## NOTES

### * Terminology

Each note in a song has an associated 10 bit frequency (except for noise notes) and 4 bit attenuation which is output to the sound chip every time PLAY_SONGS is called. The initial frequency and attenuation values (stored in 2 bytes) are part of a block of 4 to 8 bytes that describe a single note within a song's ROM note list. The remaining bytes are used to indicate sound channel, note type, duration, and various timers and values associated with swept notes.

The following are explanations of names and symbols used throughout this manual to refer to bytes, or segments of bytes, within both a note's ROM note list and a RAM song data area (see Figure 1):

|: | is a symbol used to graphically separate bits or nibbles within a byte

Bx: means bit x of a byte, bit 7 being the most significant bit

byte x: refers to the offset of a byte within a data block, byte 0 being the first byte in the block

MSN, LSN: MSN means the most significant nibble of a byte, LSN is the least significant nibble

CH#: the sound channel upon which a note is to be played; 0 = noise generator, 1 to 3 = the 3 tone generators

SONGNO: the song number of the song playing in a song data area; 1 to 61; SONGNO 62 means a special sound effect is using the data area

NEXT_NOTE_PTR: 2 byte address of the ROM location of the data block for the next note to be played in a song

F0 - F9: the 10 bit frequency data to be sent to a sound chip tone generator; F0 is the most significant bit; see data sheets, TI 76489

ATN: 4 bit attenuation data to be sent to any of the four sound generators

CTRL: 3 bit control data for sound chip noise genrator; FB NF0 NF1 (called SHIFT in this manual), see data sheets for details

NLEN: 1 byte that directly or indirectly determines the duration of a note

FPS: 4 bit frequency prescaler, used with NLEN to determine the length of a frequency sweep

FPSV: 4 bit temporary storage location for FPS; this variable gets decremented every VDP interrupt, and is reloaded from FPS

FSTEP: the size of a step in a frequency sweep, an 8 bit two's complement signed value that is added to the current 10 bit frequency at a rate determined by NLEN and FPS, 1 to 128, -1 to -128

ALEN: 4 bit number of steps in an attenuation sweep

ASTEPS: ASTEPS is the signed, 4 bit size of a step in an attenuation sweep; can take values 1 to 7, -1 to -8 (MSB = 1 means negative)

APS: 4 bit attenuation prescaler, used with ALEN to determine the length of a frequency sweep

APSV: 4 bit temporary storage location for APS; this variable gets decremented every VDP interrupt, and is reloaded from APS

* Frequency sweeps and note duration

The time duration of a note = the number of passes by PLAY_SONGS through the note times 16.7ms (the VDP interrupt period). (note that the time the note is HEARD could be shorter: see "Attenuation sweeps" discussion) The number of PLAY_SONGS passes is always determined directly or indirectly by NLEN. NLEN, however, has two meanings, depending upon whether or not a note's frequency is swept:

Fixed frequency notes - In this case, NLEN is decremented every VDP interrupt and therefore directly determines the length of a note:

duration = NLEN * 16.7ms

NLEN should have values in the range 0 to 255 (0 => 256), giving a maximum duration of ~ 4.25 seconds.

Swept frequency notes - Here, the prescaler variable, FPSV, is decremented until zero before NLEN is decremented. Once FPSV goes to zero, it's reloaded from FPS; however, an initial value for FPSV, which enters into the calculation of the the length of the first step in the sweep, is stored in ROM along with the rest of the note's initial data, and it may be different from the reload value, FPS. For a frequency swept note:

note duration = [(NLEN -1) * FPS) + initial FPSV] * 16.7ms

So, NLEN again determines note duration, but in an indirect fashion (in concert with FPS and the initial FPSV).

In the case of a frequency swept note, NLEN can be thought of as one of four parameters that describe the sweep: 1) the starting frequency, 2) FSTEP, a signed step size, i.e., a delta frequency that is periodically added to the current frequency, 3) the prescaler value (FPS) which determines the length of time at any one frequency step, and 4) NLEN, the number of steps in the sweep.

