Let's Make a Retro Game

Episode 16 – Generating Sounds

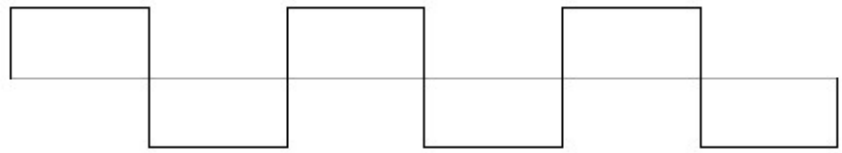
In this episode we are going to look at how we generate sound, this will allow us to add the sound effects for our game e.g. our laser shooting, explosions when things are destroyed.

This is also where we have a little bit of a difference between our platforms, that we have been covering so far, so we are going to have to 'divide and conquer', and cover them as separate tracks.

Colecovision

The Colecovision (as well as the Sega SG-1000/SC-300 and Memotech) uses a Texas Instruments SN76389A Programmable Sound Generator chip that has 3 tone generators and 1 noise generator.

The SN76389A can only generate square waves, but you can control the tone and volume of each of the four channels.



1 - SN76389A output wave shape

The frequency of each tone channel has 10 bits, so can have values from 0 to 1023. 1 is the highest frequency (111861Hz) and 1023 is the lowest frequency (109Hz). This gives a frequency range that can cover 8 octaves from A2 – 10 cents, to A12 – 12 cents.

The noise channel, can either generate periodic or white noise, and you have two bits remaining, that set a shift rate.

The volume of each channel has 4 bits, so can have values from 0 to 15. Zero represents full volume, and 15 represents silence.

This can all be a lot to absorb, so before we get too carried away, lets learn by using some code that will allow us to play with the tone and volume of each channel.

So first we need to add some code to initialise the sound chip to a known state as follows:

```
;************************************************************
; Sound routines
;************************************************************
SOUND_PORT: EQU 0FFh ; SN76489A

; Initialise the sound chip, so that no sound is playing
INIT_SOUND:
  LD A,%10011111 ; Tone 1 volume =  off
  OUT (SOUND_PORT), A
  LD A,%10111111 ; Tone 2 volume =  off
  OUT (SOUND_PORT), A
  LD A,%11011111 ; Tone 3 volume =  off
  OUT (SOUND_PORT), A
  LD A,%11111111 ; Noise volume =  off
```

```
  OUT (SOUND_PORT), A
  RET
```

As you can see there is only one port for the Colecovision's sound chip, and when setting the volume of a channel you only need to output a single byte.

Our next section of code, is a single routine that will set the volume of all three tone channels and the noise channel from stored values in RAM as follows:

```
; sets the current volume of each of the sound channels
SET_SOUND_VOLUME:
  LD HL,CH1VOL
  LD A,%10010000 ; Channel 1 volume
  OR (HL)
  OUT (SOUND_PORT),A
  INC HL
  LD A,%10110000 ; Channel 2 volume
  OR (HL)
  OUT (SOUND_PORT),A
  INC HL
  LD A,%11010000 ; Channel 3 volume
  OR (HL)
  OUT (SOUND_PORT),A
  INC HL
  LD A,%11110000 ; Noise volume
  OR (HL)
  OUT (SOUND_PORT),A
  RET
```

Next, we need a routine to set the current frequency of each channel, from the 10 bit value stored in our Ram area.

```
SET_ALL_SOUND_FREQUENCIES:
  LD HL,CH1FRQ
  LD D,%10000000
  CALL SET_SOUND_FREQUENCY
  LD D,%10100000
  CALL SET_SOUND_FREQUENCY
  LD D,%11000000
  CALL SET_SOUND_FREQUENCY
  RET

SET_SOUND_FREQUENCY:
  LD A,(HL)
  AND 00Fh
  OR D
  OUT (SOUND_PORT),A
  LD A,(HL)
  AND 0F0h
  LD D,A
  INC HL
```

```
LD A,(HL)
AND 00Fh
OR D
RRCA
RRCA
RRCA
RRCA
OUT (SOUND_PORT),A
INC HL
RET
```

So these two routines, allow all channels to be updated from our Ram values. In the 2nd routine it handles sending the channel select along with the upper bits of the frequency and then the 2nd byte of data containing the lower bits of the frequency.

