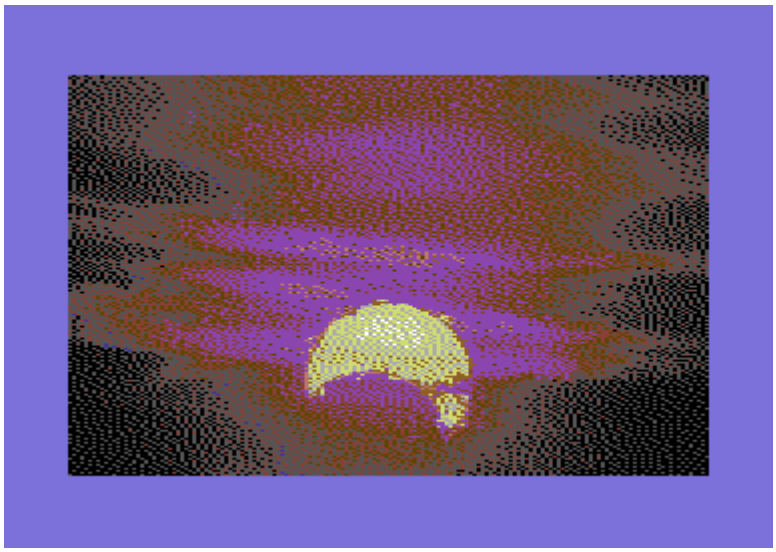


“Eclipse”

Vice snapshot with Vice palette

Made with the GIMP from a photo
and converted to C64 160x200
Multicolor Mode Bitmap
by Stefano Tognon
in 2005

“Other tiny”
...

Free Software Group

GoGoDin 9
version 1.00
14 January 2006

SIDin Contents

General Index

Editorials

Stefano Tognon <ice00@libero.it>

Hi, again.

In this number there are two technical news:

- The use of *.odt* instead of *.sxw* as document. This means that more programs can read the saved document (you may use OpenOffice 1.5 or 2.0 to open them)
- Sources are colored based onto 6502 assembly syntax. This come out automatically by the new copy/paste function of KWrite (the KDE editor in Linux that supports 6502 syntax): now it is more simple to look at sources (maybe old numbers one day could be converted to colored syntax too).

The articles are in some sort similar as in the previous number: the analysis of the 256 bytes entries of Tiny Sid Compo and the further analysis of the MK4.

You should see in the 256 bytes entries lot of differents approach to the programming and I think that you could have some nice idea in how to implement your one in the near to come Tiny Sid Compo 2.

After some further testing I was able to made the card sound better then before and finally I find the problems that remain to solve in my card.

You probably will find this article a little too much descriptive and based on hypothesis that where reduced pages after pages, but this is the way I really used for investigate to the card and I think that a simple: “the card did not work for 1=.. 2=.. 3=.. “ gives no idea of the work below this conclusion.

Bye
S.T.

News

Some various news of players, programs , competitions and hardwares:

- Goattracker 2.12/2.18
- Polly Tracker 1.2
- SwinSID
- JSIDPlay 0.3
- HVSC 43
- PollyTracker Compo
- Xmms-sid-0.8.0beta15
- SwinSID2
- The Sid Compo V
- Goattracker 2.2/2.22
- Sidplay64 v0.3
- HVSC 44
- XMPlay SID plugin beta 24b

Goattracker 2.12/2.17

Released from August to November 2005 the new versions of Goattracker PC music tracker:

v2.12
- Playroutine 1 has buffered SID-writes.
- Octave 0 is not disabled in routines with sound FX (no matter what the relocator says...)

v2.13
- Standard playroutine now both in unbuffered and buffered flavors.

v2.15
- Added SHIFT+I for inverting current pattern selection / whole pattern.

v2.16
- Fixed octave selection with / * for laptop keyboards.
- Empty patterns referenced in the orderlist will be saved when saving a song.
- Save dialog will reappear if writing the song/instrument/executable music failed.
- Optimized handling of "packed rests" in the playroutines.
- Added < > for instrument selection in instrument/table edit modes.

v2.17
- Fixed initialization of instrument vibrato.
- Playroutines size-optimized.

v2.18
- Wavetable left side values changed. Delay is now $01-$0F and inaudible waveforms (register values $00-$0F) have been mapped to table values $E0-$EF.

Download from: http://covertbitops.c64.org

Polly Tracker 1.2

Released on 20 August 2005 the new version of Polly Tracker:

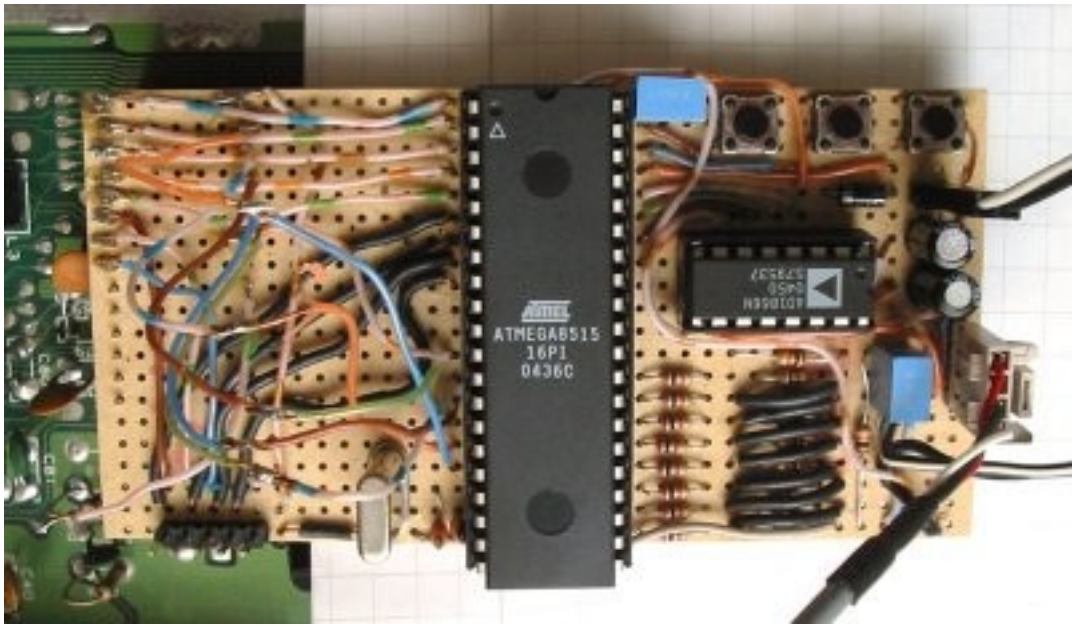
v1.2
- Added demo modules
- Loader problems with some disk drives fixed. Some old drives may still require an Initialize command (c=+M and type 'i') before loading dir

Download: http://www.kolumbus.fi/aleksieeben/pollytracker/pollytracker.zip

SwinSID

Features of SwinSID (16-bit stereo hardware sound module for C64) :

- hardware compatibly with SID chip.
- Software compatibly with SID (almost full)
- 3 main channels + 3 effect channels.
- Wavetable synthesis
- 9 instruments stored in flash ROM
- Programmable ADSR envelope generator.
- Stereo reverb
- High quality audio DAC - 16bit stereo.
- 16 bit mixing with interpolation
- Sampling rate 31,25 Khz.



http://www.swinkels.tvtom.pl/swinsid/

JSIDPlay 0.3

JSIDPlay is SID-player based onto JaC64 and version 0.3 was released in September 2005:

Features

- ADSR emulation including ADSR bug
- Emulation of combined waveforms using same method as re-sid (sample-lookup)
- Filter (LP/BP/HP)
- Synchronization, ring modulation, etc.
- Interrupts - IRQ (50 times per second) - soon support for other playspeeds
- Bank switching
- PSID sample play (Galway Noice, and ordinary samples)
- Supports .sid files (PSID, some RSIDs)
- Better "timing" than ordinary JaC64 emulator (via sound-player)
- Animation/Oscilloscope shows all SID voices, ADSR, frequency, etc while playing.

Look at http://www.dreamfabric.com/c64/jsidplay/ for the online version.

HVSC 43

HVSC Update #43 was released on 21 September 2005 at www.hvsc.c64.org

After this update, the collection should contain 30,743 SID files!

This update features (all approximates):

- 759 new SIDs
- 19 fixed/better rips
- 7 fixes of PlaySID/Sidplay1 specific SIDs
- 24 repeats/bad rips eliminated
- 500+ SID credit fixes
- 600+ SID model/clock infos
- 24 tunes from /DEMOS/UNKNOWN/ identified :-)
- 34 tunes moved out of /DEMOS/ to their composers' directories
- 5 tunes moved out of /GAMES/ to their composers' directories

New features in HVSC v43:

- Not much really groundbreaking this time, just a lot of SIDs for you to enjoy.
- Murdock quit the HVSC team, good luck with your various other projects!

Main Composers featured in this update: (Artists marked with NEW are either completely new to the HVSC or they get their own directory in this update)

```
4-Mat                    Abject
Artlace                  Kjell Nordbø
Brizz                    Pierre Conus (NEW)
Cubehead                 Data
Jeroen Koops             LDX#40 (NEW)
Madaco (NEW)             Ozzy Oldskool
Randy                    Kristopher Roebuck
Surgeon                  XPO (ex-Goner)
Peet                     Harald Rosenfeldt (NEW)
Rambones                 Hein Holt
DRAX
```

PollyTracker Compo

PollyTracker Compo ran in August/September at http://www.kolumbus.fi/aleksieeben/pollytracker/ with tunes made with the Polly Tracker editor. Final classification:

1	Pollyhunter	Abaddon/Fairlight	98/110
2	Polly wants a tracker	Randall	96.5/110
3	Supernoid	E. Jones	87.5/110
4	Melo	Hukka	70/110
5	Instant funk	SDFG	65/110
6	Heavy metal rain	Uneksija	65/110
7	Old domain	Murdock/Tropyx	59/110
8	Bakakaj		57/110
9	Extreme!!!	Murdock/Tropyx	41/110
10	Eight(lo)	Aneurysm	39/110
11	My Commodore	Bones	38/110
12	Trancer		38/110

Xmms-sid-0.8.0beta15

Released on 10 November 2005 the new version of the sid plugin for Xmms.

News in version 0.8 so far:

- Complete re-write.
- Support for multiple emulator libraries, including libSIDPlay 2.x! See instructions in "INSTALL" for more information.
- HVSC song-length database support. XMMS-SID now supports the XSIDPLAY's song-length database for tunes contained in HVSC collection. Requires downloading of a small package (few hundred kB's), that contains the database. Read "README" for more info.
- Minimum and maximum playtime settings. If enabled, song will be played for given time minimum and maximum. This can be used as a fall-back method if song-length database does not contain information for selected tune.
- Improved file information dialog.
- XMMS v1.2.5 introduced the "generic title format" that can be used to have same format titles for those plugins that support it. XMMS-SID is now one of them. There is also an overriding option, that enables you to have SID-specific titles. See the configuration dialog for more information.
- New sub-song control methods as selectable options. Does not require patching of XMMS anymore, so they are easier to use. The XMMS patch used by older versions is supported as well, though.
- Throw in a handful of bugfixes (and possibly new bugs), stir powerfully. Finally, sprinkle some minor improvements on the top and vo�a!

Download the stuff from: http://www.tnsp.org/xmms-sid.php

SwinSID2

After the released of SwinSID specifications, now are available the specifications of the next generation of the 16-bit stereo hardware sound module for C64:

- Pin compatible with C64's SID socket.
- Up to 8 voice channels
- 2 channels of white and purple noise generators
- 8 waveform oscillators with effects or 8bit stereo sample playback on each channel.
- Cyclic wave synthesis
- 16 waveforms stored in flash ROM
- Independent stereo volume control
- Stereo phase control for surround effect.
- 16 effects for waveform processing including flanger, chorus, ring modulation, phase modulation and more.
- 2 internal LFO oscillators for effect automation.
- Hardware vibrato and tremolo effects.
- One global filter with adjustable cut-off and resonance frequency.
- High quality 16 bit stereo audio DAC
- 16 bit mixing with interpolation
- Sampling rate 31,25 KHz or 24 KHz
- Requires special software to work.

Look at http://www.swinkels.tvtom.pl/swinsid/swinsid2.htm

The Sid Compo V

The 5° sid compo organized by www.c64.sk was running from 8 October to 13 November 2005.

Here the final result:

1	Assatas Song	Michal Hoffmann (Dat Nigga Randall) Poland	473
1	Bombs Over Dresden	Alexander Rotzsch (Fanta) Germany	473
2	Pick It Up	Stellan Andersson (Dane) Sweden	449
3	I Have a Knot In My Superstring	Vincent Merken (_V_) Belgium	448
4	Water Is Fun	Kamil Wolnikowski (Jammer) Poland	446
5	Valley of Dreams	Dennis LeDoux (Phase 2/) Denmark	442
6	Solar Incantation	Rafal Kazimierski (Asterion) Poland	436
7	Trainline Andromeda	Luca Carrafiello (Luca) Italy	430
8	Lemmie Eat The Rastertime	Marcin Kubica (Booker) Poland	424
9	Interstellarian Love	Arman Behdad (Intensity) Germany	423
10	Transylvanian Whipping	Lasse Öörni (Cadaver) Finland	418
11	error 23	Ronny Engmann (Dalezy) Germany	415
12	In Deep Freeze	Timo Taipalus (Abaddon) Finland	411
13	Nightbird	Sascha Zeidler (Linus) Germany	402
14	PMS	Hein Holt (Hein) Netherlands	389
15	Dazzy Levstovski	Siegfried Rudzynski (Crome) Germany	381
16	I w. to c. s. new but it s.like sid	Daniel M. Gartke (Turtle) Germany	367
17	Future Impulse	Stephan Parth (Nata) Italy	355
18	Spank!	Jan Diabelez Arnt Harries (Rambones) Denmark	348
19	Exploring New Worlds	Freedom Italy	339
19	Wall of Fire	Richard Bayliss (Richard) United Kingdom	339
20	Bossah Novah	Marcin Romanowski (Sidder) Poland	336
21	Hangman's Swing	Maciej Stankiewicz (Trompkins) The Land of Po	309
22	dATA dISCO	Robert Dörfler (LordNikon) Germany	280
23	Zion	Gerhard Flagge (G-Fellow) Germany	273
24	Run From a Ghost	Deborah (Decompracid) Netherlands	216
25	Poetry of a Lonely Mind	Peter Bergstrand / Sweden	131

More info at http://www.c64.sk/sidcompo5-results.html

Goattracker 2.2/2.22

Released on 11, 21 December 2005 and 9 January 2006 the new versions of Goattracker:

v2.2

- Added the speedtable for more precise control of vibrato, portamento and funktempo.
- Added SHIFT+O to optimize the speedtable (remove unused entries).
- Added SHIFT+R to convert between absolute/relative notes in the wavetable.
- Added SHIFT+RETURN in pattern/instrument editor to convert old style portamento, vibrato and funktempo parameters to speedtable entries.
- Song and instrument format modified for 4 tables. Old 3-table data will be loaded but not saved anymore.

v2.21

- *Fixed NTSC CIA timer value for SID files.*
- *Shift+E will copy several effect rows if pattern has been marked.*

v2.22

- Shift+N will also negate relative wavetable notes.
- When converting oldstyle parameters to speedtable entries with SHIFT+RETURN, the speedtable view will shift to the new entry if one was created.

Download from: http://covertbitops.c64.org

Sidplay64 v0.3

Released on 19 December 2005 the C64 program that can playback sid files from the HVSC collection on a real c64:

- Supports most CBM and CMD drive types and runs from the device currently selected. It is recommended to use Action Replay/Retro Replay.
- It can handle 196 files in the playlist. A 1541 disk can only handle 144 files. Other drive types can handle more.
- It detects if the file loaded is a HVSC sid file. (only version 2 sid files)
- Two playlist functions: play next tune in list or play tunes at a random selection.
- The program relocates itself according to startPage (reloc-StartPage) and pageLength (relocPages) found in the sid header. Possible relocation area is: $0400-$d000
- SID Tunes that doesnt use timers (dc04/dc05) is played back in PAL or NTSC speed. If the video standard for the tune is unknown, playback will be PAL speed.