The duration of each step in the sweep is given by the following:

| | | |
|---|---|---|
| duration 1st step | = | initial FPSV * 16.7ms |
| duration all others | = | FPS * 16.7ms |

Frequency sweep parameter ranges:

FSTEP - signed 8 bit two's complement number: 1 to 127, -1 to -128; an FSTEP of 0 tells FREQ_SWEEP that the note is not frequency swept, and the note's duration is determined by directly decrementing NLEN (prescaler is disregarded)

FPS - 4 bit frequency prescaler, used with NLEN to determine the length of a frequency sweep: 0 to 15 (0 => 16)

FPSV - 4 bit temporary storage location for FPS; this variable gets decremented every VDP interrupt, and is reloaded from FPS. 0 to 15 (0 => 16)

NLEN - 8 bit note duration for a fixed frequency note: 0 to 255 (0 => 256)
8 bit number of steps for a swept frequency note: 2 to 255 (0 =>256)

Note durations:

| | |
|---|---|
| Fixed frequency - | NLEN * 16.7ms |
| Swept frequency - | [(NLEN -1) * FPS) + initial FPSV] * 16.7ms |
| duration 1st step = | initial FPSV * 16.7ms |
| duration all others = | FPS * 16.7ms |

= Attenuation sweeps

Volume attacks and decays can be thought of as attenuation sweeps: a sweep from low to higher volume is an attack, a sweep in the other direction is a decay. Attenuation sweeps are created in a similiar fashion to the frequency sweeps described above, the primary difference being that attenuation sweep parameters don't take on 8 bit values. The full volume range for the attenuation registers on the 76489 chip is 0 (ON) to 15 (OFF), so step sizes and number of sweep steps greater than 4 bits aren't generally useful.

Just as in the case of a frequency swept note, attenuation sweeps have four parameters that describe the sweep: 1) the starting attenuation, 2) ASTEP, a signed step size, i.e., a delta attenuation that is periodically added to the current attenuation, 3) the prescaler value (APS) which determines the length of time at any one attenuation step, and 4) ALEN, the number of steps in the sweep.

The prescaler parameters, APS and APSV, are the same size (4 bits) and mean exactly the same thing as their frequency counterparts. ALEN, the number of steps in the sweep, is only 4 bits (compared to an FLEN of 8 bits), but 15 steps of even the smallest step size (+/-1) can sweep a generator from full on to full off. ASTEP, in order to squeeze it into a nibble, has been limited to a 4 bit signed number (3 bits data, 1 bit sign). This gives a range of step values from 1 to 7, -1 to -8. This shouldn't be too limiting, since most attenuation sweeps are implemented with the smallest step size.

NOTE: Recall that, as far as SND_MANAGER is concerned, the length of a note is determined, directly or indirectly, only by NLEN, a timer/counter that is decremented during FREQ_SWEEP; i.e., the duration of a note is independent of what is happening to its attenuation. Therefore, the programmer should take care to see that an attenuation sweep isn't inadvertently created that ramps the volume down to off before SND_MANAGER, through NLEN, has decided that the note is over. However, since ATN_SWEEP simply leaves the attenuation alone once it's finished a sweep, the independence of attenuation sweep length and note length may be put to good use: e.g., a sforzando can be accomplished by making an attenuation sweep (to a still audible volume) end before the rest of the note.

ALEN, like NLEN for a frequency sweep, is the number of steps in an attenuation sweep and can take on values from 0 to 15. However, since a "step" consists of a tone (which may be frequency swept) played at a fixed attenuation level, a sweep of 1 step doesn't make sense. ALEN values, then, should range from 2 to 15. An ALEN value of 0 causes a sweep of sixteen steps (NOTE: ATN_SWEEP "wraps around" at 0 and 15, i.e., subtracting 1 from 0 results in 15, and adding 1 to 15 results in 0).

The duration of an attenuation sweep can be calculated as follows:

```
duration entire sweep = [((ALEN -1) * APS) + initial APSV] * 16.7ms
duration 1st step     =          initial APSV * 16.7ms
duration all others   =          APS * 16.7ms
```

Attenuation sweep parameter ranges:

ALEN: 4 bit number of steps in an attenuation sweep; can take values from 2 to 15 (0 => 16 steps).

