Documentation Développeurs
Apple Computer France 1987

Vol. 2
Fas. 12

Document développeur numéro 49

Note Synthesizer

type d'upgrade de ce ducument : 5
1 Documentation de première catégorie inchangée
2 Documentation de deuxième catégorie mise à jour
3 Documentation de deuxième catégorie inchangée
4 Mise à jour payante de la documentation de première catégorie
5 Mise à jour gratuite de la documentation de première catégorie
6 Nouveautés payantes non vitales
7 Nouveautés gratuites et vitales

Taille : 20 page(s) environ

Domaine : Tool 25

VERSION : 00:50
DATE : 25.09.86

Date: Sept. 25, 1986

Author: Bill Mauchly

Subject: Note Synthesizer ERS

Document Version Number: 00:50

Revision History

June 30: Rev 0.2

Startup is the only one of the first eight functions which is implemented.
The priority scheme was changed slightly to get around a flaw in the original logic.
Semitone and Volume arguments are now words, not bytes.
The order of the data in the Instrument have been changed.
Supports 14 generators.

July 27, 1986: Rev 0.3

A problem with the Control Panel was eliminated by adding a new argument to the startup call: the user update routine. The problem was, the Note Synth re-enabled interrupts from within the interrupt handler and so did the Control Panel. Now, if the user wants interrupts enabled, he/she can do it inside this user update routine. The real reason for this new routine is to give sequencer programs a timer. To this end, a second parameter, the update rate, is also included in startup.
Uses SetUserVector of Sound Tools.
Implements all mandatory functions.
The interrupt timer runs all the time.

August 18, 1986: Rev 0.5

Changed structure of Instrument to save space and eliminate extra level of indirection.
NoteOn can be called from backround or Interrupt.
Reset does Shutdown.
DeAlloc turns DOC off.

GENERAL

The Note Synthesizer is a ram-based tool set, number (decimal) 25. It provides a way of making complex musical sounds on the Cortland when it is equipped with the ENSONIQ Digital Oscillator Chip (DOC).

An application program will use the note synthesizer by making tool calls at the beginning and at the end of each note to be played. The first call is to allocate one of the sound generators. Then the specified DOC generator will be set up to produce sound with a NoteOn call. During the course of the note the DOC registers for that generator will be automatically updated on a regular basis to create the shape of the sound. This happens from a timer interrupt routine which is part of the note synthesizer. The end of a note, called the release, starts when a NoteOff call is made. Sometime later, when the note has died away to zero, the generator will be automatically deallocated.

Generators and GCB's

There are 32 oscillators in the DOC. Of these, 2 are reserved for use by Apple in the future. The remaining 30 are grouped into pairs, called generators. When the Note Synth starts up it grabs one generator to use as a timer. The remaining 14 generators are allocated on a priority basis as needed.

One page of bank zero memory must be assigned to the Sound Manager Tool Set when it is started up. This area is divided into 15 blocks of 16 bytes each, which are called Generator Control Blocks (GCB). The first byte of a GCB indicates which synthesizer is using that generator (if any). The definition of the other 15 bytes depends on which synthesizer is using it.
The GCB is used as a mailbox by the note synthesizer. It contains the current values of three "knobs" or controllers which may be changed by the application program. These are for pitchbend, vibrato depth, and volume. All three controls have a range of 0 to 127. After a call to NoteOn, the GCB will be set up as follows:

GCB:	SynthID	byte	= note synth ID = 2
	GenNum	byte	= [0..14]
	Semitone	byte	as specified in call
	Volume	byte.	as specified in call
	Pitchbend	byte	64 = no bend
	VibratoDepth	byte	as specified in instrument
	Note Synth internal variables: 10 bytes		

Priority Allocation

Generators may be shared among various sound producing tools. Because the note synthesizer will, by far, use the most generators, and allocate them more often, the generator allocation is one of the note synthesizer functions. It is not uncommon for music to 'accidentally' request a new note when all the available generators are busy. The allocation scheme will allow the note synthesizer to steal generators from itself, but

not from other types of synthesizers.

A generator's priority may range from 0 to 128. When priority is zero, that means that that generator is not being used, and therefore free to be used. When it is 128, then that generator is locked and may not be stolen. Priorities between 0 and 127 are used by the note synthesizer to control the stealing of notes.

When a generator is allocated to be used by one of the synthesizers, the generator is given a new priority. The generator allocation function will return with the lowest priority generator.When the note synthesizer uses a generator, it automatically lowers its priority when the envelope hits the sustain portion, and again when it hits the release portion. When the note stops, it returns the generator to zero priority. Other synthesizers in the system (the free-form synth) should always get a generator with a priority of 128.