Download from http://home.eunet.no/~ggallefo/

HVSC 44

HVSC Update #44 was released on 24 December 2005 at www.hvsc.c64.org

After this update, the collection should contain 31,330 SID files!

This update features (all approximates):

- 620 new SIDs
- 87 fixed/better rips
- 4 fixes of PlaySID/Sidplay1 specific SIDs
- 4 repeats/bad rips eliminated
- 360 SID credit fixes
- 150 SID model/clock infos
- 9 tunes from /DEMOS/UNKNOWN/ identified :-)
- 20 tunes moved out of /DEMOS/ to their composers' directories
- 12 tunes moved out of /GAMES/ to their composers' directories

Main Composers featured in this update:

(Artists marked with NEW are either completely new to the HVSC or they get their own directory in this update)

Tomas Danko DRAX Jˆrg Rosenstiel

Akuma Pan (NEW)	A-Man	Richard Bayliss
Arman Behdad	Bluez	Booker
Drake	Hein Holt	Joan
Josstintimberlake (NEW)	Linus	MAC2
Madaco	Maktone	Mankeli
MRT (NEW)	Rayden	Shock (NEW)
Surgeon	The Blue Ninja	

XMPlay SID plugin beta 24b

XMP-SID is a plugin for XMPlay (http://www.un4seen.com/) for playing Commodore 64 SID music, featuring:

- high quality playback
- cycle exact C64 emulation by libsidplay2 and reSID engine
- PSID, RSID (+BASIC), Sidplayer (+stereo) and C64 executable files (PRG) loading
- neat sound effects (configurable stereo separation, surround and fadeout)
- SLDB (song-length database) support
- advanced song length manipulation
- STIL (SID Tune Information List) and BUGList comments displaying (also with archived HVSC and files outside of HVSC directory)
- Favourite Top 100 SIDs rank displaying
- PlaySID tags and Sidplayer comments displaying
- fast time-based seeking
- configurable C64 emulation
- configurable output
- subsong switching
- ...plus tons of XMPlay's features

Version beta 24b was releaded on 6 january 2006 and can be downloaded at:
http://dhost.info/pieknyman/bin/xmp-sid.zip

Nata (Stephan Parth) Interview!

by Stefano Tognon

This time I go to interview a very young and active Italian composer: Nata. The interview was achieved this month.

Hello, Nata,
could you give some words about you and your real life?

I'm 22 years old and live in a small village called Naturns in Italy.
Currently I have no job, since I don't know what I could do...
However, I learnt something that has to do with advertising, but unfortunately I am really fed up with everything that has to do with that. :-)

You are very young, so why did you choose to compose music with an so old chip?

That's really a good question!
I simply had to recognize that I don't get happy me with today's commercial music. I have always liked raw synthesizer sounds, but until 2003 there was no way to start composing.
It was rather hard for me to compose something, since I had no idea how the SID chip works. But now it's pretty easy and I'm always surprised what is possible to squeeze out from it.
SID music gives me everything what I need for my life – it's simply the BEST MUSIC around the world.

Did you own some Commodore computers?

In 1989 my brother bought a Commodore 64, but he sold it quite soon for unknown reasons. I really was amazed about this little machine – it was the first computer I ever saw. But I was too young (6 years old) to do something with it. Some years later I bought my own C64-II, a 1541-II disk drive and a green Monitor.
Today I hardly use the disk drive since I'm a happy owner of an MMC64. This way I easily can test my music, play games and listen to the HVSC. Unfortunately the BIOS has no save/load routines and only a simple Sidplayer.
However, I don't think there are many Italian sceners that have such a nice card, right?

I receive lot of request from friends that want to try to compose music for the C64 about what program to use, but with all the editors available the answers is not so simple. Have you already find a good (for you) tool for composing? How did you make your choice?

At the moment I compose with GoatTracker 2.2 (Yes it's cross-composing) since it offers nearly everything I need. Cadaver was very generous and implemented a lot of features that I requested (thank you very much, Cadaver!!).
I know there are a lot of people that believe GT is bad or something between (since it does not work on a real c64), but the truth is there is hardly any difference. You just have to use 8580 emulation and reSID interpolation turned on. Even them most people don't know that some of the today's best composers like Hein, Jammer and Randall use GT.
At the time I started to compose SID music I tested some c64 based editors (f.e. Voicetracker, DMC, JCH...), but they were simply too hard for me – first choice was GoatTracker.

What are your future projects about the sid music that you had already planned to make for

you or your group?

Samar Productions (The Group I'm currently in) will release soon (I really hope that!) my first music collection with about 8 tunes. It contains own songs, remixes and some covers.
Furthermore I did a nice Stereo-SID tune (Yes, 6 SID channels!!!). I don't know what Samar plans do with it, but I probably do not wait much longer for the release since I did it already in summer 2005.
One tune I will write for the HVSC-Crew (for the 10th anniversary), and another one is planned for the next issue of the diskmag "Attitude"
For sure I will do again some 1:1 covers of some SIDs I like and fix filters for the 8580. :-)

Now some quick final (standard) questions:
Real machine vs emulator: what do you think of?

There isn't much difference between an emulator and a real machine.
I quite often use Vice (CCS64 SID emulation is too inaccurate), since it offers so many features. For example it allows me to run IDE64 fixed games from PC desktop, simulate a REU or switch between different SID models.
But when playing games with friends, it makes much more fun to use the real thing.

6581 vs 8580 chip: any (musical) preference?

I prefer the 8580. It's a heavily improved SID, but unfortunately the most old game tunes sounds very bad on it.
There are some tunes (Ghouls 'n' Ghosts – subtune 6, The Last Ninja – subtune 9 and some more) around, which use some odd filter settings that aren't audible on a new SID.
On the other hand the 8580 has very nice waveform combinations and filters that are accurate.

What is the worst sid that you compose and the better one?

Well, every tune in the "c64music\VARIOUS\M-R\Nata\Early_Tunes" directory is really worse (except maybe "Nata is Back". It's a bit experimental)
One of my favorites is "Stargate" (done in GoatTracker 2.0 Beta). It has quite a simple structure but is finest trance-techno. As far I can remember it was THE FIRST song that has been released with the improved GT system.
Another one is "Sentimentale Tc-Mix". It's based on the "Sentimentale" and "Sentimentale 8580" tunes. The techno part starts at 00.33 min. and is quite nice.

Who are your best sid authors?

Hmm, that's really hard to answer, because there are so many good composers around. From the classical game-composers I prefer Chris Huelbeck, Tim Follin, Martin Galway, Jeroen Tel and Rob Hubbard. But I like music of scene musicians much more. The best ones are Cane, DOS, Jeff, Jammer, Randall, Hein Holt, Reed, *_V_, and Welle: Erdball / Honey.*

What are the best sids ever in your opinion?

I like every song of Cane & DOS. My most favorite song of Cane is BRAINBALL-GAME". Unfortu-

nately it isn't in the HVSC (and probably it never will be), since it bugs after some minutes.

Ok, here a list of my FAVOURITE SIDs (in no particular order):

"Aurora" by PseudoGrafx
"Dancesque" by TBB
"Draxish" by Taki
"Smilygirl" by Chubrock
"Yes Comment" by Tomas Danko
"Psy Six" by _*V*_
"Timmy-Boy" by Tim von Straaten
"Nicht schlecht" by Stephan Schmid
„Dutch Breeze: Soft and Wet" by Reyn Ouwehand
"Selfemade.exe.amj" by AMJ
"Shades" by Chris Huelsbeck
"Foatee" by Jammer
"Higher State of SID" by Cyberbrain
"Reanim8ed" by The Syndrom
"Phronk" by Nebula
"Gliding" by Kristian Røstøen

and a lot more…

The probably BEST SIDs:

"Guerrilla War" by Jonathan Dunn
"Wizball" by Martin Galway
"The Last Ninja" by Ben Daglish & Anthony Lees
"Aerobics" by Bill Mauchly
"Rainbow Islands" by Jason Page
"R-type" by Chris Hülsbeck & Ramiro Vaca
"IK+" by Rob Hubbard

Finally, many thanks for the time you give for this interview, and now you can say any things you want that the people will read from you!

First let me say that it really was a pleasure to answer your questions (hehe, my first interview!). I'm always happy to get a new issue of your mag – it's simply great!!!

Furthermore, I'm always glad the receive any mails from people out there (for any purpose):

<natac64 [sid] web.de>

Ok, my 4 biggest wishes:

- GoatTracker 64 [The C64 version of GT]
- Writing music for a C64 Zelda or Bomberman
- My own C64 music group
- SID music forever ;-)

Tiny Sid Compo 256b entries

by Stefano Tognon <ice00@libero.it>

In this article I go to show and comment the entries of Tiny Sid Compo for the 256b category.
This was the list of entries:

1. Myblock... One Block
2. Random Ninja
3. Crue Gurl Freestyle
4. 128 Byte Blues
5. Imperial March
6. New Kid On The Block
7. Repeat Me
8. Electronic
9. Repetitive Tune BASIC
10. Splatform256

Myblock... One Block

This is the 256 bytes tune by Agemixer and it is a co-winner.

The player, after disable interrupt with SEI, simple starts to copy his code to zero page addresses and then switches the execution on it. This is a big solution as now all the variables can be accessed from zero page and so using only one byte for the address!

The player now initializes all the sid voices using a table (initTable): voice 1 and 3 are rectangular and a filter is apply in voice 1.

As interrupt is disable, it synchronized the code with raster line $80.

Register y is used as an index with 32 values that always is incremented. According with his value (or better his bits values) some actions are taken.

Y Value	*Description*
0..15	Read high/low voice 1 note frequency from actual voice 1 index position (location $0A)
16..31	Make a frequency (high/low value affected) effect in voice 1 using XOR and LSR operations onto Y values become: 15, 14, 13, 12, 11, 10, 9, 8 and they are double.

The drum of voice 2 is made according with the bit 2 of register y:

- bit 0/1 drum=on/off

So every 4 values of Y, drum state is inverted

Voice 3 uses a 4 bytes pattern table and each pattern has 8 notes in it.
It normally uses bits 3-4 as index for pattern table, but every 8 times that an index is incremented, it goes to use bits 2-3 instead (it uses self modified code for this). At the same time, the low/high byte of duty cycle of Voice 3 is changed.

The upper bits of register Y are also used for setting the high byte of filter cut-off frequency and even this time every 8 times the code is self-modified to change his behavior.

Maybe you will find more behaviors by looking at the player more carefully...

```
.org $0801

.byte $0B, $08, $D5, $07, $9E
```

```
  .org $0806
  .byte $32, $30, $35, $39, $00

; copy all the code to zeropage!
  .org 2058
      ldy  #$00
      sei
      ldx  #$00
loop
      dex
      lda  $0806,x
      sta  $00,x
      cpx  #$17
      bne  loop
      jmp  $0017

  .org $0017
entry:
      lda  initTable-1,x
      sta  $D401,x               ; initialize sid registers
      dex
      bne  entry

sync:
      lda  #$80
wait:
      cmp  $D012                 ; Reading/Writing IRQ balance value
      bne  wait

      tya
      and  #$1F
      cmp  #$10
      bcc  freqEff

      ldx  $0A                   ;  Flag: 0=LOAD, 1=VERIFY
      lda  v1LoFreq,x
      sta  $D400                 ; Voice 1: Frequency control (lo byte)
      lda  v1HiFreq,x
      sta  $D401                 ; Voice 1: Frequency control (hi byte)
      jmp  cont

freqEff:
      eor  #$0F
      lsr
      sta  $D401                 ; Voice 1: Frequency control (hi byte)
      sta  $D400                 ; Voice 1: Frequency control (lo byte)

; byte format
;  aaaa abcc
; b=0/1 drum on/off

cont:
      tya
Pos2:
      and  #$07                  ; $07 <-> $0F
      asl                        ; ASL <-> NOP
      asl
      asl
      asl
      eor  #$7F
      sta  $D416                 ; Filter cut frequency: hi byte

      tya
      and  #$04
      lsr
      lsr
      eor  #$81
      sta  $D40B                 ; Voice 2: Control registers

      tya
      lsr
Pos3:
      lsr                        ; LSR <-> NOP
      sta  $D411                 ; Voice 3: Wave form pulsation amplitude (hi byte)
      ror  $D410                 ; Voice 3: Wave form pulsation amplitude (lo byte)
      lsr
      and  #$03                  ; limit the value to 0..3
      tax
```

```
      lda  pattern,x
      sta  Pos+1
      ldx  $0A
Pos:
      lda  pat1,x
      tax
      lda  v3LoFreq,x
      sta  $D40E                ; Voice 3: Frequency control (lo byte)
      lda  v3HiFreq,x
      sta  $D40F                ; Voice 3: Frequency control (hi byte)

      iny
      tya
      and  #$3F
      bne  goSync
;0081
      inc  Ind+1
Ind:
      lda  #$08                 ; this is a local index
      and  #$0F                 ; let value to be 0..15
      bne  skipInv3
                                ; invert for voice 3 every 8 times
;0089
      tax
      lda  Pos3
      eor  #$A0                 ; LSR <-> NOP
      sta  Pos3
      txa

skipInv3:
      and  #$07                 ; let value to be 0..7
      sta  $0A
      bne  goSync
                                ; invert for voice 1 every 8 times
      lda  Pos2+1
      eor  #$08                 ; #$07 <-> #$0F
      sta  Pos2+1

      lda  Pos2+2
      eor  #$E0                 ; ASL <-> NOP
      sta  Pos2+2

goSync:
      jmp  sync

pattern:
  .byte <pat3, <pat1
  .byte <pat3, <pat2

v1LoFreq:
  .byte $00
  .byte $56
  .byte $FF
  .byte $7C
  .byte $00
  .byte $56
  .byte $FE
  .byte $20

v1HiFreq:
  .byte $08
  .byte $05
  .byte $02
  .byte $04
  .byte $04
  .byte $05
  .byte $05
  .byte $07

v3LoFreq:
  .byte $00
  .byte $F6
  .byte $06
  .byte $5B
  .byte $F9
  .byte $E9
  .byte $82
```

```
pat1:
  .byte $00
  .byte $03
  .byte $01
  .byte $01
  .byte $00
  .byte $00
  .byte $01
  .byte $01

pat2:
  .byte $02
  .byte $05
  .byte $04
  .byte $03
  .byte $02
  .byte $03
  .byte $04
  .byte $03

pat3
  .byte $04
  .byte $07
  .byte $06
  .byte $05
  .byte $04
  .byte $05
  .byte $06
  .byte $06

v3HiFreq:
  .byte $10
  .byte $11
  .byte $13
  .byte $15
  .byte $17
  .byte $1A
  .byte $1C

initTable:
  .byte $20         ; wave low byte voice 1
  .byte $0A         ; wave high byte voice 1
  .byte $41         ; control reg of voice 1
  .byte $09         ; attack/decay voice 1
  .byte $E9         ; sustain/release voice 1
  .byte $FE         ; freq low of voice 2
  .byte $FF         ; freq high of voice 2
  .byte $09         ; wave low byte voice 2
  .byte $09         ; wave high byte voice 2
  .byte $08         ; control reg of voice 2
  .byte $02         ; attack/decay voice 2
  .byte $35         ; sustain/release voice 2
  .byte $00         ; freq low of voice 3
  .byte $00         ; freq high of voice 3
  .byte $09         ; wave low byte voice 3
  .byte $09         ; wave high byte voice 3
  .byte $41         ; control reg of voice 3
  .byte $FF         ; attack/decay voice 3
  .byte $7F         ; sustain/release voice 3
  .byte $00         ; filter freq low
  .byte $00         ; filter freq high
  .byte $C1         ; filter in voice 1
  .byte $1F         ; thief pass filter and volume max
```

Random Ninja

This is the my 256 bytes entry. It tries to performs the end part of “Last Ninja II – Central Park” in the Chris Holm edition. From my analysis and a little Goattracker test tune, I saw that with 2 voices I can have a good feeling about the generated music.

In particular I find those instruments values the ones that work better (without using very complex effects):

```
voice 2:
$81 C8
$41 A8
$40 00

voice 1:
$41 00
$21 00
```

The first is a drum that gives the rhythm to the tune, while the second is the melody part.

