1. Introduction

The sound device driver provides a high level interface to the DAVE chip which allows the user to use most features of the chip.

It only allows one EXOS channel to be open to it at a time. The sound chip has four sound sources - three tone channels and a noise channel. The sound driver maintains a queue of sounds for each of these sound channels. Each sound in the queue will be played in turn when the one before it is finished. These sound queues are of a fixed maximum size of 25 sounds in each queue and are stored in the channel RAM.

As well as the sound queues, the sound device maintains a list of envelopes each with an envelope number 0...254. Each sound in the queues refers to a specific one of the envelopes or to the "null" envelope (number 255). When the sound is actually played the specified envelope controls changes in pitch and left and right amplitude of the sound throughout its duration. The storage for envelopes is in the channel RAM and the size of it must be specified with an EXOS variable before opening the channel.

2. General Device Interface

A channel to the sound driver can be opened by giving the device name "SOUND:". Any filename or unit number will be ignored and the open will be rejected (with error code .2NDCH) if there is already a channel open to the sound driver.

An EXOS variable BUF_SND is used to specify how much storage is required in channel RAM for envelopes and must be set up before a channel to the sound driver is opened. It is specified in phases and the sound driver will obtain enough channel RAM to guarantee that the user can define envelopes with a total of the requested number of phases. Thus if the user requests 20 phases then he will be able to define one envelope with 20 phases in it or 20 envelopes each of one phase. Because of the overhead associated with each envelope, the former case takes up rather less RAM than the latter, so if 20 phases are requested then probably more than that number can be defined before storage will be exhausted since most envelopes have more than one phase. The number of phases requested can be from 2 to 255.

The sound driver is write only - it will not accept any read function calls. Printing characters are ignored. All the sound functions are controlled by various control codes and escape sequences. It accepts write block function calls, treating them exactly as if the characters had been written with separate write character calls.

The sound driver has an interrupt routine which is entered 50 times per second (once every TV frame). This scans the sound queues and processes any sounds which are waiting or are currently being played. Each 20ms period is called a 'TICK' and all timing is in terms of ticks.

3. Envelopes

3.1 General Description of Envelopes

An envelope consists of a series of between 1 and 40 phases each of which will be executed in turn when the envelope is used. Each phase is defined by four numbers:

PD - Duration of this phase in ticks (16 bits).
CP - Change value for pitch in 1/512 semitones (16 bits).
CL - Change value for left amplitude (8 bits).
CR - Change value for right amplitude (8 bits).

The change values are signed numbers to allow a change in either direction, up or down in pitch and louder or softer in amplitude. The pitch change can be any signed 16-bit number -32768...32767. The amplitude change values must be in the range -63...+63. They specify the total change in the appropriate parameter which is required during this phase. The specified change in pitch or amplitude will be spread evenly over the duration of this phase.

One particular phase of the envelope may be distinguished as the start of the release phase. The effect of this will be described in detail later but basically controls the dying away of the sound after the note has really finished.

3.2 Format of Envelope Definition

Envelopes are defined by an escape sequence sent to the sound channel:

esc E <en> <ep> <er> [<cp> <cl> <cr> <pd>]*

<en> = Envelope number 0...254 (8 bits).

<ep> = Total number of phases 1...40 (8 bits).

<er> = Number of phases before release (8 bits). If no release phase is required then this should be 0FFh which will result in the sound finishing as soon as the sound duration is expired.

<cp> = Pitch change (16 bits) \
<cl> = Left amp change (8 bits) \ Repeated EP times,
<cr> = Right amp change (8 bits) / once for each phase.
<pd> = Phase duration (16 bits) /

When an envelope definition is received, if there is an existing definition of that envelope then it is deleted. The new definition is then added to the list. If there is insufficient space to store the new one then an error code (.SENBF) will be returned. If this happens then the old definition of that envelope will be lost.

4. Sound Production

To actually produce a sound an escape sequence must be sent which defines the sound. The format of this is:

esc S <env> <p> <vl> <vr> <sty> <ch> <d> <f>

The meaning of each field is:

<env> - (8 bit) Envelope to use for this sound. An envelope number of 255 will produce a "beep" type sound which is of constant amplitude and pitch for the duration of the sound.