All the code above can just be typed in or copied and pasted into the template, just before the LOAD_CHR_SET code is a good spot.

Only the routines above, that talks to the sound chip, are different between our two tracks, so jump to this episode's example.

Let's Make a Retro Game

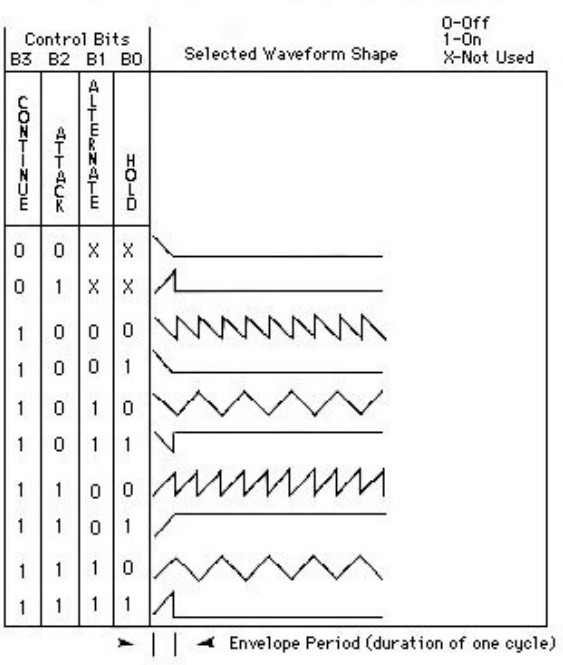
Episode 16 – Generating Sounds

MSX and Spectravideo

The MSX and Spectravideo use a General Instruments AY-3-8910 Programmable Sound Generator chip that also has 3 tone generators and 1 noise generator.

This at 1st glance seems pretty much the same as the Colecovisions SN76389A sound chip, but there are some key differences as follows:

- The generation of noise can be mixed with one or more of the other channels and has a larger range of frequencies i.e. 32 values, instead of four.
- Rather than a single set volume one or more of the channels can also be set to use one of 8 different envelope patterns. The rate of change through the envelope shape can be set using a period of 0-65535, this allows the sounds to continue to play without support from the processor.



The Colecovision Super Game Module, adds 32k of Ram and an AY-3-8910 sound chip, making it very much like a consolised MSX.

The AY-3-8910 chip is controlled using three ports, PSG_LATCH, PSG_WRITE, PSG_READ.

The PSG_LATCH controls which one of the 16 registers we want to write or read a value to/from and then the PSG_WRITE or PSG_READ transfers the data.

Register \ bit		B7	B6	B5	B4	B3	B2	B1	B0
R0	Channel A Tone Period	8-Bit Fine Tune A							
R1						4-Bit Coarse Tune A			
R2	Channel B Tone Period	8-Bit Fine Tune B							
R3						4-Bit Coarse Tune B			
R4	Channel C Tone Period	8-Bit Fine Tune C							
R5						4-Bit Coarse Tune C			
R6	Noise period				5-Bit Period control				
R7	Enable (bit 0 = on, 1 = off)	IN/OUT		Noise			Tone		
		IOB	IOA	C	B	A	C	B	A
R8	Channel A Envelope on/off, Volume			Env	volume				
R9	Channel B Envelope on/off, Volume			Env	volume				
R10	Channel C Envelope on/off, Volume			Env	volume				
R11	Envelope Period	8-Bit Fine Tune Envelope							
R12						4-Bit Coarse Tune Envelope			
R13	Envelope Shape/Cycle					CONT	ATT	ALT	HOLD
R14	I/O Port A Data Store	8-Bit Parallel I/O on Port A							
R15	I/O Port B Data Store	8-Bit Parallel I/O on Port B							

Let's Make a Retro Game

Episode 16 – Generating Sounds

So now let's add some code to either the MSX or Spectravideo templates, as both use the same code.