After some experimentation I find that the best way to describe the notes was to use a unique pattern where one byte represent both voice 1 and voice 2 values, using fix duration for note:

Value	***Description***
Upper bit	Play drum in voice 2 if it is 1
Lower 7 bits	Play note in voice 1 if not zero, else continue with previous note

This is possible as for the drum we use fixed frequency for the note to use and so only a flag is needed for it.

At this point the problem is how to create the timbre of the instrument, as the decode of note and pattern reading is very simple as there is a unique pattern and so a unique pointer for all the voices.

The main idea used here is to have some synchronization loop with Vic II raster low position 0 (that will come double in a frame) and to perform some instrument steps if it is necessary, but after a fixed number of synchronization, we loop to begin for managing a new note.

So the interrupt is disable (SEI instruction) and the fakeSyncro routine is used with the passed number of time to use it. fakeSyncro uses kernel delay to skip raster position 0 and so it may be called with the number of synchronizations to use as needed.

This table shows so the operations in details:

Raster 0 ***number of Ticks***	***Operations***
0	read current note/duration execute voice 2 (if it is the case) execute voice 1 (if it is the case)
1	Put $21 as waveform for voice 1 Change high frequency of voice 2

Raster 0 ***number of Ticks***	***Operations***
2	Change high frequency of voice 2
9	Voice 2 hardrestart (this is needed, otherwise no sound is produced) Restart from beginning

At his point the tune was ready, but I want to make some special effects in it. One was to add a fade out, but then I migrate to the one that give the title to the tune: random changes into the tune (the random number is taken from voice 3 output)

This was simple to achieve: as soon as a complete pattern is sounded, the wave to use for voice 1 timbre (the second value of table) is randomly taken from $21 and $11 (a self modified code is even used). This gives some kind of life to the music.

If you want to see how fakeSyncro works, just uncomment the "inc 53280" into the code: you will see a very nice graphical effects!!!!

Here the code:

```
; Random Ninja
; 256b SID music
; This is a remix of the ending
; "Last Ninja II - Central Park" of Matt Grey
; in the remix version of Chris Holm

; this tune use voice 1 and 2 for music
; voice 3 is used for random number that change voice 1
; this make this tune quite different at each minute
; I listen to it for hours..... due to LN II magic sound

  processor 6502

   org 2049

  .byte $0b,$08,$e8,$03,$9e,"2061",0,0,0

  .org 2061


point =   $84        ; +$85 pointer
patt     = $44       ; 4B index to current pattern

RDELAY = 9        ; reload delay

; note declaration
G3  = 1
A3  = 2
C4  = 3
D4  = 4
E4  = 5
G4  = 6


;voice 2 should be like:
;$81 C8
;$41 A8
;$40 00
; in goattracker format for the best drum effect.

; voice 1 is
; $41 00
; $21 00
; in goattracker format


      ;stx  patt+7
      ;inx
      ;stx  delay              ; initial delay to virtually 0

      lda  #$8F               ; voice 3 off
      sta  $D418              ; volume max
```

```
      sei                       ; did not allow interrupt
      ;lda  #<irq
      ;sta  $0314
      ;lda  #>irq
      ;sta  $0315
      ;cli
      ;rts
                                ; don't use IRQ, as too many code is needed for making the dinamic
                                ; waveform change at each tick
                                ; this is best achieved with fake interrupt syncronizatin code

;irq:
      ;dec  delay
      ;beq  music
      ;jmp  exit

resetPat:
      lda  $D41B                ; "random" byte
      lsr                       ; "random" bit
      lda  #$11                 ; triangular waveform
      bcs  skipAdd

      adc  #$10                 ; $21 waveform
skipAdd:
      sta  pos+1                ; self modified code

      ldx  #0                   ; reset pattern
      stx  patt
music:

      ldy  patt                 ; load pattern index of voices
      cpy  #64
      beq  resetPat

nextInd:
      iny                       ; increment index (so pattern must start one position before)
      sty  patt
                                ; set pattern address
      lda  #<(pat-1)
      sta  point
      lda  #>(pat-1)
      sta  point+1

      lda  (point),y            ; read current note/duration
      bmi  voice2_note
      bpl  newNote

voice2_note:
      ldx  #157
      stx  $d400+7
      ldx  #69
      stx  $d401+7

      ;ldx  #$08
      ;stx  $D403+7             ; wave high

      ldx  #$81
      stx  $D404+7              ; control voice 2
      stx  $D404+14             ; control voice 3 (random number)

      ;ldx  #$00                ; we can skip this
      ;stx  $D405+7             ; AD is already 0

      ldx  #$b8 ;e8
      stx  $D406+7              ; SR voice 2

newNote:
      and  #$0F                 ; take only low nibble
      beq  skipNew
      tay

      lda  #00                  ; reset gate
      sta  $D404
                                ; put right frequency
      lda  freqLo-1,y
      sta  $D400                ; low voice 1 frequency
      lda  freqHi-1,y
      sta  $D401                ; high voice 1 frequency
      sta  $D401+14             ; put even some frequency for the random number

      lda  #$41
      sta  $D404                ; voice 1 control
      lda  #$76
      sta  $D405                ; AD voice 1
      sta  $D402                ; wave low
      lda  #$9B
      sta  $D406                ; SR voice 1
skipNew:
```

```
      ldx  #1
      jsr  fakeSyncro

pos:
      lda  #$21
      sta  $D404


      ;ldx  #196                ; skip this for saving bytes: values is not so different
      ;stx  $d400+7
      ldx  #9
      stx  $d401+7              ; change only high frequency of voice 2

      ;lda  #$41
      ;sta  $D404+7

      ldx  #2
      jsr  fakeSyncro

      ;ldx  #113                ; skip this for saving bytes: values is not so different
      ;stx  $d400+7
      ldx  #2
      stx  $d401+7              ; change only high frequency of voice 2
      ;lda  #$40
      ;sta  $D404+7


      ldx  #RDELAY
      jsr  fakeSyncro

      lda  #$09                 ; voice 2 hard restart
      sta  $D404+7


      jmp  music

;================================
; A IRQ like syncronization
; we use a call to KERNAL delay
; for leave the (low) raster 0
;================================
fakeSyncro:
      ;inc 53280                ; this give some very good effect
      lda  $D012
      bne  fakeSyncro           ; sync with 0 of D012 (can come double in a frame)
      jsr  $eeb3                ; some cycle of delay for skipping d012 to 0
      dex
      bne  fakeSyncro
      rts


freqLo:
  .byte 10        ; G3
  .byte 162       ; A3
  .byte 103       ; C4
  .byte 137       ; D4
  .byte 237       ; E4
  .byte 20        ; G4


freqHi:
  .byte 13        ; G3
  .byte 14        ; A3
  .byte 17        ; C4
  .byte 19        ; D4
  .byte 21        ; E4
  .byte 26        ; G4


; patterns
; $80=drum for voice 2
; low byte=note for voice 1
pat:
  .byte $80+E4, 0, D4, $80, $80+G3, 0, $80+E4, 0
  .byte $80+D4, 0, G3, $80, $80+E4, 0, $80+D4, 0
  .byte $80+E4, 0, D4, $80, $80+A3, 0, $80+E4, 0
  .byte $80+D4, 0, $80+A3, $80, $80+E4, 0, $80+D4, $80

  .byte $80+E4, 0, C4, $80, $80+A3, 0, $80+E4, 0
  .byte $80+C4, 0, A3, $80, $80+E4, 0, $80+C4, 0
  .byte $80+E4, 0, C4, $80, $80+G3, 0, $80+E4, 0
  .byte $80+C4, 0, $80+G3, $80, $80+E4, 0, $80+C4, $80
```

Crue Gurl Freestyle

This is the 256 bytes version of the tune by “A Life in Hell”. Read the previous SIDin number for the review of the 512 bytes version, as this is the same base player.

As in the previous number, the code colored in blue is the one not used into the tune (the player has some compilation flag for adding/removing some features).

I just give you some hint about the code:

- Interrupt is set by loading the tune in position $326 where is the IRQ routine called by the Kernal, however, then IRQ is disable with SEI and code is synchronized with raster line $80
- Some of the data about the tune is copied to zeropage and this allow to read it with 8bits address. This is a good technique for saving space.
- The rest of the player is about as the 512b version, but here it is used a random way for generating the notes for save space.

The code:

```
; Tiny Player v0.5
; Player by A Life in Hell
; Additional optimizations by Jockstrap and Sorex
; Music by A Life in Hell
; (c) 2004-2005, Warriors of the Wasteland

; chords -1
; wave -1   -- both of these are to save memory, since 0 is silence!

sid_v0_freq_lo              = $d400         ; voice 0 frequency LO
sid_v0_freq_hi              = $d401         ; voice 0 frequency HI
sid_v0_pwidth_lo            = $d402         ; voice 0 pulse width LO
sid_v0_pwidth_hi            = $d403         ; voice 0 pulse width HI (only bits
sid_v0_ctrl                 = $d404         ; voice 0 control register
sid_v0_ad                   = $d405         ; voice 0 attack / decay
sid_v0_sr                   = $d406         ; voice 0 sustain / release
sid_ctrl                    = $d418         ; general control register

susFrames = 4

exe = 1
lotsOfZpage = 1
useTranspose = 0
tranposeUp = 0
defaultSpeed = 5
gateEnd = 3
speedEor = 8
useSpeedEor = 1
stupidlyCompact = 1
rlines = 0
defaultSr = $e9
defaultWave = $21
accurateChords = 0

noFilter =  1
filterSweep = 0
filterReset = 0

usePwm = 0
useRealPulse = 0
pwmMin = 3
pwmMax = 6
pwmSpeed = 3            ; must be (n^2)-1!!!!

useRestart = 0
useOrderList = 0
doGateReset = 1
useRandomNotes = 1
randomNotesAnd = 7

chordEor = 1
chordEorByte = 1
chordEorVal = 2

.if lotsOfZpage = 0
        zp1=$fe
        zp2=$fc
        chordPtr=$fb
```

```
.else
        tempZp1=$fe
        tempZp2=$fc
        tempZp3=$fa
        curDur          .symbol force8
        zp1             .symbol force8
        zp2             .symbol force8
        chordPtr        .symbol force8
        pos     .symbol force8
        durTable        .symbol force8
        orderPos        .symbol force8
        fltr            .symbol force8
        wave            .symbol force8
        chords          .symbol force8
.endif

.if exe = 1
        ;* = $7ae7
        ; thanks to steve judd's xip for the tip on starting smally :)
        ; unfortunatly, it tends to generate files which are
        ; larger than the original for this size - i've never had it
        ; generate smaller... tho often the same size, indicating that
        ; you'll win by the depack routine size at 1k :-p
        *=$326
        .word entry             ;BSOUT vector
        .byte $ed,$f6           ;STOP vector
entry
        ;sei
.else
        * = $1000
        jmp initPlayer
        jmp playPlayerAll
.endif

initPlayer
.if stupidlyCompact = 0
        lda #0
        ldy #23+128
yloop1
        sta $d400-128,y
        dey
        bmi yloop1
.endif

        ; setup filter
.if noFilter = 0
        lda #$1f
        sta sid_ctrl
.if filterReset = 0
        lda #$f4
        sta $d417
.endif
        lda #$40
        sta $d416
.else
        .if stupidlyCompact = 0
                lda #$0f
                sta $d418
        .endif
.endif
        ; set channel #1 adsr now!
.if stupidlyCompact = 0 || useRestart = 0
        lda #defaultSr
        sta sid_v0_sr
        sta sid_v0_sr+14
        sta sid_v0_sr+7
.endif
        ; this should always be on if you're not using PWM, i guess...
        lda #$8
        sta sid_v0_pwidth_hi+7
.if usePwm = 0
        .if defaultWave = $41
                sta sid_v0_pwidth_hi
                sta sid_v0_pwidth_hi+7
        .endif
.endif
.if lotsOfZpage = 0
        lsr
        sta zp2
        lda #<pat1
        sta zp1
        lda #>pat1
        sta zp1+1
.else
        ldx #(dataEnd-dataStart)+1
zpcloop lda dataStart-1,x
        sta $01,x
        dex
        bne zpcloop
```

```
.endif

.if exe = 0
        ; return from init!
        rts

playPlayerAll
        tax
.else
playPlayerAll
.if rlines = 1
        inc $d020
.endif
        ldx #0
.endif
.if useOrderList = 1
        lda #<orderList1
        sta olsm+1
.else
        stx patsm+1
.endif
        jsr playPlayer

.if filterSweep = 1
        inc fltr
        lda fltr
        sta $d416
.endif

.if usePwm = 1
        ; do pulse width now
        ; why now?  why not?  just not
        ; at start, so we can save one
        ; byte with tax :)
        inc zp2+1
        lda zp2+1
        and #pwmSpeed
        bne nopulseinc
        ldx zp2
inxbit  inx
        cpx #pwmMax
.if useRealPulse = 1
        beq plsdown
        cpx #pwmMin                     ; possible saving: just reset pulse in
        beq plsup               ; plsdown instead of flipping direction!
.else
        bne ddd
        ldx #pwmMin
.endif
ddd     stx sid_v0_pwidth_hi
        stx sid_v0_pwidth_hi+14
        stx zp2
nopulseinc
.endif

        ; now finish the channels!
        ldx #7
.if useOrderList = 1
        lda #<orderList2
        sta olsm+1
.else
        lda #<(pat2-pat1)
        sta patsm+1
.endif
        jsr playPlayer

        ldx #14
.if noFilter = 1 & stupidlyCompact = 1
        ; save two bytes by doing this each frame... heh!
        stx sid_ctrl
.endif
.if useOrderList = 1
        lda #<orderList3
        sta olsm+1
.else
        lda #<(pat3-pat1)
        sta patsm+1
.endif
.if exe = 1
        ; don't fall through on exe - instead loopy!
        jsr playPlayer
.if rlines = 1
        dec $d020
.endif
        lda #80
        cmp $d012
        bne *-3
        beq playPlayerAll
.endif
```

```
playPlayer
        ; check if we need to play a new note
        ldy pos,x
        lda durTable,x
        beq newNote

.if useRestart = 1
        ; to turn off hard restart on channel #1,
        ; uncomment this!
        ;cpx #0
        ;beq norestartchan

        ; update channel #1
        cmp #2
        beq restart
        bcc restart             ;branch if a==1
.endif
        cpx #7
.if doGateReset = 0
        bne owt2
.else
        beq notC1upd
norestartchan
        cmp #3
        bne noGateReset
        ;bne owt2
        lda #((defaultWave)&$fe)
        sta sid_v0_ctrl,x
noGateReset
        ;jmp owt2
        ; not restarting - ensure filter is set!
.if filterReset = 1
        cpx #0
        bne nofiltr4
        lda #$f4
        sta $d417
nofiltr4
.endif
        bne owt2                ;branch always
.endif
notC1upd
.if accurateChords = 1
        cmp #3
        bne ng2
        lda #$fe
        sta gater+1
ng2
.endif
        ldy chordPtr
        lda chords-1,y
        bne allGoodC
        lda (chords-1)+1,y
        tay
        lda chords-1,y
allGoodC
.if useTranspose = 1
        sty tempZp3
        jsr transpose
        ldy tempZp3
.endif
        sta sid_v0_freq_hi,x
        lda wave-1,y
.if accurateChords = 1
gater   and #$ff
.endif
        sta sid_v0_ctrl,x
        iny
        sty chordPtr
owt2
        ; otherwise, update the player and retrun
        dec durTable,x
        rts

.if useRealPulse = 1
        ; flip pulse off
plsup
        lda #$e8
        sta inxbit
        bne ddd

plsdown
        lda #$ca
        sta inxbit
        bne ddd
.endif

.if useRestart = 1
restart2
```

```
        ;lda (zp1),y
        ;beq owt2
        ;lda #$ff
        sta sid_v0_freq_hi,x
        lda #defaultSr
        sta sid_v0_sr,x
.if filterReset = 1
        cpx #0
        bne nofiltr3
        lda #$0
        sta $d417
nofiltr3
.endif
        lda #$81

savemore
        sta sid_v0_ctrl,x
        bne owt2                ;branch always

restart                         ;if durTable==2, then carry is set, if durTable==1, then carry is cleared
        lda (zp1),y
        beq owt2
        lda #$ff
        bcc restart2
        ;sta sid_v0_sr,x
        lda #$08                ;Set to different (non-zero) values to get various restart types.
        ;sta sid_v0_ctrl,x
        ;lda #$ff

        bne savemore            ;branch always
.endif

newNote
        ; get current byte
        lda (zp1),y

        beq out

        cmp #$ff
        bne valid
.if useOrderList = 1
        ldy orderPos,x
        iny
xxx     sty orderPos,x
olsm    lda orderList1,y
        bpl noreset
        ldy #0
        beq xxx                 ;branch always
noreset
        sta pos,x
        tay
        bpl newNote             ;branch always
.else
        ; this gets executed either twice or three times for one in every
        ; eight loops
        .if chordEor = 1
                lda chords+chordEorByte
                eor #chordEorVal
                sta chords+chordEorByte
        .endif
patsm   lda #0
        sta pos,x
        tay
        bpl newNote
.endif
valid
        cpx #7
        bne notChannel2
        ; channel 2 is the hard one, actually!
.if accurateChords = 1
        tay
        lda chords-1,y
        sta sid_v0_freq_hi,x
        lda wave-1,y
        sta sid_v0_ctrl,x
        iny
        sty chordPtr
        lda #$ff
        sta gater+1
.else
        sta chordPtr
.endif
        bne out                 ;branch always

notChannel2
.if useTranspose = 1
        php
        jsr transpose
        plp
```

```
.endif
        bcc c1out               ;branch if x<7
        pha
.if useTranspose = 0
        and #$f0
.else
        asl
        asl
        asl
        asl
.endif
        sta sid_v0_freq_lo,x    ; freq low
        pla
.if useTranspose = 0
        and #$0f
.else
        lsr
        lsr
        lsr
        lsr
.endif
c1out
        sta sid_v0_freq_hi,x
.if useRandomNotes = 1
        lda $d41b
        and #randomNotesAnd
        sta pos
        ;inc pos,x
.endif
        lda #defaultWave
        sta sid_v0_ctrl,x
out
        lda curDur,x
        sta durTable,x
.if useSpeedEor = 1
        eor #speedEor
        sta curDur,x
.endif
;.if useRandomNotes = 0
        inc pos,x
;.endif
        rts

.if useTranspose = 1
transpose
        sta tempZp1
        lda #0
        sta tempZp2
        sta tempZp2+1
.if tranposeUp = 1
        ; magic number :)  137 adds :")
        ldy #136                ; actually 135.6112760779898
.else
        ldy #242                ; actually 241.6
.endif
transposeLoop
        lda tempZp2
        clc
        adc tempZp1
        sta tempZp2
        lda tempZp2+1
        adc #0
        sta tempZp2+1
        dey
        bne transposeLoop
.if tranposeUp = 1
        ; take the high byte and
        ; shift right for eight bits!
        asl
.endif
        rts
.endif


.if useOrderList = 1
orderList1
        .byte pat1-pat1
        .byte pat1-pat1
        .byte pat5-pat1
        .byte pat5-pat1
        .byte pat7-pat1
        .byte pat7-pat1
        .byte pat7-pat1
        .byte pat7-pat1
        .byte pat6-pat1
        .byte pat6-pat1
        .byte pat6-pat1
        .byte pat6-pat1
        .byte $ff
```

```
orderList2
        .byte pat2-pat1
        .byte pat2-pat1
        .byte pat4-pat1
        .byte pat4-pat1
        .byte $ff

orderList3
        .byte pat3-pat1
        .byte $ff
.endif


pat1
.if useOrderList = 0
        .byte $00, $0f, $14, $22, $24, $28, $1e, $18, $ff
.else
        .byte $28, $00, $14, $28, $14, $24, $22, $14, $28, $00, $0f, $14, $28, $00, $0f, $14, $ff
.endif
pat2
.if useOrderList = 0
        .byte $0b, $00, $01, $01, $0b, $06, $01, $01, $ff
.else
        .byte $14, $00, $01, $01, $14, $06, $01, $01
        .byte $14, $00, $14, $01, $14, $06, $01, $06, $ff
.endif
pat3
.if useOrderList = 0
        .byte $05, $00, $05, $0a, $05, $e8, $05, $68, $ff
.else
.if useTranspose = 1
        .byte $50, $00, $50, $a0, $50, $8e, $50, $86, $50, $77, $50, $00, $6a, $00, $6a, $00, $ff
.else
        .byte $05, $00, $05, $0a, $05, $e8, $05, $68, $05, $77, $05, $00, $a6, $00, $a6, $00, $ff
.endif
.endif

.if useOrderList = 1
pat4
        .byte $12, $00, $0e, $0e, $12, $06, $0e, $0e
        .byte $12, $00, $12, $0e, $12, $06, $0e, $0e, $ff
pat5
        .byte $14, $24, $22, $24, $22, $00, $00, $14, $28, $00, $0f, $14, $0f, $14, $1e, $1e, $ff
pat6
        .byte $0f, $14, $16, $18, $14, $1e, $1b, $1e, $1b, $0f, $1b, $0d, $1b, $0b, $1b, $0a, $ff
pat7
        .byte $28, $1e, $18, $14, $00, $16, $18, $00, $1b, $1e, $1b, $1e, $1b, $1e, $1b, $28, $ff
.endif

dataStart
; why sepecated?  beacuse we can move all of this up to put the patterns into
; zpage
.if lotsOfZpage = 1
        eop=*
        *=$02
        .offs eop-$02
wave
        .byte $21, $21, $21, $00, $01   ; 6
        .byte $ff, $41, $40, $80, $40, $80
.if useOrderList = 1
        ; repeat from last ins
        .byte $00, $0b
        .byte $21, $21, $21, $00, $0e   ; 6
        .byte $81, $41, $41, $41, $11, $08, $00, $19
.endif
chords
        .byte $28, $2f, $3c, $00, $01   ; 6
        .byte $81, $0b, $0b, $b5, $0a, $ff
.if useOrderList = 1
        ; repeat from last ins - chords is actually used
        ; for looping, so you shouldn't really do this.  but
        ; because i wrote the player, i know what's going to happen - for
        ; everyone else, i just can't recommend this.  use less notes or
        ; something :).
        .byte $00, $0b
        .byte $28, $2d, $3c, $00, $0e  ; $11
        .byte $ff, $08, $06, $03, $09, $09, $00, $19
.endif

.endif


.if lotsOfZpage = 0
curDur  .byte defaultSpeed
pos     .byte pat1-pat1
durTable        .byte $00
orderPos .byte 0
fltr    .byte $40
free2   .byte 0
```

```
free3   .byte 0
        .byte defaultSpeed
        .byte pat2-pat1
        .byte 0
        .byte 0
        .byte 0
        .byte 0
        .byte 0
        .byte defaultSpeed,pat3-pat1,0,0 ;,0,0,0
.else
fltr    .byte $40
curDur  .byte defaultSpeed
pos     .byte pat1-pat1
durTable        .byte $00
orderPos .byte 0
zp1     .byte <pat1
zp1hi   .byte >pat1
chordPtr        .byte 0

        .byte defaultSpeed              ; curdur2
        .byte pat2-pat1 ; pos2
        .byte 0         ; durTable2
        .byte 0         ; orderpos2
zp2     .byte 4         ; zp2
        .byte 0         ; zp2hi
        .byte 0         ; chordPtrHi
        .byte defaultSpeed,pat3-pat1,0,0 ;,0,0,0
zpageLen=*-$02
        *=eop+zpageLen
.offs 0
.endif

dataEnd
```

