SoftDin 8

“Alessia Merz”

Vice snapshot with Vice palette

**Made with the GIMP from a AM photo
and converted to C64 320x200
Hires Mode Bitmap
by Stefano Tognon
in 2005**

**“Tiny and Tiny”
...**

Free Software Group

SSDin 8
version 1.00
30 July 2005

SIDin Contents

General Index

Editorials

Stefano Tognon <ice00@libero.it>

Hi, again.

This time we go to look inside at four tiny (512 bytes) sid players (they come from the Tiny Sid Compo). In the next number we will see all the other 256 byte players, so for the next year Tiny Sid Compo, you will have many players examples that could give you some idea for a new player.

In the second part, I go to look at the Catweasel Mk4 card that I brought as a present for my birthday some time ago. The card have a very high potential, but as you will see, at the moment it cannot be used for sid music :(

The best new is that Martin Galway had read the article about Arkanoid in SidIn #4 and found it interesting. Thanks Martin.

I will hope that both us will found some time to discuss it further (sorry Martin, but free time is at negative rate in this period :( ) and so we can have some interesting news into next number.

Unfortunately this number is a lot in delay as I want to release it in May, but maybe next number should be available soon...

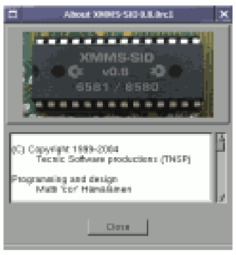
Bye
S.T.

News

Some various news of players, programs, competitions:

- XMMS-SID plugin v0.8.0beta14
- ACID64 v2.0.3
- TFX v2.97
- HVSC 5.9
- XMPlay's SIDPlay2 plugin
- PSID64 0.7
- Tiny Sid Compo
- Goattracker 1.53
- HVMEC 0.4
- Sidplay2w 020605
- HVSC 42
- Goattracker v2.0/2.1
- CGSC v1.13
- Polly Tracker v0.9 beta/v1.0
- TFX version 2.99
- Audio::SID v3.10
- Polly Tracker v1.1
- Goat Tracker 2 Music Compo

XMMS-SID plugin v0.8.0beta14



(C) Copyright 1999-2004
Tecnic Software productions (TNSP)

Programming and design
Matti 'ccr' Hämäläinen

Available from January 2005 the new version of XMMS plugins for playing SID:

- Run-time selectable emulation library.
- 2-8x oversampling support.
- Supports multiple emulation libraries:
  - libSIDPlay v1
  - libSIDPlay v2 with reSID-builder
- Several sub-tune selection/control methods as configurable options.
- Automatic sub-tune changer; plays through all sub-tunes in file, starting from default.
- Configurable filter-settings.
- Supports HVSC Song-length database and has a maximum playtime check.
- Optionally configurable title-string like in MPG123 plugin (supports also XMMS 1.2.5+ generic titles)
- Standard audio output quality settings.
- STIL (SID Tune Information List) support.
- File information dialog <CTRL+3>, shows normal SID-tune info and STIL info if enabled.

Download the source at http://www.tnsp.org/xs-files/xmms-sid-0.8.0beta14.tar.gz

ACID64 v2.0.3

Wilfred Bos has released in January 2005 the new version 2.0.3 of ACID64 player:

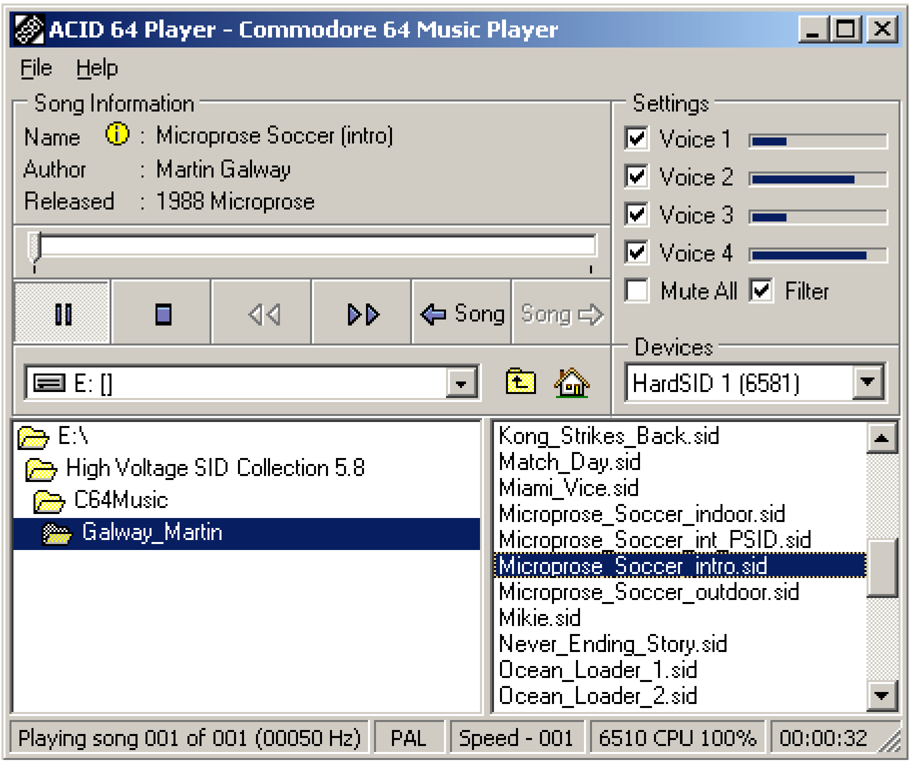
Fixes
- Possible crash at startup on some systems
- File mask was changing after drag and drop
- Changing device right after startup didn't work
- Trackbar position indicator was sometimes not pointing to current track
- Going back or forward through history list could result in an error
- Buttons sometimes stayed raised when going to the next sub-tune
- Pressing CTRL-P and CTRL-N after each other could slow down player

Improvements
- Small emulation improvements
- Directory-up now selects previous directory for easy browsing
- Improved keyboard controlling of directory and file box
- After drag and drop ACID64 gets the focus
- Optimized code

New
- Added STIL support
- Added file property menu

The next version will support the Catweasel MK4

Download the player from http://www.acid64.com/



Name : Microprose Soccer (intro)
Author : Martin Galway
Released : 1988 Microprose

☑ Voice 1
☑ Voice 2
☑ Voice 3
☑ Voice 4
☐ Mute All ☑ Filter

E: []

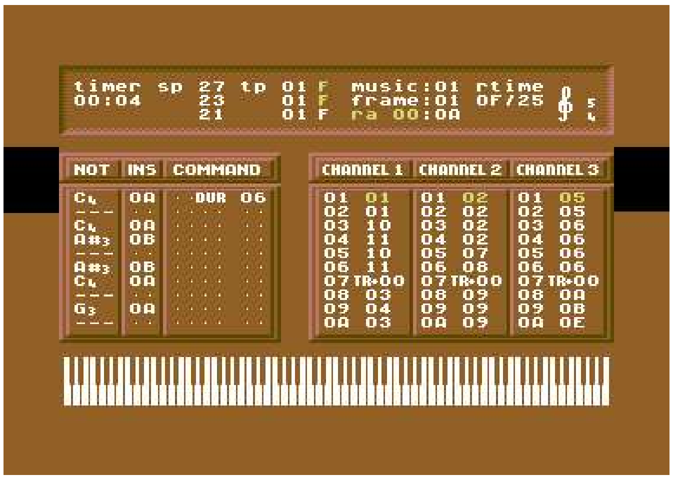
HardSID 1 (6581)

E:\
High Voltage SID Collection 5.8
C64Music
Galway_Martin

Kong_Strikes_Back.sid
Match_Day.sid
Miami_Vice.sid
Microprose_Soccer_indoor.sid
Microprose_Soccer_int_PSID.sid
Microprose_Soccer_intro.sid
Microprose_Soccer_outdoor.sid
Mikie.sid
Never_Ending_Story.sid
Ocean_Loader_1.sid
Ocean_Loader_2.sid

Playing song 001 of 001 (00050 Hz) | PAL | Speed - 001 | 6510 CPU 100% | 00:00:32

