Amiga Mail
Volume II
March/April 1993

Creating Virtual Voices with Amiga Audio

by Dan Baker

Every Amiga model comes standard with 4-channel, 8-bit stereo audio hardware. This hardware provides every application with the capability of producing 4-part, stereo sound. Some applications however may want to exceed the 4-channel limit. For games and other applications that use sound effects extensively, it may be desirable to trigger more than 4 sounds simultaneously. This article demonstrates two techniques you can use to implement virtual voices on the Amiga's audio hardware effectively doubling the number of voices available to 8.

Audio Hardware Limits

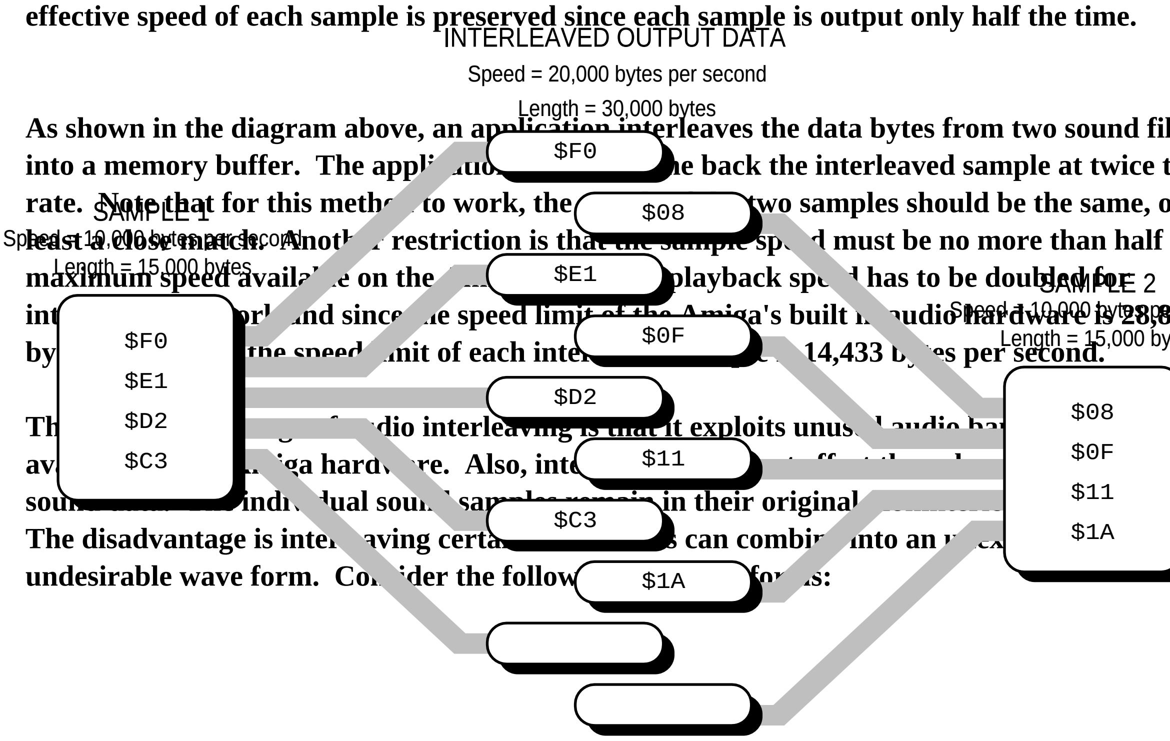
Using DMA, the audio hardware can fetch about 2 bytes per scan line for each channel without processor intervention. This means that each audio channel can play back sampled data at up to 28,867 bytes per second. It turns out that this 28,867 byte per second limit far exceeds the requirements of most audio sample files.

The typical 8SVX file contains data sampled at a rate of about 10,000 bytes per second. In that case, the audio hardware only uses about 1/3 of the available audio bandwidth. The audio hardware is capable of fetching much more sound data without affecting system performance. The question then arises, can this extra horsepower be harnessed in some way? The answer is yes.

Audio Interleaving

An application can use any extra audio bandwidth to interleave the bytes from two separate sample files. This technique allows an application to play both samples simultaneously on a single audio channel.