128 Byte Blues

As you can see by reading the note that is within this tune (that I copy below), freakyDNA has try to make a tune that read his notes from the C64 memory and so has a very hight compact ratio (the code is only 128 bytes) and that it is influenced by user actions.

But here the description:

```
When I first had a look at the 512b/1k Tiny SID compo, I thought it'd be
a great way to get back into doing some assembly language on the C=64
(which I haven't done for a long time).  I spent a fair amount of time
thinking of the best way to reduce the size and decided that one of the
best ways would be to really challenge myself and to squeeze some music
out of 128 bytes just for fun.

I really wanted to feature the nature of the C=64, so I thought about
several ways to best store the note and instrument data.  After a few
sketches, I found that I seemed to be wasting a lot of space trying to
store both the note and duration data, so I decided that the best way
would be to derrive the notes and durations directly from the C=64 memory
and just index into a basic blues scale and generate some music.  It took
a bit of playing around with techniques, but I found that utilizing
individual bits to step up or down within the scale generated the most
musical results.  Somewhat of a 1-bit DPCM technique except for successive
music notes in a scale instead of successive samples.

So, the code basically cycles through the bits which are displayed in the
screen memory and uses them to generate the notes.  The note duration is
set by combining the jiffy clock and the cursor countdown to give some
flex to the tempo.  I found that staying in one note sequence was boring,
so I made it step through two blues scales.  I tried to squeeze out more
space by removing the low byte of the frequency but found that it either
made the music too out of tune or too high in pitch.  I also attempted to
add a second voice, but ran out of space.

I really liked the idea of adding something to watch while it's playing, so
the colours change as it is reading through the bytes of screen memory.
Rolling the bits of screen memory had the additional benefit that the bytes
would return to their original form after passing over them eight times
when the sequence repeats.  Since the tempo is partially derrived from the
cursor blink, it is possible to change the tempo while moving the cursor.
The sequence of notes can also be modified by typing in the upper portion
of the screen.
```

But now it's time to comment the code:

- The IRQ is set with the BASIC poke instruction like we see in SIDin #7 that is very compact for having the IRQ located at $0831
- The two blues scales are defined into 4 frequencies tables *freq_table_xx* (low/high for each scale)
- Uses triangular waveform and set only Sustain/Release of note

I find this tune very interesting for 128 bytes and I think that maybe using all the space allowed it will be possible to made this kind of player to play other beautiful sound.

```
; Cross-compiled using ACME assembler and Relaunch64
;
; April 9, 2005
; "128 Byte Blues" SID - freakyDNA
;
; Notes:
; - try moving cursor around to change tempo
; - type letters at top of screen to change notes
;
;------------------------------------------------------------------------
;          _/_/_/_/_/  _/_/_/_/    _/_/_/_/    _/_/_/    _/_/  _/_/  _/_/  _/_/
;         _/_/        _/_/  _/_/  _/_/      _/_/  _/_/  _/_/  _/_/  _/_/  _/_/
;        _/_/_/_/    _/_/_/_/    _/_/_/_/  _/_/_/_/_/  _/_/_/_/    _/_/  _/_/
;       _/_/        _/_/  _/_/  _/_/      _/_/  _/_/  _/_/  _/_/      _/_/
;      _/_/        _/_/  _/_/  _/_/_/_/  _/_/  _/_/  _/_/  _/_/      _/_/
;
;                      _/_/_/_/    _/_/    _/_/    _/_/_/
;                     _/_/  _/_/  _/_/_/  _/_/  _/_/  _/_/
;                    _/_/  _/_/  _/_/  _/_/_/  _/_/_/_/_/
;                   _/_/  _/_/  _/_/  _/_/_/  _/_/  _/_/
;                  _/_/_/_/    _/_/    _/_/  _/_/  _/_/
;
;
;------------------------------------------------------------------------
;                w w w . f r e a k y d n a . c o m
;------------------------------------------------------------------------

; list of defines for SID
!src "../../acme/ACME_Lib/sid/sid.a"
; compile to
!to "fdna128b.prg"

note_count = $02                        ; $0
note_val = $fd
cursor_blink = $cd              ; 1-14

note_table = $400               ; index into screen space
colour_mem = $D800              ; colour memory

jiffy = $a2                     ; 1/60 counter

debug_flag = 0                  ; no debuggin'

              * = $0801

              !word $080c
              !byte $05, $00
              !byte $97
              !tx "789,8"                       ; POKE 789,8 ;set interrupt to $0831
              !byte $0,$0,$0                            ; as seen in xxlarge in SIDin #7

taggit:       !tx "fDNA"

freq_table_hi:
              !8 16,   19, 22, 25, 29,   33,  39,  44
freq_table_lo:
              !8 195, 239, 96, 30, 223, 135, 223, 193

;  |      64     |      C-4      |     4291    |      16     |     195     |
;  |      67     |      D#-4     |     5103    |      19     |     239     |
;  |      69     |      F-4      |     5728    |      22     |      96     |
;  |      71     |      G-4      |     6430    |      25     |      30     |
;  |      74     |      A#-4     |     7647    |      29     |     223     |
;  |      80     |      C-5      |     8583    |      33     |     135     |
;  |      83     |      D#-5     |    10207    |      39     |     223     |
;  |      85     |      F-5      |    11457    |      44     |     193     |

freq_table_hi2:
                !8  22,  26,  29,  31,  33,  44,  59,  53
freq_table_lo2:
           !8  96, 156, 223, 165, 135, 193,  190
; top note is a bit out of tune, but allows for tag above
;  |      69     |      F-4      |     5728    |      22     |      96     |
;  |      72     |      G#-4     |     6812    |      26     |     156     |
;  |      74     |      A#-4     |     7647    |      29     |     223     |
;  |      75     |      B-4      |     8101    |      31     |     165     |
;  |      80     |      C-5      |     8583    |      33     |     135     |
;  |      85     |      F-5      |    11457    |      44     |     193     |
;  |      90     |      A#-5     |    15294    |      59     |     190     |

; start is at $0831
mus_irq:
!if debug_flag=1 { inc $d020 }
                lda #$1F
                sta SID_MODE_VOL                ; set volume to max and use LP filter

                lda jiffy                       ; get 1/60 tick
                and cursor_blink                ; combine with cursor countdown for tempo modulation
                bne exit
```

```
add_lda:
                lda note_count

                tax
                inc note_count

rotate_note:
                clc                             ; clear any garbage in carry reg
                lda note_table, x               ; get the current note from the table
                rol                             ; rotate left, carry reg top bit
                adc #0                  ; add carry to low bit
                sta note_table, x               ; restore rotated note
                sta colour_mem, x               ; change colours too

                and #01                 ; check if low bit is one
                bne note_up                     ; go up or down in pitch based on rotated low bit

note_down:
                dec note_val            ; one-bit note index down
                bvc do_note                     ; save a byte over jmp

note_up:
                inc note_val
do_note:
                lda note_val
                bpl do_key1                     ; change key if halfway through sequence
                and #07
                eor #16
                bvc do_freq
do_key1:
                and #07                 ; index into second freq table
do_freq:
                tax
                lda freq_table_hi, x    ; stuff low and high freq values
                sta SID_V1_F_HI
                lda freq_table_lo, x
                sta SID_V1_F_LO

set_adsr:
                ldy #$F1                        ; loud with quick release
                sty SID_V1_S_R

turn_on_note:
                ldy #$15                        ; set wave and turn on note
                sty SID_V1_C_REG

exit:
!if debug_flag=1 { dec $d020 }
              jmp $ea31                 ; exit interrupt
```

Imperial March

This is a cover of Star Wars theme done by Tapio Viitanen. Here I present a reverse engineering source code for following better the player.

- The player use the SEI instruction to disable the interrupt. Synchronization is so done using raster line $64.
- The table sidTable is used for putting all the sid values to registers each time (and at beginning it contains the initial values).
- Voice 1 and 2 use the same duration table (length) and notes table (note). A note is packed with:
    - high nibble = note value of voice 1
    - low nibble = note value of voice 2
- Voice 3 uses a unique table (pat3) of values:
    - high nibble = note value of voice 3
    - low nibble = note duration of voice 3
- All notes are decoded using the common decodeFreq routine.
- On voice 3 it is made an effect that affect the control value to use for the voice each time the pattern is over. This made the tune more various like you can heart.