ASTEP: ASTEP is the 4 bit signed (two's complement) size of a step in an attenuation sweep; can take values from 1 to 7, -1 to -8.

APS: 4 bit attenuation prescaler, used with ALEN to determine the length of a frequency sweep: 0 to 15 (0 => 16).

APSV: 4 bit temporary storage location for APS; this variable gets decremented every VDP interrupt, and is reloaded from APS: 0 to 15 (0 => 16)

Descriptions of each of the six note types follow:

* Rests

See Figure 4.

```
byte 0:   B5 set indicates a rest, to be played on CH# in B7 - B6
          duration = (B4 - B0) * 16.7ms, can take values 1 to 31
```

* Type 0: Fixed frequency, fixed attenuation

See Figure 4.

```
byte 0:   header, CH# in B7 - B6, note type = 0 in B1 - B0
byte 1:   least significant 6 bits of the 10 bit frequency data (constant)
byte 2:   MSN = 4 bit ATN data (constant throughout the note)
          LSN = 0 0 F0 F1, the top 2 bits of the frequency data (constant)
byte 3:   NLEN, duration of the note = NLEN * 16.7ms
```

* Type 1: Swept frequency, fixed attenuation

See Figure 5.

```
byte 0:   header, CH# in B7 - B6, note type = 1 in B1 - B0
byte 1:   least significant 6 bits of the initial 10 bit frequency data
byte 2:   MSN = 4 bit ATN data (constant throughout the note)
          LSN = 0 0 F0 F1, the top 2 bits of the initial frequency data
byte 3:   NLEN, number of steps in the frequency sweep, 1 to 255 (0 => 256)
byte 4:   FPS : FPSV, prescaler reload value and initial FPSV
byte 5:   FSTEP, sweep step size, 1 to 127, -1 to -128
```

* Type 2: Fixed frequency, swept attenuation

See Figure 6.

```
byte 0:   header, CH# in B7 - B6, note type = 2 in B1 - B0
byte 1:   least significant 8 bits of the 10 bit frequency data (constant)
```

byte 2: MSN = 4 bit ATN data (initial value)
LSN = 0 0 F0 F1, the top 2 bits of the frequency data (constant)
byte 3: NLEN, duration of the note = NLEN * 16.7ms
byte 4: ALEN ! ASTEP
ALEN = number of steps in the attenuation sweep
ASTEP = step size, 1 to 7, -1 to -8
byte 5: APS ! APSV, prescaler reload value and initial APSV, 1 to 15 (0 => 16)

* Type 3: Swept frequency, swept attenuation

See Figure 7.

byte 0: header, CH# in B7 - B6, note type = 3 in B1 - B0
byte 1: least significant 8 bits of the initial 10 bit frequency data
byte 2: MSN = 4 bit ATN data (initial value)
LSN = 0 0 F0 F1, the top 2 bits of the initial frequency data
byte 3: NLEN, number of steps in the frequency sweep, 2 to 255 (0 => 256)
byte 4: FPS ! FPSV, prescaler reload value and initial FPSV, 0 - 15 (0 => 16)
byte 5: FSTEP, sweep step size, 1 to 127, -1 to -128
byte 6: ALEN ! ASTEP
ALEN = number of steps in the attenuation sweep
ASTEP = step size, 1 to 7, -1 to -8
byte 7: APS ! APSV, prescaler reload value and initial APSV, 1 to 15 (0 => 16)

* Noise notes: special case Type 2 notes

See Figure 8.

Noise notes are notes that are played on the sound chip noise generator (CH#0). They are stored in ROM as a special case of a Type 2 note, fixed frequency and swept attenuation. They consist of white noise with superimposed attenuation decay, which creates a percussive effect, such as a snare drum note.

Instead of frequency information, noise notes are stored with three bits of noise control data: FB NF0 NF1 (see TI data sheets 76489), which remain constant throughout the note. Experimentation with various values can result in credible percussion effects.