For instance, if you have two files sampled at 10,000 bytes per second, you could set the playback speed to 20,000 bytes per second and alternate playing bytes from each sample. T effective speed of each sample is preserved since each sample is output only half the time.



As shown in the diagram above, an application interleaves the data bytes from two sound fil into a memory buffer. The appl[illegible] the back the interleaved sample at twice t rate. Note that for this meth[illegible] to work, the [illegible] two samples should be the same, o least a close match. Another restriction is that the sample sp[illegible]d must be no more than half maximum speed availa[illegible]e on the [illegible] playback sp[illegible] has to be doubled for int[illegible]rk and since [illegible]e speed limit of the Amiga's built [illegible] audio hardware is 28,8 by[illegible] the speed [illegible]mit of each inte[illegible] 14,433 b[illegible]es per second.

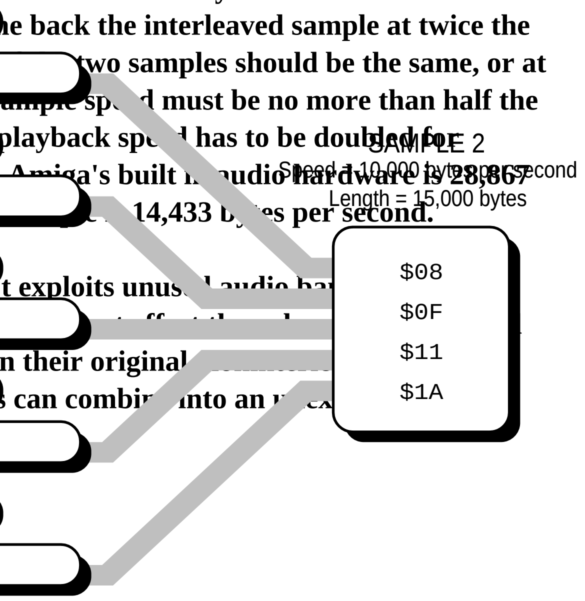
Th[illegible] [illegible]dio interleaving is that it exploits unus[illegible] audio ba av[illegible] [illegible]iga hardw[illegible]re. Also, inte[illegible] sou[illegible] indiv[illegible]dual sou[illegible] sa[illegible] in their original [illegible] The disadvantage is inte[illegible]aving certa[illegible]s can combi[illegible] into an [illegible] undesirable wave form. Co[illegible]ider the follow[illegible]for[illegible]s:

When the Amiga interleaves these two wave forms, it has to alternate between wave form A and wave form B. Because the Amiga is constantly oscillating between two wave forms, it produces a completely different wave form:

to interleave the bytes from two separate
o play both samples simultaneously on a

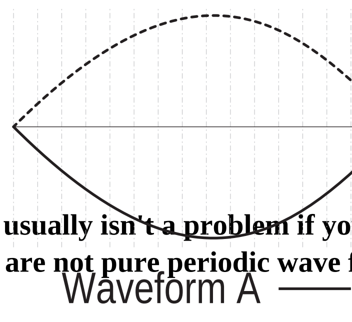
) bytes per second, you could set the
nate playing bytes from each sample. The
h sample is output only half the time.

rleaves the data bytes from two sound files
ne back the interleaved sample at twice the
two samples should be the same, or at
ample speed must be no more than half the
playback speed has to be doubled for
Amiga's built in audio hardware is 28,867
14,433 bytes per second.



t exploits unused audio har
n their original
s can combine into an unex

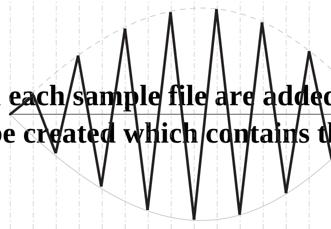
it has to alternate between wave form A
oscillating between two wave forms, it



In practice, this usually isn't a problem if yo
sampled sounds are not pure periodic wave f

Audio Averaging

It is not always possible to interleave two sam
be more than 14,433 bytes per second, so dou
limit. In that case, there is another trick you
on one channel.