Interrupt Timer

The note synthesizer will use one oscillator as a free running timer to provide the update rate for the envelopes. This timer can also be accessed by an application program by using the hook provided in the Startup call. The update rate is usually between 60 and 200 Hz. The timer runs all the time, until a reset or a shutdown call is made.

Generators used for sound production by the note synthesizer should have their interrupts disabled in the DOC control register defined in the instrument. The Note Synth will treat all interrupts from its oscillators as timer interrupts.

An application which uses incoming MIDI will have a hard time on the Cortland if it uses tools. Most tools disable interrupts for long periods of time. The Note Synthesizer normally runs an interrupt service routine with interrupts disabled, and this routine may take several milliseconds when many notes are playing. Since MIDI can generate interrupts as often as every 333 usec, there is a problem. The application can force the note synth interrupt service routine to run with interrupts ENABLED. This is accomplished by installing a user timer routine. It will get called in the interrupt, immediately before the note synth service routine. If the user subroutine returns with irqs enabled, that state will remain throughout the note synth interrupt service routine. This should prevent loss of incoming MIDI data.

DOC Memory

It is up to the application to get the needed waveforms into DOC MEM, using the WriteRamBlock function. At no time should a zero be placed in the first 256 bytes of DOC memory. That would cause the timer oscillator to halt.

NSBootInit

Function Number:	$01
Input:	none
Output:	none
Errors:	none

Should be called only by Tool Loader.

NSStartup

Function Number:	$02	
Input:	*Update Rate*	push WORD
Input:	*User Update Rtn*	push LONG
Errors:	AlreadyInit = $1901	
	SoundNotInit = $1902	

The Sound Tools (Tool #8) should be started first or this startup will fail. The note synth shares a page of bank zero with the sound tools. NSStartup is necessary before using the other functions.

The Update Rate is the rate at which the envelopes and LFOs will be generated, and the rate at which the User Update Routine will get called. The rate value is in units of .4 Hertz. Some typical rates would be:

Rate: 60 Hz	use 60/.4 = 150
Rate: 100 Hz	use 100/.4 = 250
Rate: 200 Hz	use 200/.4 = 500 default

Use low rates for low overhead. Use a higher rate for smoother sounding envelopes and better sequencer timing resolution.

The User Update Routine is the address of a routine which will be called on every timer interrupt. It is intended to be used by a sequencer program. There are various ways that a sequencer can create various tempos from a fixed clock. The routine will be called will a jsl from native mode, with index and memory long, and should return with an rtl. If Interrupts have been re-enabled by the user update routine, they will remain that way throughout the note synthesizer update routine. If the user chooses to enable interrupts inside this routine, then it should be reentrant. If this argument is zero, then no routine will be called.

EXAMPLE

```
            pushword #150                ; 60 Hz update rate
            pushlong #MyRoutine
            _NSStartup
            .
            .
MyRoutine   lda >MyTimeCount
            dec a
            sta >MyTimeCount
            rtl
```

NSShutdown

Function Number:	$03
Input:	none
Output:	none
Errors:	NotInit = $1923

Shutdown turns off all generators *used by the note synth* and clears their priority. It replaces the sound IRQ vector with the one which was present before startup.

NSVersion

Function Number:	$04	
Input:	none	
Output:	*Version Number*	WORD
Errors:	None	

EXAMPLE

NSReset

Function Number:	$05
Input:	none
Output:	none
Errors:	none

This performs the same function as Shutdown.

NSStatus

Function Number:	$06	
Input:	none	
Output:	StartStatus	word ($FFFF=started,0=not)
Errors:	none	

This function returns 0 or $FFFF depending on whether the Note Synth was started yet.

AllocGen

Function Number:	$09	
Input:	RequestPriority	WORD
Output:	GenNum	WORD
Errors:	NoneAvailable = $1921	

AllocGen is a request for a sound generator. If successful, it returns a generator number, from 0 to 13. Which generator is returned is determined by the current priorities of the 14 generators. If one of the generators is free, that is, it has a priority of zero, then the first free one is returned. If none are free, then it looks for one to 'steal'. It finds the lowest priority generator. If that generator's priority is lower than, or equal to, the *RequestPriority,* then that generator is 'stolen'. If all generators are already of a higher priority, then the request fails. There is one exception; a generator with a priority of 128 is never stolen.

A fail is indicated by carry set on return. When successful, the generator is assigned a priority equal to *RequestPriority.*

DeallocGen

Function Number:	$0A	
Input:	GenNum	WORD
Output:	none	
Errors:	BadGenNum = $1922	

This function sets the named generator's priority to zero. It also makes sure that the oscillators are halted.