TFX v2.97



```
timer  sp 27 tp 01 F  music:01 rtime
00:04     23    01 F  frame:01 0F/25  5
          21    01 F  ra 00:0A        4

NOT  INS  COMMAND      CHANNEL 1  CHANNEL 2  CHANNEL 3
C4   0A   ..DUR  06    01 01      01 02      01 05
---  ..   .....  ..    02 01      02 02      02 05
C4   0A   .....  ..    03 10      03 02      03 06
A#3  0B   .....  ..    04 11      04 02      04 06
---  ..   .....  ..    05 10      05 07      05 06
A#3  0B   .....  ..    06 11      06 08      06 06
C4   0A   .....  ..    07TR+00    07TR+00    07TR+00
---  ..   .....  ..    08 03      08 09      08 0A
G3   0A   .....  ..    09 04      09 09      09 0B
---  ..   .....  ..    0A 03      0A 09      0A 0E
```

On 21 January 2005 Unreal has released the new 2.97 version of TFX music editor.

Along with the usual features of TFX, including a modern player/editor, while actively avoiding having to remember irrelevant hex numbers for commands when a symbolic name will do, multiframe (up to 10x on all channels, and 72x on a single channel) support, the player has undergone major cosmetic surgery for this release. New features since our last release, v2.9, include:

- Less bugs, Better Tasting, More Filling. Most of the bizarre bugs that plagued some of the older versions are now dead.
- Complete English language documentation.
- A new hard restart system, allowing for more flexibility. Hard restart anywhere from 0 to 255 frames, with the sid parameters of your choice.
- Import for songs created using v2.7, v2.8, and v2.9
- An optional PC based external packer and relocator, which recompiles the player with only necessary features.
- Heavily enhanced Pulse and Filter programs, including
    - Pulse programs no longer overshoot as they did in older versions
    - Optional dual pulse/filter tables running simultaneously, modulated together.
    - Variables which are modifiable from within the tracks.
    - Full 11bit precision on filter programs now.
    - Special filter type #0, which turns off the filter
    - Jump while Gate Held.
- Enhanced Wave table programs, include:
    - Hard Note mode, to allow you to specify a note (as opposed to a frequency) directly
    - Variable support as in the Pulse/Filter tables, which can be set not only by number, but also as a symbolic note name (i.e. "var1 c-4"), in order to allow for complex harmonies in arpeggios.
    - Ability to change the speed of the wavetable execution (even to a variable)
    - Jump while Gate Held.
- New commands in the sector editor, including:
    - Nopulse, Nofilter - don't reset pulse on new note trigger
    - Control Override - modify the control bits (Gate, Ring, Sync, Test) from within the song itself.
    - Greater vibrato control from within the song.
    - Variable setting commands, switchable between note input and number input.

TFX v2.97 can be downloaded from http://www.unreal64.net/tfx

HVSC 5.9

On 26 January HVSC 5.9 was released at http://www.c64.hvsc.org

We've got the music compo entries from the following parties in this pack: X'2004, Black Birdie 2004, Datahelg 2004, TUM 2004, SID Compo IV (c64.sk)

Booker, the last active Amorphis member whished for the Amorphis directory to be dissolved, and so we did.

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)

```
DRAX             Ashley Hogg
Ati              Bax
Richard Bayliss  Bzyk
Cadaver          CMP
Compod           Djinn
Dokken           Ed
Fanta            Froyd
Gregfeel         Hein Holt
Heinmuck         Herr Tie (new)
Gerard Hultink   Isildur
JFK              Jaymz Julian
Kosa             Maktone
Mamba            Bekir Ogurlu (Slowhand)
Olsen            PCH
Phobos           Praiser
Raffi            Randy
Rap              Sad
Shogun           Surgeon
TMG              Vegeta
Zeus
```

After this update, the collection should contain 29,055 SID files!

