Introduction

Next produces sound using the AY-3-8912 sound chip. The Next has 3 AY chips and hence maintains compatibility with the ZX Spectrum 128K.

Each AY chip has 3 sound channels or “voices” and so the 3 AY chips combined allows for 9 channel sounds.

By default, the AY chip generates square waves. This is the most basic sound, but the amplitude (or volume) can be varied over time using the Envelope settings.

Programming

Introduction

In order to create a sound, the AY chip needs to be selected first via the “Turbo Sound Next Control” register $FFFD

Opionally see also various parameters through Peripheral 3 $08 and Peripheral 4 $09 registers.

Then programming each sound involves multiple port writes. The Next controls the sound chip registers by first placing the register number on port $FFFD and then placing the parameter value on port $BFFD. . There are 14 different AY chip registers that control the tone, noise, volume, envelope etc of the 3 channels as follows.

Register	Description	Note
0/1	Channel A Tone Period	Low byte (fine value) High byte (coarse value) - top nibble 0 See Tone Period below
2/3	Channel B Tone Period	As Channel A
4/5	Channel C Tone Period	As Channel A
6	Noise Period	Bits 0-4 allowing values of 0 - 31 See Noise Period below
7	Mix Flags This register controls which channels/signals are included in the mix	Bits 0-2 Channel selector for ABC Bits 3-5 Noise selector for ABC Note a zero bit means that the channel/signal is included.
8	Channel A volume / envelope The volume is set by writing a 4 bit value to this register. If bit 4 is	Bits 0-3 volume value Bit 4 activate envelope for this channel

Register	Description	Note
	set then the volume is ignored and the envelope is switched on for the channel	
9/10	Channel B/C volume / envelope	As Channel A
11/12	Envelope Period	Low byte (fine value) High byte (coarse value) See Envelope below
13	Envelope Shape Determine how the volume / amplitude varies over time	Bit 0 - Hold Bit 1 - Alternate Bit 2 - Attack Bit 3 - Continue See Envelope below

Tone Period

The tone period is a 12 bit value that controls the sound frequency and is stored in two registers for each channel. The top 4 bits of the high byte (coarse value) are unused.

Sound frequency is calculated as:
F = AYC / (16 * TP). Where AYC is the AY clock speed and TP is the tone period.

The Next has a AY clock frequency of 1.7735 MHz so to obtain a sound frequency of 440 Hz (musical note A above middle C) the tone period is set to 1773500 / (16 * 440) giving a tone period of 252

Noise Period

Each AY chip has one white noise channel that may be coupled to one or more of the 3 tone channels. The period of the noise is a value going from 1 to 31 (0 producing the same result as 1, 1 being a very high-pitched noise, 31 a low-pitched one). Include the Noise channel into the Tone channel via the Mix Flags register.

Envelope

The amplitude of a sound wave determines its loudness or volume. A large amplitude gives a louder sound and a small amplitude a softer sound. The Envelope settings allow you to vary the amplitude / volume over time.

Only one envelope can be applied at any time across all three channels but it can be switched on or off per channel using the Volume / Envelope register for that channel.

Envelope Period

The Envelope period determines the speed (frequency) of the volume Envelope. The formula is EF = AYC / (256 * EP). Where EF = Envelope frequency and EP = Envelope Period.

To obtain an Envelope frequency of 0.5 Hz the Envelope Period should be set to 177350 / (256 * 0.5) = 13855. 1 Hz is 1 cycle per second, therefore the Envelope duration would be 1 / 0.5 = 2 seconds. That is set register 11 (envelope period fine value) to 31 and register 12 (envelope period coarse value) to 54.

Envelope Shape

Instead of a standard square wave the AY chip can vary the volume / amplitude over the Envelope Period with the envelope shape, register 13. Values of 0-16 are allowed.