<p> - (16 bit) Starting pitch of sound in 1/512 semitones. Only exact quartertones will necessarily be musically correct. The others are generated by linear interpolation. Ignored for noise channel.

<vl> - (8 bit) Overall left amplitude. (0...255)
<vr> - (8 bit) Overall right amplitude. (0...255)

<sty> - (8 bit) Sound style byte. For the noise channel, this byte is put into the noise control register for the duration of the sound. For a tone channel the top four bits are put into the four sound control bits in the sound frequency register for that channel, they thus control filtering, distortion and ring modulation. Zero gives a pure tone or white noise. See separate DAVE chip specification for meaning of each bit in these registers.

<ch> - (8 bit) Source for this sound. 0, 1 or 2 for the appropriate tone channel and 3 for the noise channel.

<d> - (16 bit) Duration of this sound in ticks.

<f> - (8 bit) Flags byte.
b0...b1 - SYNC count for this sound. See later section on synchronisation.
b2...b6 - Not used, should be zero.
b7 - Set to force over-ride of any sound in queue for this channel. Clear to make sound wait its turn.

When a sound is received it is added to the end of the appropriate queue (killing the queue first if bit-7 of the flags byte is set). If the queue is full then there are two courses of action which depend on the state of the EXOS variable WAIT_SND. If this is zero then the sound driver will just wait until there is space in the sound queue, testing the stop key to allow it to be interrupted. If this EXOS variable is non-zero then it will return an error code (.SQFUL) to the user.

5. Processing of Sounds

This section describes how the sound and envelope definitions are interpreted to actually produce a sound when the sound definition reaches the head of its queue. This involves controlling the pitch and amplitude throughout the duration of the sound and also deciding when to start the sound initially (synchronisation).

5.1 Overall Envelope Processing

If the envelope specified in a sound definition is not defined then the appropriate channel will be silent for the duration of the sound. The same thing applies if the envelope definition vanishes during the course of processing the sound.

If envelope number 255 is specified in the sound definition, then the pitch value and left and right amplitude values from the sound definition will be put into the appropriate DAVE chip registers (with the amplitude values divided by 4 to get then into the range 0...63) for the duration of the sound. This produces a simple "beep" type sound.

The production of a sound under control of an envelope requires various current values to be kept. There is a current pitch value which is initialised to the pitch value given in the sound definition. There are also left and right current amplitude values, which are initialised to zero at the start of the sound.

5.2 Envelope Phase Processing

With the current pitch and amplitude values initialised, processing of the envelope can begin. Each phase of the envelope in turn is executed, each one lasting for the number of ticks specified in the envelope definition. At each tick, the current pitch and amplitude values are modified so that the change value in the envelope definition is spread linearly over the duration of the phase. The result at the end of each phase is that the signed change value has been added to the current value at the start of the phase, for each of the three parameters.

In the case of the current amplitude parameters the value is limited to the range 0 to 63. If an attempt is made by an envelope to make the amplitude go above 63 then the actual amplitude will just stick at 63. For pitch values there is no checking, the value will simply wrap around from 65535 to 0 and vice versa.

At each tick the current values of pitch and amplitude are output to the DAVE chip registers. The details of how the register values are calculated are different for pitch and amplitude.

For amplitude, the current left amplitude (6 bits) is multiplied by the overall left amplitude specified in the sound definition (8 bits). The resulting 14 bit number is the required left amplitude. The top six bits of this value are written to the appropriate DAVE register. The right amplitude is of course treated similarly.

The 16-bit pitch value goes through a logarithmic conversion process to produce in a counter value to go into the appropriate DAVE registers. The top byte of the pitch defines a quarter-tone number from 0 to 255 (range 9-10 octaves). The counter value for this quarter-tone is found from a lookup table, with appropriate shifting depending on the octave. The top four bits of the lower byte of the pitch value are used to linearly interpolate between the selected counter value and the next one up. This gives a resolution of approximately 1/64 tone which is certainly adequate to produce smooth pitch changes.

