FREQ_SWEEP

FREQ_SWEEP is used by SND_MANAGER and special effects routines to create frequency sweeps. It operates upon frequency data stored within a song data area, and is normally called (by SND_MANAGER or a special effect routine) once every VDP interrupt (16.7ms). The start of the data area (address of byte 0) is passed in IX.

FREQ_SWEEP assumes data has been stored as follows (names which may be used to describe the various bytes or byte segments within the data area are indicated; see Figure 1):

byte 3: the least significant 8 bits of that note's frequency (F2 - F9)

byte 4: top 2 bits of that note's frequency: B1 = F0, B2 = F1

byte 5: NLEN - determines the note's duration:
1) if frequency is to be swept, NLEN = number of steps in the sweep:
2 to 255 (0 => 256)
2) if fixed frequency, NLEN * 16.7 ms = duration of the note:
1 to 255 (0 => 256)

byte 6: FPS ! FPSV - frequency sweep duration prescaler:
FPS = prescaler reload value: 0 to 15 (0 => 16)
FPSV = temp storage nibble for FPS: init ROM value, 0 to 15 (0 => 16)
duration of sweep (& note) = [((NLEN-1) * FPS) + initial FPSV] * 16.7ms
duration 1st step = initial FPSV * 16.7ms
duration all other steps = FPS * 16.7ms

byte 7: FSTEP - frequency sweep step size: signed 8 bit number, two's complement: 1 to 127, -1 to -128
if FSTEP = 00, frequency is not to be swept, but NLEN is decremented each time called

Parameter limitations:
1) In a frequency sweep, a "step" consists of a single fixed frequency tone; therefore, the minimum number of steps a frequency sweep can have is two (otherwise the frequency wouldn't have "swept").
2) If a note is to be frequency swept, FSTEP must not be 0.
3) The minimum length fixed frequency note has NLEN = 1.
4) Maximum NLEN 0, which is equivalent to 256.

FREQ_SWEEP returns with the Z flag SET if the note (swept or fixed) is over, RESET if the note is not over. (PROCESS_DATA_AREA decides that a note is over when FREQ_SWEEP returns with the Z flag set)

INPUT: 16 bit address of a song data area in CPU RAM
PASSED: in IX
DESCRIPTION: FREQ_SWEEP operates upon frequency data within this song data area

OUTPUT: 1) duration and sweep counters are decremented
2) freq data in bytes 3&4 is modified if note is freq swept
3) returns with Z flag SET if note over, RESET if note not over

ATN_SWEEP

ATN_SWEEP is used to create attenuation sweeps. It operates upon attenuation data stored within a song data area, and is normally called (by PROCESS_DATA_AREA or a special effect routine) once every VDP interrupt (16.7ms). The start of the data area (address of byte 0) is passed in IX.

ATN_SWEEP assumes data has been stored as follows (see Figure 1):

byte 4: ATN - the MSN = 4 bit attenuation

byte 8: ALEN ! ASTEP - no-sweep code or sweep length and step size:
1) if byte 8 = 00, ATN is not to be swept and counters aren't changed
2) if byte 8 non zero, attenuation is to be swept:
a) ALEN = number of steps in the sweep: 1 to 15 (0 => 16)
b) ASTEP = sweep step size: 1 to 7, -1 to -8 (signed, 4 bit two's complement)

byte 9: APS ! APSV - attenuation sweep duration prescaler:
1) if attenuation is not swept, byte 9 is not used by ATN_SWEEP
2) if attenuation is to be swept:
APS = prescaler reload value: 1 to 15 (0 => 16)
APSV = temp storage nibble for APS: init ROM value, 1 to 15 (0 => 16)
duration of swept attenuation part of note =
[((ALEN - 1) * APS) + initial APSV] * 16.7 ms
duration 1st step = initial APSV * 16.7ms
duration all other steps = APS * 16.7ms

Parameter limitations:
1) In an attenuation sweep, a "step" consists of a tone (swept or not) played at a fixed attenuation level; so, the minimum number of steps an attenuation sweep can have is two (otherwise the attenuation wouldn't have "swept"). Therefore, the minimum ALEN value is 2 (0 => 16)
2) If a note is to be attenuation swept, byte 8 must not be 00.
3) The absolute value of ASTEP must be) = 1.

