IIGS Sound

1993 A2-Central Summer Conference

Presented by Steve Gunn (GEnie A2PRO.STEVE) and Nate Trost (GEnie N.TROST)

I. Overview of the Apple IIGS sound hardware
  A. Basic Hardware
  B. Hardware add-ons

II. Using the Sound Tool Set and Sound Control Panel

III. Sound File Formats
  A. rSound resource
  B. HyperStudio
  C. AIFF, AIFF-C
  D. ASIF
  E. Unofficial 'formats'

IV. Low Level Ensoniq Programming
  A. Overview of DOC registers
  B. Setting/getting registers and moving data to/from DOC RAM
  C. Playing sounds
  D. Oscillator interrupts and timing
  E. Special Effects, Stereo

V. Music
  A. Overview of music software
  B. SoundSmith format
  C. Playing SoundSmith sequences
  D. MOD format
  E. Parsing/Playing MOD sequences

REFERENCES

Nate and Steve recommend you have these books if you are interested in GS sound programming:

Apple IIGS Toolbox Reference 3 (_MUST_ have for IIGS sound programming, includes more detailed information on sound hardware than TB I and adds documentation for ACE, Note Synthesizer, Note Sequencer, and MIDI Tools---although last three are mostly obsolete)

Apple IIGS Toolbox Reference 2 (Reference for Sound Tool Set)

Apple IIGS Hardware Reference Manual (description of sound hardware)

Programmers Reference for System 6.0 (MIDISynth reference)

For more information on sound related programming, check out the A2Pro Roundtable on GEnie. In addition to having Bulletin Board topics on everything from the Sound Tool Set to Low Level Ensoniq Access, A2Pro sponsors a weekly Real Time Conference devoted to graphics and sound programming. For more information on A2Pro and GEnie, talk to one of the A2/A2Pro staff here at the conference!

This text was originally published in the September 1991 issue of 8/16 Central.

Unleashing the 'S' in IIGS
by Nate Trost

The Apple IIGS Sound Hardware

The heart of the Apple IIGS sound hardware is the Ensoniq 5503 Digital Oscillator Chip (hereby referred to as the DOC). An important thing to remember about the DOC is that neither it, or the 64K of dedicated sound RAM is directly linked to the 65816 microprocessor. A custom chip called the Sound General Logic Unit (GLU) acts as a bridge between the DOC and the rest of the IIGS hardware.

Please note that we will be using the words 'sound' and 'waveform' interchangeably throughout the article to mean the same thing-the digital representation, in the memory of the GS, of the noise we wish to create .

The DOC contains 32 oscillators, each of which generates sound by passing 8-bit values from the sound RAM, through a digital-to-analog converter, and out to the various sound outputs. The DOC oscillators are not oscillators in the traditional sense, but Apple named them oscillators, so that's what we'll call them.

Each DOC oscillator has several one-byte registers assigned to it. These registers control the frequency that the oscillator will play, the playback volume, pointers to the sounds in DOC RAM, and other various things. We will discuss the 'nitty-gritty' of the sound hardware in more detail in an upcoming installment.

The Sound Toolset

As with many other parts of GS hardware, Apple has created a toolset (appropriately titled the Sound Toolset) to make the playback of digitized sounds much easier for the programmer.

The Sound Toolset combines 30 of the 32 DOC oscillators into 15 generators each of which consists of two oscillators working together. Oscillators 31 and 32 are reserved for the Sound Toolset's private use.

There are three main Sound Toolset calls used to play digitized sounds. The most important is FFStartSound.

FFStartSound and You

Let's look at an example of a FFStartSound call (example code is in assembly):

```
     PushWord #genNum     ;specify channel, generator...
     PushLong #paramBlock ;pointer to sound parameter block
     _FFStartSound

 * our data

 genNum =  $0201  ;play sound channel 0,with generator 1...

 paramBlock
 waveStart   adrl  SoundPtr ;pointer to sound in main memory
 waveSize    dw    #64      ;sound size in pages (64pages=16K)
 freqOffset  dw    #200     ;playback at 10281 hertz...
 docBuffer   dw    $2000   ;start sound at $2000 in DOC RAM
 bufferSize  dw    #5       ;buffer is 8K
 nextWavePtr adrl  #0       ;no other waveforms to play linked
with this
 volSetting  dw    #$FF     ;maximum volume
 * dw=2byte value, adrl=4byte value
```