The code:

```
  DUR   = $50
  DUR3  = $51
  IPAT  = $52
  IPAT3 = $53

 .org $0801
  .byte $0B, $08
  .byte $00, $00
  .byte $9E, $32, $30
  .byte $36, $31
  .byte $00
  .byte $00
  .byte $00

 .org $080D
      sei
      lda  #$00
      sta  IPAT
      sta  IPAT3
      sta  DUR3

loopExt:
      jsr  sync

      ldx  #$18
loopIT
      lda  sidTable,x
      sta  $D400,x              ; Voice 1: Frequency control (lo byte)
      dex
      bpl  loopIT

      dec  DUR                  ; decrement note duration
      bne  goLoop
readFromPat:
      ldx  IPAT                 ; load the pattern index
      lda  length,x
      bne  noReset

      sta  IPAT                 ; reset the pattern index
      beq  readFromPat
noReset:
      sta  DUR                  ; store note length
      ldy  #$00                 ; voice 1
      lda  note,x               ; read packet notes
      pha
      and  #$0F
      jsr  decodeFreq
      ldy  #$07                 ; voice 2
      pla
      lsr
      lsr
```

```
      lsr
      lsr
      jsr  decodeFreq
      dec  DUR3                  ; decrement duration 3
      bpl  incITAP
readPat3:
      ldx  IPAT3                 ; pattern index voice 3
      lda  pat3,x
      bne  decodePat3

      sta  IPAT3                 ; pattern index voice 3
      rol  ctrl3
      lda  ctrl3
      rol
      ora  #$01
      sta  ctrl3

      lda  pos+1
      asl
      rol  pos+1

      clc
      bcc  readPat3
decodePat3:
      and  #$0F
      sta  DUR3                  ; store duration voice 3
      ldy  #$0E
      lda  pat3,x
      lsr
      lsr
      lsr
      lsr
      jsr  decodeFreq
      inc  IPAT3                 ; increment pattern 3
incITAP:
      inc  IPAT                  ; increment the pattern index
goLoop:
      jmp  loopExt
;0884
      rts

sync:
      lda  #$64
wait:
      cmp  $D012                 ; Reading/Writing IRQ balance value
      bne  wait
      rts

;=================================
; Decode frequency of note
;=================================
decodeFreq:
      beq  skip
      tax
      lda  loFreq,x
      sta  sidTable,y            ; low of frequency

      lda  hiFreq,x
      sta  sidTable+1,y          ; high of frequency

pos:
      lda  #$10                  ; gate off
      sta  $D404,y               ; Voice 1: Control registers
skip:
      rts

; notes
; high nibble = voice 2
; low  nibble = voice 1
note:
  .byte $14, $14
  .byte $14, $14
  .byte $14, $14
  .byte $14, $14
  .byte $14, $23
  .byte $23, $23
  .byte $23, $23
  .byte $23

length:
  .byte $20, $10
  .byte $06, $06
  .byte $06, $10
  .byte $06, $06
  .byte $06, $06
  .byte $06, $06
  .byte $06, $06
  .byte $06, $00
```

```
; high nibble = note voice 3
; low  nibble = duration voice 3
pat3:
  .byte $0F, $0D
  .byte $40, $43
  .byte $43, $23
  .byte $51, $40
  .byte $21, $51
  .byte $49, $60
  .byte $63, $63
  .byte $73, $41
  .byte $30, $21
  .byte $51, $49
  .byte $00

loFreq:
  .byte $00
  .byte $F6, $07
  .byte $A1, $F9
  .byte $82, $EB
  .byte $0E

hiFreq:
  .byte $00
  .byte $11, $13
  .byte $16, $17
  .byte $1C, $23
  ,byte $26

sidTable:
  .byte $00        ; freq low of voice 1
  .byte $00        ; freq high of voice 1
  .byte $10        ; wave low byte voice 1
  .byte $08        ; wave high byte voice 1
  .byte $21        ; control reg of voice 1
  .byte $0A        ; attack/decay voice 1
  .byte $20        ; sustain/release voice 1

  .byte $00        ; freq low of voice 2
  .byte $00        ; freq high of voice 2
  .byte $20        ; wave low byte voice 2
  .byte $05        ; wave high byte voice 2
  .byte $11        ; control reg of voice 2
  .byte $0A        ; attack/decay voice 2
  .byte $50        ; sustain/release voice 2

  .byte $00        ; freq low of voice 3
  .byte $00        ; freq high of voice 3
  .byte $40        ; wave low byte voice 3
  .byte $02        ; wave high byte voice 3
ctrl3:
  .byte $11        ; control reg of voice 3
  .byte $2A        ; attack/decay voice 3
  .byte $00        ; sustain/release voice 3

  .byte $00        ; filter freq low
  .byte $00        ; filter freq high
  .byte $00        ; no filter
  .byte $0F        ; volume
```

New Kid On The Block

This tune is made by Frantic and is a co-winner of 256 byte category. Even if instruments are very simple in their definitions (only $11 and $21 waveform used), this tune is however one of the winner, so it is great made.

Here a little analysis, even if code is simple to understand.

Instruments are made with 3 values:

Value	Description
Attack/Decay	Attack/Decay to use. Sustain/Release is not used (0)
Control	Control value to use for sid
Duration	How long is the duration of one note in pattern for this instrument. A pattern is S*8 tick long, so this allow to use an instrument to change values at each tick (if S*1 is used).

This trick of use duration inside the instruments is very interesting, as with one instruction you set the duration of note and the instrument to use. However, this will require to define more instruments that you will need, but one instruments cost you only 3 bytes.

A pattern is made by:

Value	Description
00 jump	A (relative) jump to another pattern section
xx	The high nibble is the instrument to use (and so duration), the low nibble is the note to play

The above information is all you need for understanding the engine, however let we see some in detail points:

- Interrupt routine is initialized automatically as the program is loaded into $326, and so the first word sets the interrupt vector that Kernal IRQ will call automatically.
- A filter is used in melody voice: note that the frequency is varied and it is used a BASIC call to do it. This save lot of instructions (and this made the length of tune to be: 13*3*4*128*0,02=399,36 seconds).
- The undocumented instruction sbx is used into the engine for decrementing the x register by 7

Here the code:

```
; Title:  "NewKidOnTheBlock"
; Chip:   6581 C64 used when composing.
; Author: Frantic/Hack'n'Trade

;---------------------------------
; CONSTANTS
;
DEBUG                           = 0

TICKCOUNTERS                    = $02 ;Use zp for tickcounters.. TICKCOUNTERS+0, TICKCOUNTERS+7, TICKCOUNTERS+14
will be used.
DATAPOS                         = $03 ;Use zp for SONGPOSITIONS... 0, 7, 14..
FILTERHI                        = $23
NUMBEROFVOICES                  = 3   ;Using all three Voices
```

```
S                               = $10 ;Song "speed"

;--------------------------------
; CODE
;
                * = $326

        .word start   ;Four byte init code. Shamelessly ripped from Alihs entry, who ripped it from Steven
        .word $f6ed   ;Judd... :) Sibce Alih "ripped" this I guess I could just as well use it too.???
                      ;I hope that won't break any competition rules. Ninjas method, used in xxxlarge was
                         ;nice too, but still some bytes larger even if one includes the extra code here
                         ;needed for raster sync..
start:  sei

                #if DEBUG
                lda #<$ce00     ;NMI interrupt pointer to Retro Replay debugstub by Groepaz.
                sta $0318       ;RR.EXE is a great tool and it allows me to get rid of using
                lda #>$ce00     ;VICE altogether when coding. Why aren't more ppl using this???
                sta $0319
                #endif

                ldy #$02     ;Set it to $02 so the first incorrectly timed iteration won't affect anything
                sty FILTERHI ;Init filtersweep to make it a bit more deterministic.
                ;Init counters and sonpositions
                lda #(NUMBEROFVOICES-1)*7       ;Voice indeX
@initlp:tax
                sta DATAPOS,x           ;Init datapos+0 to 0, datapos+7 to 7 and datapos+14 to 14.
                sty TICKCOUNTERS,x      ;y can be anything.. doesn't matter, but now I happen to
                                        ;init the filter so set it to $01 too
                sec
                sbc #7
                bpl @initlp

;--
; Main player loop
@outerloop:
@wrast:         cpx $d012       ;After the loop, x is a negative number, thus well above $3e
                                ;(or whatever the critical raster value is again..)
                bne @wrast      ;First iteration won't be correct though, but since the counters
                                ;are set to 2 initially it doesn't matter.

                #if DEBUG
                lda #1
                sta $d020
                sta $d021
                #endif

                jsr $b5ff       ;hijack some BASIC ROM shit to do the filter sweep
                                ;b5ff looks like this:
                                ;       inc $23
                                ;       ldx $23
                                ;       ldy #$00
                                ;       rts
                stx $d416       ;filter hi

                ldx #(NUMBEROFVOICES-1)*7       ;Voice indeX
                ;Right here is a good place for Voice specific code, if there is any reason for that.
@innerloop:
                dec TICKCOUNTERS,x
                bne @loopend

                ;Time for new sound settings, turn gate and oscillator off..
                lda #8
                sta $d404,x

                ;Turn on global volume to make sure we'll hear anything at all.
                ;Reason for having this code inside the loop:
                ; To make sure the player won't run too fast and get executed
                ; twice on the same rasterline.
                lda #$5f        ;Hi-pass + Lo-pass filter on.
                sta $d418
                lda #$a2        ;Filter used on middle (melody) voice.
                sta $d417

                ;Parse data sequence data
                ldy DATAPOS,x
        @newseq:
                iny
                lda @musicdata-1,y      ;Get databyte
                bne @nonewseq
                lda @musicdata-0,y      ;Get jumpval if it's jumptime
                tay                                     ;..and use new one instead
                bpl @newseq                     ;This means data may not be larger than $80 bytes
        @nonewseq:
                sty DATAPOS,x
                pha
                and #$0f                ;Note value
                tay
                lda @freqhi,y
```

```
                sta $d401,x             ;Freq hi
                lda @freqlo,y
                sta $d400,x             ;Freq lo
                pla
                lsr
                lsr
                lsr
                lsr
                tay
                lda adtab,y
                sta $d405,x
                lda durtab,y
                sta TICKCOUNTERS,x
                lda ctrltab,y
                sta $d404,x

@loopend:
                lda #$ff
                .byte $cb,7             ;axs #7 / sbx #7 / whatever..
                bpl @innerloop

                #if DEBUG
                dec $d020
                dec $d021
                jmp @outerloop
                #endif

                bmi @outerloop

;---------------------------------
; "Instruments"
;
; Using AD only to save space. (SR is set to 00 as default)
;
; A pattern is S*8 ticks long, so using "instrument" 5 we
; can represent a whole empty pattern by just one byte in
; the sequence data. At the same time, this format allows
; for changes to the waveform every tick, which means we
; can also make drums and such things. But, not in this
; tune. Perhaps in the next one..
;
;                       00   01   02   03   04   05   06
@adtab:         .byte $1c, $1b, $cd, $2b, $1a, $00, $ad
@ctrltab:       .byte $11, $11, $21, $21, $21, $00, $21
@durtab:        .byte S*1, S*3, S*6, S*2, S*1, S*8, S*8

;---------------------------------
; Sequence data
;
; Voc0 starts at 0
; Voc1 starts at 7
; Voc2 starts at 14
;
; Note and duration stored in one byte and the byte following a $00 (JP)
; byte is interpreted as the destination of a jump to another place in
; the data.
;
@musicdata:
@Voc0start:
        .byte $00 | G4
        .byte $00 | Az4
        .byte $00 | C5
        .byte $00 | D5
        .byte $00 | Dz5
        .byte JP
        .byte <(@Voc0komp-@musicdata)
@Voc1start:
        .byte $20 | D5
        .byte $30 | Dz5

        .byte $20 | D5
        .byte $40 | A4
        .byte $40 | Az4

        .byte JP
        .byte <(@Voc1melody-@musicdata)
@Voc2start:
        ;-
        .byte $50 | 0
        .byte $50 | 0
        .byte $50 | 0
        .byte $50 | 0

        ;-
        .byte $50 | 0
@Voc2loop:
        .byte $50 | 0
        .byte $50 | 0
        .byte $20 | C5
        .byte $40 | Dz4
```

```
        .byte $40 | F4
        ;-
        .byte $50 | 0
        .byte $60 | D5
        .byte $20 | Dz5
        .byte $30 | Az4
        .byte $20 | A4
        .byte $30 | F4

        ;-
        .byte $60 | G4
        .byte $50 | 0
        .byte $50 | 0
        .byte $60 | C5

        ;-
        .byte $50 | 0
        .byte $60 | D5
        .byte $60 | Dz5
        .byte $60 | F5

        ;-
        .byte $60 | G5
        .byte JP
        .byte <(@Voc2loop-@musicdata)
@Voc0komp:
        .byte $10 | G5

        .byte $00 | D4
        .byte $00 | Fz4
        .byte $00 | A4
        .byte $00 | Az4
        .byte $00 | C5
        .byte $10 | D5

        .byte $00 | Dz4
        .byte $00 | F4
        .byte $00 | G4
        .byte $00 | Az4
        .byte $00 | C5
        .byte $10 | Dz5

        .byte $00 | F4
        .byte $00 | A4
        .byte $00 | C5
        .byte $00 | D5
        .byte $00 | Dz5
        .byte $10 | F5

        .byte JP
        .byte <(@Voc0start-@musicdata)

@Voc1melody:
        .byte $20 | C5
        .byte $30 | Az4

        .byte $20 | A4
        .byte $30 | F4

        ;-
        .byte $60 | G4

        .byte $50 | 0

        .byte $20 | F4
        .byte $30 | Dz4

        .byte $20 | F4
        .byte $30 | Dz4

        ;-
        .byte $20 | D4
        .byte $30 | Dz4

        .byte $20 | D4
        .byte $30 | Fz4

        .byte $20 | G4
        .byte $30 | Dz4

        .byte $00 | C4
        .byte $00 | Dz4
        .byte $00 | F4
        .byte $00 | G4
        .byte $00 | A4
        .byte $00 | C5
        .byte $00 | G5
        .byte $00 | F5

        .byte JP
```

```
        .byte <(@Voc1start-@musicdata)
;--------------------------------
;Note freq data
;
; Only using needed notes.
                JP      =  0
                C4      =  1
                D4      =  2
                Dz4     =  3
                ;E4     =  2;Not used
                F4      =  4
                Fz4     =  5
                G4      =  6
                ;Gz4=  6;Not used
                A4      =  7
                Az4     =  8
                ;B4     =  8;Not used
                C5      =  9
                ;Cz5=  9;Not used
                D5      = 10
                Dz5     = 11
                ;E5     = 11;Not used
                F5      = 12
                ;Fz5= 13;Not used
                G5      = 13
                ;Gz5    ;Not used
                ;A5             ;Not used
                ;Az5= 14;Not used
@freqlo = *-1
;               .byte $0c,$1c,$2d,$3e,$51,$66
;               .byte $7b,$91,$a9,$c3,$dd,$fa
;               .byte $18,$38,$5a,$7d,$a3,$cc
;               .byte $f6,$23,$53,$86,$bb,$f4
;               .byte $30,$70,$b4,$fb,$47,$98
;               .byte $ed,$47,$a7,$0c,
                .byte $77;,$e9

                .byte $61,$e1;,$68,
                .byte $f7,$8f,$30
                ;.byte $da,
                .byte $8f,$4e;,$18,
                .byte $ef;,$d2
                .byte $c3,$c3;,$d1,
                .byte $ef;,$1f,
                .byte $60
                ;.byte $b5,$1e,
                ;.byte $9c;,$31,$df,$a5

;               .byte $87,$86,$a2,$df,$3e,$c1
;               .byte $6b,$3c,$39,$63,$be,$4b
;               .byte $0f,$0c,$45,$bf,$7d,$83
;               .byte $d6,$79,$73,$c7,$7c,$97
;               .byte $1e,$18,$8b,$7e,$fa,$06
;               .byte $ac,$f3,$e6,$8f,$f8,$2e

@freqhi:
;               .byte $01,$01,$01,$01,$01,$01
;               .byte $01,$01,$01,$01,$01,$01
;               .byte $02,$02,$02,$02,$02,$02
;               .byte $02,$03,$03,$03,$03,$03
;               .byte $04,$04,$04,$04,$05,$05
;               .byte $05,$06,$06,$07,
                .byte $00 ;WRAP

                .byte $07;,$07

                .byte $08,$08;,$09,
                .byte $09,$0a,$0b
                ;.byte $0b,
                .byte $0c,$0d;,$0e,
                .byte $0e;,$0f
                .byte $10,$11;,$12,
                .byte $13;,$15,
                .byte $16
                ;.byte $17,$19,
                ;.byte $1a;,$1c,$1d,$1f

;               .byte $21,$23,$25,$27,$2a,$2c
;               .byte $2f,$32,$35,$38,$3b,$3f
;               .byte $43,$47,$4b,$4f,$54,$59
;               .byte $5e,$64,$6a,$70,$77,$7e
;               .byte $86,$8e,$96,$9f,$a8,$b3
;               .byte $bd,$c8,$d4,$e1,$ee,$fd
```

Repeat Me

This is the 256b tune by Laxity.