If byte 8 is 00, ATN_SWEEP returns immediately with Z flag SET (the sweep is over or the note was never swept), and doesn't modify any counters. When a sweep finishes, ATN_SWEEP sets byte 8 to 00 and returns with the Z flag SET. If a sweep is in progress, ATN_SWEEP returns with the Z flag RESET. (NOTE: PROCESS_DATA_AREA decides that a note is over when FREQ_SWEEP returns with Z set: the length of a note has nothing to do with when its attn sweep is over)

INPUT: 16 bit address of a song data area in CPU RAM
PASSED: in IX
DESCRIPTION: ATN_SWEEP operates upon frequency data within this song data area

OUTPUT: 1) duration and sweep counters are decremented if sweep in progress
2) atn data in byte 4 is modified if note is atn swept
3) RETs C SET, byte 8 = 0 if sweep is over or note was never swept
RETs C RESET if sweep in progress

PROCESS_DATA_AREA

PROCESS_DATA_AREA is called by SND_MANAGER. For an active data area (address of byte 0 passed in IX), PROCESS_DATA_AREA modifies the timers, sweep counters, frequency, and attenuation data by calling FREQ_SWEEP and ATN_SWEEP. If a note finishes during the current pass through PROCESS_DATA_AREA, the next note in the song is examined and its data is loaded into the data area (calls LOAD_NEXT_NOTE). Then, in order to maintain the song data area priority structure, the CH# : SONGNO of the newly loaded note is compared to the CH# : SONGNO of the previous note: if there is a difference, UP_CH_DATA_PTRS is called to adjust the channel data area pointers in response to the change caused by loading the next note.

If the data area is being used by a special sound effect, PROCESS_DATA_AREA calls the sound effect routine whose address is stored in bytes 1&2 of the data area (the actual address called is routine + 7: see discussion of special sound effects).

If the data area is inactive, PROCESS_DATA_AREA returns immediately (no processing occurs).

INPUT: address of byte 0 of a song data area
PASSED: in IX

CALLS: ATN_SWEEP, FREQ_SWEEP, LOAD_NEXT_NOTE, UP_CH_DATA_PTRS, AREA_SONG_IS

OUTPUT: 1) if active, modifies song data area's timer, freq, and atn data
2) loads the next note's data when a note is finished
3) if special sound effect routine using data area, calls it
4) when necessary, updates the channel data area pointers

LOAD_NEXT_NOTE

Called by PROCESS_DATA_AREA and JUKE_BOX, LOAD_NEXT_NOTE examines the next note to be played in a data area (address byte 0 passed in IX) and moves its data into the area. It fills in bytes (e.g., to indicate swept or not swept) where appropriate, based upon note type. If the next "note" is a special sound effect, its address is saved in bytes 1&2 and the address of the routine + 0 is called, with the address of the note to follow the effect passed in HL and SONGNO passed in A. This will cause the special effect routine to save both these values. Then, the special effect routine + 7 is called, which allows the routine to initialize the song data area for the first pass through PLAY_SONGS. (see discussion of special sound effects)

Prior to moving the next note data, LOAD_NEXT_NOTE saves the data area's byte 0 (CH# ! SONGONO) and stores the song inactive code (0FFH) there. The last thing LOAD_NEXT_NOTE does is restore byte 0, loading CH# with the CH# ! SONGNO of the new note (usually the same as the old note). If the new note is a special sound effect, 62 is returned as the SONGNO part of byte 0.