AllocGen and DeallocGen can be used to gain control of generators in the DOC for any of the synthesizer functions in the Cortland Tools, or even a user defined synthesizer function. The programmer can guarantee success in this allocation by always requesting the same priority. This means that there will never be a situation where a note will not sound because all the generators are busy. That is the simplest way to use the dynamic allocation. A more advanced use of priority would be to put higher priorities on bass notes and melody lines.

Note that this call is not necessary for generators that have played notes. The note synth automatically deallocates generators when their envelopes have dropped to zero.

An Error code is returned if the Generator number is greater that 13.

NoteOn

Function Number:	$0B	
Input:	*GenNum*	WORD
	Semitone	WORD
	Volume	WORD
	InstrumentPtr	LONG
Output:	None	
Errors:	AlreadyOn = $1924	

This function initiates the sounding of a note on the specified instrument.
GenNum is a generator number from 0 to 14. The *GenNum* used in the call should usually be obtained immediately prior to the call from a call to AllocGen.
The *Semitone* is specified in MIDI standard format: a value from 0 to 127, where middle C is 60.
The *Volume* parameter is also in the range of 0 to 127, and can be treated like MIDI velocity. This volume parameter is copied into the GCB. It is used as a scaler of the amplitude envelope. Each 16 steps of this parameter correspond to a 6 Db change in volume. Note that when the sum of the volume and the instantaneous envelope is less than 128, no sound will be heard because of the 48 Db dynamic range of the DOC.
The *InstrumentPtr* is a pointer to an Instrument structure, which is defined below, on page 15.

EXAMPLE

```
        pushword #0                 ;space for the GenNum
        pushword #64                ; Priority of this note: average
        _AllocGen
        pla
        sta GenNum

        pushword GenNum
        pushword Semitone
        pushword #127               ; max volume
        pushlong #Instrument        ; LONG ptr to inst definition
        _NoteOn

... wait a while
        pushword GenNum
        pushword Semitone
        _NoteOff
```

NoteOff

Function Number:	$0C	
Input:	*GenNum*	WORD
	Semitone	WORD
~~WORD~~		
Output:	None	
Errors:	None	

This function causes the envelope generator of the given note to go to the release stage. The release usually causes the volume to drop to zero in a short time. When the envelope reaches zero, the note will no longer be heard and the note is considered off. The generator's priority will then be set to zero, indicating that it is free.

The *GenNum* and *Semitone* should be the same ones that were specified in the corresponding NoteOn call. There are cases where the note is no longer sounding; for example, if the envelope had already dropped to zero or if the generator had been stolen to play another note. NoteOff checks to make sure that the named generator is indeed playing the named semitone.

AllNotesOff

Function Number:	$0D
Input:	none
Output:	none
Errors:	none

This function turns off all the notes that the note synthesizer is playing and returns them to zero priority. It will not shut down other generators, such as those used by the free-form synthesizer.

INSTRUMENT DEFINITION

An Instrument is a data structure which resides somewhere in Cortland memory. A NoteOn call must pass a pointer to an instrument.

Instrument:

offset	Field Name	size
0	*Envelope*	24 BYTES
24	*ReleaseSegment*	BYTE
25	*PriorityIncrement*	BYTE
26	*PitchBendRange*	BYTE
27	*VibratoDepth*	BYTE
28	*VibratoSpeed*	BYTE
29	*Spare*	BYTE
30	*AWaveCount*	BYTE
31	*BWaveCount*	BYTE
32	*AWaveList:*	AWaveCount * 6 BYTES
X	*BWaveList:*	BWaveCount * 6 BYTES

(X = 32 + AWaveCount * 6)

The *Envelope* is composed of up to eight linear segments. Each segment is described by a breakpoint and an increment. During each segment, the volume of the note will ramp from its current value to the breakpoint value. The slope, and thus the time of the ramp, is determined by the increment.

The breakpoint should be a byte between 0 and 127. It represents sound level in a logarithmic scale: each 16 steps change the amplitude by 6 Db.

The slope is described by an increment which will be added or subtracted from the current level at the update rate (100 times a second). The increment is a two byte fixed point number, that is, the lower 8 bits represent a fraction. Thus when the increment is 1 it represents 1/256. In this case, the increment would have to be added 256 times (2.56 seconds) to cause the envelope level to go up by 1.