Bit	Description / Effect
0	“Hold” 1 = envelope generator performs 1 cycle then holds the end value 0 = cycles continuously
1	“Alternate” If “hold” set 1 = the value held is initial value 0 = the value held is the final value If “hold” not set 1 = envelope generator alters direction after each cycle 0 = resets after each cycle
2	“Attack” 1 = the generator counts up 0= the generator counts down
3	“Continue” 1 = “hold” is followed 0 = the envelope generator performs one cycle then drops volume to 0 and stays there, overriding “hold”

In practice, this gives us 16 combinations with the following sound profiles

Value	Profile	Description
0 - 3	__________	decay - silence

<table>
<tr><th>Value</th><th>Profile</th><th>Description</th></tr>
<tr><td>4 - 7</td><td>/|________</td><td>attack - silence</td></tr>
<tr><td>8</td><td>\|\|\|\|\|\</td><td>repeated decay</td></tr>
<tr><td>9</td><td>_________</td><td>Decay - silence</td></tr>
<tr><td>10</td><td>\/\/\/\/\/\</td><td>Repeated decay/attack</td></tr>
<tr><td>11</td><td>\|^^^^^</td><td>Decay - max volume</td></tr>
<tr><td>12</td><td>/|/|/|/|/|/</td><td>Repeated attack</td></tr>
<tr><td>13</td><td>/^^^^^</td><td>Attack - max volume</td></tr>
<tr><td>14</td><td>/\/\/\/\/\/</td><td>Repeated attack/decay</td></tr>
<tr><td>15</td><td>/|________</td><td>Attack - silence</td></tr>
</table>

State Machine

An important point is that the sound chip works like a state machine, meaning that if you set a parameter to a value, this parameter will stay the same until you change it. So, you can't tell the sound chip to produce a signal for a certain time period - you have to tell it exactly when to turn the signal on and off.

This means that, unless the envelope shape ends with silence, the last sound will continue to be heard. One way to turn off the sound is to write 0 to registers 8, 9 & 10 that control the volume / envelope settings.

Sound Ports and Registers

Turbo Sound Next Control - $FFFD

When bit 7 is 1:
Bit(s)
7: 1
6: 1 = to enable left audio
5: 1 = to enable right audio
4-2: = Must be 1
1-0: Selects active AY chip:
00 - reserved
01 - AY3
10 - AY2
11 - AY1

When bit 7 is 0:
Bit(s)
7: 0
6-0: Selects given AY register number for read or write from active sound chip

NextReg Peripheral 3 $08

Bit	Effect
7	1 unlock / 0 lock port Memory Paging Control $7FFD (page 41) paging
6	1 to disable RAM and I/O port contention (0 after soft reset)
5	AY stereo mode (0 = ABC, 1 = ACB) (0 after hard reset)
4	Enable internal speaker (1 after hard reset)
3	Enable 8-bit DACs (A,B,C,D) (0 after hard reset)
2	Enable port $FF Timex video mode read (0 after hard reset)
1	Enable Turbosound (currently selected AY frozen when disabled) (0 after hard reset)
0	Implement Issue 2 keyboard (port $FE reads as early ZX boards) (0 after hard reset)

Peripheral 4 $09

Bit	Effect
7	1 to enable AY3 “mono” output (A+B+C is sent to both R and L channels, makes it a bit louder than stereo mode)
6	1 to enable AY2 “mono” output, 0 default
5	1 to enable AY1 “mono” output (0 after hard reset)
4	1 to lockstep Sprite Port-Mirror Index $34 and Sprite Status/Slot Select $303B
3	1 to reset mapram bit in DivMMC
2	1 to silence HDMI audio (0 after hard reset)
1-0	Scanlines weight (0 after hard reset)

Example Z80 code - play a single note A

```
LD BC, $FFFD               ; Turbo Sound Next Control Register
LD A, %11111111                 ; Enable left+right audio, select
AY1
OUT (C), A                 ; write to port

LD HL, sounddata
LD E, 0

NoteLoop:
LD D, (HL)
LD A, E
CP 14
RET Z
CALL WriteDToAYReg
INC HL
INC E
JR NoteLoop

WriteDToAYReg:
; A = AY register number to write to 0 - 13
; D = value to write
LD BC, $FFFD               ; Turbo Sound Next Control Register
OUT (C), A                 ; write to port

; Write given value
LD A, D
LD BC, $BFFD
OUT (C), A

RET

Sounddata:
DB 252, 0, 0, 0, 0, 0           ; tone ch1, ch2, ch3
DB 0                            ; noise period
DB 1                            ; mix flag,
DB %00001111, 0, 0              ; volume 1,2,3
DB 0, 0, 0                      ; envelope
```

Sources

Various sources, but mainly :

Who	Source
Magnus Krook	SoftSpectrum48 https://softspectrum48.weebly.com/notes/category/ay389108912
Tomaz Kragelj	ZX-Next-Dev-Guide https://github.com/tomaz/zx-next-dev-guide