Sound Tools ERS

REV 1.7

Developed by:
Gus Andrade

Revision History:

rev.1.0 Initial release

rev. 1.1 Start sound call added.

rev. 1.2 Changed Start sound call.
Added added a description for generators.
Added Sound tools status call for sound tools startup call.
Added individual generator status call.

rev. 1.3 Updated the Start Sound call to include the new parameter block.
Added FFSoundDoneStatus function ($14).

rev. 1.4 Changed the Frequency formula in the FFStartSound call to use integer values instead of floating point values for the frequency register calculations.
Changed the descriptions in the low level routines to access the DOC registers or ram.

rev. 1.5 Updated the Generator/Mode word parameter in the FFStartSound call.

rev. 1.6 Changed the paramerter block format for the FFStartSound call to conform with the word aligment of parameters passed to functions.
Reserved oscillators thirty and thirty one for use by Apple Computer. These two oscillators can NOT be used by application programs.

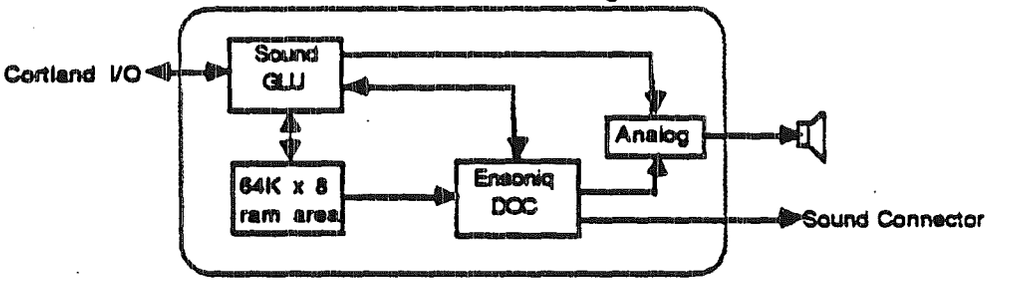
rev. 1.7 Updated stop sound call to show 1 = stop corresponding generator.
Added examples for each of the Sound Tools function calls.

1.0 Introduction.

The sound tool package gives developers the ability to access the Sound hardware without having to know specific hardware I/O addresses. The Cortland sound hardware comes in two configurations. The first configuration is 100% compatible with the Apple //e sound capabilities. In this mode applications toggle a soft switch, which in turn generates clicks in a speaker. Also, with Cortland it is possible for an application to control the volume of the speaker.

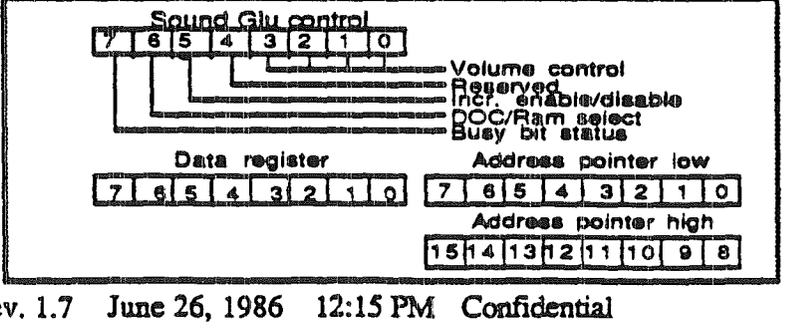
The second configuration requires the Ensoniq (DOC) digital oscillator chip and two 64K x 4 ram chips. The sound tools will contain all of the firmware routines required to access the hardware in the Ensoniq configuration. The following block diagram shows the major functional blocks of the sound hardware.

Cortland Sound Block diagram



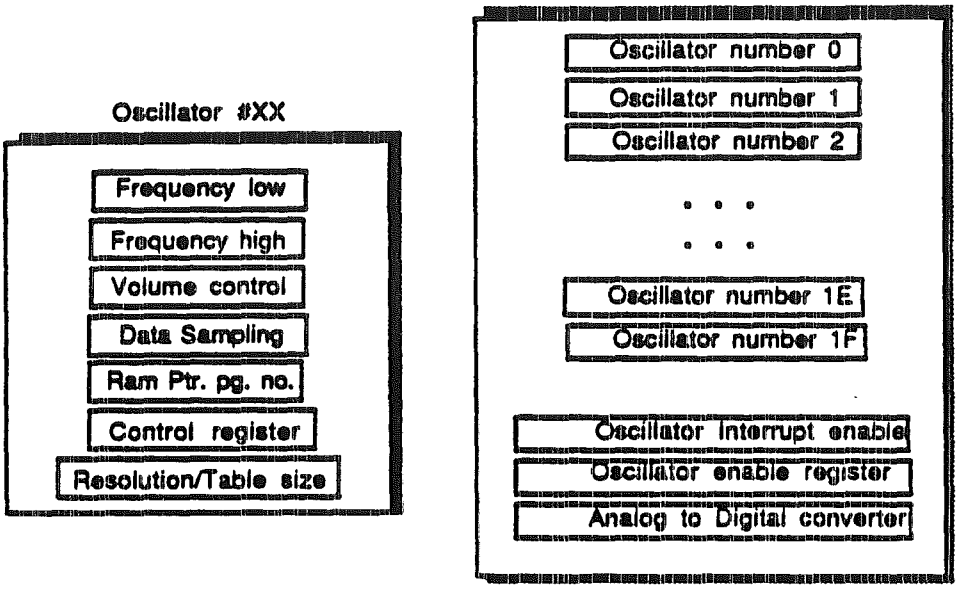
The sound GLU acts as the interface chip between the Cortland I/O and system volume, ram chips or the DOC. The following diagram shows a diagram for the register breakdown for the sound glu:

Sound glu register breakdown



The DOC ram is used to store waveforms which will be used by the DOC for sound generation. The DOC is the work horse of the sound system. With this chip we can create sounds of any pitch and duration. A register breakdown of the DOC follows:

Ensoniq Digital Oscillator Chip
Register Allocation



DOC register table

REG#	Function	D7	D6	D5	D4	D3	D2	D1	D0
00-1F	FREQ. LOW	FL7	FL6	FL5	FL4	FL3	FL2	FL1	FL0
20-3F	FREQ. HI	FH7	FH6	FH5	FH4	FH3	FH2	FH1	FH0
40-5F	VOLUME	V7	V6	V5	V4	V3	V2	V1	V0
60-7F	DATA SAMPLE	W7	W6	W5	W4	W3	W2	W1	W0
80-9F	WAVEFORM TABLE PTR	P7	P6	P5	P4	P3	P2	P1	P0
A0-BF	CONTROL	CA3	CA2	CA1	CA0	1E	M2	M1	H
C0-DF	BANK SEL/TBL. SIZE/RES.	X	BS	T2	T1	T0	R2	R1	R0
E0	OSCILLATOR INTERRUPT	IFC	1	04	03	02	01	00	1
E1	OSCILLATOR ENABLE	X	X	E4	E3	E2	E1	E0	X
E2	A/D CONVERTER	S7	S6	S5	S4	S3	S2	S1	S0

Please refer to the Ensoniq DOC Ers for a detailed description of the part.

The analog section contains all the circuitry needed to amplify and filter the signal coming from the Sound Glu or the DOC, which will be sent to the speaker.

Finally, the sound connector gives developers the ability to design

interface cards, which can take the tones generated by the DOC and modify them further. Two examples of possible sound cards are, programmable filter stereo interface cards, and sound sampling cards. The remainder of this document will deal with a detailed description of the Sound tool calls and how they can be used to access the hardware to generate sounds.

1.1 Sound Tools Definitions.

An *oscillator* is defined as the basic sound generating unit in the DOC. The DOC contains thirty two oscillators; each of which can function independently from all the other oscillators.

One of the modes the DOC can be set to is called swap mode. In this mode each pair of oscillators is grouped together to form a swap pair of oscillators. This is the mode used by the Free Form synthesizer to generate sounds. Each of these swap pair of oscillators is called a *Generator.* There are fifteen generators defined in the Cortland sound system. Oscillators thirty and thirty one a reserved for use by Apple Computer and should not be used by application programs. An oscillator to generator translation table has been defined to get the generator number corresponding to a particular oscillator number.

Before a generator can be accessed, a sound tools startup call must be made. This call assigns a work area for the sound tools. The work area is broken down into sixteen groups of sixteen bytes each. Each sixteen byte group is defined to be a *generator control block (GCB).* The first byte of each GCB is defined to contain the synthesizer mode being used by that generator. The low nibble of the byte contains the mode. The high nibble is reserved for use by the system. The remaining fifteen bytes are user defineble.

The Sound tools set is made up of four main blocks; the Free Form Synthesizer, the Note Synthesizer, the Note Sequencer and the Instrument generator.

1.0 Free form synthesizer tool set definition.

As mentioned before, the tool set gives a developer the ability to control the sound hardware without having to access the hardware registers directly. The tool set is defined from the point of view of a complete sound system. The tool set must be able to read and write to ram, read and write to the DOC registers and raise and lower the volume.

The sound tool package is accessed through the Tool locator. This tool locator lets an application set up parameter lists on the stack, call tool functions and return to the caller with return parameters on the stack. It is the responsibility of the caller to make room on the stack for values which may be returned to the caller from the tool calls.