INPUT: address of byte 0 of a song data area
PASSED: in IX

OUTPUT:
1) sets up song data area with data from next note to be played
2) for next note = special sound effect, calls the effect twice, first with the address of the following note in the song and the song's SONGNO, and then once more to allow the effect to initialize the song data area
3) if next note is "normal", loads CH# ! SONGNO in byte 0 with CH# ! SONGNO of new note
4) returns with byte 0 = 0FFH if song over, SONGNO = 62 if next note is a sound effect

UTILITIES

The following are O/S utility routines, used by the main O/S sound programs, that may be of use to the cartridge programmer:

*** AREA_SONG_IS ***

The address of byte 0 of a song data area is passed in IX. The song number of the song using that area is returned in A (0FFH if inactive). If a special effect was using that area, 62 is returned in A and HL is returned with the address of the special sound effect routine.

*** UPATNCTRL ***

Perform single byte update of the snd chip noise control register or any attenuation register. IX is passed pointing to byte 0 of a song data area. MSN register C = formatted channel attenuation code.

*** UPFREQ ***

Perform double byte update of a sound chip frequency register. IX is passed pointing to byte0 of a song data area, MSN register D = formatted channel frequency code.

*** DECLSN ***

Without affecting the MSN, decrement the LSN of the byte pointed to by HL. HL remains the same.
RET with Z flag set if dec LSN results in 0, reset otherwise.
RET with C flag set if dec LSN results in -1, reset otherwise.

*** DECMSN ***

Without affecting the LSN, decrement the MSN of the byte pointed to by HL. HL remains the same.
RET with Z flag set if dec MSN results in 0, reset otherwise.
RET with C flag set if dec MSN results in -1, reset otherwise.

*** MSNTOLSN ***

Copy MSN of the byte pointed to by HL to the LSN of that byte. HL remains the same.

*** ADD816 ***

Adds 8 bit two's complement signed value passed in A to the 16 bit location pointed to by HL. Result is stored in the 16 bit location.

*** PT_IX_TO_SxDATA ***

A SONGNO is passed in B. PT_IX_TO_SxDATA returns with IX pointing to the song data area which is used by that SONGNO.

UP_CH_DATA_PTRS

UP_CH_DATA_PTRS adjusts each channel data pointer to point to the highest priority (ordinal last) song data area that uses that channel. It is called whenever a change has been made to a song data area that requires modification of the channel data pointers.

All 4 channel data pointers (PTR_TO_S_ON_x) are initially pointed to a dummy inactive area, DUM_AREA. Then, moving in order from the first data area to the last, CHS in byte 0 of each data area is examined, and the corresponding channel data pointer is pointed to that data area. Thus, by the time the routine is done, each channel data pointer is pointing to the last active data area that contains data to be sent to that channel. If none of the active data areas used a particular channel, then that channel remains pointing to DUM_AREA (and therefore its generator will be turned off next time through PLAY_SONGS).

LEAVE_EFFECT

LEAVE_EFFECT, called by a special sound effect routine when it's finished, restores the SONGNO of the song to which the effect belongs to B5 - B0 of byte 0 in the effect's data area, and loads bytes 1 & 2 with the address of the next note in the song. The address of the 1 byte SONGNO (saved by the effect when it was first called) is passed in DE. The 2 byte address of the next note in the song, also saved by the effect, is passed in HL. IX is assumed to be pointing to byte 0 of the data area to which the song number is to be restored. Bits 7 & 6 of the saved SONGNO byte are not stored into byte 0, and therefore may be used during the course of the effect to store any useful flag information.

SPECIAL SOUND EFFECTS

* Sound effects as notes within a song

Sounds which do not fit one of the six categories of "normal" musical notes can be created and played throughout the course of a song as "special effect" notes. Unlike normal musical notes, which are stored in ROM as tables of frequency/control and attenuation data, a special effect's data are determined algolrithmically by a custom routine written by the cartridge programmer. Special effect notes can also be used to generate sounds that could have been comprised of many normal notes, but which are more efficiently (in terms of ROM space used) computed by a short program.