In this case you will see that there is a unique type of instrument hardcore into the code. The instrument uses $21 as waveform, it has a sort of hardrestart at the end of note and it uses a fixed $55 value for Attack/Decay and Sustain/Release.

The pattern for note/duration is defined according with this table:

Value	*Description*
Positive number	The lower nibble is the base note to play, high nibble is the octave to use. 00 is for rest
Negative number	The low 7bits are the duration of note
$FF	Repeat the pattern

The engine so memorized only 12 notes frequency and the others are calculated at runtime.

The method used for initializing the IRQ is to simple disable it (SEI instruction) and to have a loop at raster line $55 for proper time passed calculation.

Here the code:

```
;--------------------------------------
;TinyPlayer 02.g0
;By Laxity of Vibrants/Maniacs of Noise
;--------------------------------------
;Coded on 13th of April 2005
;--------------------------------------
         ; Player variables
;--------------------------------------
zp       = $fb

clrbeg   = $38

seqpoi   = clrbeg+3
cnt      = seqpoi+3
dur      = cnt+3
note     = dur+3

spdcnt   = note+3
clrend   = spdcnt+1
;--------------------------------------
         *= $07fd
;--------------------------------------
         jmp start
;--------------------------------------
         .byte 0
;--------------------------------------
         .byte $0b,$08,$00,$00,$9e
         .text "2061"
         .byte $00,$00,$00
;--------------------------------------
start    sei
         lda #$0f
         sta $d418
;--------------------------------------
init     ldx #clrend-clrbeg
         lda #0

il01     sta clrbeg,x
         dex
         bpl il01
;--------------------------------------
waitsl   ; A is $55 (except for first
         ; frame which doesn't matter)
```

```
        cmp $d012
        bne waitsl
;--------------------------------------
        dec spdcnt
        bpl noreset
        lda #8
        sta spdcnt
;--------------------------------------
noreset
        ldx #$02
nexttrk
        lda spdcnt
        bne updsnd

        dec cnt,x
        bpl updsnd

        ldy seqpoi,x
next
        lda s0,y

        bpl isnote
        cmp #$ff
        bne nowrap
        ldy sofs,x
        bpl next     ;bpl should work
nowrap
        and #$7f
        sta dur,x
next2   iny
        bpl next     ;bpl should work
isnote
        sta note,x
        lda dur,x
        sta cnt,x
        iny
        tya
        sta seqpoi,x
;--------------------------------------
updsnd
        ldy #$20
        lda cnt,x
        beq gateoff
        lda note,x
        beq gateoff
        iny
gateoff
        tya
        ldy voice,x
        sta $d404,y

        lda note,x
        pha
        and #$0f
        tay
        lda frqlo,y
        sta zp
        lda frqhi,y
        sta zp+1
        pla
        lsr a
        lsr a
        clc
        adc spdcnt
        lsr a
        lsr a
        tay
        dey
        dey
        bmi nooct
shift
        lsr zp+1
        ror zp
        dey
        bpl shift
nooct
        ldy voice,x
        lda zp
        sta $d400,y
        lda zp+1
        sta $d401,y
        lda #$55
        sta $d405,y
        sta $d406,y
;--------------------------------------
        dex
        bpl nexttrk
;--------------------------------------
```

```
        jmp waits1
;---------------------------------------
voice   .byte 0,7,14
;---------------------------------------
sofs    .byte s1-s0,s2-s0,s3-s0
;---------------------------------------
s0      .byte $ff

s1      .byte $83
        .byte $40,$30,$43,$33
        .byte $40,$30,$43,$47,$ff

s2      .byte $81
        .byte $00,$27,$26,$00
        .byte $00,$28,$27,$00
        .byte $00,$27,$26,$20
        .byte $22,$23,$22,$00
        .byte $ff

s3      .byte $bf,$00
        .byte $83
        .byte $00,$12,$10,$1a
        .byte $87,$17,$00
        .byte $83
        .byte $00,$1a,$18,$17
        .byte $87,$13,$00
        .byte $83
        .byte $12,$13,$10,$17
        .byte $12,$13,$27,$18
        .byte $8f,$17,$00
        .byte $ff
;---------------------------------------
frqlo   .byte $a0,$b7,$20,$bc,$ac,$e4
        .byte $70,$4c,$84,$18,$10,$70
frqhi   .byte $45,$49,$4e,$52,$57,$5c
        .byte $62,$68,$6e,$75,$7c,$83
;---------------------------------------
```

Electronic

This is the 256 bytes tune by Aleksi Eeben. Here I present a reverse engineering source code. But now some comments to the code:

- It relocates itself in page 0 with the same technique of Agemixer
- Interrupt is disable with SEI instruction and synchronization is done with raster line $81
- It uses a table of values for all sid registers (even for initialize it)
- It uses lot of self modified code for storing and then using index values
- It uses 7 notes (tables loFreq and hiFreq)
- Voice 1 is rectangular with a duty cycled that is increase each time
- Noise is putted into voice 1, 2 and 3 according to certain rules
- Cut off frequency of thief pass filter in voice 3 is done using voice 3 output
- It has the tables for note and duration (note/dur)

However, as you can see, the code is more articulated and the description of all his behaviors will need more time for sure!

The pseudo-code:

```
  .org $0801
  .byte $0B, $08
;0803
  .byte $00, $37, $9E, $32
  .byte $30, $35, $39, $00
  .byte $A2, $00
                              ; 14080 sys 2059
  .org $080D
      sei
      lda  $0803,x
      sta  $00,x
      inx
      bne  $080E
      jmp  $0084

   .org $0016
sync:
      ldy  #$81
wait:
      cpy  $D012              ; Reading/Writing IRQ balance value
      bne  wait

      lda  $D41C              ; Generator output
      lsr
      adc  #$20
      sta  sidTable4+1        ; filter freq high
      lda  sidTable+2         ; wave low byte voice 1
      adc  #$0C
      sta  sidTable+2         ; wave low byte voice 1
      bcc  skipWH
      inc  sidTable+3         ; wave high byte voice 1
skipWH:
      lda  sidTable2+1        ; freq high of voice 2
      eor  #$40
      sta  sidTable2+1        ; freq high of voice 2
      lda  IndW+1             ; index of note to play
      clc
      ldx  IndZ+1
      adc  tmp-1,x
      cmp  #$07
      bcc  skipSub
      sbc  #$07
skipSub:
      tax
      lda  loFreq,x
      asl
      sta  sidTable           ; low freq voice 1
      lda  hiFreq,x
      rol
      sta  sidTable+1         ; high freq voice 1

      ldx  #$18
loopST
      lda  sidTable,x
      sta  $D400,x            ; Voice 1: Frequency control (lo byte)
```

```
      dex
      bpl  loopST

      dec  IndZ+1
IndZ:
      ldx  #$06
      bne  goSync

      lda  #$06
      sta  IndZ+1
      asl  sidTable2+2            ; low wave voice 2
      bcc  skipNote
IndW:
      ldx  #$00                   ; index of note to play
      lda  loFreq,x
      sta  sidTable3              ; low freq voice 3
      lda  hiFreq,x
      sta  sidTable3+1            ; high freq voice 3
      sty  $D412                  ; Voice 3: Control registers
      inc  sidTable2+2            ; low wave voice 2
skipNote:
      ldx  IndA+1
      cpx  #$06
      bne  skipC1
      sty  $D404                  ; Voice 1: Control registers
skipC1:
      cpx  #$04
      bne  skipC2
      sty  $D40B                  ; Voice 2: Control registers
skipC2:
      dec  IndA+1
IndA:
      ldx  #$01
      bne  goSync

      lda  #$08
      sta  IndA+1
      dec  IndB+1
IndB:
      lda  #$01
      bne  goSync

      inc  IndC+1
IndC:
      ldx  #$FF
      lda  note,x
      sta  IndW+1                 ; index of note to play
      bne  notZero
      sta  IndC+1
      inc  sidTable+4             ; control reg of voice 1
      bne  IndC
notZero:
      lda  pulse,x
      sta  sidTable2+2            ; low wave voice 2
      lda  dur,x
      sta  IndB+1
goSync:
      jmp  sync

sidTable:
  .byte $00        ; freq low of voice 1
  .byte $00        ; freq high of voice 1
  .byte $00        ; wave low byte voice 1
  .byte $04        ; wave high byte voice 1
  .byte $40        ; control reg of voice 1
  .byte $08        ; attack/decay voice 1
  .byte $18        ; sustain/release voice 1

sidTable2:
  .byte $00        ; freq low of voice 2
  .byte $1A        ; freq high of voice 2
  .byte $00        ; wave low byte voice 2
  .byte $00        ; wave high byte voice 2
  .byte $80        ; control reg of voice 2
  .byte $04        ; attack/decay voice 2
  .byte $04        ; sustain/release voice 2

sidTable3:
  .byte $00        ; freq low of voice 3
  .byte $00        ; freq high of voice 3
  .byte $00        ; wave low byte voice 3
  .byte $00        ; wave high byte voice 3
  .byte $20        ; control reg of voice 3
  .byte $06        ; attack/decay voice 3
  .byte $06        ; sustain/release voice 3

sidTable4:
  .byte $00        ; filter freq low
```

```
  .byte $00         ; filter freq high
  .byte $F4         ; resonance + filter in voice 3
  .byte $1F         ; volume + tief pass filter

loFreq:
  .byte $9B, $0C, $8B, $D0, $67, $10, $CE

hiFreq:
  .byte $03, $04, $04, $04, $05, $06, $06

pulse:
  .byte $92, $90, $92, $94, $92
  .byte $90, $A4, $A4, $92, $AA
  .byte $92

note:
  .byte $05, $03, $01, $04, $05
  .byte $03, $01, $06, $05, $02
  .byte $01

tmp:
  .byte $00, $00, $04, $04

dur:
  .byte $02, $02, $02, $02, $02
  .byte $02, $02, $02, $06, $02
  .byte $08
```

Repetitive Tune BASIC

This tune made in BASIC was found by Peter Weighill in some very old stuffs. The code were all BASIC and don't fit in 256 bytes. However with some management (like removing data statement), the code was restricted to the right size.

```
1 m=54272:poke m+24,15:poke m+5,9:poke m+6,15
2 p=2226
3 h=peek(p):l=peek(p+1):d=peek(p+2):p=p+3:if d=0then 5
4 poke m+1,h:poke m,l:poke m+4,33:for t=1to d:next :poke m+4,32:
  for t=1to 300:next :goto 3
5 c=c+1:if c=1then c=-1:goto 2
6 goto 3
```

This is the code in basic that is located from $0801 to $08B1. It uses m and p pointers: m points to the sid registers, while p points to the note data of the song:

- In line 1, the sid 1 voice is initialized.
- In line 3 three bytes are read from current position in memory: high/low note frequency and note duration. If duration is 0, line 5 is reached.
- In line 4 the note to play is performed letting the gate on for a time that is polled by a cycle based onto the note duration. After the gate is made off for a fixed amount of time.
- In line 5 is coded that the tune is restarted when we reached duration 0 for the second time.

```
.org 2226
 .byte $03, $f4, $18
 .byte $04, $b4, $18
 .byte $05, $47, $90
 .byte $04, $b4, $c0
 .byte $03, $f4, $18
 .byte $04, $b4, $18
 .byte $05, $47, $90
 .byte $04, $b4, $60
 .byte $04, $b4, $60
 .byte $00, $00, $00

 .byte $05, $47, $18
 .byte $06, $47, $18
 .byte $07, $0c, $90
 .byte $06, $47, $c0
 .byte $05, $47, $18
 .byte $06, $47, $18
 .byte $07, $0c, $90
 .byte $06, $47, $60
 .byte $06, $47, $60
 .byte $00, $00, $00
```

Even if this tune uses only one voice, you can see what is needed for programming sound using BASIC program, and so the polling like technique to simulate note duration.

Splatform256

This is the tune of Splatform minigame, but Steve Judd had rewrite the code for fitting into 256 bytes.

Let we see some points of the player:

- The player loads itself in position that already set the IRQ vector. However, then the IRQ is disable with SEI and it synchronized with raster line $FF.
- With a loop it copies some values for initialize voice 1 and 2 and some zeropage variables. Voice 3 is never used.
- It copies (and splits a byte in his nibbles) notes in zero page. This allow to use only 8 bits for acceding to the notes.
- The code is self modified in two points: one for made voice 1 to play a noise fixed note (drum) when duration of note is over, and one for loading a new note address when a 0 note is reached. The trick is to use a BIT instruction that contains inside the LDY or LDA instruction.
- The “drill” sound effect in voice 2 is made using note $0e and making a ramp of sound followed by a fixed note.

Now the reverse engineering code:

```
   .org $0326

  .byte $2A, $03
  .byte $ED, $F6

  .org $032A
      sei
      ldy  #$0D
      sty  $D418                  ; Select volume and filter mode
loopCopy:
      lda  Values,y
      sta  $D402,y                ; Voice 1: Wave form pulsation amplitude (lo byte)
      sta  $D409,y                ; Voice 2: Wave form pulsation amplitude (lo byte)
      sta  $00AB,y
      dey
      bpl  loopCopy

      ldx  #$00
      stx  $78
decode:
      iny
      lda  decript,y
      pha
      lsr
      lsr
      lsr
      lsr
      jsr  dCopy
      pla
      and  #$0F
      jsr  dCopy
      cpy  #$21
      bcc  decode

sync:
      lda  #$FF
wait:
      cmp  $D012                  ; Reading/Writing IRQ balance value
      bne  wait

      ldx  #$00                   ; use voice 1
      jsr  getNote
      sty  $B4                    ; cur pattern pointer voice 1
      lda  $B0                    ; actual duration voice 1
      bne  useRect

      ldx  #$0C                    ; note to play
      lda  #$81                    ; noise waveform
      .byte $2C    ;bit  $41A9
useRect:
      lda  #$41                    ; rectangular waveform
```

```
      ldy  #$00                   ; voice 1
      jsr  outNote

      ldx  #$01                   ; use voice 2
      jsr  getNote
      sty  $B5                    ; cur pattern pointer voice 2

      ldy  $B8
      iny
      iny
      sty  $D40A                  ; Voice 2: Wave form pulsation amplitude (hi byte)
      sty  $B8

      lda  #$41                   ; rectangular waveform
      ldy  #$0C
      cpx  #$0E                   ; note for special effect
      bcc  skipDrill
                                  ; make the "drill" effect into the tune
      ldx  $8F                    ; temp note
      inx                         ; inc temp note
      cpx  #$0D                   ; max to check
      bcc  skipFixed
      ldx  #$0A                   ; fixed note
skipFixed:
      stx  $8F                    ; temp note
      ldy  #$60
      lda  #$0C
      cmp  $B1                     ; actual duration voice 2
      adc  #$04
skipDrill:
      sty  $B3                     ; note duration voice 2
      ldy  #$07                    ; voice 2
      jsr  outNote
      bcc  sync

dCopy:
      sta  $22,x
      sta  $34,x
      cpy  #$19
      bcs  skip02
      sta  $02,x
skip02:
      inx
      rts

;==================================
; Get the next note
; in x=voice to use (0/1)
; out x=note
; out y=next index to note
;==================================
getNote:
      ldy  $B4,x                  ; cur pattern pointer
      inc  $B0,x                  ; actual duration
      lda  $B0,x
      sec
      sbc  $B2,x                  ; note duration
      bne  readNote
      sta  $B0,x                  ; clear actual duration
      iny                         ; and go to read next note

      .byte $2C    ; bit  $B6B4
isZero:
      ldy  $B6,x

readNote:
      lda  $0002,y
      beq  isZero
      tax                         ; note to play
      rts

;==================================
; Out the note:
; y=voice offset
; a=cntr of voice
; x= note to play
;==================================
outNote:
      sta  $D404,y                ; Voice 1: Control registers
      lda  loFreq-1,x
      sta  $D400,y                ; Voice 1: Frequency control (lo byte)
      lda  hiFreq-1,x
      sta  $D401,y                ; Voice 1: Frequency control (hi byte)
      rts

Values:
; copied from $AB
  .byte $00
```

```
  .byte $07
  .byte $41
  .byte $31
  .byte $FB

  .byte $0B    ; actual note duration voice 1
  .byte $0B    ; actual note duration voice 2
  .byte $0C    ; note duration voice 1
  .byte $0C    ; note duration voice 2
  .byte $0F    ; cur pattern pointer voice 1
  .byte $75    ; cur pattern pointer voice 2
  .byte $00
  .byte $12

loFreq:
  .byte $00
  .byte $F4, $30
  .byte $47, $61
  .byte $8F, $C3
  .byte $1F, $87
  .byte $3E, $3C
  .byte $0F

hiFreq:
  .byte $00
  .byte $03, $04
  .byte $06, $08
  .byte $0C, $10
  .byte $15, $21
  .byte $2A, $32
  .byte $43

; decriptied and copied to $22, $34 and $02
decript:
  .byte $31, $13, $11, $41
  .byte $31, $13, $11, $21
  .byte $00, $9A, $19, $B1
  .byte $9C, $19, $B1, $A1
  .byte $91, $51, $65, $16
  .byte $71, $65, $76, $56
  .byte $57, $E8

; decripted and copied to $22 and $34
  .byte $76, $B7, $98, $BE
  .byte $87, $67, $56, $51
```

Conclusion

Well, this is all about the previous year compo. But now it's time to think to the new one that is being running as soon as you read this chapter.