This update features (all approximates):

```
1405 new SIDs
  29 fixed/better rips
   2 fixes of PlaySID/Sidplay1 specific SIDs
   6 repeats/bad rips eliminated
 135 SID credit fixes
 250 SID model/clock infos
  10 tunes from /DEMOS/UNKNOWN/ identified  :-)
  41 tunes moved out of /DEMOS/ to their composers' directories
  17 tunes moved out of /GAMES/ to their composers' directories
```

XMPlay's SIDPlay2 plugin

This is a beta version of a SIDPlay2 plugin for XMPlay plater (www.un4seen.com)

It manages:

- balls-on SID reproduction (libsidplay2+reSID)
- PSID, RSID, MUS/STR support
- SLDB support for PSID's (MD5 fingerprint)
- SLDB support for RSID's (path comparison if verbose)
- STIL and BUGList comments displaying

- STIL for files outside HVSC directory (using the SLDB)
- Favourite Top 100 SIDs rank displaying
- subsong switching
- seeking (painfully slow, though)
- configurable stereo separation
- groovy surround
- fadeout
- archive, DSP and visualisation support via XMPlay plugin system
- tags and other info
- nice configuration

A thread about the plugin is here: http://un4seen.com/forum/?topic=3838.0

Else, an unofficial SLDB for HVSC v5.9 is available here:
http://xthost.info/pieknyman/download.html
or at
http://www.pieknyman.tk

PSID64 0.7

Roland Hermans ha released the new version of PSID64. PSID64 is a program that makes it possible to listen to your favourite SID music on a real Commodore 64 computer.

The download section on http://psid64.sourceforge.net/ provides the new source tarball as well as precompiled binaries for Linux and DOS/Windows.

The precompiled binary for DOS/Windows was compiled using DOSEMU, DR-DOS 7.03 and DJGPP. The Linux RPM binary has been compiled on Fedora Core 3.

- News summary:
- Improved RSID compatibility.
- Use INST/DEL to toggle screen on and off.
- Use + and - keys to select next and previous song respectively.
- Display the song number of song currently being played.
- Show all 31 characters of the PSID header.
- Fixed NMI bank register bug.
- Pass default song number via boot loader.

Tiny Sid Compo

Running from 15 January to 8 May 2005, the compo where only 256 and 512 bytes music were allowed.

```
Pos. Points    Title                         Author
--------------------------------------------------------------------------
 1    186      My Block .. one block         Agemixer
 1    186      New Kid on the block          Frantic
 3    152      Repeat me                     Laxity
 4    151      Electronic                    Aleksi Eeben
 5    145      Splatform256                  Steve Judd
 6    118      Imperial March                Tapio Viitanen
 7    117      Random Ninja                  Stefano Tognon
 8    105      Crue Gurl Freestyle           Jaymz Julian (A Life in Hell)
 9     91      128 Byte Blues                  Freaky DNA
10     75      Repetitive Tune BASIC           Peter Weighill

Pos. Points    Title                           Author
----------------------------------------------------------------------------
 1     206     Empty                           4mat
 2     171     Twone Five                      Laxity
 3     154     15BB                            Stefano Tognon
 4     118     Crue Gurl                       Jaymz Julian (A Life in Hell)
```

More details at http://digilander.iol.it/ice00/tsid/tinysid

Goattracker 1.53

The new version of PC tracker were release on May from http://covertbitops.cjb.net/

New:

- Fixed packer behaviour with no hardrestart/no pulseinit instruments
- Optimized pulseinit routine a few cycles (at cost of one byte)

HVMEC 0.4

The new version of High Voltage Music Engine Collection was released on 2 June 2005.

Now there are 83 editors, 53 trackers and 24 other kinds of music programs.
Added in this version:

Editor:

DMC v4.0y	Future Composer v2.2	Music Construction Set ?
Music Mixer 6	SoundBooster v1.0	Soundmaker IV
TFX 2.97	The Music Shop X	The Music Shop Y
The Music Studio ?	Voicetracker V4.0	Voicetracker V5.0

Tracker:

Digitronix V1	Hardtrack Composer V1.0	Music-Player V2.0
Playstar v2.0	Rockmonitor VI (1)	Soundmaster II

Other:

Cynthcart ?	Digi Music Editor V2	Microdisco ?
Microrhythm ?	Microtuned ?	Microvocals ?
Music Editor 64 ?	Music Maker ?	Pro-Drum V2.0
Sample Mixer V1.2	Sound Maker ?	