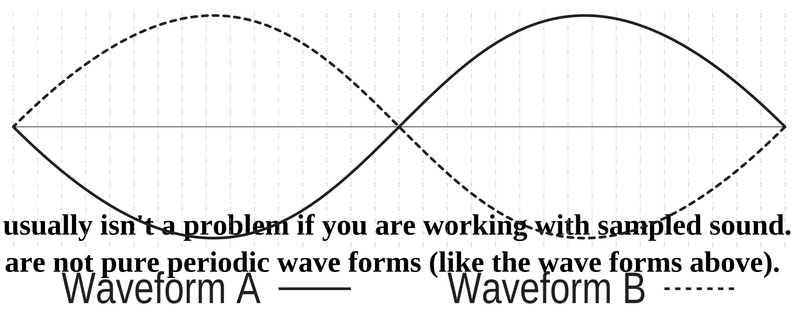


If the values from each sample file are added
data stream can be created which contains t

Note that in the example above, data values a
+127 using twos complement format. The re
though the ioa_Data field of the IOAudio str
as (UBYTE *) in the include file *<devices/aud*
signed bytes.

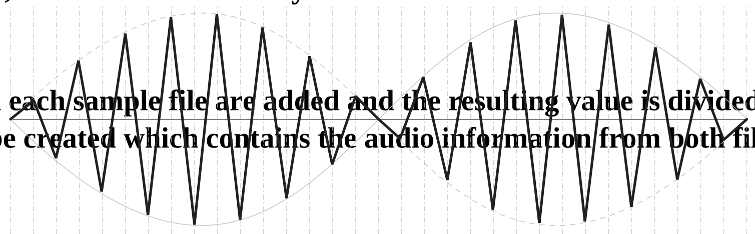
Audio averaging introduces some noise into
summed values by two effectively reduces th
bit (from eight to seven bits). Also, when the
is truncated, hence, some information is lost.

Despite these drawbacks, the results of audio
hardware are comparable to the interleaving
techniques are virtually indistinguishable on
may not always be true--see the ``Audio Exp



ractice, this usually isn't a problem if you are working with sampled sound. Typically, pled sounds are not pure periodic wave forms (like the wave forms above).

not always possible to interleave two samples. For example, the frequency of a sample may nore than 14,433 bytes per second, so doubling it would exceed the 28,867 bytes per second t. In that case, there is another trick you can do to the audio data to combine two samples

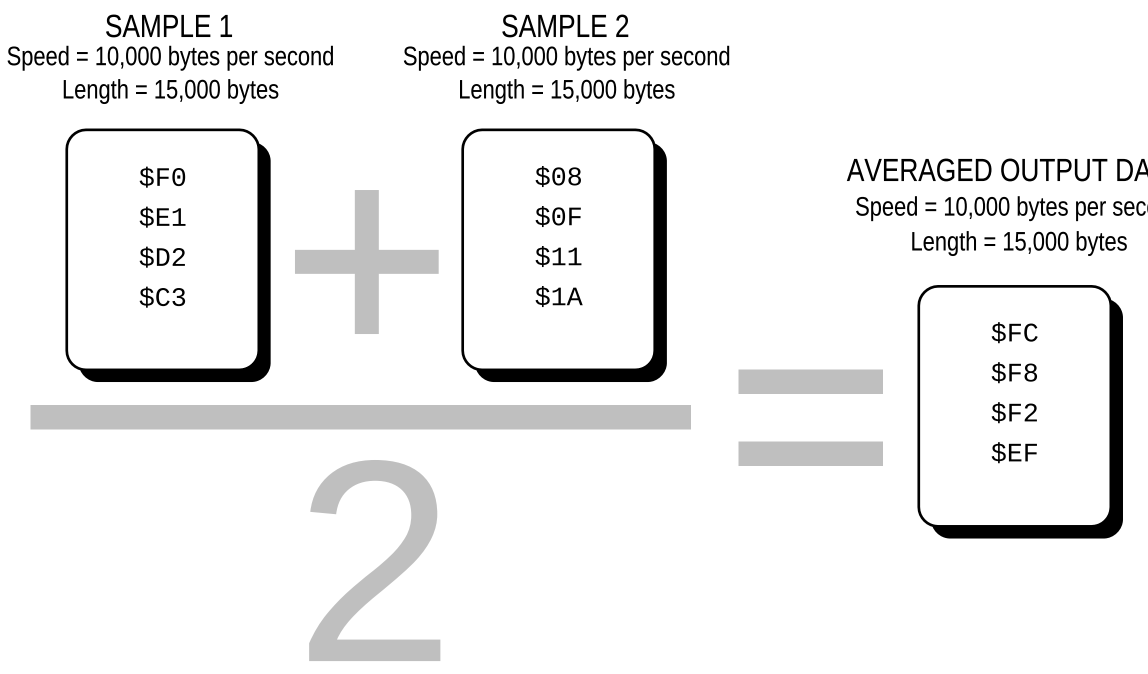


e values from each sample file are added and the resulting value is divided by two, a new stream can be created which contains the audio information from both files.