NLEN, as is the case for a regular Type 2 note, directly determines the duration of a noise note.

The sound chip noise generator is unlike the other generators in that sending it redundant data (i.e., the same data that it has stored in its internal registers) has an audible effect on its output. In particular, whenever control data, redundant or not, is sent to the noise generator, its internal shift register is reset, causing a short pop or click to be heard. This isn't annoying on an occasional basis, or when a new noise starts, but remember: PLAY_SONGS is sending data to all four channels every 16.7ms. This would cause a noticeable lack of "whiteness" in the noise generator's output.

PLAY_SONGS avoids this problem by referencing a dedicated CART RAM location, SAVE_CTRL (see Figure 9) each time before it sends control data to the noise generator. SAVE_CTRL contains the control data that was output to the noise generator the last time through PLAY_SONGS. If the noise control data in the pointed to song data area = SAVE_CTRL, PLAY_SONGS doesn't send it out again. If there is new data to be sent, that data is output and SAVE_CTRL is updated.

byte 0: header, CH#0 in B7 - B6, note type = 2 in B1 - B0
byte 1: MSN = 4 bit ATN data (initial value)
LSN = 0 FB NF0 NF1, noise control data
byte 2: NLEN, duration of note: NLEN * 16.7ms
byte 3: ALEN : ASTEP
ALEN = number of steps in the attenuation sweep
ASTEP = step size, 1 to 7, -1 to -8
byte 4: APS : APSV, prescaler reload value and initial APSV, 1 to 15 (0 => 16)

## OPERATING SYSTEM ROUTINES

### INIT_SOUND

Contains ENTRY POINT: ALL_OFF

INIT_SOUND, usually called right after power on, turns off the sound generators, initializes the CART RAM locations to be used as song data areas, and sets up the four channel data area pointers. Specifically, it:

1) directly turns off all four sound generators.
2) initializes PTR_TO_LST_OF_SND_ADDRS, a dedicated 16 bit CPU RAM pointer which other sound routines expect to contain the base address of a list in CART ROM (called LST_OF_SND_ADDRS) of the starting addresses of each song's data area and note list. The address of LST_OF_SND_ADDRS is passed to INIT_SOUND in HL.
3) stores the sound-inactive code (0FFH) into byte 0 of n song data areas. n is passed in B and = the total number of song data areas used by the game.
4) stores an end of data area code (00) following the last data area.
5) sets the four pointers to the data areas for the songs to be played on each channel, PTR_TO_S_ON_x (x = 0-3), to a dummy inactive area (DUM_AREA, which is actually a single 0FFH byte within INIT_SOU).
6) sets SAVE_CTRL to an initial value of 0FFH

INPUT: n
TYPE: 8 bit constant
PASSED: in B
DESCRIPTION: number of song data area used by the game

INPUT: LST_OF_SND_ADDRS
TYPE: 16 bit address
PASSED: in HL
DESCRIPTION: LST_OF_SND_ADDRS is the base address of a list of the starting addresses of each song's data area and note list.

OUTPUT: 1) turns off all sound generators
2) initializes PTR_TO_LST_OF_SND_ADDRS
3) writes inactive code to byte 0 of n song data areas
4) stores 00 at end of song data areas
5) sets the 4 channel song pointers to the inactive DUM_AREA
6) sets SAVE_CTRL to 0FFH

### ALL_OFF

ALL_OFF directly turns off all four sound generators, but does nothing to any song data areas or the 4 channel data pointers.

INPUT: none

OUTPUT: turns off all sound generators

## JUKE_BOX

JUKE_BOX is called to start a song. Using a song number passed in B, JUKE_BOX loads the data for the song's first note into the appropriate song data area (the address of the area is found by using the song number as an index into the LST_OF_SND_ADDRS table). It also formats the data area's header and sets up the next note pointer. If the song is a special sound effect, its next note pointer is set to the address of the special effect routine. The next time PLAY_SONGS is called, that song's first note will be processed (thereby truncating whatever song had been "playing" in that data area), and the song will have started.

