate / Limitations on Selection of Sampling Period

The sampling period is limited by the number of DMA cycles allocated to an audio channel. Each audio channel is allocated one DMA slot per horizontal scan line of the screen display. An audio channel can retrieve two data samples during each horizontal scan line. The following calculation gives the maximum sampling rate in samples per second.

2 samples/line * 262.5 lines/frame * 59.94 frames/second = 31,469 samples/second

The figure of 31,469 is a theoretical maximum. In order to save buffers, the hardware is designed to handle 28,867 samples/second. The system timing interval is 279.365 nanoseconds, or .279365 microseconds. The maximum sampling rate of 28,867 samples per second is 34.642 microseconds per sample (1/28,867 = .000034642). The formula for calculating the sampling period is:

$$\text{Period value} = \frac{\text{sample interval}}{\text{clock interval}} = \frac{\text{clock constant}}{\text{samples per second}}$$

Thus, the minimum period value is derived by dividing 34.642 microseconds per sample by the number of microseconds per interval:

$$\text{Minumum period} = \frac{34.642 \text{ microseconds/sample}}{0.279365 \text{ microseconds/interval}} = 124 \text{ timing intervals/sample}$$

or:

$$\text{Minumum period} = \frac{3{,}579{,}545 \text{ ticks/second}}{28{,}867 \text{ samples/second}} = 124 \text{ ticks/sample}$$

Therefore, a value of at least 124 must be written into the period register to assure that the audio system DMA will be able to retrieve the next data sample. If the period value is below 124, by the time the cycle count has reached 0, the audio DMA will not have had enough time to retrieve the next data sample and the previous sample will be reused.

28,867 samples/second is also the maximum sampling rate for PAL systems. Thus, for PAL systems, a value of at least 123 ticks/sample must be written into the period register .

### Clock Values

| | NTSC | PAL | units |
|---|---|---|---|
| Clock Constant | 3579545 | 3546895 | ticks per second |
| Clock Interval | 0.279365 | 0.281937 | microseconds per interval |

NOTE:
---
The Clock Interval is derived from the clock constant, where:

$$\text{clock interval} = \frac{1}{\text{clock constant}}$$

then scale the result to microseconds. In all of these calculations "ticks" and "timing intervals" refer to the same thing.

## 1.11 5 / / Selecting the Data Output Rate / Specifying the Period Value

After you have selected the desired interval between data samples, you can calculate the value to place in the period register by using the period formula:

$$\text{Period value} = \frac{\text{desired interval}}{\text{clock interval}} = \frac{\text{clock constant}}{\text{samples per second}}$$

As an example, say you wanted to produce a 1 KHz sine wave, using a table of eight data samples (four data words) (see Figure 5-3).

Figure 5-3: Example Sine Wave

| Sampled Values: |
|---|
| 0 |
| 90 |
| 127 |
| 90 |
| 0 |
| -90 |
| -127 |
| -90 |

To output the series of eight samples at 1 KHz (1,000 cycles per second), each full cycle is output in 1/1000th of a second. Therefore, each individual value must be retrieved in 1/8th of that time. This translates to 1,000 microseconds per waveform or 125 microseconds per sample. To correctly produce this waveform, the period value should be:

$$\text{Period value} = \frac{125 \text{ microseconds/sample}}{0.279365 \text{ microseconds/interval}} = 447 \text{ timing intervals/sample}$$

To set the period register, you must write the period value into the register AUDxPER, where "x" is the number of the channel you are using.

For example, the following instruction shows how to write a period value of 447 into the period register for channel 0.

```
SETAUD0PERIOD:
        LEA     CUSTOM,a0
        MOVE.W  #447,AUD0PER(a0)
```

To produce high-quality sound, avoiding aliasing distortion , you should observe the limitations on period values that are discussed in the section called "Producing Quality Sound."

For the relationship between period and musical pitch, see the section at the end of the chapter, which contains a listing of the equal-tempered musical scale .

## 1.12 5 / Forming and Playing a Sound / Playing the Waveform

After you have defined the audio data location , length , volume and period , you can play the waveform by starting the DMA for that audio channel. This starts the output of sound. Once started, the DMA continues until you specifically stop it. Thus, the waveform is played over and over again, producing the steady tone. The system uses the value in the location registers each time it replays the waveform.

For any audio DMA to occur (or any other DMA, for that matter), the DMAEN bit in DMACON must be set. When both DMAEN and AUDxEN are set, the DMA will start for channel x. All these bits and their meanings are shown in table 5-3.

Table 5-3: DMA and Audio Channel Enable Bits

DMACON Register

| Bit | Name | Function |
|---|---|---|
| 15 | SET/CLR | When this bit is written as a 1, it sets any bit in DMACONW for which the corresponding bit position is also a 1, leaving all other bits alone. |
| 9 | DMAEN | Only while this bit is a 1 can any direct memory access occur. |
| 3 | AUD3EN | Audio channel 3 enable. |
| 2 | AUD2EN | Audio channel 2 enable. |
| 1 | AUD1EN | Audio channel 1 enable. |
| 0 | AUD0EN | Audio channel 0 enable. |

For example, if you are using channel 0, then you write a 1 into bit 9 to enable DMA and a 1 into bit 0 to enable the audio channel, as shown below.

```
BEGINCHAN0:
        LEA     CUSTOM,a0
        MOVE.W  #(DMAF_SETCLR!DMAF_AUD0!DMAF_MASTER),DMACON(a0)
```

## 1.13 5 / Forming and Playing a Sound / Stopping the Audio Dma

You can stop the channel by writing a 0 into the AUDxEN bit at any time. However, you cannot resume the output at the same point in the waveform by just writing a 1 in the bit again. Enabling an audio channel almost always starts the data output again from the top of the list of data pointed to by the location registers for that channel. If the channel is disabled for a very short time (less than two sampling periods ) it may stay on and thus continue from where it left off.