Of the two parameters passed on the stack to FFStartSound, the first word sized value, genNum, obviously needs some explanation. genNum holds settings for various options for the playing of the sound, the structure of the parameter is as follows:

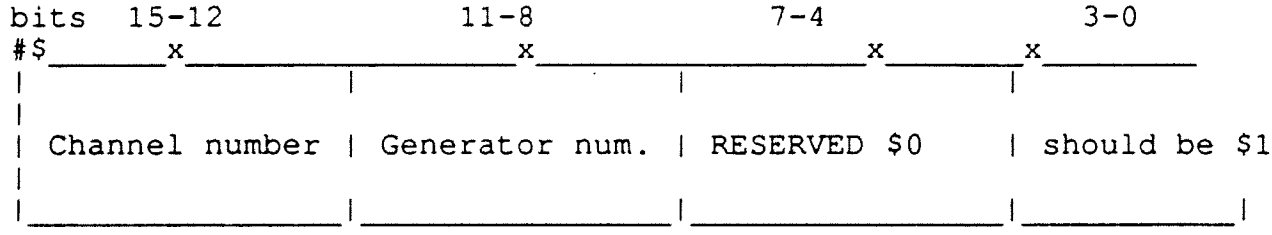
--> BITS 15-12 (#$x000) CHANNEL NUMBER: the number in these 4 bits specifies which of the 8 DOC channels plays the sound. It is usually best to set this to zero. The DOC channels, basically, have no meaning with the standard IIGS; stereo cards work by routing even channels to the right and odd channels to the left.

--> BITS 11-8 (#$0x00) GENERATOR NUMBER: this 4 bit value may range from $0 to $e and specifies on which of the Sound Tools 15 generators the sound will be played.

--> BITS 7-4 (#$00x0) RESERVED: must be set to 0

--> BITS 3-0 (#$000x) MODE: the value in these 4 bits determines the method used for playing the sound, at the moment the only supported value is $1 (so always set this to 1).

FIGURE 1: (layout of parameter)



In almost all cases when you want to play a sound, you will want to leave the genNum parameter value like: #$0x01, replacing the 'x' with your selection of the generator number ($0-$E) to play the sound with.

After we push the genNum parameter on the stack, we have to push a pointer to a parameter block (called paramBlock in our examples). The format of this parameter block is as follows:

```
paramBlock
waveStart    adrl   SoundPtr ;pointer to our sound in main memory
waveSize     dw     #64      ;our sound size in pages (64pages=16K)
freqOffset   dw     #200     ;playback sound at 10281 hertz...
docBuffer    dw     #$2000   ;start sound at $2000 in DOC RAM
bufferSize   dw     #5       ;I want my buffer in DOC RAM to be 8K
nextWavePtr  adrl   #0       ;no other waveforms to play now...
volSetting   dw     #$FF     ;maximum volume
```

#1--waveStart LONG (pointer to sound in main memory)...
This parameter is a long pointer to the sound data somewhere in main memory. Your sound data should terminate with several bytes of $00's to mark the end of the sound (practically all sound digitizing/modifying program will do this automatically).

#2--waveSize WORD (size of sound in main memory---in _pages_)
This parameter specifies the size of the sound in main memory. A very, very important note, this parameter specifies the size in PAGES--_not_ bytes. For high-level folks who may not know, a 'page' is equal to 256 bytes. Although you must specify your sound size in blocks of 256 bytes, as long as the sound data terminates with several $00 bytes, any left over garbage from beyond your sound won't be picked up ($00 tells the DOC to stop playing the sound) even if the last page is only partially used.