These notes use the same song data area as the song within which they are contained, and they are stored in the song's ROM note list with a one byte header as are normal notes. However, the bytes following the ROM header do not contain data to be directly loaded into the song data area. The header (see Figure 2), which specifies the channel upon which to play the effect (which is usually the same as the channel used by the rest of the notes in the song), is followed by a two byte address of a routine written by the cartridge programmer which will be called every 16.7ms by PROCESS_DATA_AREA. When called, this special effect routine should compute data values and store them at the appropriate locations within the song data area. (In fact, many effect routines may call the O/S routines FREQ_SWEEP or ATN_SWEEP, which also require that data be ordered appropriately within a song data area) This computed data will then be output on the next pass through PLAY_SONGS (assuming that this song data area has the highest priority of any data area using the same channel).

Variables required by the effect which will not be output may be stored wherever the programmer desires. Free locations within the song's data area might as well be used for effect variable storage, since the entire ten byte area is reserved for the song anyway. If no free locations exist within a data area, which would be the case if an effect required both frequency and attenuation to be swept, the effect can store the remaining needed variables wherever convenient.

In order to interact properly with the O/S sound routines, each special effect routine must conform to a certain format. A description of that format, and how an effect interacts with the O/S routines, follows:

WHEN AN EFFECT BEGINS - When loading a new note, if LOAD_NEXT_NOTE sees that the note to be loaded is a special effect:

1) It stores in byte 0 of the song's data area the effect's CH# and a SONGNO of 62. SONGNO = 62 is used later by PROCESS_DATA_AREA to detect the fact that an effect is using the data area.

2) It then takes the address of the special effect routine (let's call it SFX) from ROM and puts it into bytes 1&2 (NEXT_NOTE_PTR).

3) LOAD_NEXT_NOTE then calculates the ROM address of the header of the next note in the song, stores that address in HL, puts the song's SONGNO in A, and calls SFX + 0. In every special effect routine at SFX + 0, there MUST be the following code which saves the two passed values (see Figure 9):

```
SFX:      LD (SAVE_x_NNP),HL
          LD (SAVE_x_SONGNO),A
          RET
SFX+7:    code for sound effect starts here
          ...
```

where SAVE_x_NNP is a two byte location used by all the sound effect notes in the current song to save the address of the next note in the song, and SAVE_x_SONGNO is the address of a byte where the song number is saved. The programmer may put SAVE_x_NNP and SAVE_x_SONGNO wherever desired, including somewhere within the song data area.

Thus, calling SFX + 0 allows each effect routine to save the next note's address and the song's SONGNO.

1ST PASS THROUGH EFFECT - After calling SFX + 0, LOAD_NEXT_NOTE calls SFX + 7 for the first pass through the body of the routine. At this location, there should be code which initializes the appropriate bytes within the song data area, as the next pass through PLAY_SONGS, subject to the data area priority system, will output this initial data in normal fashion.

As will be seen below, this same location (SFX + 7) will be called every 16.7ms by PROCESS_DATA_AREA to modify the data within the area. Therefore, the code at SFX + 7 must know which pass is in effect, so that the song data area will be initialized only on the first pass. A convenient way of doing this is to test bit 7 of SAVE_x_SONGNO, the byte which contains the saved song number. On the 1st pass through the effect, bit 7 (and bit 6) will be zero, since the largest possible SONGNO (62) would not set this bit. If bit 7 is zero, then, code to initialize the data area can be executed and bit 7 reset to prevent re-initialization. I.e.,

```
SFX+7:      LD HL,SAVE_x_SONGNO
            BIT 7,(HL)
            JR NZ,NOT_PASS_1
              SET 7,(HL)          ;to prevent further passes thru inits
              ...                 ;initialize bytes within the data area here
              RET                 ;to LOAD_NEXT_NOTE
NOT_PASS_1: ....                  ;code for pass 2 or greater starts here
```

PRIORITY UPDATE - After calling SFX + 0 and SFX + 7, LOAD_NEXT_NOTE will return to PROCESS_DATA_AREA, which checks to see if loading a new note has caused a change in either the channel used by the song (this happens with noise notes within a musical song) or the song number. If a change has occured, UP_CH_DATA_PTRS will be called, which updates the data pointers on the basis of priority within the block of song data areas (see description of this routine in the preceeding "UTILITIES" section). Since a special effect note will cause a change in the song number (from whatever it was to 62), UP_CH_DATA_PTRS will always be called whenever an effect note is loaded.