The following example shows how to stop audio DMA for one channel.

```
STOPAUDCHAN0:
        LEA     CUSTOM,a0
        MOVE.W  #(DMAF_AUD0),DMACON(a0)
```

## 1.14 5 / Forming and Playing a Sound / Audio Summary

These are the steps necessary to produce a steady tone:

1. Define the waveform.
2. Create the data set containing the pairs of data samples (data words). Normally, a data set contains the definition of one waveform.
3. Set the location registers:

   AUDxLCH (high three bits)
   AUDxLCL (low fifteen bits)

4. Set the length register, AUDxLEN , to the number of data words to be retrieved before starting at the address currently in AUDxLC .
5. Set the volume register, AUDxVOL .
6. Set the period register, AUDxPER
7. Start the audio DMA by writing a 1 into bit 9, DMAEN , along with a 1 in the SET/CLR bit and a 1 in the position of the AUDxEN bit of the channel or channels you want to start.

## 1.15 5 Audio Hardware / Producing Complex Sounds

In addition to simple tones, you can create more complex sounds, such as different musical notes joined into a one-voice melody, different notes played at the same time, or modulated sounds.

Joining Tones
Playing Multiple Tones at the Same Time
Modulating Sound

## 1.16 5 / Producing Complex Sounds / Joining Tones

Tones are joined by writing the location and length registers , starting the audio output, and rewriting the registers in preparation for the next audio waveform that you wish to connect to the first one. This is made easy by the timing of the audio interrupts and the existence of back-up registers. The location and length registers are read by the DMA channel before audio output begins. The DMA channel then stores the values in back-up registers.

Once the original registers have been read by the DMA channel, you can change their values without disturbing the operation you started with the original register contents. Thus, you can write the contents of these registers, start an audio output, and then rewrite the registers in preparation for the next waveform you want to connect to this one.

Interrupts occur immediately after the audio DMA channel has read the location and length registers and stored their values in the back-up registers. Once the interrupt has occurred, you can rewrite the registers with the location and length for the next waveform segment. This combination of back-up registers and interrupt timing lets you keep one step ahead of the audio DMA channel, allowing your sound output to be continuous and smooth.

If you do not rewrite the registers, the current waveform will be repeated. Each time the length counter reaches zero, both the location and length registers are reloaded with the same values to continue the audio output.

Audio DMA Example

## 1.17 5 / / Joining Tones / Audio DMA Example

This example details the system audio DMA action in a step-by-step fashion.

Suppose you wanted to join together a sine and a triangle waveform, end-to-end, for a special audio effect, alternating between them. The following sequence shows the action of your program as well as its interaction with the audio DMA system. The example assumes that the period , volume , and length of the data set remains the same for the sine wave and the triangle wave.

```
                        Interrupt Program
                        -----------------
If (wave = triangle)
               write  AUD0LCL  with address of sine wave data.

Else if (wave = sine)
               write  AUD0LCL  with address of triangle wave data.


                         Main Program
```

---

1. Set up volume , period , and length .
2. Write AUD0LCL with address of sine wave data.
3. Start DMA.
4. Continue with something else.

System Response

---

As soon as DMA starts,

a. Copy to "back-up" length register from AUD0LEN .
b. Copy to "back-up" location register from AUD0LCL (will be used as a pointer showing current data word to fetch).
c. Create an interrupt for the 680x0 saying that it has completed retrieving working copies of length and location registers .
d. Start retrieving audio data each allocated DMA time slot.

## 1.18 5 / Producing Complex Sounds / Playing Multiple Tones at the Same Time

You can play multiple tones either by using several channels independently or by summing the samples in several data sets, playing the summed data sets through a single channel.

Since all four audio channels are independently programmable, each channel has its own data set; thus a different tone or musical note can be played on each channel.

## 1.19 5 / Producing Complex Sounds / Modulating Sound

To provide more complex audio effects, you can use one audio channel to modulate another. This increases the range and type of effects that can be produced. You can modulate a channel's frequency or amplitude , or do both types of modulation on a channel at the same time.

Amplitude modulation affects the volume of the waveform. It is often used to produce vibrato or tremolo effects. Frequency modulation affects the period of the waveform. Although the basic waveform itself remains the same, the pitch is increased or decreased by frequency modulation.

The system uses one channel to modulate another when you attach two channels. The attach bits in the ADKCON register control how the data from an audio channel is interpreted (see the table below). Normally, each channel produces sound when it is enabled. If the "attach" bit for an audio channel is set, that channel ceases to produce sound and its data is used to modulate the sound of the next higher-numbered channel. When a channel is used as a modulator, the words in its data set are no longer treated as two individual bytes. Instead, they are used as "modulator" words. The data words from the modulator channel are written into the corresponding registers of the modulated channel each time the

period register of the modulator channel times out.

To modulate only the amplitude of the audio output, you must attach a channel as a volume modulator. Define the modulator channel's data set as a series of words, each containing volume information in the following format:

| Bits | Function |
|---|---|
| 15 - 7 | Not used |
| 6 - 0 | Volume information, V6 - V0 |

To modulate only the frequency, you must attach a channel as a period modulator. Define the modulator channel's data set as a series of words, each containing period information in the following format:

| Bits | Function |
|---|---|
| 15 - 0 | Period information, P15 - P0 |

If you want to modulate both period and volume on the same channel, you need to attach the channel as both a period and volume modulator. For instance, if channel 0 is used to modulate both the period and frequency of channel 1, you set two attach bits -- bit 0 to modulate the volume and bit 4 to modulate the period . When period and volume are both modulated, words in the modulator channel's data set are defined alternately as volume and period information.