Since starting a new song may have altered the priority structure within the song data areas, JUKE_BOX also calls UP_CH_DATA_PTRS to modify the channel data pointers accordingly.

If JUKE_BOX is called with a song number of a song already in progress, it returns immediately (i.e., it doesn't restart the song).

INPUT: song number to be started
TYPE: 8 bit constant, 1 to 61
PASSED: in B

CALLS: PT_IX_TO_SxDATA, LOAD_NEXT_NOTE, UP_CH_DATA_PTRS

OUTPUT: 1) moves the song's first note data to the appropriate song data area
1) formats byte 0 header of the song's data area
2) points next note pointer in data area (bytes 1&2) to address of first note in song, or address of special sound effect routine
3) updates the channel data pointers on basis of song priorities

## SND_MANAGER

SND_MANAGER should be called every VDP interrupt (every 16.7 ms). It assumes that the song data areas are stored contiguously in a data block beginning with the data area assigned to song number one. For each data area, SND_MANAGER, or routines which it calls, processes the appropriate timer and sweep counters and modifies the frequency and attenuation data accordingly. If the data area is assigned to a special effect, SND_MANAGER simply calls that effect, and doesn't modify any data. When a note is finished, SND_MANAGER, using the data area's next note pointer, moves data for the next note of the song into the area and fills in keys bytes within the area to allow proper processing of the data area by the sweep routines it calls (FREQ_SWEEP and ATN_SWEEP). (SND_MANAGER considers a note finished when its frequency duration timers have timed out; see the descriptions of the FREQ_SWEEP and ATN_SWEEP routines) A special effect is responsible for deciding when its over and initiating the next note in the song.

After the operations upon a data area have been performed, the channel data area pointers (PTR_TO_S_ON_x) may be updated (see description of UP_CH_DATA_PTRS in "UTILITIES" section).

If SND_MANAGER reads a header byte (in CART ROM) that has bits 3&4 set, indicating repeat song, it will start the song again by reloading the first note in the song, using the SONGNO portion (B5-B0) of byte 0 in the song's data and the LST_OF_SND_ADDRS to find it.

SND_MANAGER does not output the modified frequency and attenuation data. PLAY_SONGS is usually called just before SND_MANAGER to do this.

Special codes in byte 0 of the song data area indicate:

| | | |
|---|---|---|
| 0FFH | - | data area inactive, do no processing, do not modify channel data area pointer |
| B5-B0 = 62 | - | a special effect is to be played; SND_MANAGER calls the effect routine |
| 00H | - | end of song data areas (SND_MANAGER processes data areas until it sees 0 in byte 0) |

NOTE: Song number 1 MUST use the first data area in the block of song data areas.

INPUT: none

CALLS: PROCESS_DATA_AREA, PT_IX_TO_SxDATA

OUTPUT:
1) decrements song duration and sweep timers
2) modifies swept frequency and attenuation values
3) calls special effects routines where necessary
4) restarts the song if indicated
5) may update the channel data area pointers (PTR_TO_S_ON_x)

## PLAYSONGS

PLAY_SONGS takes the frequency and attenuation data pointed to by the four channel data area pointers (PTR_TO_S_ON_x) and outputs it to the four sound chip generators. Action is taken on the basis of the each data area's byte 0:

1) If the pointed to data area is active, the frequency and attenuation data are sent to the channel indicated by B7-B6 (CH#) of byte 0 of the pointed to data area.
2) If byte 0 is 0FFH (inactive), the channel to which that pointer is dedicated is sent the OFF attenuation code.
3) If CH# = 0 (noise), the attenuation data is output. If there is no new noise control data to be output (determined by checking dedicated CART RAM location SAVE_CTRL), no control data is sent out. Otherwise, the new control data is output and SAVE_CTRL is updated.

INPUT: none

OUTPUT: through SOUND_PORT,
1) current freq and atn data is output to each tone generator, if song/effect on that channel is active
2) noise generator is sent current atn data, and control data, if new
3) modifies SAVE_CTRL if necessary