The Sound Tool set has a tool number assigned to it. With this tool number the Tool Locator can access the sound tools.

The sound tool calls are broken down into two groups. The first group of calls is made through the Tool Locator. Each of these calls has a function number assigned to it. With this function number the Tool locator knows which function to call within the tool set. All parameters for these calls are passed on the stack. Function results are returned to the caller in the stack. The number of parameters will vary depending on the type of call being made. It is the responsibility of the individual tool functions to do the stack manipulation to keep it aligned. Also the accumulator and the carry bit will reflect the success or failure of the function call. Please refer to the "Tool locator" documentation for a detailed description of the interface.

The second group is a set of routines which can be accessed through a jump table located somewhere in ram. Parameters are passed to these routines in the processors registers. Results from these calls are passed back in registers. The following list gives a breakdown of the sound tools.

Sound tool function calls:

Group A (Function calls)		Group B (Low level routines)**
SoundBootInit	= $01	Read Register
SoundStartup	= $02	Write register
SoundShutdown	= $03	Read Ram
SoundVersion	= $04	Write Ram
SoundReset	= $05	Read Next
SoundToolStatus	= $06	Write Next
WriteRamBlock	= $09	
ReadRamBlock	= $0A	
GetTableAddress	= $0B	
GetSoundVolume	= $0C	
SetSoundVolume	= $0D	
FFStartSound	= $0E	
FFStopSound	= $0F	
FFSoundStatus	= $10	
FFGeneratorStatus	= $11	
SetSoundMIRQV	= $12	
SetUserSoundIRQV	= $13	
FFSoundDoneStatus	= $14	

** The low level routines are entered through a jump table. The table address can be obtained through a call to "Get Address" function. The format of the jump table is as follows:

	Offset				
Read Register	$00	Addr low	Addr high	Bank	$00
Write Register	$04	Addr low	Addr high	Bank	$00
Read Ram	$08	Addr low	Addr high	Bank	$00
Write Ram	$0C	Addr low	Addr high	Bank	$00
Read Next	$10	Addr low	Addr high	Bank	$00
Write Next	$14	Addr low	Addr high	Bank	$00
Osctable	$18	Addr low	Addr high	Bank	$00
Generator table	$1C	Addr low	Addr high	Bank	$00
Gcb.addr. table	$20	Addr low	Addr high	Bank	$00

SoundBootInit **function #01**

This call is made on system powerup or system reset to bring the sound hardware to powerup state. The call is made by the firmware and can NOT be made by an application program! This call will reset all of the DOC sound memory to $80, zero out the sound tools work areas, halt all the oscillators and turn the volumes down to zero.

Error Codes: None

SoundStartup **function #02**

The Sound tools startup call is made by an application to set up a sound tools work area. This call must be the first call made by the application program. The call initializes a work area to be used by the sound tools. The pointer to the work area must be passed as a parameter to the call. This work area will be used as a zero page. This page will be allocated by calling the memory manager. It must be page aligned and locked until a shutdown call is made. The stack configuration for the call is as follows:

```
Stack configuration for SAppInit

Wap:word ; Work area pointer in Bank $00
```

Error Codes:

$10 = No DOC chip found
$18 = Sound tools already started

Example:

```
PEA Label        ; One page work area in bank $00
_SoundStartup    ; Sound Tools startup macro call
```

SoundShutdown **function #03**

This call will shut down the sound tools. It shuts off all of the oscillators resets the WAP back to $0000 and zeros out the sound tools work memory to zero. There are no parameters passed to the call on the stack and no values returned It is the responsibility of the application to relase the memory allocated to the work area back to the memory manager.

Error Codes: None

SoundVersion **function #04**

This call returns the Sound tools version number. The format of the version number is as specified in the Tool Locator documentation. There are no parameters passed to the call but room must be made on the stack for one word of version information returned to the caller.

Error Codes: None

Example:

```
PEA $0000          ; make room for version
_SoundVersion      ; Sound Tools version call
```

SoundReset **function #05**

This call stops all of the generators which may be generating sound. This call can not be used by an application to stop sound generation. It is intended for use by the firmware to control the shutdown of generators. An application program should use the stop sound call to shut down a generator. This call does not require any parameters on the stack or returns any values back to an application. This call does not update the active generators flag.

Error Codes: None

SoundToolStatus function #06

This call will return the status of the sound tools. It returns a $FFFF if a SAppInit ($02) call has been made; otherwise it returns $0000. Room must be made on the stack for a one word value which will be returned to the caller.