The sample set of data in Table 5-4 shows the differences in interpretation of data when a channel is used directly for audio, when it is attached as volume modulator, when it is attached as a period modulator, and when it is attached as a modulator of both volume and period .

Table 5-4: Data Interpretation in Attach Mode

| Data Words | Independent (not Modulating) | Modulating Both Period and Volume | Modulating Period Only | Modulating Volume Only |
|---|---|---|---|---|
| Word 1 | \|data\|data\| | \|volume for other channel\| | \|period\| | \|volume\| |
| Word 2 | \|data\|data\| | \|period for other channel\| | \|period\| | \|volume\| |
| Word 3 | \|data\|data\| | \|volume for other channel\| | \|period\| | \|volume\| |
| Word 4 | \|data\|data\| | \|period for other channel\| | \|period\| | \|volume\| |

The lengths of the data sets of the modulator and the modulated channels are completely independent.

Channels are attached by the system in a predetermined order, as shown in Table 5-5. To attach a channel as a modulator, you set its attach bit to 1. If you set either the volume or period attach bits for a channel, that channel's audio output will be disabled; the channel will be attached to the next higher channel, as shown in Table 5-5. Because an attached channel always modulates the next higher numbered channel, you cannot

attach channel 3. Writing a 1 into channel 3's modulate bits only disables its audio output.

Table 5-5: Channel Attachment for Modulation

| | ADKCON Register | |
|---|---|---|
| Bit | Name | Function |
| 7 | ATPER3 | Use audio channel 3 to modulate nothing (disables audio output of channel 3) |
| 6 | ATPER2 | Use audio channel 2 to modulate period of channel 3 |
| 5 | ATPER1 | Use audio channel 1 to modulate period of channel 2 |
| 4 | ATPER0 | Use audio channel 0 to modulate period of channel 1 |
| 3 | ATVOL3 | Use audio channel 3 to modulate nothing (disables audio output of channel 3) |
| 2 | ATVOL2 | Use audio channel 2 to modulate volume of channel 3 |
| 1 | ATVOL1 | Use audio channel 1 to modulate volume of channel 2 |
| 0 | ATVOL0 | Use audio channel 0 to modulate volume of channel 1 |

## 1.20 5 Audio Hardware / Producing High-quality Sound

When trying to create high-quality sound, you need to consider the following factors:

* Waveform transitions.
* Sampling rate.
* Efficiency.
* Noise reduction.
* Avoidance of aliasing distortion.
* Limitations of the low pass filter.

Making Waveform Transitions
Sampling Rate
Efficiency
Noise Reduction
Aliasing Distortion
Low-Pass Filter

## 1.21 5 / Producing High-quality Sound / Making Waveform Transitions

To avoid unpleasant sounds when you change from one waveform to another, you need to make the transitions smooth. You can avoid "clicks" by making sure the waveforms start and end at approximately the same value. You can avoid "pops" by starting a waveform only at a zero-crossing point. You can avoid "thumps" by arranging the average amplitude of each wave to be about the same value. The average amplitude is the sum of the bytes in

the waveform divided by the number of bytes in the waveform.

## 1.22 5 / Producing High-quality Sound / Sampling Rate

If you need high precision in your frequency output, you may find that the frequency you wish to produce is somewhere between two available sampling rates, but not close enough to either rate for your requirements. In those cases, you may have to adjust the length of the audio data table in addition to altering the sampling rate.

For higher frequencies, you may also need to use audio data tables that contain more than one full cycle of the audio waveform to reproduce the desired frequency more accurately, as illustrated in Figure 5-4.

Figure 5-4: Waveform with Multiple Cycles

## 1.23 5 / Producing High-quality Sound / Efficiency

A certain amount of overhead is involved in the handling of audio DMA. If you are trying to produce a smooth continuous audio synthesis, you should try to avoid as much of the system control overhead as possible. Basically, the larger the audio buffer you provide to the system, the less often it will need to interrupt to reset the pointers to the top of the next buffer and, coincidentally, the lower the amount of system interaction that will be required. If there is only one waveform buffer, the hardware automatically resets the pointers, so no software overhead is used for resetting them.

The Joining Tones section illustrated how you could join "ends" of tones together by responding to interrupts and changing the values of the location registers to splice tones together. If your system is heavily loaded, it is possible that the response to the interrupt might not happen in time to assure a smooth audio transition. Therefore, it is advisable to utilize the longest possible audio table where a smooth output is required. This takes advantage of the audio DMA capability as well as minimizing the number of interrupts to which the 680x0 must respond.

## 1.24 5 / Producing High-quality Sound / Noise Reduction

To reduce noise levels and produce an accurate sound, try to use the full range of -128 to 127 when you represent a waveform. This reduces how much noise (quantization error) will be added to the signal by using more bits of precision. Quantization noise is caused by the introduction of round-off error. If you are trying to reproduce a signal, such as a sine wave, you can represent the amplitude of each sample with only so many digits of accuracy. The difference between the real number and your approximation is round-off error, or noise.

By doubling the amplitude , you create half as much noise because the

size of the steps of the wave form stays the same and is therefore a smaller fraction of the amplitude.