The envelope is a list of breakpoints and increments:

stage 1: breakpoint	increment
stage 2: breakpoint	increment

stage 3:	breakpoint	increment
stage 4:	breakpoint	increment
stage 5:	breakpoint	increment
stage 6:	breakpoint	increment
stage 7:	breakpoint	increment
stage 8:	breakpoint	increment

Increment 1 is used to go from the initial level of 0 up to the level of breakpoint 1. Increment 2 is used to go from breakpoint 1 to breakpoint 2, and so on. The sustain level of the envelope, if there is one, is created by setting the increment to 0, causing the envelope to get "stuck" on that level. The release segment of the envelope is specified by the *ReleaseSegment* parameter, which must be a number from 0 to 7. The release may take several segments to get to zero. The last breakpoint should always be zero.

To compute the time of a segment:

$$\text{time} = \frac{|\text{last breakpoint} - \text{new breakpoint}| * 256}{\text{increment} * \text{update rate}}$$

For example, to ramp from 30 to 40, with an increment of 25, and an update rate of 100 Hz:

$$\text{time} = \frac{|30 - 40| * 256}{25 * 100\ \text{Hz}} = \frac{2560}{2500} = 1.02\ \text{seconds}$$

PriorityIncrement is a number which will be subtacted form the generator priority when the envelope reaches the sustain segment. Whe it reaches the release segment the priority will be cut in half. The proirity of each generator will also be decremented by one each time a new generator is allocated. This causes 'older' notes of to be preferred for stealing.

PitchbendRange is the number of semitones that the pitch will be raised when the 'pitchwheel' reaches 127 (the center value is 64). The valid values for PitchbendRange are 1, 2, and 4.

VibratoDepth is the (initial) fixed depth of vibrato, ranging from 0 to 127. Vibrato is a triangle shaped LFO modulating the pitch of both oscillators in a generator. A vibrato depth of zero will turn the vibrato mechanism OFF, which saves some CPU time.

VibratoSpeed controls the rate of the vibrato LFO. It can be any byte value. The actual frequency will depend of the update rate set during Startup.

AWaveCount and BWavecount tell the note synth how many waves there are in the following wavelists. There can be up to 255 waves in each list.

The **Wavelist** structure is a variable length array where each entry is 6 bytes long. Each entry represents a **Waveform**. The information is particular to the DOC; the user should refer to the DOC specification when creating instruments. Each 6 byte entry represents a waveform and contains information about the allowable pitch range of the waveform. This means that the waves can be "multi-sampled" across (an imaginary) keyboard. When a note is played, the WaveList A and B will be examined and ONE waveform will be picked out and assigned to each oscillator.

Each **Waveform** in a Wavelist has the following 6 byte format:

TopKey	byte
WaveAddress	byte
WaveSize	byte
DOCMode	byte
RelPitch	word

TopKey is the highest MIDI semitone that will be played by this waveform. The synth will examine the topkey field of each waveform until it finds one greater than or equal to the note it is trying to play. The items in the Wavelist should be in order of increasing TopKey values. The last wave in a Wavelist should have a TopKey of 127. The TopKey value is, then, the split point between waveforms.

The next three bytes will be picked up and stuffed directly into the DOC registers. The WaveAddress is the high byte of the waveform address. The WaveSize sets both the size of the wavetable and the frequency resolution. The DOC mode goes into the mode register. The interrupt enable bit will be ignored.

Briefly, some of the ways that DOCMode may be used:

Synthesizer: both oscillators, A and B, in free run mode. ($00)

Sampled, no loop: Osc A in single cycle and trigger peer mode ($06); Osc B in single cycle and halt mode, with halt set ($03). Osc A will complete and start Osc B, which will play to the end and stop.

Sampled, with loop: Osc A in single cycle and trigger peer mode ($06); Osc B in free run mode, with halt set ($01). Osc A will complete and start Osc B, which will play continuously until the note ends.

RelPitch is a two-byte word which is used to tune the waveform. This will compensate for different sample rates and waveform sizes. The high byte is in semitones, but can be a signed number. The low byte is in 1/256 semitone increments. Note that the low byte is first in memory on the 65816... A setting of

zero is the default for sounds that have one cycle per page of waveform memory.

The wavelist structure is designed to give great flexibility in creating realistic instrument timbres. It allows 'multi'sampling' with different samples of sounds on different ranges of pitch. It allows mixing of various size waveforms, with different tuning on each one. One special application might be to use a different **Waveform** entry on each semitone, to allow seperate tuning of each note. This would be a way to duplicate special tuning systems, like just temperment. The wave pointers need not be different in this case, just the RelPitch fields.

Tuning is accurate to 1/128 of a semitone in the note synth software, subject to the resolution setting of the DOC. For accurate tuning on lower notes it may be necessary to use higher settings in the DOC resolution register.