Error Codes: None

Example:

```
PEA $0000          ; make room For sound tools status
_SoundToolStatus   ; Sound Tools Started status
```

WriteRamBlock function #09

The Write Ram Block call will write a specified number of bytes from system ram into DOC ram. The parameter list is made up of the starting address, and a byte count to move. If the sum of the starting address and the byte count are greater than 64K, an error status will be returned.

Stack configuration for write ram block:

```
Source_ptr:Long word   ; data source start address
DOC.start:word         ; DOC buffer start address
Byte_count:word        ; number of bytes to move
```

Error Codes:

$0011 = DOC address range error.

Example:

```
Pushlong Label      ; Source buffer address
PEA DOC.buff        ; DOC ram buffer start address
PEA byte.count      ; number of bytes to move
_WriteRamBlock      ; Write ram block macro call
```

ReadRamBlock **function #0A**

This call reads any number of locations from the 64K DOC ram area into a user specified buffer. The number of bytes and the starting location must not add up to a value greater than 64K, otherwise a range error will be generated. The format of the parameter list is as follows:

Stack configuration for Read Ram block

```
Dest_ptr:Long word  ; Destination system buffer address
DOC.start:word      ; Source start address in DOC ram.
Byte_count:word     ; number of bytes to move
```

Error Codes:

$0011 = DOC address range error.

Example:

```
Pushlong Label      ; System ram buffer start address
PEA DOC.buff        ; DOC ram buffer start address
PEA byte.count      ; number of bytes to move
_ReadRamBlock       ; Read ram block macro call
```

GetTableAddress **function #0B**

This call returns the jump table address for the fast access routines. The table of low level routines is defined as follows:

	Offset				
Read Register	$00	Addr low	Addr high	Bank	$00
Write Register	$04	Addr low	Addr high	Bank	$00
Read Ram	$08	Addr low	Addr high	Bank	$00
Write Ram	$0C	Addr low	Addr high	Bank	$00
Read Next	$10	Addr low	Addr high	Bank	$00
Write Next	$14	Addr low	Addr high	Bank	$00
Osctable	$18	Addr low	Addr high	Bank	$00
Generator table	$1C	Addr low	Addr high	Bank	$00
Gcb.addr. table	$20	Addr low	Addr high	Bank	$00

With the exception of the last three entries, each of these routines are defined later in this document.

The Osctable translates from generator number to oscillator number, . The oscillator number returned through this table is the first oscillator of the pair. The Gcb address table points to the first location of the GCB corresponding to a generator, and the Generator table translates from oscillator number to generator number.

The application making this call must make room on the stack for a long word returned from the call.

Error codes:None

Example:

```
Pushlong $00000000      ; Make room for long address
_GetTableAddress        ; Get table address macro call
```

GetSoundVolume **Function $0C**

This call will read the volume setting for a generator. The possible range of values read back are between $00-$FF. All eight bits are valid for DOC volume registers.

If the generator specified is greater than fourteen ($0E), then the system volume setting will be returned. The hardware for the system volume control uses the low nibble of a byte to set the volume. In order to be consistent with the DOC volume registers, we map the low nibble into the upper nibble of a byte. We end up with each possible system volume setting mapped sixteen times. Volume settings $00-$0F correspond to system volume setting $00, values $10-$1F correspond to system volume $01, etc.

Room must be made on the stack for a one word value which will be returned from the call.

Stack configuration for Get Volume call:

```
Gen_number:word          ; Generator number
```

Error codes:None

Example:

```
PEA $0000          ; room for volume setting
PEA gen.num        ; Generator number
_GetSoundVolume    ; Get volume macro call
```

SetSoundVolume — Function $0D

The set volume call changes the volume setting for the volume registers in the DOC and the system volume. Generator numbers $00-$0E will set the volume on pairs of generators in the DOC. Generator numbers $0F or greater will set the system volume control. The range of values for the volume setting are $00-$FF. The DOC volume registers use all eight bits of resolution. The system volume control will use the upper nibble of the setting to determine the setting.

Stack configuration for SetVolume call:

```
Volume_setting:word     ; new volume setting
Gen_number:word         ; Generator number to  set
```

Error codes: None

Example:

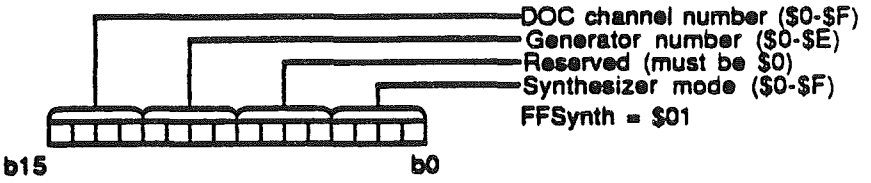
```
PEA New.volume      ; new volume setting
PEA gen.num         ; Generator number
_SetSoundVolume     ; Get volume macro call
```