In other words, if you try to represent a waveform using, for example, a range of only +3 to -3, the size of the error in the output would be considerably larger than if you use a range of +127 to -128 to represent the same signal. Proportionally, the digital value used to represent the waveform amplitude will have a lower error. As you increase the number of possible sample levels, you decrease the relative size of each step and, therefore, decrease the size of the error.

To produce quiet sounds, continue to define the waveform using the full range, but adjust the volume . This maintains the same level of accuracy (signal-to-noise ratio) for quiet sounds as for loud sounds.

## 1.25 5 / Producing High-quality Sound / Aliasing Distortion

When you use sampling to produce a waveform, a side effect is caused when the sampling rate "beats" or combines with the frequency you wish to produce. This produces two additional frequencies, one at the sampling rate plus the desired frequency and the other at the sampling rate minus the desired frequency. This phenomenon is called aliasing distortion.

Aliasing distortion is eliminated when the sampling rate exceeds the output frequency by at least 7 KHz. This puts the beat frequency outside the range of the low-pass filter , cutting off the undesirable frequencies. Figure 5-5 shows a frequency domain plot of the anti-aliasing low-pass filter used in the system.

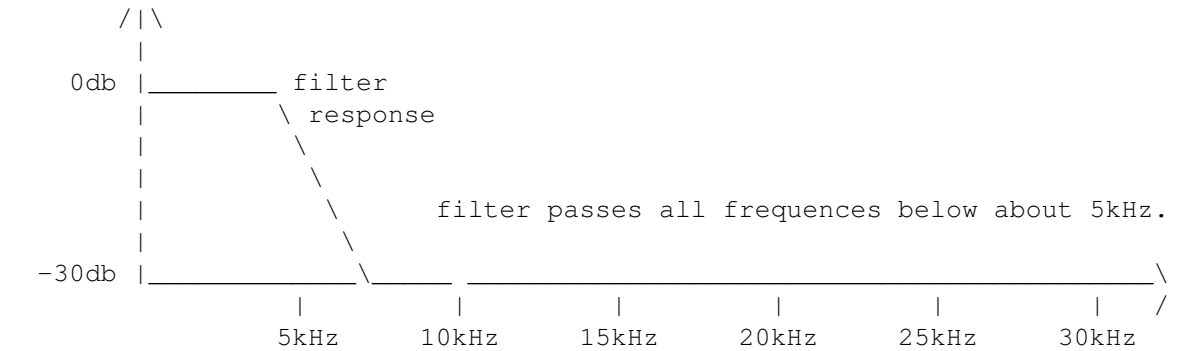
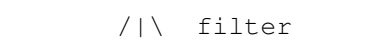



Figure 5-5: Frequency Domain Plot of Low-Pass Filter

Figure 5-6 shows that it is permissible to use a 12 KHz sampling rate to produce a 4 KHz waveform. Both of the beat frequencies are outside the range of the filter, as shown in these calculations:

12 + 4 = 16 KHz
12 - 4 = 8 KHz



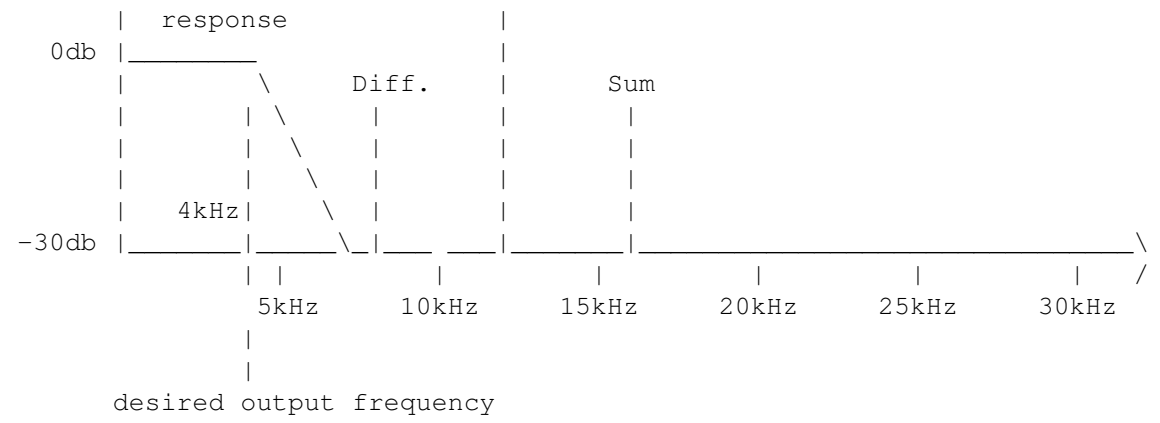



Figure 5-6: Noise-free Output (No Aliasing Distortion)

You can see in Figure 5-7 that is unacceptable to use a 10 KHz sampling rate to produce a 4 KHz waveform. One of the beat frequencies (10 - 4) is within the range of the filter, allowing some of that undesirable frequency to show up in the audio output.