e that in the example above, data values are represented as signed bytes in the range -128 to 7 using twos complement format. The resulting average values are correct as shown. Even gh the ioa_Data field of the IOAudio structure used for all audio.device requests is shown UBYTE *) in the include file *<devices/audio.h>*, do not be misled. The data values are

io averaging introduces some noise into the resulting combined signal. Dividing the med values by two effectively reduces the dynamic range of the component samples by one from eight to seven bits). Also, when the values are divided by two, any fractional amount uncated, hence, some information is lost.

pite these drawbacks, the results of audio averaging on the Amiga's eight bit audio dware are comparable to the interleaving technique described above. In fact, the two niques are virtually indistinguishable on the current generation of Amigas (although this not always be true--see the ``Audio Experiments'' section later in this article).



Virtual Voices

When two samples are combined on a single channel using interleaving or averaging, both samples are clearly audible but, subjectively, it sounds as if one bit of volume control has be lost on each sample. Because of this, it is not wise to carry these virtual voice techniques to a extreme. Combining samples without limit will result in a badly degraded composite in whi the component signals are no longer clearly audible.
It is however quite feasible to double the number of available voices to 8 using interleaving o averaging techniques. The loss of fidelity with 8 virtual voices is quite tolerable. The code listed below shows how this can be implemented.

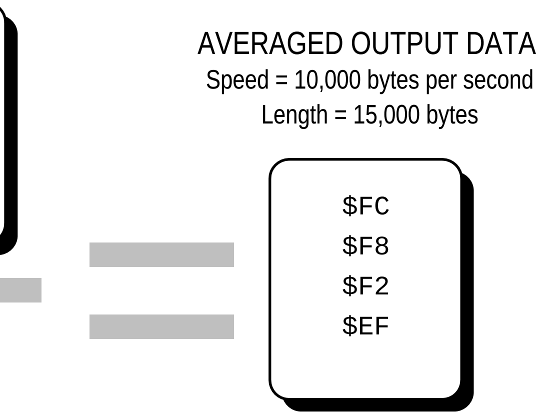
Using the Interplay Program

The program, named *interplay.c*, allows the playback of standard IFF 8SVX files in three different ways:

1. **Normal playback of a single file on one channel**
2. **Interleaved playback of two files on one channel**
3. **Averaged playback of two files on one channel**

For normal playback of a single 8SVX file, enter the following command at the Shell promp

```
1>interplay sample.8svx
```



el using interleaving or averaging, both
nds as if one bit of volume control has been
o carry these virtual voice techniques to an
ult in a badly degraded composite in which

available voices to 8 using interleaving or
tual voices is quite tolerable. The code

k of standard IFF 8SVX files in three

e following command at the Shell prompt:

This feature allows you to find out how a sa
or averaged counterpart.

For playback of two 8SVX files on a single c
command at the Shell prompt:

```
1>interplay voice.8svx music.8svx
```

The program reads the two files, figures out
audio device to twice that value. If the calcu
per second, then *interplay* sets the speed to t
data from the two files so that the bytes play
the other. If the data from one file runs out
interleaved with zero.

For playback of two 8SVX files on a single c
command at the Shell prompt:

```
1>interplay voice.8svx music.8svx SUM
```

The ``SUM'' keyword enables averaging ins
whichever file uses the faster playback rate.
added and then divided by two. The resulti
one file runs out before the other, any remai

Interplay can play samples of any size. If a
pressing Ctrl-C.

How Interplay Works

Interplay uses a double-buffered approach f
While one data buffer is playing, the other d
averaging or interleaving technique descrie

Most of the code in the main loop within ma
two playback buffers and their correspondi
averaging or interleaving of bytes actually t
main().