You can now choose from a 256 bytes, 512 bytes and even 1KB, so maybe you will find your right size for competing.

Catweasel Mk4 (follow)

by Stefano Tognon <ice00@libero.it>

Some times is passed and some progress were made to make the card sound good in my system, as if you remember from last article, the produced sound by the card was very horrible and it depends by what you are doing with other applications in your system.

However, I had decided to experiment with the card by myself even with the possibility to damage the sid chip in case I had a card with the DC-DC problem as I have not jet get an answer from Individual Computer.

Driver

If you remember for Linux system, Simon White had made two hardsid kernel driver module: Head and Experimental:

- Head did not use the hardware buffer of the card (classic manage of the card)
- Experimental: use the hardware buffer of the card (this is the best driver to use)

We will speck about the hardware buffer later in more details, however in the near future Simon will merge the two codes for having a unique driver that can use the two system together.

As I had *sidplay2* console that play the tune at hyper speed, while the (patched) *Vice* played it at right speed, I try to investigate why the driver gives these two different behaviors.

The funny things was that when I modify *libsidplay2* to put a debug string at every commands it passed to the kernel driver, and so this make *sidplay2* to eat all the cpu power, then the card starts to sound good (well, good as now, that it is not so good, but we will see this later) even if the tune plays slower that the original speed.

The problem seems so that there was an incorrect speed managements into the kernel driver: only if notes are emitted at the right time (due to how *Vice* send commands or with *sidplay2* that are delayed by cpu overworking) sound is played almost correctly.

Looking at the kernel module source I find soon the problem: in the experimental branch Simon temporally disable chip 1 timing and let it be synchronized with chip 0. This was a way to test hardware buffer without having the problem of the mutual synchronization of two chips mounted in the same card.

Maybe was my fault to insert the chip in position 1 and not to position 0, but I liked the position 1 as it was more cooler in my system and even it is more easy to extract from this socket the chip, if needed, using a simple screwdriver.

I so patch the driver to make it uses correct timer even for chip 1 (as I don't have two chips in the same card, and this was easier to be done that swapping the sid chip).

Well, now the sound is stable: *sidplay2* plays at the right speed and (unfortunately for the moment) I didn't have *Vice* 1.17 compiled with the hardsid patch to test it again, due to some compilation errors.

Hardware buffer

Before analyzing how the card sounds now, let we look at the hardware buffer of the card.

If you want that a card with a SID chip play a tune as in your C64 you need:

1. The chip must be clocked at the right PAL/NTSC speed. If you don't have this, internal logic shouldn't be temporized correctly.
2. You must put a sid value in a sid register at the same time the real C64 puts it into the sid.
3. You should have some external capacitors for filters that are like in C64, otherwise the filter (of the same chip) will have different behavior.

Point 1 and 3 is done by the hardware of the the card, but for point 2 it was necessary to sent the command to the card at the right time.

Maybe this is not a problem: we know that actual emulators/sidplayers are good timing, and so they will send the right sid commands at the right moment. However, as an I/O operation is to be done for sending one command to the card, passing thrown the PCI bus, and this will use the Kernel of the operating system, we could expect that if the cpu is overworking and there are lot of I/O operations and kernel activities, we should experiment sound timing problems.

The hardware buffer of the card is so the solution of this problem. Imagine that you run a sidplayer at the max speed and register at each (virtual) clock the commands you had to sent to the sid and put them in a software buffer. Then, as soon as the card is ready, you sent this buffer of commands to the card hardware buffer, and start to produce another buffer to send the next time the card is ready (well, this work even if you send directly the commands to the hardware buffer without storing it -and maybe this is how the driver is implemented).

The card, as soon as it receives the commands, will starts to play the sid commands at the time is specified for each instructions, something like:

```
DELAY   $xxxx
PUT     $yy     TO register $kk
DELAY   $hhhh
PUT     $dd     TO register $ll
```

until the buffer is empty.

This methods will so prevent any sort of timing delay due to your SO or system overworking.

However if your system is overworking over a certain level, there are no hardware buffer that can help you: if the buffer is empty before you give another one, the sound will be broken, but this is however an advantage over the classic system that is more cpu depends.

Sound

At this point you will want to know how the sound play using the hardware buffer to his power after resolving the timing issue of the driver.

Well: good and horrible, depending from the tune!! I'm trying to understand why some tunes play good and other play horrible and I will try to think of the problem.

However the first word to say is that the tune play always the same way, e.g. the sound is ever

the same (horrible or not) and so there is probably something in the card, or the drive, or the chip that made this happen.

If we want to think that the card is working perfectly, just test these tunes (remember that I mount a 6581 chip):

/Dunn_Jonathan/Ocean_Loader_4.sid
/Dunn_Jonathan/Ocean_Loader_5.sid

The tunes are almost 99% equals to the C64 listening. There is sometimes a distortion in some sound (but as I don't have listen to that tunes in the C64 with the chip I mounted in the card, I could accept the sound).

Oh, good, try with Matt, as I know by memories his tune:

/Gray_Matt/Tusker.sid

#1:

```
0:15        Perfectly the initial sound!
0:20-0:24:  Distortion (sound seems to go slow in one voices)
0:30-0:38   Distortion (sound seems to go slow in one voices)
1:35-1:44   Arpeggio is perfect, but the long sound is distorted
other       Could be happy, sound good
5:25-5:30   A little slow in one voice
```

#4:

```
0:30-0:48   The main sound is a little   around frequency   as I remember
1:10 -1:35  Slow in one voice
1:35-2:10   The solo voice is perfect
2:25        Slow in one voice
2:39-2:55   Distorted in frequency
3:23-3:29   Slow in one voice
4:25-4:40   Final voice is perfect
```

Not so good, around 75% of the right tune, but try with
Gray_Matt/Last_Ninja_2.sid

#1:

```
0:00-0:06  Perfectly
0:06- 0:20 Sound goes killed in volume in the voice that start
0:20-0:30  Voices is distorted, volume sometimes go 0 for all voices
0:54-1-08  Volume is right, play correctly
1:08 -1:30 Goes silent, inaudible sounds
1:30 -2:32 Sound good, almost correctly
2-32-3:44  A little distorted and not so synchronized
3:50-4:10  Goes up/down in volume
4-10- 4:28 Low volume from the expected, but correctly
```

I think 40% right, the rest is horrible. However I could assured that the sid chip mounted in this card sound perfectly this tune is the C64 (I always test a C64 with this tune, then I goes to test other interesting tunes).

What is the problem?

At this time I'm wondering if what I listen is related to Sid ADSR bug. I remember that the first tune I wrote to cover Driller using Hubbard sound driver where full of ADSR bug (sound goes killed everywhere in volume). That was as I try to use ADSR values like in Driller using instruments without a proper hardrestart.

Else if I remember correctly, after patching a sidplay2 console some time ago for showing ADSR bug in tunes while playing, the Matt Gray tunes like Driller where near the ADSR bug is some points.

Maybe there is the possibility that the tune I listen almost correctly are the ones that use hardrestart, while the others (maybe the old one) without hardrestart will go in ADRS bug due to a not precise timing from driver/card firmware?

Test

At this point I need more test for understanding what is going wrong with the card.

One of the thing I done was to let *xsidplay* to use hardsid driver, so I could use it for a more precise timing (e.g *sidplay2* displayed clock goes at high speed, then slow down and so on, probably due to the process used to send commands to the card).

The operation was simple, as Simon describe: substitute the *reSidBuilder* definition with the *hardsidBuilder* one in the wrapper used by *xsidplay*. It works perfectly.

The other operations was to compile the *Vice* 1.18 that supports the hardsid (experimental and only in unix). I compile it as usual, as it detects the presence of hardsid driver and adds support to it without adding configuration parameters.

If you start *x64* you will now see the voice Hardsid below Resid in the sid settings.

I so try it and see that now you can move windows, and do other activities into your desktop, and the sound is played and not disturbed as with the last test I made (with driver with chip 0 timing and no chip 1 timing).

Take present that if you click in closing emulator, when the windows with yes/no/cancel appears, the sound do the same things: the last one is played (with Resid, the sound is stopped).

The same thing appears even in *xsidplay* if you play pause (but not stop) or *sidplay2* when you exit. The reason is that if you start again from the pause, the sound must be started from where it was. Maybe it could be more convenient to kill the sound, because having the last note played could be very annoying and maybe when the sound restarts, it could not restart properly in every case.

Now that *x64* goes, it could be possible to speech even about RSID sid with samples in it.

In my system (266Mhz) I have that normally a tune played with *xsidplay* eats 30% with Resid (with the faster option, not the most accurate) and 15% with hardsid. So, hardsid let me save 15% of cpu working.

But if I listen to a sample RSID with *xsidplay*, all the cpu is working in emulation of cpu (here there is not the max optimization) and so sound with Resid is played slowed. Using hardsid, the 15% of cpu save is not sufficient and the sound I listen is totally crap.

With *x64* (that has a different cpu emulation), I could now listen to RSID sid having not the 100% cpu working.

Even if now here I could listen to sample music, there is always the distortion/volume problem in the sid part, and so listen to Arkanoid is quite different and the sound is not so good, even if sam-

ples are played and you can move the window without having a single note delayed.

However the hardware buffer should give the best result with sample based music as here there are lot of volume settings per second for having the sid generating the sample music.

In order to test the card I try again to use Head kernel driver (if you remember the last snapshot freeze my kernel, by now I have an updated system), in this way I will see if bypassing hardware buffer the card sound the same.

The driver manifest soon some problems (well, maybe they were even the first time, but I didn't investigate):

1. It detected my sid chip as 8580. Simon fixed soon this problem. However Head end Experimental used little different approach to determine the sid chip type.
2. It detected a chip even in position 0 that was empty (and with the same type of position 1). This was quite intricate, as I test different values of delay and constant values to use in the part of driver that detect if a socket is empty but with the same result. Simon fixed it after knowing that exists two MK4 version and one like the mine did not have pull ups/downs in the bus.

However at the moment the driver freeze again the kernel, but as Simon is merging his code this is to be fixed soon.

Analysis

Even if I describe the cases very common, like in Last_Ninja_2, where sound volume goes low and there is a distortion, there are some cases where voices are completely missed.

Try an example: /VARIOUS/S-Z/Starlost/Nullone.sid

In this you not heard anything until 0:20 where you listen only a voice at a high volume. So look at a *sid2midi* output of the tune (it is condensed):

```
          Voice 1                    Voice 2                    Voice 3
Time      Note  Freq  PW  WF ADSR VL  Note  Freq  PW  WF ADSR VL  Note  Freq  PW  WF ADSR VL  Filter
=============================================================================================================
00:00.00   ---     0    0 00 0000 --   ---     0    0 00 0000 --   ---     0    0 00 0000 --  L__ 1__    0 f
00:00.00  >B-1<   61 2296 40 0000 --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__    0 f
00:00.01   +++    61 2328 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 3a8 f
00:00.01   +++    61 2360 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 398 f
00:00.02   +++    61 2392 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 388 f
00:00.03   +++    61 2424 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 378 f
00:00.03   +++    61 2456 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 368 f
00:00.04   +++    61 2488 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 358 f
00:00.04   +++    61 2520 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 348 f
00:00.05   +++    61 2552 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 338 f
00:00.06   +++    61 2584 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 328 f
00:00.06   +++    61 2616 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 318 f
00:00.07   +++    61 2648 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 308 f
00:00.07   +++    61 2680 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 2f8 f
00:00.08   +++    61 2712 40 0fef --   ---    16    0 00 0000 --   ---    16    0 00 0000 --  L__ 1__ 2e8 f
```

Voice 1 starts with a low pass filter (with variable frequency) and a very low note frequency. The pulse is modulated (he increases). Attack is the minimum, while decay and release are the maximum.

```
00:10.07  >B-2<  123 2184 40 0fef --  >B-1<   61 2296 40 0000 --   ---    16    0 00 0000 --  L__ 12_ 398 f
00:10.08   +++   123 2200 40 0f0e --   +++    61 2328 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 3a8 f
00:10.09   +++   123 2216 40 0f0e --   +++    61 2360 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 398 f
00:10.09   +++   123 2232 40 0f0e --   +++    61 2392 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 388 f
00:10.10   +++   123 2248 40 0f0e --   +++    61 2424 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 378 f
00:10.10   +++   122 2264 40 0f0e --   +++    61 2456 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 368 f
00:10.11   +++   121 2280 40 0f0e --   +++    61 2488 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 358 f
00:10.12   +++   122 2296 40 0f0e --   +++    61 2520 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 348 f
00:10.12   +++   123 2312 40 0f0e --   +++    61 2552 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 338 f
```

```
00:10.13   +++   124 2328 40 0f0e --   +++    61 2584 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 328 f
00:10.13   +++   125 2344 40 0f0e --   +++    61 2616 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 318 f
00:10.14   +++   124 2360 40 0f0e --   +++    61 2648 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 308 f
00:10.15   +++   123 2376 40 0f0e --   +++    61 2680 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 2f8 f
00:10.15   +++   122 2392 40 0f0e --   +++    61 2712 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 2e8 f
00:10.16   +++   121 2408 40 0f0e --   +++    61 2744 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 2d8 f
00:10.16   +++   121 2408 40 0f0e --   +++    61 2776 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 2c8 f
00:10.17  >F#3<  185 2184 40 0f0e --   +++    61 2808 40 0fef --   ---    16    0 00 0000 --  L__ 12_ 2b8 f
```

Voice 2 follows the same setting of voice 1.

```
00:20.15  >B-2<  123 2184 40 0f0e --  >B-1<   61 2296 40 0fef --  >B-4<  494 2296 40 0000 --  L__ 12_ 398 f
00:20.15   +++   123 2200 40 0f0e --   +++    61 2328 40 0fef --   +++   494 2280 40 0fe0 --  L__ 12_ 3a8 f
00:20.16   +++   123 2216 40 0f0e --   +++    61 2360 40 0fef --   +++   494 2264 40 0fe0 --  L__ 12_ 398 f
00:20.16   +++   123 2232 40 0f0e --   +++    61 2392 40 0fef --   +++   494 2248 40 0fe0 --  L__ 12_ 388 f
00:20.17   +++   123 2248 40 0f0e --   +++    61 2424 40 0fef --   +++   494 2232 40 0fe0 --  L__ 12_ 378 f
00:20.18   +++   122 2264 40 0f0e --   +++    61 2456 40 0fef --   +++   490 2216 40 0fe0 --  L__ 12_ 368 f
00:20.18   +++   121 2280 40 0f0e --   +++    61 2488 40 0fef --   +++   486 2200 40 0fe0 --  L__ 12_ 358 f
00:20.19   +++   122 2296 40 0f0e --   +++    61 2520 40 0fef --   +++   490 2184 40 0fe0 --  L__ 12_ 348 f
00:20.19   +++   123 2312 40 0f0e --   +++    61 2552 40 0fef --   +++   494 2168 40 0fe0 --  L__ 12_ 338 f
00:20.20   +++   124 2328 40 0f0e --   +++    61 2584 40 0fef --   +++   498 2152 40 0fe0 --  L__ 12_ 328 f
00:20.21   +++   125 2344 40 0f0e --   +++    61 2616 40 0fef --   +++   501 2136 40 0fe0 --  L__ 12_ 318 f
00:20.21   +++   124 2360 40 0f0e --   +++    61 2648 40 0fef --   +++   498 2120 40 0fe0 --  L__ 12_ 308 f
00:20.22   +++   123 2376 40 0f0e --   +++    61 2680 40 0fef --   +++   494 2104 40 0fe0 --  L__ 12_ 2f8 f
00:20.22   +++   122 2392 40 0f0e --   +++    61 2712 40 0fef --   +++   490 2088 40 0fe0 --  L__ 12_ 2e8 f
00:20.23   +++   121 2408 40 0f0e --   +++    61 2744 40 0fef --   +++   486 2072 40 0fe0 --  L__ 12_ 2d8 f
00:20.24   +++   121 2408 40 0f0e --   +++    61 2776 40 0fef --   +++   490 2056 40 0fe0 --  L__ 12_ 2c8 f
00:20.24  >F#3<  185 2184 40 0f0e --   +++    61 2808 40 0fef --   +++   494 2040 40 0fe0 --  L__ 12_ 2b8 f
```

Voice 3 starts with a little different value: release is 0, and as Sustain is $E, the volume is high as listen in the player. However filter is not apply in this voice.