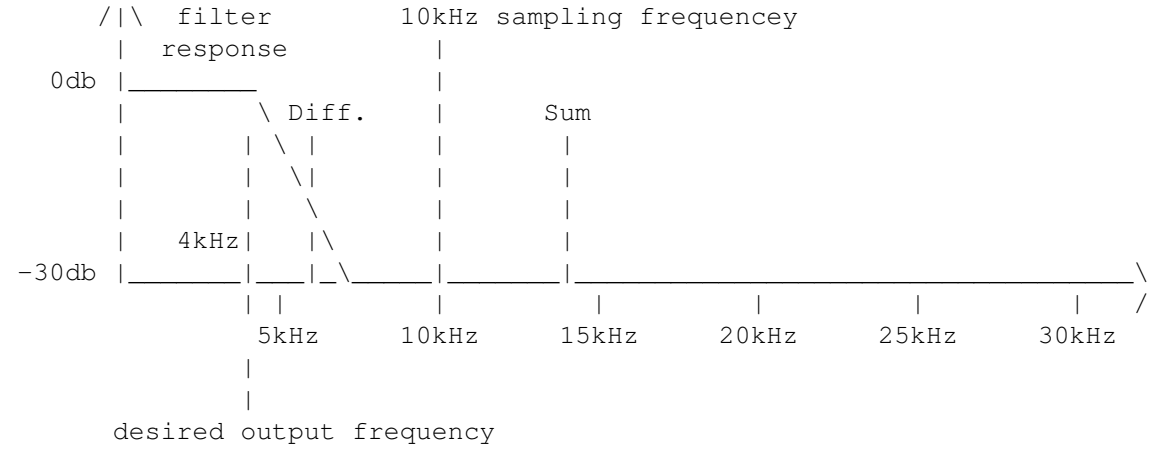



Figure 5-7: Some Aliasing Distortion

All of this gives rise to the following equation, showing that the sampling frequency must exceed the output frequency by at least 7 KHz, so that the beat frequency will be above the cutoff range of the anti-aliasing filter :

Minimum sampling rate = highest frequency component + 7 KHz

The frequency component of the equation is stated as "highest frequency component" because you may be producing a complex waveform with multiple frequency elements, rather than a pure sine wave.

## 1.26 5 / Producing High-quality Sound / Low-Pass Filter

The system includes a low-pass filter that eliminates aliasing distortion as described above. This filter becomes active around 4 KHz and gradually begins to attenuate (cut off) the signal. Generally, you cannot clearly hear frequencies higher than 7 KHz. Therefore, you get the most complete frequency response in the frequency range of 0 - 7 KHz. If you are making frequencies from 0 to 7 KHz, you should select a sampling rate no less than 14 KHz, which corresponds to a sampling period in the range 124 to 256.

At a sampling period around 320, you begin to lose the higher frequency values between 0 KHz and 7 KHz, as shown in Table 5-6.

Table 5-6: Sampling Rate and Frequency Relationship

| | Sampling Period | Sampling Rate (KHz) | Maximum Output Frequency (KHz) |
|---|---|---|---|
| Maximum sampling rate | 124 | 29 | 7 |
| Minimum sampling rate for 7 KHz output | 256 | 14 | 7 |
| Sampling rate too low for 7 KHz output | 320 | 11 | 4 |

In A2000's with 2 layer motherboards and later A500 models there is a control bit that allows the audio output to bypass the low pass filter. This control bit is the same output bit of the 8520 CIA that controls the brightness of the red "power" LED (CIA A $BFE001 - Bit 1: /LED). Bypassing the filter allows for improved sound in some applications, but an external filter with an appropriate cutoff frequency may be required.

## 1.27 5 Audio Hardware / Using Direct (Non-DMA) Audio Output

It is possible to create sound by writing audio data one word at a time to the audio output addresses, instead of setting up a list of audio data in memory. This method of controlling the output is more processor-intensive and is therefore not recommended.

To use direct audio output, do not enable the DMA for the audio channel you wish to use; this changes the timing of the interrupts . The normal interrupt occurs after a data address has been read; in direct audio output, the interrupt occurs after one data word has been output.

Unlike in the DMA-controlled automatic data output, in direct audio output, if you do not write a new set of data to the output addresses before two sampling intervals have elapsed, the audio output will cease changing. The last value remains as an output of the digital-to-analog converter .

The volume and period registers are set as usual.

## 1.28 5 Audio Hardware / The Equal-tempered Musical Scale

Table 5-7 gives a close approximation of the equal-tempered scale over one octave when the sample size is 16 bytes. The " Period " column gives the period count you enter into the period register . The length register AUDxLEN should be set to 8 (16 bytes = 8 words). The sample should represent one cycle of the waveform.

Table 5-7: Equal-tempered Octave for a 16 Byte Sample

| NTSC Period | PAL Period | Note | Ideal Frequency | Actual NTSC Frequency | Actual PAL Frequency |
|---|---|---|---|---|---|
| 254 | 252 | A | 880.0 | 880.8 | 879.7 |
| 240 | 238 | A# | 932.3 | 932.2 | 931.4 |
| 226 | 224 | B | 987.8 | 989.9 | 989.6 |
| 214 | 212 | C | 1046.5 | 1045.4 | 1045.7 |
| 202 | 200 | C# | 1108.7 | 1107.5 | 1108.4 |
| 190 | 189 | D | 1174.7 | 1177.5 | 1172.9 |
| 180 | 178 | D# | 1244.5 | 1242.9 | 1245.4 |
| 170 | 168 | E | 1318.5 | 1316.0 | 1319.5 |
| 160 | 159 | F | 1396.9 | 1398.3 | 1394.2 |
| 151 | 150 | F# | 1480.0 | 1481.6 | 1477.9 |
| 143 | 141 | G | 1568.0 | 1564.5 | 1572.2 |
| 135 | 133 | G# | 1661.2 | 1657.2 | 1666.8 |

The table above shows the period values to use with a 16 byte sample to make tones in the second octave above middle C. To generate the tones in the lower octaves, there are two methods you can use, doubling the period value or doubling the sample size.