#3--freqOffset WORD (frequency for playback)
This parameter specifies the playback rate of the sound. In other words, the value in this will determine whether that digitized voice you recorded will sound like a monster, a human, or a duck! This value can easily be converted to and from Hertz:

Convert value to Hertz: VALUE*1645/32= Hertz

Convert Hertz to value: HERTZ*32/1645

#4--docBuffer WORD (address in Sound RAM of buffer for sound)
This parameter is a pointer to the start of a buffer in the 64K of sound RAM. This parameter tells the Sound Toolset where in the DOC RAM we want our buffer for the DOC to use for playing the sound. As all sounds in DOC RAM must start on a page boundary, the low byte is not used and should be set to zero. It is important to note that your buffer for playing the sound must be aligned in memory in increments equal to the size of the buffer, e.g. if you your buffer size is 4K, this parameter must point to a part of DOC memory that rests on a 4K boundary (like $0000, $1000, $2000...), a 8K buffer must be aligned on 8K boundaries ($0000,$2000,$4000...).

#5--bufferSize WORD (size of buffer in Sound RAM to play sound in)
The low three bits of this word specify the size of the buffer in Sound RAM you wish your sound to be played in. The larger the buffer, the less often the Sound Toolset has to refill it from main memory. The eight possible settings are as follows (other thirteen bits should be zero):

FIGURE 2

```
if bits 2-0 are:   then RAM buffer is:
000 (0)             1 page (256 bytes)
001 (1)             2 pages (512 bytes)
010 (2)             4 pages (1024 bytes)
011 (3)             8 pages (2048 bytes)
100 (4)            16 pages (4096 bytes)
101 (5)            32 pages (8192 bytes)
110 (6)            64 pages (16384 bytes)
111 (7)           128 pages (32768 bytes)
```

#6--nextWavePtr LONG (pointer to optional next parameter block)
This parameter is a pointer to a parameter block you wish to use after the current sound is finished. For example, if this pointer is set to the start of the parameter block, the sound would be played continuously.

#7--volSetting WORD (volume at which to play sound)
The low byte of this value (ranging from $00-$FF) specifies the volume at which to play the sound.

Stop I say, Stop!

Before calling FFStartSound, you must make sure that the generator you selected will be ready to play. The FFStopSound call stops whatever sound (if any) the selected generators are playing and make them ready to play new sounds. There is only one parameter to the FFStopSound call, a word sized parameter that specifies which generators to stop. There are 16 (0-15) bits in a word, and 15 generators (numbered 0-14), so with the exception of bit 15, each bit in the word parameter corresponds to a generator, e.g. setting bit 4 will stop generator 4, bit 0 and generator 0, etc. Here is a small code example:

```
PushWord #%0001_0001_0010_0001  ;stop generators 12, 8, 5 and 0
_FFStopSound
```

Play It Again Sam

Although FFStartSound is a very useful call, there are two other tool calls for playing sounds--FFSetUpSound and FFStartPlaying. They were added with the release of System Software 5.0 and work together to produce sound. FFSetUpSound takes the same parameters as FFStartSound, and, like FFStartSound, does all the necessary setup without actually playing the sound. After a call to FFSetUpSound, you begin playing your sound by calling FFStartPlaying. FFStartPlaying uses the same generator flag word as FFStopSound does. In order to play the correct sound, specify the same generator with FFStartPlaying that you did with FFSetUpSound. You do not have to call FFStopSound to a generator before issuing a FFStartPlaying call, unlike FFStartSound. Here is a brief assembly code-segment:

```
PushWord #genNum      ;specify channel, generator...
PushLong #paramBlock ;pointer to sound parameter block
_FFSetUpSound

PushWord #genMask     ;select generator(s)
_FFStartPlaying       ;and begin playing setup sound
```

These calls are most useful when you have a sound you wish to play over and over again; you can simply set it up with FFSetUpSound using whatever generator you like, and call FFStartPlaying specifying the generator to play the sound.

Bag O' Tricks

Here is one trick you can do, using FFSetUpSound and FFStartPlaying to play sounds without taking up any of the main RAM (after an initial setup phase). If the sound you want to play is smaller that the size of the DOC RAM buffer (i.e. a 12K sound would fit in a 16K buffer), then after calling FFSetUpSound, you can dispose of the memory the sound takes up in main memory. FFSetUpSound puts the sound data in the DOC RAM, and since it fits in the buffer, FFStartPlaying will never have to go back to main memory to get more data. There are limitations. Since there is only 64K of sound RAM, and since the maximum buffer size is 32K, it may be useful only when you want to add some nifty little sound effects.