5.3 Sound Duration control

While all the above processing of envelopes is going on, another counter is timing the length of the sound itself. This is a 16 bit counter initialised at the start of the sound to the sound duration specified in the sound definition, and decremented at each tick. If the end of the envelope is reached before this counter reaches zero then the sound will be silenced and remain silent until the counter does reach zero.

If the sound duration counter reaches zero before the envelope has finished then two things happen. Firstly if the release phase of the envelope has not yet been reached then control skips to the start of the release phase. Secondly a flag is set to allow this sound to be over-ridden by another sound which is waiting in the queue or appears in the queue at a later time.

5.4 Synchronisation

This section describes the use of the SYNC count which is one of the fields in the flag byte in the sound definition. It is used to control when sounds which reach the head of a sound queue will start and is useful for ensuring that certain sounds (such as chords) start simultaneously with each other.

The SYNC count is a two bit count which is ignored when the sound is put in the queue. The sound driver maintains an internal SYNC COUNT which is normally zero. When a sound reaches the head of a queue and is thus ready to be played, the SYNC byte in the sound is examined in conjunction with the internal SYNC counter.

If the SYNC count in the sound is zero then the sound is just started in the normal way with no synchronisation. If the SYNC count in the sound is non-zero then the sound is held up and the internal SYNC counter is examined. If it is zero then the SYNC counter from the sound is copied into it. If the internal SYNC counter is non-zero then it is decremented by one and if it goes to zero then all held up sounds are released simultaneously.

The effect of all this is that if three sound are to be played simultaneously then they should be queued up on their appropriate channels each with a SYNC count of two (one less than the number of sounds). The sound driver will then ensure that they are started simultaneously even if one of them gets to the head of its queue slightly earlier. This facility can thus be used to iron out slight timing differences in multi-voice tunes.

6. Other Control Functions

There are various other functions which the sound device provides which are listed here.

^Z	-	Flush all sound queues.
Esc Z <n>	-	Flush an individual sound queue. n = 0, 1 or 2 for tone channel 3 for noise channel
^X	-	Flush all of envelope storage
^G	-	Triggers a PING on tone channel 2. Does nothing if tone channel 2 is already in use.

6.1 Internal Speaker Control

The sound produced by the sound driver, and anything else using the DAVE chip, can normally be heard through the built in speaker. This speaker can be disabled by setting the EXOS variable MUTE_SND to a non-zero value. This only affects the internal speaker, it does not prevent the sound from coming through headphones or anything connected to the monitor socket.

7. Interaction With Other Devices

7.1 Other Devices Accessing Sound Registers

Ideally the sound device would have exclusive use of all the sound registers. However the cassette device uses some of the sound registers for output and timing purposes. The sound driver is designed to set up all sixteen sound registers on each interrupt so it will recover very quickly if its registers are corrupted.

7.2 Keyboard Click

The keyboard device has to make an audible click when a key is pressed. This event is triggered from the keyboard device's interrupt routine and so cannot be done with an EXOS call. Instead the keyboard device calls a globally defined routine KEYCLICK in the sound driver which uses tone channel zero to produce a click without interfering with any other sound which may be on this channel. The key click sound takes about 1/100 second and the routine does not return until it has finished.

8. Quick Reference Summary

8.1 EXOS calls.

OPEN/CREATE CHANNEL - Treated identically. Only one channel. Device name "SOUND:". Filename and unit number ignored. EXOS variable BUF_SND must be set before open.

CLOSE/DESTROY CHANNEL - Treated identically.

READ CHARACTER/BLOCK - Not supported.

WRITE CHARACTER/BLOCK - Printing characters ignored. Control codes and escape sequences interpreted to provide envelopes and sound.

READ STATUS - Not supported

SET STATUS - Not supported.

SPECIAL FUNCTION - No special functions

8.2 EXOS Variables

BUF_SND - Envelope buffer size. Must be set before opening channel.

MUTE_SND - Non-zero to silence internal speaker.

WAIT_SND - Non-zero to return .SQFUL error if sound queue is full. Zero to wait until not full.

++++++++++ END OF DOCUMENT ++++++++++