When you double the period , the time between each sample is doubled so the sample takes twice as long to play. This means the frequency of the tone generated is cut in half which gives you the next lowest octave. Thus, if you play a C with a period value of 214, then playing the same sample with a period value of 428 will play a C in the next lower octave.

Likewise, when you double the sample size, it will take twice as long to play back the whole sample and the frequency of the tone generated will be in the next lowest octave. Thus, if you have an 8 byte sample and a 16 byte sample of the same waveform played at the same speed, the 16 byte sample will be an octave lower.

A sample for an equal-tempered scale typically represents one full cycle of a note. To avoid aliasing distortion with these samples you should use period values in the range 124-256 only. Periods from 124-256 correspond to playback rates in the range 14-28K samples per second which makes the most effective use of the Amiga’s 7 KHz cut-off filter to prevent noise. To stay within this range you will need a different sample for each octave.

If you cannot use a different sample for each octave, then you will have to adjust the period value over its full range 124-65536. This is

easier for the programmer but can produce undesirable high-frequency noise in the resulting tone. Read the section called Aliasing Distortion for more about this.

The values in Table 5-7 were generated using the formula shown below. To calculate the tone generated with a given sample size and period use:

$$\text{Frequency} = \frac{\text{Clock Constant}}{\text{Sample Bytes} * \text{Period}} = \frac{3579545}{16 * \text{Period}} = 880.8 \text{ Hz}$$

The clock constant in an NTSC system is 3579545 ticks per second. In a PAL system, the clock constant is 3546895 ticks per second. Sample bytes is the number of bytes in one cycle of the waveform sample. (The clock constant is derived from dividing the system clock value by 2. The value will vary when using an external system clock, such as a genlock.)

Using the formula above you can generate the values needed for the even-tempered scale for any arbitrary sample. Table 5-8 gives a close approximation of a five octave even tempered-scale using five samples. The values were derived using the formula above. Notice that in each octave period values are the same but the sample size is halved. The samples listed represent a simple triangular wave form.

Table 5-8: Five Octave Even-tempered Scale

| NTSC Period | PAL Period | Note | Ideal Frequency | Actual NTSC Frequency | Actual PAL Frequency |
|---|---|---|---|---|---|
| 254 | 252 | A | 55.00 | 55.05 | 54.98 |
| 240 | 238 | A# | 58.27 | 58.26 | 58.21 |
| 226 | 224 | B | 61.73 | 61.87 | 61.85 |
| 214 | 212 | C | 65.40 | 65.34 | 65.35 |
| 202 | 200 | C# | 69.29 | 69.22 | 69.27 |
| 190 | 189 | D | 73.41 | 73.59 | 73.30 |
| 180 | 178 | D# | 77.78 | 77.68 | 77.83 |
| 170 | 168 | E | 82.40 | 82.25 | 82.47 |
| 160 | 159 | F | 87.30 | 87.39 | 87.13 |
| 151 | 150 | F# | 92.49 | 92.60 | 92.36 |
| 143 | 141 | G | 98.00 | 97.78 | 98.26 |
| 135 | 133 | G# | 103.82 | 103.57 | 104.17 |
| Sample size = 256 bytes, AUDxLEN = 128 | | | | | |
| 254 | 252 | A | 110.00 | 110.10 | 109.96 |
| 240 | 238 | A# | 116.54 | 116.52 | 116.43 |
| 226 | 224 | B | 123.47 | 123.74 | 123.70 |
| 214 | 212 | C | 130.81 | 130.68 | 130.71 |
| 202 | 200 | C# | 138.59 | 138.44 | 138.55 |
| 190 | 189 | D | 146.83 | 147.18 | 146.61 |
| 180 | 178 | D# | 155.56 | 155.36 | 155.67 |
| 170 | 168 | E | 164.81 | 164.50 | 164.94 |
| 160 | 159 | F | 174.61 | 174.78 | 174.27 |
| 151 | 150 | F# | 184.99 | 185.20 | 184.73 |
| 143 | 141 | G | 196.00 | 195.56 | 196.52 |

| | | | | | |
|---|---|---|---|---|---|
| 135 | 133 | G# | 207.65 | 207.15 | 208.35 |

Sample size = 128 bytes, AUDxLEN = 64

| | | | | | |
|---|---|---|---|---|---|
| 254 | 252 | A | 220.00 | 220.20 | 219.92 |
| 240 | 238 | A# | 233.08 | 233.04 | 232.86 |
| 226 | 224 | B | 246.94 | 247.48 | 247.41 |
| 214 | 212 | C | 261.63 | 261.36 | 261.42 |
| 202 | 200 | C# | 277.18 | 276.88 | 277.10 |
| 190 | 189 | D | 293.66 | 294.37 | 293.23 |
| 180 | 178 | D# | 311.13 | 310.72 | 311.35 |
| 170 | 168 | E | 329.63 | 329.00 | 329.88 |
| 160 | 159 | F | 349.23 | 349.56 | 348.55 |
| 151 | 150 | F# | 369.99 | 370.40 | 369.47 |
| 143 | 141 | G | 392.00 | 391.12 | 393.05 |
| 135 | 133 | G# | 415.30 | 414.30 | 416.70 |