First let's add code to initialise the sound chip to a known state as follows:

```
;************************************************************
; Sound routines
;************************************************************

; Initialise the sound chip, so that no sound is playing
INIT_SOUND:
  LD A,7        ; write to reg 7 mixer control
  LD E,0B8h     ; disable noise, enable all 3 tones
  CALL WRTPSG
  RET
```

Note: On the MSX WRTPSG is calling a routine in the BIOS, as the Spectravideo ROM Bios is switched out, is technically in the same place as our code and to make things easier, you will find a copy of the MSX code has been added to our SVIRom-Lib.asm file in the starting template.

Next, we will add a routine to set the current volume of each of our three tone channels from the values stored in our Ram variables as follows:

```
; sets the current volume of each of the sound channels
SET_SOUND_VOLUME:
  LD HL,CH1VOL
  LD E,(HL) ; Channel 1 volume
  LD A,8
  CALL WRTPSG
  INC HL
  LD E,(HL) ; Channel 2 volume
  LD A,9
  CALL WRTPSG
  INC HL
  LD E,(HL) ; Channel 3 volume
  LD A,10
  CALL WRTPSG
  RET
```

Next, we need a routine to set the current frequency of all our tone channels as follows:

```
SET_ALL_SOUND_FREQUENCIES:
  LD HL,CH1FRQ
  XOR A
  CALL SET_SOUND_FREQUENCY
  LD A,2
  CALL SET_SOUND_FREQUENCY
  LD A,4
  CALL SET_SOUND_FREQUENCY
  RET

SET_SOUND_FREQUENCY:
```

```
  LD E,(HL)
  CALL WRTPSG
  INC HL
  LD E,(HL)
  INC A
  CALL WRTPSG
  INC HL
  RET
```

So, these two routines, allow all channels to be updated from our Ram values. In the 2nd routine it handles sending the channel select along with the upper bits of the frequency and then the 2nd byte of data containing the lower bits of the frequency.

All the code above can just be typed in or copied and pasted into the template, just before the LOAD_CHR_SET code is a good spot.

Only the routines above, that talks to the sound chip, are different between our two tracks, so jump to this episode's example.

Let's Make a Retro Game

Episode 16 – Generating Sounds

Sound Example

The code provided with this episode has the usual Start and End folders for each system.

The Start folder contains a template application, that sets up the system and loads some tiles so we can display some text on screen.

This example application will just allow us to play with the sound chips tone and volume levels for each channel, using our joystick controller.

The code to move a sprite pointer between each of the channels tone and volume settings is already included, so if you run the application it should look like this:

And you should be able to increase and decreasing the volume of the current channel by pressing up or down on the joystick, plus pressing left or right will decrease or increase the frequency of the current channel. Pressing the fire button will move the pointer to the next channel.

Now add the code for your chosen platform from earlier in this document, either from the Colecovision section or the MSX/Spectravideo section. It can be added just before the LOAD_CHR_SET routine.

Now let's add some code to call each of these routines.

Right near the start of our code, find where the we "initialise clock" and add the following call to INIT_SOUND:

```
  CALL INIT_SOUND

  ; initialise clock
  LD  HL,TIMER_TABLE
  LD  DE,TIMER_DATA_BLOCK
  CALL INIT_TIMER
```

Next find the comment "set cursor position" and insert the following code, that sets some initial values, before it:

```
  LD A,12
```

```
  LD (CH1VOL),A

  CALL SET_ALL_SOUND_FREQUENCIES
  CALL SET_SOUND_VOLUME

  ; set cursor position
```

Now our starting code already changes the volume and frequency values in Ram, so we just need to call our routines to send them to the sound processor as follows:

```
  CALL SELECT_CHANNEL
  CALL PLAYER_ACTIONS
  CALL SET_ALL_SOUND_FREQUENCIES
  CALL SET_SOUND_VOLUME

  JR SPLASH_TITLE2
```

This code will increase or decrease the currently selected Tone or Volume value and then update the sound chip to be the same value.

If you run the application, it should look the same but now if you move the joystick up and down on a field you should hear a continuous sound being output.

So, we have covered a lot of concepts in this episode, and probably what seems like only a small amount of code, but hopefully something interesting to play with.

Next time we will generate some actual sound effects for our game.