FFStartSound — Function $0E

This call will enable the DOC to start generating sound on a particular generator based on the parameter list passed to the call. If a generator is already active and a start sound call is made for it, then the previous sound generation process will be terminated and the new sound process will be started. The parameter list for the Start Sound call is as follows:

The stack configuration for StartSound call:

```
GenNumb/FFsynth:word ; Channel no./generator number/mode
```




```
Pblock_ptr:Longword    ; Parameter block pointer
```

The parameter block format:

```
Wave_start:Longword    ; Start address of wave
Wave_size:word         ; Waveform size in pages[1]
Freq_offset:word       ; waveform playback frequency[2]
DOC_buffer:word        ; DOC buffer start address[4]
DOC_buffer_size:word   ; DOC buffer size code[3]
Nextw_addr:Longword    ; Next wave parameter block ptr[5]
Volume_setting:word    ; DOC volume setting.[4]
```

1. The smallest which can be played back is one page. A waveform size of $FFFF will play back 65536 pages.
2. The Frequency register setting can be calculated with the following formula: FR=((32*PF)/1645), where PF=Playback frequency in hertz & FR=Frequency register value.
3. This code assigns a size for the DOC buffer used for the waveform being played. One of these buffers is assigned for each oscillator in the generator pair being used to play the waveform. The DOC start address for the second oscillator is assigned at start address + DOC buffer size.
4. For further information on these settings, please refer to the Ensoniq DOC ERS.
5. These three bytes point to another waveform parameter block. If the setting of the Nextw_addr and Nextw_bank are zero, then there are no more Free-Form synthesizer waveforms to be played back through this start sound call.

Error Codes:

$0012 = NO SAppInit call made
$0013 = Invalid generator number
$0014 = Synthesizer mode error
$0015 = Generator busy
$0017 = Master IRQ not assigned

Example:

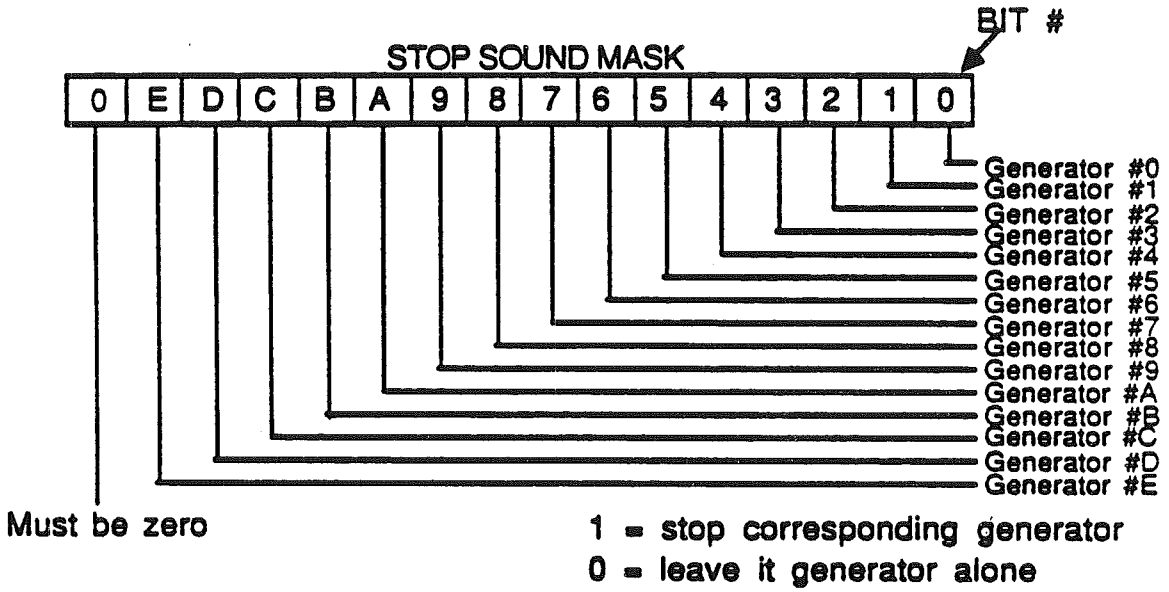
```
                PEA Gen.mode          ; Generator/mode word
                Pushlong Pblock       ; Parameter block pointer
                _FFStartSound         ; Free Form Synth start sound macro
                    ....
                    ....
                    ....
Pblock          equ *                 ; Waveform parameter block
                DC I4'Wave.start'     ; Waveform start address
                DC I2'Wave.size'      ; Wave size in pages (1 page min.)
                DC I2'DOC.Freq'       ; DOC frequency register value
                DC I2'DOC.buffer'     ; DOC ram buffer start address
                DC I2'DOC.buf.code'   ; DOC buffer size code $00-$07
                DC I4'Next.wave'      ; next wave parameter block ptr.
                DC I2'DOC.volume'     ; DOC volume register setting
                ....
                ....
                ....
Next.wave       equ *                 ; Next wave parameter block
                ....
                ....
                ....
```