Sample size = 64 bytes, AUDxLEN = 32

| | | | | | |
|---|---|---|---|---|---|
| 254 | 252 | A | 440.0 | 440.4 | 439.8 |
| 240 | 238 | A# | 466.16 | 466.09 | 465.72 |
| 226 | 224 | B | 493.88 | 494.96 | 494.82 |
| 214 | 212 | C | 523.25 | 522.71 | 522.83 |
| 202 | 200 | C# | 554.37 | 553.77 | 554.20 |
| 190 | 189 | D | 587.33 | 588.74 | 586.46 |
| 180 | 178 | D# | 622.25 | 621.45 | 622.70 |
| 170 | 168 | E | 659.26 | 658.00 | 659.76 |
| 160 | 159 | F | 698.46 | 699.13 | 697.11 |
| 151 | 150 | F# | 739.99 | 740.80 | 738.94 |
| 143 | 141 | G | 783.99 | 782.24 | 786.10 |
| 135 | 133 | G# | 830.61 | 828.60 | 833.39 |

Sample size = 32 bytes, AUDxLEN = 16

| | | | | | |
|---|---|---|---|---|---|
| 254 | 252 | A | 880.0 | 880.8 | 879.7 |
| 240 | 238 | A# | 932.3 | 932.2 | 931.4 |
| 226 | 224 | B | 987.8 | 989.9 | 989.6 |
| 214 | 212 | C | 1046.5 | 1045.4 | 1045.7 |
| 202 | 200 | C# | 1108.7 | 1107.5 | 1108.4 |
| 190 | 189 | D | 1174.7 | 1177.5 | 1172.9 |
| 180 | 178 | D# | 1244.5 | 1242.9 | 1245.4 |
| 170 | 168 | E | 1318.5 | 1316.0 | 1319.5 |
| 160 | 159 | F | 1396.9 | 1398.3 | 1394.2 |
| 151 | 150 | F# | 1480.0 | 1481.6 | 1477.9 |
| 143 | 141 | G | 1568.0 | 1564.5 | 1572.2 |
| 135 | 133 | G# | 1661.2 | 1657.2 | 1666.8 |

Sample size = 16 bytes, AUDxLEN = 8

256 Byte Sample

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |
| 64 | 66 | 68 | 70 | 72 | 74 | 76 | 78 | 80 | 82 | 84 | 86 | 88 | 90 | 92 | 94 |
| 96 | 98 | 100 | 102 | 104 | 106 | 108 | 110 | 112 | 114 | 116 | 118 | 120 | 122 | 124 | 126 |

```
 128 126 124 122 120 118 116 114 112 110 108 106 104 102 100  98
  96  94  92  90  88  86  84  82  80  78  76  74  72  70  68  66
  64  62  60  58  56  54  52  50  48  46  44  42  40  38  36  34
  32  30  28  26  24  22  20  18  16  14  12  10   8   6   4   2
   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
 -64 -66 -68 -70 -72 -74 -76 -78 -80 -82 -84 -86 -88 -90 -92 -94
 -96 -98-100-102-104-106-108-110-112-114-116-118-120-122-124-126
-127-126-124-122-120-118-116-114-112-110-108-106-104-102-100 -98
 -96 -94 -92 -90 -88 -86 -84 -82 -80 -78 -76 -74 -72 -70 -68 -66
 -64 -62 -60 -58 -56 -54 -52 -50 -48 -46 -44 -42 -40 -38 -36 -34
 -32 -30 -28 -26 -24 -22 -20 -18 -16 -14 -12 -10  -8  -6  -4  -2
```

128 Byte Sample

```
   0   4   8  12  16  20  24  28  32  36  40  44  48  52  56  60
  64  68  72  76  80  84  88  92  96 100 104 108 112 116 120 124
 128 124 120 116 112 108 104 100  96  92  88  84  80  76  72  68
  64  60  56  52  48  44  40  36  32  28  24  20  16  12   8   4
   0   4   8  12  16  20  24  28  32  36  40  44  48  52  56  60
  64  68  72  76  80  84  88  92  96 100 104 108 112 116 120 124
-127-124-120-116-112-108-104-100 -96 -92 -88 -84 -80 -76 -72 -68
 -64 -60 -56 -52 -48 -44 -40 -36 -32 -28 -24 -20 -16 -12  -8  -4
```

64 Byte Sample

```
   0   8  16  24  32  40  48  56  64  72  80  88  96 104 112 120
 128 120 112 104  96  88  80  72  64  56  48  40  32  24  16   8
   0  -8 -16 -24 -32 -40 -48 -56 -64 -72 -80 -88 -96-104-112-120
-127-120-112-104 -96 -88 -80 -72 -64 -56 -48 -40 -32 -24 -16  -8
```

32 Byte Sample

```
   0  16  32  48  64  80  96 112 128 112  96  80  64  48  32  16
   0 -16 -32 -48 -64 -80 -96-112-127-112 -96 -80 -64 -48 -32 -16
```

16 Byte Sample

```
   0  32  64  96 128  96  64  32   0 -32 -64 -96-127 -96 -64 -32
```

## 1.29 5 Audio Hardware / Decibel Values for Volume Ranges

Table 5-9 provides the corresponding decibel values for the volume ranges of the Amiga system.