SECOND PASS OR GREATER - The next time PROCESS_DATA_AREA is called (from SND_MANAGER), which will be 16.7ms after PLAY_SONGS has sent out the effect's initial data, it will detect the fact that an effect is using the song data area (by seeing a SONGNO of 62) and will JUMP to SFX + 7, rather than calling the frequency and attenuation sweep routines as it would for a normal note. This will result in the first pass through the part of the body of the effect routine that actually does computation and adjusts the data values within the data area. When the effect routine has completed its modifications to the data area and performs a RET, control is transferred back to SND_MANAGER, which then moves on to the next song data area to be processed.

This process will be repeated every 16.7ms until the effect routine itself decides that it's over and takes action to load the next note in the song.

WHEN AN EFFECT IS OVER - Prior to performing a RET, the effect routine must decide whether the effect note has finished. If it has, NEXT_NOTE_PTR within the data area must be set to the address of the next note in the song and SONGNO must be restored to byte 0. This can be done by calling the O/S routine LEAVE_EFFECT which does this. The address of SAVE_x_NNP must be passed in HL and the address of SAVE_x_SONGNO must be passed in DE. Finally, the effect should JUMP to EFXOVER, a location within PROCESS_DATA_AREA which would normally be reached once a note is over. The code there takes care of loading the next note in the song. Thus, the final code of each effect routine will look as follows:

```
RET if effect not over
  LD HL,(SAVE_x_NNP)          ;HL := addr next note in song
  LD DE,SAVE_x_SONGNO         ;DE := addr saved song number
  CALL LEAVE_EFFECT           ;to restore them to bytes 0 - 2 in data area
  JP EFXOVER                  ;in PROCESS_DATA_AREA to load song's next note
```

The entire above described sequence is summarized in Figure 9.

* A sound effect as a single sound

A stand alone sound effect can be implemented within the previously mentioned structures simply by creating a single note song. The single note is the effect

and would be followed by an end of song code (or repeat code if you wish the effect to go on forever).

Many stand alone effects may want to use more than one tone generator channel: e.g., a special laser zap that momentarily requires all three tone generators, or, as is often the case, a white noise effect of particular character that requires the noise channel shift rate to be modified by channel three (see TI data sheets). In these cases, the effect's routine will have to modify data in several data areas whenever called. The song data areas used by such effects are subject to the normal priority structure. E.g., if you wish a two channel effect to temporarily overwrite the harmony and bass lines of a repeating song, the effect must have been assigned two data areas of higher priority (ordinally later in the block of song data areas). If it is not necessary to maintain any underlying songs, an effect can share data areas to conserve RAM space, with the understanding that, as usual, songs or sounds that share the same song data area truncate each other. A multi-channel effect (a chord note, say) may be included as a note within a song, but, again, the song data area priority structure determines what will finally be heard.

Providing for a typical game's sound generation needs might require eight song data areas: four for an underlying, repeating song(s) (three areas for the three tone generators and one for the noise generator used for percussion notes), and four for higher priority, occasional sound effects (which would temporarily overwrite the repeating songs, but truncate each other).

* Pseudo code listings of main routines

The following two pages contain pseudo code descriptions of most of the O/S sound routines. Some computational details are not shown, but all jumps, calls, returns, pushes and pops are listed.

Terminology:
":=" is used as the assignment statement, and "(xx)" means the contents of the memory location pointed to by xx, where "xx" is HL, IX, etc.