Maybe could the missing sound caused by a filter problem? There is only an operation to do: patch Nullone tune to not use filter and listen what changes.

Just change all 17 D4 to 17 E4 in the tune and test again: now I listen voice 1 and 2 correctly. So the problem for this is filter related.

So, now the question is: from what this happen and all problems are due to filter?

Filter

Maybe all sounds distortion come out from filter and so the volume killed for voices?
No, not possible: tunes like of Matt Gray did not make high use of filter, so we must look for other problems into the driver/card/chip.

However, the filter problems should derive from one of this causes:

- Driver did not pilot correctly the sid chip when using the filter. I tend to exclude this possibility
- Chip is broken in filter managing or its frequency working is out of a middle standard sid. As it sounded good in C64 this should be to exclude, but maybe the chip could be damaged by inserting into the card (even if I manage it with lot of carefully). However I did not test lot of tunes that made high use of filter with this chip into the C64 before insert it into the card, so there is the possibility that I test tunes where cutting frequency worked well.
- Filter jumpers in the card are not set correctly: they are inserted as from the card manual so this should be to exclude.
- Capacitors in the card are broken/or not in right capacity? As changing the chip socket we have the same effect, maybe it is to exclude that they are broken together, but there could be the remote possibility that they are not into the right capacity even if I tend to exclude even this.

At this point the right way is maybe to test again the chip into the C64 or inserted another 6581 chip into the card.

But before try this way, it it better to look and try to lean the causes of the other music problems

No Filter

Well, how look like LN II tune? Simple:

```
          Voice 1                        Voice 2                        Voice 3
Time      Note  Freq  PW  WF ADSR VL   Note  Freq  PW  WF ADSR VL   Note  Freq  PW  WF ADSR VL  Filter
=============================================================================================================
00:00.00   ---     0    0 00 0000 --    ---     0    0 00 0000 --    ---  1132    0 00 0000 --  ___ ___   0 0
00:00.00   ---     0    0 00 0000 --    ---     0    0 00 0000 --  >F#4<  377    0 10 0000 --  ___ ___   0 0
00:00.01   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:00.01   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++   755    0 10 00a9 --  ___ ___   0 0
00:00.02   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++   377    0 10 00a9 --  ___ ___   0 0
00:00.03   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:00.03   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++   755    0 10 00a9 --  ___ ___   0 0
00:00.04   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++   377    0 10 00a9 --  ___ ___   0 0
00:00.04   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:00.05   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++   755    0 10 00a9 --  ___ ___   0 0
00:00.06   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++   377    0 10 00a9 --  ___ ___   0 0
00:00.06   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:00.07   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++   755    0 10 00a9 --  ___ ___   0 0
00:00.07   ---     0    0 00 0000 --    ---     0    0 00 0000 --  >Db5<  566    0 10 00a9 --  ___ ___   0 0

00:01.27   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++   566  186 10 00a9 --  ___ ___   0 0
00:01.27   ---     0    0 00 0000 --    ---     0    0 00 0000 --   +++  2264  186 10 00a9 --  ___ ___   0 0
00:01.28   ---     0   16 00 0000 --    ---     0   16 00 0000 --  >F#4<  377    0 10 00a9 --  ___ ___   0 0
00:01.28   ---     0   16 00 0000 --    ---     0   16 00 0000 --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:01.29   ---     0   16 00 0000 --    ---     0   16 00 0000 --   +++   755    0 10 00a9 --  ___ ___   0 0
00:02.00   ---     0   16 00 0000 --    ---     0   16 00 0000 --   +++   377    0 10 00a9 --  ___ ___   0 0

00:03.24   ---     0   16 00 0000 --    ---     0   16 00 0000 --   +++   566  202 10 00a9 --  ___ ___   0 0
00:03.25   ---     0   16 00 0000 --    ---     0   16 00 0000 --   +++  2264  202 10 00a9 --  ___ ___   0 0
00:03.25  >F#2<    94  384 40 0000 --  >F#1<    47  384 40 0000 --  >F#4<  377    0 10 00a9 --  ___ ___   0 0
00:03.26   +++     94  416 40 0f00 --   +++     47  416 40 0f00 --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:03.27   +++     94  448 40 0f00 --   +++     47  448 40 0f00 --   +++   755    0 10 00a9 --  ___ ___   0 0
00:03.27   +++     94  480 40 0f00 --   +++     47  480 40 0f00 --   +++   377    0 10 00a9 --  ___ ___   0 0
00:03.28   +++     94  512 40 0f00 --   +++     47  512 40 0f00 --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:03.28   +++     94  544 40 0f00 --   +++     47  544 40 0f00 --   +++   755    0 10 00a9 --  ___ ___   0 0
00:03.29   +++     94  576 40 0f00 --   +++     47  576 40 0f00 --   +++   377    0 10 00a9 --  ___ ___   0 0
00:04.00   +++     94  608 40 0f00 --   +++     47  608 40 0f00 --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:04.00   +++     94  640 40 0f00 --   +++     47  640 40 0f00 --   +++   755    0 10 00a9 --  ___ ___   0 0
00:04.01   +++     94  672 40 0f00 --   +++     47  672 40 0f00 --   +++   377    0 10 00a9 --  ___ ___   0 0
00:04.01   +++     94  704 40 0f00 --   +++     47  704 40 0f00 --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:04.02   +++     94  736 40 0f00 --   +++     47  736 40 0f00 --   +++   755    0 10 00a9 --  ___ ___   0 0
00:04.03   +++     94  768 40 0f00 --   +++     47  768 40 0f00 --  >Db5<  566    0 10 00a9 --  ___ ___   0 0

00:19.05   +++     74 3008 40 0f00 --   +++     37 3008 40 0f00 --   +++   599  250 10 00a9 --  ___ ___   0 0
00:19.06   +++     74 3040 40 0f00 --   +++     37 3040 40 0f00 --   +++  2398  250 10 00a9 --  ___ ___   0 0
00:19.06  >F#2<    94  384 40 0f00 --  >Bb5<   951  560 40 0f00 --  >F#4<  377    0 10 00a9 --  ___ ___   0 0
00:19.07   +++     94  416 40 0f00 --   +++    951  656 40 008d --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:19.07   +++     94  448 40 0f00 --   +++    951  752 40 008d --   +++   755    0 10 00a9 --  ___ ___   0 0
00:19.08   +++     94  480 40 0f00 --   +++    951  848 40 008d --   +++   377    0 10 00a9 --  ___ ___   0 0
00:19.09   +++     94  512 40 0f00 --   +++    951  944 40 008d --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:19.09   +++     94  544 40 0f00 --   +++    951 1040 40 008d --   +++   755    0 10 00a9 --  ___ ___   0 0
00:19.10   +++     94  576 40 0f00 --   +++    951 1136 40 008d --   +++   377    0 10 00a9 --  ___ ___   0 0
00:19.10   +++     94  608 40 0f00 --   +++    951 1232 40 008d --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:19.11   +++     94  640 40 0f00 --   +++    951 1328 40 008d --   +++   755    0 10 00a9 --  ___ ___   0 0
00:19.12   +++     94  672 40 0f00 --   +++    951 1424 40 008d --   +++   377    0 10 00a9 --  ___ ___   0 0
00:19.12   +++     94  704 40 0f00 --   +++    951 1520 40 008d --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:19.13   +++     94  736 40 0f00 --   +++    947 1616 40 008d --   +++   755    0 10 00a9 --  ___ ___   0 0
00:19.13   +++     94  768 40 0f00 --   +++    942 1712 40 008d --  >Db5<  566    0 10 00a9 --  ___ ___   0 0

00:42.06   +++     94 3200 40 0f00 --   +++     47 3200 40 0f00 --   +++   566  186 10 00a9 --  ___ ___   0 0
00:42.07   +++     94 3232 40 0f00 --   +++     47 3232 40 0f00 --   +++  2264  186 10 00a9 --  ___ ___   0 0
00:42.07  >F#1<    47  384 40 0f00 --   +++     16    0 80 0f00 --  >F#4<  377    0 10 00a9 --  ___ ___   0 0
00:42.08   +++     47  416 40 0f00 --  >A-7< 3611    0 80 00e0 --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:42.09   +++     47  448 40 0f00 --   +++     16    0 80 00e0 --   +++   755    0 10 00a9 --  ___ ___   0 0
00:42.09   +++     47  480 40 0f00 --   +++     16    0 80 00e0 --   +++   377    0 10 00a9 --  ___ ___   0 0
00:42.10   +++     47  512 40 0f00 --   +++     16    0 80 00e0 --   +++  1511    0 10 00a9 --  ___ ___   0 0
00:42.10   +++     47  544 40 0f00 --   +++     16    0 80 00e0 --   +++   755    0 10 00a9 --  ___ ___   0 0
00:42.11   +++     47  576 40 0f00 --   +++     16    0 80 00e0 --   +++   377    0 10 00a9 --  ___ ___   0 0
00:42.12   +++     47  608 40 0f00 --  >A-7< 3611    0 80 00e0 --   +++  1511    0 10 00a9 --  ___ ___   0 0
```

As you see, tune is almost with same instruments settings until 0:19, after it changes the ADSR, of one instrument. However, the sound problem could be heart before this, and so this seems confirm my hypothesis of a sort of ADSR bug due to invalid timing.

In this case *sidplay2* console was better of *xsidplay* for showing the problem as, if you remember the clock goes faster and then slower ans so on.

You can so look at this:

time 0	sidplay time 0
time 5	sidplay time 9
time 6	sidplay time10

This is interesting: I heart the start of the problem as soon as I'm around (real) time 5/6 seconds from the begging, when for the first time *sidplay2* simulated clock is slowing down. However this could be a coincidence.

Now it's time to made Head works. I so resume an old version of Head (from April) that I remember it did not freeze the system (but at that time it not produces sound for the problem of detecting a sid in chip 0 that was empty). But it is not now a problem: I had inserted the sid in position 0.

How it sound now? Essentially the sound is like the hardware buffer's one, with the same problems.

However now I can see the advantage of hardware buffer: playing Arkanoid in X64 with this driver made a 20% use of kernel for I/O operations (it was 1% with hardware buffer). If you move windows in the desktop, now after some while, notes becomes killed.

This last test made some last possibility about the wrong sound to be tested:

- Firmware of the card (or the card itself) did not work properly (and in the same manner while using hardware buffer of not)
- Hardsid driver in the part that dialogs with the kernel driver did not work properly (and so, using of not hardware buffer make not difference)
- Chip was becoming broken after inserting into the card

Another 6581

Even if now I have a 8580 chip to test, I prefer to not damaging it if some things did not work. So I test the old 6581 chip I use the first time (with no sound) in a C128: it sounds correctly.

I test Last Ninja 2, and even if the sound is not so clean (the chip is of 84, while the other one is of 86) I can say that the tune is done in a good manner. So I insert again this chip into the card.

The most difficult task was to extract the chip from the socket in position 0 of the card: you must manage carefully as you have only one point to make force into the chip. However after some minutes the new chip was mounted and the system started.

How it sound now?

Well, thinking that now the sound is a little more disturbed as in the C128, it sound as the previous chip and so:

- Same muting/distortion problem as in Last Ninja 2
- Low filter make mute the voices as the other chip

At this point it is evident that there is an hardware issue regarding filter and maybe a software or hardware related issue about the muting problem.

One thing to say about sound output is that, as sid chip is mono, chip 0 is passed into right sound card canal, while chip 1 is passed into left channel.

This allow you to listen to stereo tune if you have 2 sid chips, but maybe it could be better that you can listen in both channels if you are using one sid chip only.

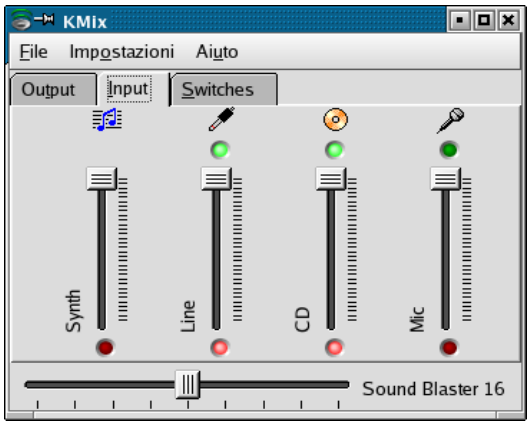
Conclusion

Well at this point there is a major problem in my card.

One my friend with a MK4 with an 8580 say me that Last Ninja 2 is played correctly in his system with *Windows* driver, so this prevent that this is due to a firmware driver problem (we have the same and last one version).

I re-contact Jens and have a replay within 2 hours (well done) and for looking to the problem he require (if possible) to have pictures of card + samples captured from the card.

Maybe the last operation is something that I should describe here as this is not a so common operation that programs done by default in Linux.



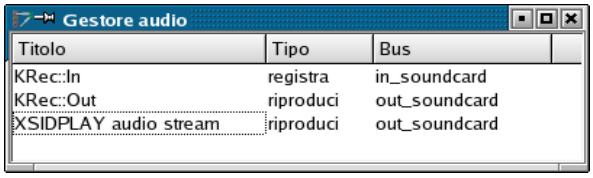
First I have to made the CD line controlled by Alsa driver to capture the sound by:

```
amixer set CD cap
```

You can also use *KMix* and click to the red led in it under CD line to allow this (look at the image in this page).

Then I have set *xsidplay* to use *aRts* sound (instead of OSS or ESD). Then opening *KRec*, you now can record the sample from MK4.

In the Audio Manager you should now see the *xsidplay* audio stream and the v-meter shows the sound from the MK4.



You can see from this image how low is the sound volume level that come out from the sid in MK4.

However sound where captured in wav and then converted into mp3 with *lame* (if one wants to listen to it I can send the mp3 by email).

Now it is time to attend Jens response, but in the meantime I have download the new Simon merged driver called *rt_async*. Even if you can find that the driver changes every days, it is already stable (I have only a driver lookup in sound, but the day after the driver was already corrected).

It is lot better of Head version (it uses asynchronous stream) as it competes with the Experimental version, even if hardware buffer gives always the best with sample music.

SOTDin 9 end