Table 5-9: Decibel Values and Volume Ranges

| Volume | Decibel Value | Volume | Decibel Value |
|---|---|---|---|
| 64 | 0.0 | 32 | -6.0 |
| 63 | -0.1 | 31 | -6.3 |
| 62 | -0.3 | 30 | -6.6 |
| 61 | -0.4 | 29 | -6.9 |
| 60 | -0.6 | 28 | -7.2 |
| 59 | -0.7 | 27 | -7.5 |
| 58 | -0.9 | 26 | -7.8 |
| 57 | -1.0 | 25 | -8.2 |
| 56 | -1.2 | 24 | -8.5 |
| 55 | -1.3 | 23 | -8.9 |
| 54 | -1.5 | 22 | -9.3 |
| 53 | -1.6 | 21 | -9.7 |
| 52 | -1.8 | 20 | -10.1 |
| 51 | -2.0 | 19 | -10.5 |
| 50 | -2.1 | 18 | -11.0 |
| 49 | -2.3 | 17 | -11.5 |
| 48 | -2.5 | 16 | -12.0 |
| 47 | -2.7 | 15 | -12.6 |
| 46 | -2.9 | 14 | -13.2 |
| 45 | -3.1 | 13 | -13.8 |
| 44 | -3.3 | 12 | -14.5 |
| 43 | -3.5 | 11 | -15.3 |
| 42 | -3.7 | 10 | -16.1 |
| 41 | -3.9 | 9 | -17.0 |
| 40 | -4.1 | 8 | -18.1 |
| 39 | -4.3 | 7 | -19.2 |
| 38 | -4.5 | 6 | -20.6 |
| 37 | -4.8 | 5 | -22.1 |
| 36 | -5.0 | 4 | -24.1 |
| 35 | -5.2 | 3 | -26.6 |
| 34 | -5.5 | 2 | -30.1 |
| 33 | -5.8 | 1 | -36.1 |
| | | 0 | Minus infinity |

## 1.30 5 Audio Hardware / The Audio State Machine

For an explanation of the various states, refer to Figure 5-8. There is one audio state machine for each channel. The machine has eight states and is clocked at the clock constant rate (3.58 MHz NTSC). Three of the states are basically unused and just transfer back to the idle (000) state. One of the paths out of the idle state is designed for interrupt-driven operation (processor provides the data), and the other path is designed for DMA-driven operation (the "Agnus" special chip provides the data).

In interrupt-driven operation, transfer to the main loop (states 010 and 011) occurs immediately after data is written by the processor. In the 010 state the upper byte is output, and in the 011 state the lower byte is output. Transitions such as 010->011->010 occur whenever the period counter counts down to one. The period counter is reloaded at these transitions. As long as the interrupt is cleared by the processor in time, the machine remains in the main loop. Otherwise, it enters the idle

state. Interrupts are generated on every word transition (011->010).

In DMA-driven operation, transition to the 001 state occurs and DMA requests are sent to Agnus as soon as DMA is turned on. Because of pipelining in Agnus, the first data word must be thrown away. State 101 is entered as soon as this word arrives; a request for the next data word has already gone out. When the data arrives, state 010 is entered and the main loop continues until the DMA is turned off. The length counter counts down once with each word that comes in. When it finishes, a DMA restart request goes to Agnus along with the regular DMA request. This tells Agnus to reset the pointer to the beginning of the table of data. Also, the length counter is reloaded and an interrupt request goes out soon after the length counter finishes (counts to one). The request goes out just as the last word of the waveform starts its output.

DMA requests and restart requests are transferred to Agnus once each horizontal line, and the data comes back about 14 clock cycles later (the duration of a clock cycle is 280 ns).

In attach mode, things run a little differently. In attach volume, requests occur as they do in normal operation (on the 011->010 transition). In attach period, a set of requests occurs on the 010->011 transition. When both attach period and attach volume are high, requests occur on both transitions.

If the sampling rate is set much higher than the normal maximum sampling rate (approximately 29 KHz), the two samples in the buffer register will be repeated. If the filter on the Amiga is bypassed and the volume is set to the maximum ($40), this feature can be used to make modulated carriers up to 1.79 MHz. The modulation is placed in the memory map, with plus values in the even bytes and minus values in the odd bytes.

The symbols used in the state diagram are explained in the following list. Upper-case names indicate external signals; lower-case names indicate local signals.

| | |
|---|---|
| AUDxON | DMA on "x" indicates channel number (signal from DMACON ). |
| AUDxIP | Audio interrupt pending (input to channel from interrupt circuitry). |
| AUDxIR | Audio interrupt request (output from channel to interrupt circuitry) |
| intreq1 | Interrupt request that combines with intreq2 to form AUDxIR.. |
| intreq2 | Prepare for interrupt request. Request comes out after the next 011->010 transition in normal operation. |
| AUDxDAT | Audio data load signal. Loads 16 bits of data to audio channel. |
| AUDxDR | Audio DMA request to Agnus for one word of data. |
| AUDxDSR | Audio DMA request to Agnus to reset pointer to start of block. |

| | |
|---|---|
| dmasen | Restart request enable. |
| percntrld | Reload period counter from back-up latch typically written by processor with AUDxPER (can also be written by attach mode). |
| percount | Count period counter down one latch. |
| perfin | Period counter finished (value = 1). |
| lencntrld | Reload length counter from back-up latch. |
| lencount | Count length counter down one notch. |
| lenfin | Length counter finished (value = 1). |
| volcntrld | Reload volume counter from back-up latch. |
| pbufld1 | Load output buffer from holding latch written to by AUDxDAT. |
| pbufld2 | Like pbufld1, but only during 010->011 with attach period. |
| AUDxAV | Attach volume. Send data to volume latch of next channel instead of to D->A converter. |
| AUDxAP | Attach period. Send data to period latch of next channel instead of to the D->A converter. |
| penhi | Enable the high 8 bits of data to go to the D->A converter. |
| napnav | /AUDxAV * /AUDxAP + AUDxAV -- no attach stuff or else attach volume. Condition for normal DMA and interrupt requests. |
| sq2,1,0 | The name of the state flip-flops, MSB to LSB. |

Figure 5-8: Audio State Diagram

ECS Audio.
---

For information on the audio hardware in the Enhanced Chip Set, see the ECS register map in Appendix C.