The structure of each description is as follows:

```
*** name of routine ***
the value expected for passed parameters (if any)

pseudo code description
of the routine, uninterrupted
by blank lines
RET
```

```
*** INIT_SOUND ***
HL = addr of LST_OF_SND_ADDRS
B = number of song data areas used by game

set RAM word PTR_TO_LST_OF_SND_ADDRS to value passed in HL
pt HL to byte 0 in 1st song data area
B1: (HL) := inactive code (0FFH)
HL := HL + 10
DJNZ B1 (set B areas inactive)
(HL) := end of data area code (0)
load all 4 channel data area pointers with the
addr of a dummy inactive area (DUMAREA)
SAVE_CTRL := 0FFH
ALL_OFF: turn off all 4 generators directly
RET

DUMAREA DEFB 0FFH


  *** UP_CH_DATA_PTRS ***

PUSH IX to save it
set all 4 ch data ptrs to a dummy, inactive area
CALL PT_IX_TO_SrDATA, song 0 1
LOOP
  IF byte 0 indicates the end of the song data areas JR DONE
    IF byte 0 indicates an active area
      set HL to address of this area's channel data pointer
      (i e., HL := addr PTR_TO_S_CH_0 + (CH# this area * 2))
      PUSH IX
      POP BC (BC := addr byte 0 this area)
      (HL) := C, (HL+1) := B
    ENDIF
    IX := IX + 10
  ENDIF
REPEAT LOOP
DONE: POP IX to restore it
RET
```

```
*** FREQ_SWEEP ***
IX = addr byte 0 of a song data area

IF FSTEP = 0 note is not to be swept
  A := NLEN
  DEC A
  RET Z (leave if note over, Z flag SET)
    NLEN := A (store decremented NLEN)
    RET (note not over, Z flag RESET)
ENDIF
PUSH IX, POP HL (pt HL to byte 0)
HL := HL + offset within data area of FPSV
CALL DECLSN to decrement FPSV
IF Z flag SET, FPSV has timed out
  CALL RSNTOLSN to reload FPSV
  A := NLEN
  DEC A
  RET Z (leave if sweep over with Z flag SET)
    NLEN := A (store decremented NLEN)
    point HL to FREQ
    A := FSTEP
    CALL ADD816 to add FSTEP to FREQ
    RESET bit Z in hi byte FREQ
    (in case of overflow from addition)
    OR 0FFH to RESET Z flag
ENDIF
RET

*** ATN_SWEEP ***
IX = addr byte 0 of a song data area
                    .
RET with Z flag SET if byte 0 = 0
(i e., note atn not to be swept)
  PUSH IX, POP HL (pt HL to byte 0)
  HL := HL + offset within data area of APSV
  CALL DECLSN to decrement APSV
  IF Z flag SET, APSV has timed out
    CALL RSNTOLSN to reload APSV
    pt HL to ALEN (# of steps in atn sweep)
    CALL DECLSN to decrement ALEN
    IF Z flag RESET, sweep not over yet
      ATN := ATN + ASTEP
      (4 bit add, overflow ignored)
      OR 0FFH to RESET Z flag
    ELSE Z flag is SET (sweep is over)
      byte 0 := 0 to indicate sweep over
    ENDIF
  ENDIF
RET
```

```
*** TONE_OUT ***

IX = (PTR_TO_S_CH_x), i.e., IX pts to byte 0 data area of song for CHx
A set for CHx OFF code
MSN C set for CHx attenuation
MSN D set for CHx frequency

IF area INACTIVE
  turn off CHx
ELSE
  CALL UPATNCTRL (send out attenuation)
  CALL UPFREQ (send out frequency)
ENDIF
RET


*** PT_IX_TO_SrDATA ***
B = a song number

HL := addr of LST_OF_SND_ADDRS
HL := HL - 2
BC := 4 * SONGNO
HL := HL + BC (i e., HL now points to SrDATA's entry in LST_OF_SND_ADDRS)
E := (HL), D := (HL+1)
PUSH DE
POP IX (IX := addr byte 0 of this song's data area)
RET
```