FFStopSound **Function $0F**

This call will stop sound generators which may be running. A generator running is defined to be one playing a waveform or one which has completed playing a waveform. The generator will stay busy until a stop sound call is made, even though waveform playback has ended. Depending on the setting of a sixteen bit flag passed as a parameter to the function any of fifteen generators will be stopped if running. Each bit position in the stop generator mask corresponds with a sound generator. Bit zero corresponds to generator zero, bit one corresponds with generator number one, and so on. There are only fifteen generators defined. This call does not return any error information back to the caller. The format of the parameter list is as follows:

Stack configuration for Stop Sound:

Gen_mask:word ; generators to stop



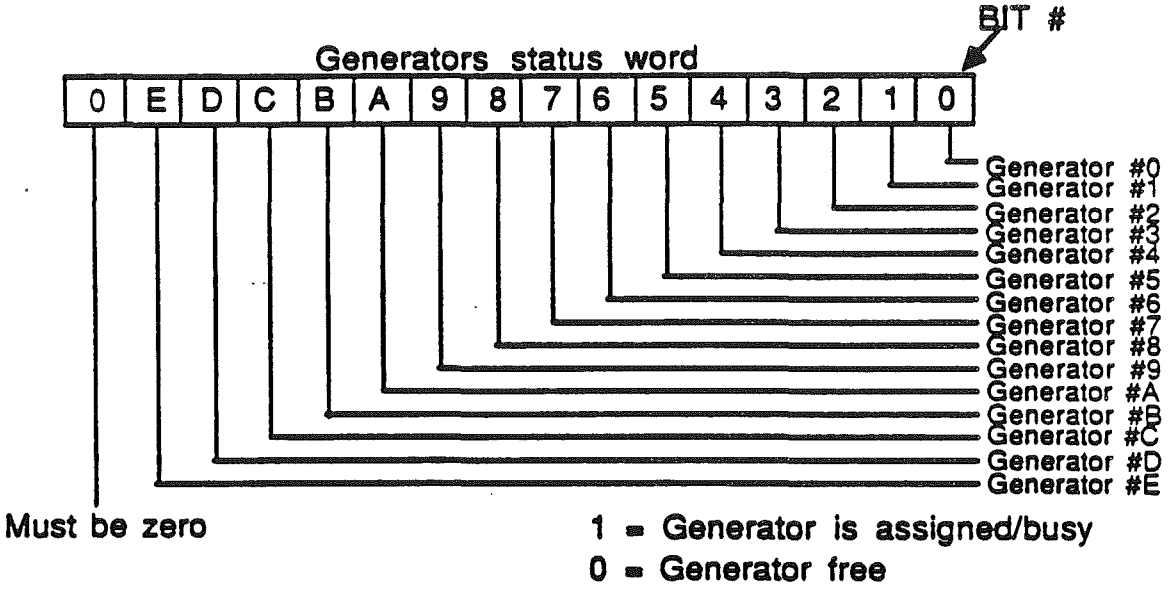
Error Status: None

Example:

```
PEA Stop.mask      ; mask for stop generators
_FFStopSound       ; Free Form Synth stop sound
```

FFSoundStatus — Function $10

This call will return the status of the all fifteen generators. Any bit position in the status word returned from the function call signifies that the corresponding generator is active. There are no parameters passed to the function. The format of the word returned form the call is as follows:



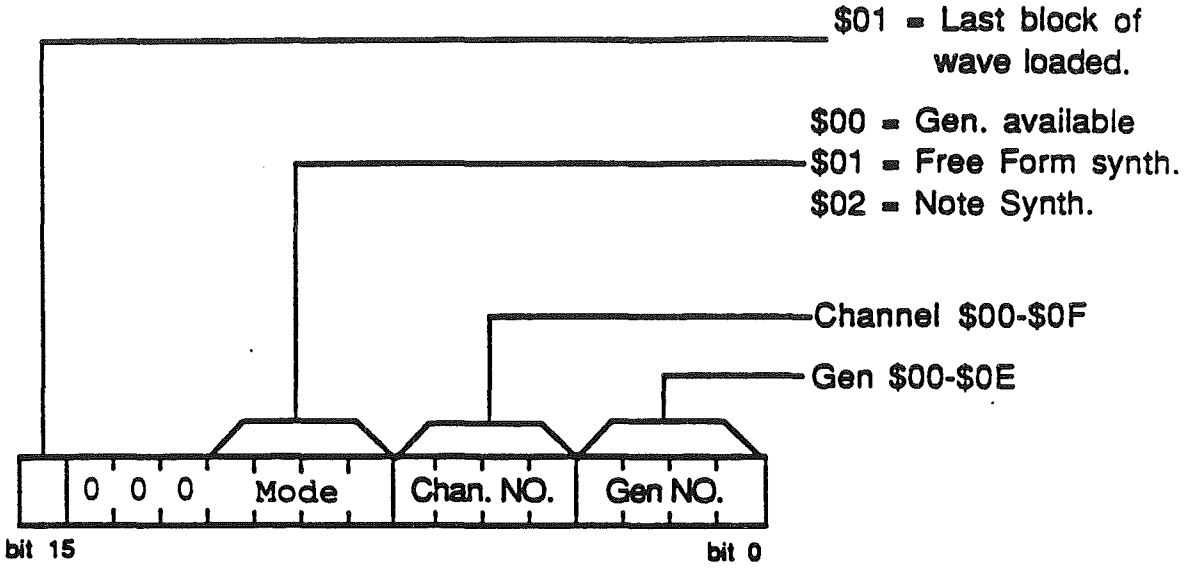
Error Status: None

Example:

```
PEA $0000          ; make room for status word
_FFSoundStatus     ; Generators status macro call
```

FFGeneratorStatus **Function=$11**

This call will read the first two bytes of the GCB corresponding to the generator specified. Room must be allocated on the stack for the word returned from the call. For the Free form synthesizer these two bytes have the following format:



Stack configuration for Gen. status call:

```
Gen_number:word      ; generator number for status
```

Error Codes: None

Example:

```
PEA $0000            ; room for Generator status
PEA Gen.number       ; Generator number
_FFGeneratorStatus; generator status macro call
```

SetSoundMIRQV **Function=$12**

This calls sets up the entry point into the sound interrupt handler. This routine will be accessed every time an interrupt is generated by the DOC. The processor will be in full native mode when the sound interrupt handler is entered. The parameter list for a set sound IRQ vector is as follows:

```
Set Master Sound Irq vector stack config.
        SMaster_irq:Longword  ; Sound Master IRQ vector
```

Error Codes: None

Example:

```
Pushlong Master.irq.vect ; Set master irq vector macro
```

SetUserSoundIRQV **Function=$13**

This calls sets up the entry point for a users synthesizer interrupt handler. When an interrupt occurs for a user defined synthesizer then control will be passed to the ram based synthesizer code through this vector. The old vector installed will passed back to the caller. This old vector must be preserved by the caller. If control is passed to the user vector and the synthesizer mode is not his, then control will passed further down the chain through this vector. Control will be passed through a JSL, therefore the user must return control through an RTL instruction. Room must be made on the stack for long word returned on the stack.

Stack configuration for Set User's Sound IRQ vector.

```
User_irq_vector:Longword        ; New user IRQ vector
```

Error Codes: None

Example:

```
Pushlong $00000000      ; make room for old vector
Pushlong New.vector     ; new vector
_SetUserSoundIRQV       ; set user sound irq vector macro
```

FFSoundDoneStatus **Function #14**

This call will return the status of the Free Form synthesizer sound playing status. If the generator specified is currently playing out a waveform, then the status returned to the caller will be $0000. If the generator is done playing then the status will be $FFFF. Room must be made on the stack for one word of status returned to the caller.

Stack configuration for FFSoundDoneStatus

```
Gen_number:word                ; Generator number
```

Error codes:

$0013 = Invalid generator number

Example:

```
PEA $0000          ; Make room for status
PEA Gen.number     ; Generator number to check
_FFSoundDoneStatus; FFsynth Sound done stat. macro
```

Read register**

This low level routine lets an application read any DOC register. The routine is entered through the Jump table provided by the "GetTableAddress" function call. This call will return to the caller through an RTL instruction. After this call is made the Sound Glu register is left in register access mode with auto increment enabled.

Through the generator to "oscillator" table, an application can assertain the setting of any register corresponding to an oscillator.