The reading and parsing of the 8SVX file ar
takes as a parameter an InterPlay structure
information that the program needs to man
be one filled-in InterPlay structure for each
combined playback, the two InterPlay struc

s feature allows you to find out how a sample sounds alone as compared with its interleaved veraged counterpart.

playback of two 8SVX files on a single channel using interleaving, enter the following mand at the Shell prompt:

```
 1>interplay voice.8svx music.8svx
```

 program reads the two files, figures out which has the faster sampling rate, and sets the o device to twice that value. If the calculated rate exceeds the maximum of 28,867 bytes second, then *interplay* sets the speed to the maximum. The program then interleaves the from the two files so that the bytes played by the audio channel alternate from one file to other. If the data from one file runs out before the other, any remaining data bytes are

playback of two 8SVX files on a single channel using averaging, enter the following mand at the Shell prompt:

```
 1>interplay voice.8svx music.8svx SUM
```

``SUM'' keyword enables averaging instead of interleaving. In this case, the speed is set to chever file uses the faster playback rate. One byte is taken from each file, the two bytes are ed and then divided by two. The resulting average value is played back. If the data from file runs out before the other, any remaining data bytes are averaged with zero.

rplay can play samples of any size. If a sample is too long, you can terminate playback by

w Interplay Works

rplay uses a double-buffered approach for the playback of samples of arbitrary length. le one data buffer is playing, the other data buffer is being prepared using either the aging or interleaving technique descried above.

t of the code in the main loop within main() is concerned with switching between one of the playback buffers and their corresponding I/O request blocks and message ports. The aging or interleaving of bytes actually takes place in the FillAudio() subroutine, not in

 reading and parsing of the 8SVX file are handled by the Parse8svx() subroutine which s as a parameter an InterPlay structure. The InterPlay struture holds all the state rmation that the program needs to manage playback of the sampled data. Thus there will ne filled-in InterPlay structure for each file to be played back. If the user requests a bined playback, the two InterPlay structures are linked together via the

InterPlay.next_iplay field. Otherwise this field is set to NULL.

Housekeeping for the audio.device channels used is handled by the SiezeChannel() and ReleaseChannel() subroutines.

Audio Experiments

Using the *interplay.c* program listed below, we found that there was very little difference in t audio quality between the two methods of combining samples. We also found that for best results, the dynamic ranges within the samples themselves had to be closely matched or the result would be one sample drowning out the other. Of course, it also helps if the speeds are close match. If they aren't then one or the other of the samples will be too slow or too fast.

Although both of these methods work comparitively well on the Amiga's eight-bit audio hardware, doing the same tricks with 16-bit samples on 16-bit hardware would yield a differ result. As mentioned earlier, the drawback of the averaging method is it loses a bit from the dynamic range of the sound samples. Compared to eight-bit sound, this loss is much less significant when working with 16-bit sound. As the dynamic range increases, the impact of losing a single bit from the dynamic range decreases. On the other hand, the drawback to th interleaving method is that it has to oscillate between two samples, which produces a wavefo equal to the sample playback rate. The waveform is independent of the dynamic range, so it remains constant as the dynamic range increases. The result is the avaeraging method will produce superior results on systems with greater dynamic range.

Perhaps the best thing about the audio techniques demonstrated by *interplay.c* is that they a not limited to the Amiga architecture. In fact, you can use interleaving and averaging with a system that supports the variable speed playback of digitally sampled audio. These methods will work not only in the current generation of Amigas but in any future system that suppor digital audio, although as the dynamic range increases, the additive method will provide superior sound quality.

handled by the SiezeChannel() and

d that there was very little difference in the
g samples. We also found that for best
nselves had to be closely matched or the
Of course, it also helps if the speeds are a
the samples will be too slow or too fast.

y well on the Amiga's eight-bit audio
s on 16-bit hardware would yield a different
veraging method is it loses a bit from the
eight-bit sound, this loss is much less
dynamic range increases, the impact of
s. On the other hand, the drawback to the
n two samples, which produces a waveform
s independent of the dynamic range, so it
The result is the avaeraging method will

emonstrated by *interplay.c* is that they are
can use interleaving and averaging with any
digitally sampled audio. These methods
gas but in any future system that support
ses, the additive method will provide