Import:

e = 0 ; native mode
m = 1; 8 bit accumulator
x = 0 ;16 bit index registers
X = DOC register to read

Export:

AL = contents of register requested

Error codes: None

DOC register table

REG#	Function	D7	D6	D5	D4	D3	D2	D1	D0
00-1F	FREQ LOW	FL7	FL6	FL5	FL4	FL3	FL2	FL1	FL0
20-3F	FREQ HI	FH7	FH6	FH5	FH4	FH3	FH2	FH1	FH0
40-5F	VOLUME	V7	V6	V5	V4	V3	V2	V1	V0
60-7F	DATA SAMPLE	W7	W6	W5	W4	W3	W2	W1	W0
80-9F	WAVEFORM TABLE PTR	P7	P6	P5	P4	P3	P2	P1	P0
A0-BF	CONTROL	CA3	CA2	CA1	CA0	1E	M2	M1	H
C0-DF	BANK SEL/TBL. SIZE/RES.	X	BS	T2	T1	T0	R2	R1	R0
E0	OSCILLATOR INTERRUPT	IRQ	1	04	03	02	01	00	1
E1	OSCILLATOR ENABLE	X	X	E4	E3	E2	E1	E0	X
E2	A/D CONVERTER	S7	S6	S5	S4	S3	S2	S1	S0

Note: Register types are grouped into register classes. Within each register class, the register number for each oscillator is assigned in assending order. For example: the low byte of the frequency register for oscillator zero is register $00, the low byte of the frequency for oscillator number is register $01. The high frequency register for oscillator number zero is accessed through register number $20, oscillator one uses register number $21 etc... The register numbers are provided in the table defined above.

Write register **

The Write DOC call will write a one byte parameter to any register in the DOC chip. The call will be made through the jump table provided to the application by the tool call "Get Address". To write to an oscillator register corresponding to a generator we get the oscillator number from the oscillator table, bump it by one if we want to access the odd oscillator of the pair, add the base register of the specific register we want to access and then make the write register call through the Write register routine address in the jump table. This call will return to the caller through an RTL instruction. After this call is made the Sound Glu register is left in register access mode with auto increment enabled. Please refer to the "Note" in the Read register description for information on register assignments for each oscillator.

Import:

e = 0 ; native mode
m = 1; 8 bit accumulator
x = 0 ; 16 bit index registers
AL = data to write
X = DOC register number

Error codes: None

DOC register table

REG#	Function	D7	D6	D5	D4	D3	D2	D1	D0
00-1F	FREQ. LOW	FL7	FL6	FL5	FL4	FL3	FL2	FL1	FL0
20-3F	FREQ. HI	FH7	FH6	FH5	FH4	FH3	FH2	FH1	FH0
40-5F	VOLUME	V7	V6	V5	V4	V3	V2	V1	V0
60-7F	DATA SAMPLE	W7	W6	W5	W4	W3	W2	W1	W0
80-9F	WAVEFORM TABLE PTR	P7	P6	P5	P4	P3	P2	P1	P0
A0-BF	CONTROL	CA3	CA2	CA1	CA0	1E	M2	M1	H
C0-DF	BANK SEL/TBL. SIZE/RES.	X	BS	T2	T1	T0	R2	R1	R0
E0	OSCILLATOR INTERRUPT	IRQ	1	04	03	02	01	00	1
E1	OSCILLATOR ENABLE	X	X	E4	E3	E2	E1	E0	X
E2	A/D CONVERTER	S7	S6	S5	S4	S3	S2	S1	S0

Read Ram**

This call will read any Ensoniq ram location specified by the caller. This call leaves the address pointer register in the Sound Glu in auto increment mode and in ram access mode. The call does not do any type of error checking on the address, or data. This call exits back to the caller through an RTL instruction. After this call is made the Sound Glu register is left in RAM access mode with auto increment enabled.

Import:

e = 0 ; native
m = 1 ; 8 bit accumulator
x = 0 ; 16 bit index registers
X = Ensoniq ram address to read

Error codes: None

Write Ram**

This call will write a one byte value to any Ensoniq ram location specified. The call does not do any type of error checking on the address or data value to be written. This call returns to the caller through an RTL instruction. After this call is made the Sound Glu register is left in RAM access mode with auto increment enabled.

Import:

e = 0 ; native
m = 1 ; 8 bit accumulator
x = 0 ; 16 bit index registers
AL = data value to be written
X = Ensoniq ram address to write to

Error codes: None

Read Next**

This call will read the next location pointed to by the Sound Glu address register. The previous call must have been a Read register, write register, read ram, or a write ram call for this call to work properly. Any of these four calls will leave the Sound Glu set to auto increment and pointing to DOC register or ram access mode. After the read is made the Sound Glu address/DOC register pointer will be incremented to the next location.

Import:

None

Export:

AL = data byte read

Error codes: None

Write Next**

This call will write one byte of data to the next DOC register or ram location depending on the setting of the Sound Glu control register. The call will write to DOC registers or ram and then increment the address pointer register in the Sound Glu, if the address pointer register was enabled for auto increment. If a Read register, read ram, write register or write ram call is made then that call will leave the Sound Glu control register in that type of access mode and with auto increment enabled.

Import:

AL = byte value to be written

Error codes: None