Sidplay2w 020506

The new version of windows sid player was released on 2 June 2005.

News in this version:

- ReSID 0.16
- PSID samples variables reset properly

- Emulation improvements fixes a few tunes

Download from http://www.gsldata.se/c64/spw/

HVSC 42

HVSC Update #42 was released in June and can be download from www.hvsc.c64.org

After this update, the collection should contain 30,001 SID files!

This update features (all approximates):

- 943 new SIDs
- 211 fixed/better rips
- 3 fixes of PlaySID/Sidplay1 specific SIDs
- 7 repeats/bad rips eliminated
- 400 SID credit fixes
- 472 SID model/clock infos
- 3 tunes from /DEMOS/UNKNOWN/ identified :-)
- 216 tunes moved out of /DEMOS/ to their composers' directories
- 85 tunes moved out of /GAMES/ to their composers' directories

We've got the music compo entries from the following parties in this pack:
Tiny SID Compo, Breakpoint 2005, Deadline 2007, Floppy 2005, Forever Hex, Maximum Overdose 6.

New features in HVSC v42:

- New version numbering (you guessed it already), which is now consistent with the number of the current Update. This should make it easier to find out which update you need in order to upgrade to a newer HVSC in the future. And believe me, there are a lot more updates yet to come, we haven't by far ripped everything that's out there. :-)

- Complete songlength database now included! Compiled by Laust Brock-Nannestad (the PSID part and some RSIDs) and Stephan Schmid (the majority of the RSID tunes and manual correction to some PSID entries that were wrong). Kudos to Michael Schwendt for making his songlength tool publicly available and to LaLa who sorted the PSID and RSID entries to make the SLDB better readable.

- From now on it only takes 3 tunes to get a composer his own directory! Three is a nice number, we thought. And it also cleans up the /GAMES and /DEMOS directories considerably.

- We decided to dissolve the following group directories in order to make the HVSC more consistent and make locating certain composers more intuitive: ADSR, Graffity, Natural Beat Right, there are still a few group directories left, but for various reasons we felt it's better to leave these composers grouped together (splitting up 20CC or Blues Muz' would really feel odd, for example).

- New update tool 2.8.3 (the Win32 version is included in the update). The C++ source code saw a reasonable clean-up, so it can now be compiled on more platforms without dirty hacks or patches to the source code. Check out the HVSC downloads page, there you find the new tool for the platform of your choice, as well as the source code, if you want to compile it for yourself. We're still looking for a precompiled binary of the update tool for the Amiga, so if you manage to compile it, please send it over. That of course also accounts for any other platform that it's not available for in the downloads section of the website.

- And of course we have another juicy update intro for you. Check out the d64 file at the bottom of your root directory (in /C64Music) and watch the intro to get into the right mood! :-)
- The HVSC crew got 2 new members: Pawel Ruczko aka Murdock/Tropyx and Mariusz Rozwadowski aka Ramos/Samar, both from Poland, have recently joined the team to support and enhance us.
- Kristoffer Johansson has retired from the team due to private and familiar reasons. Kristoffer, it's been a blast having you around, good luck and take care!

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)

```
Stephen C. Biggs          DRAX
Laxity                    Ace64 (NEW)
B0rje                     Richard Bayliss
Bionic Hands              Cleve
Digger                    Dos
Gangstar                  G-Fellow
Maktone                   Mc Zak
Miss Secret               Moog
Nata                      Ne7
No-XS                     Oedipus
Praiser                   PVCF
Racer X                   Rap
TC                        Mr. L (NEW)
Asterion                  Trays
Warnock                   Remarque (NEW)
Linus Akesson (NEW)       Radiantx (NEW)
Eloquence (NEW)           Gomez (NEW)
Mac_ALivers (NEW)
```

Goattracker v2.0/2.1

On 23 June 2005 Cadaver released a new version of Goattracker that extends the version 1.xx and break the compatibility with file format.

GoatTracker v2 adds more commands and uniform step programming tables for waveform/arpeggio, pulse effects, and filter effects.

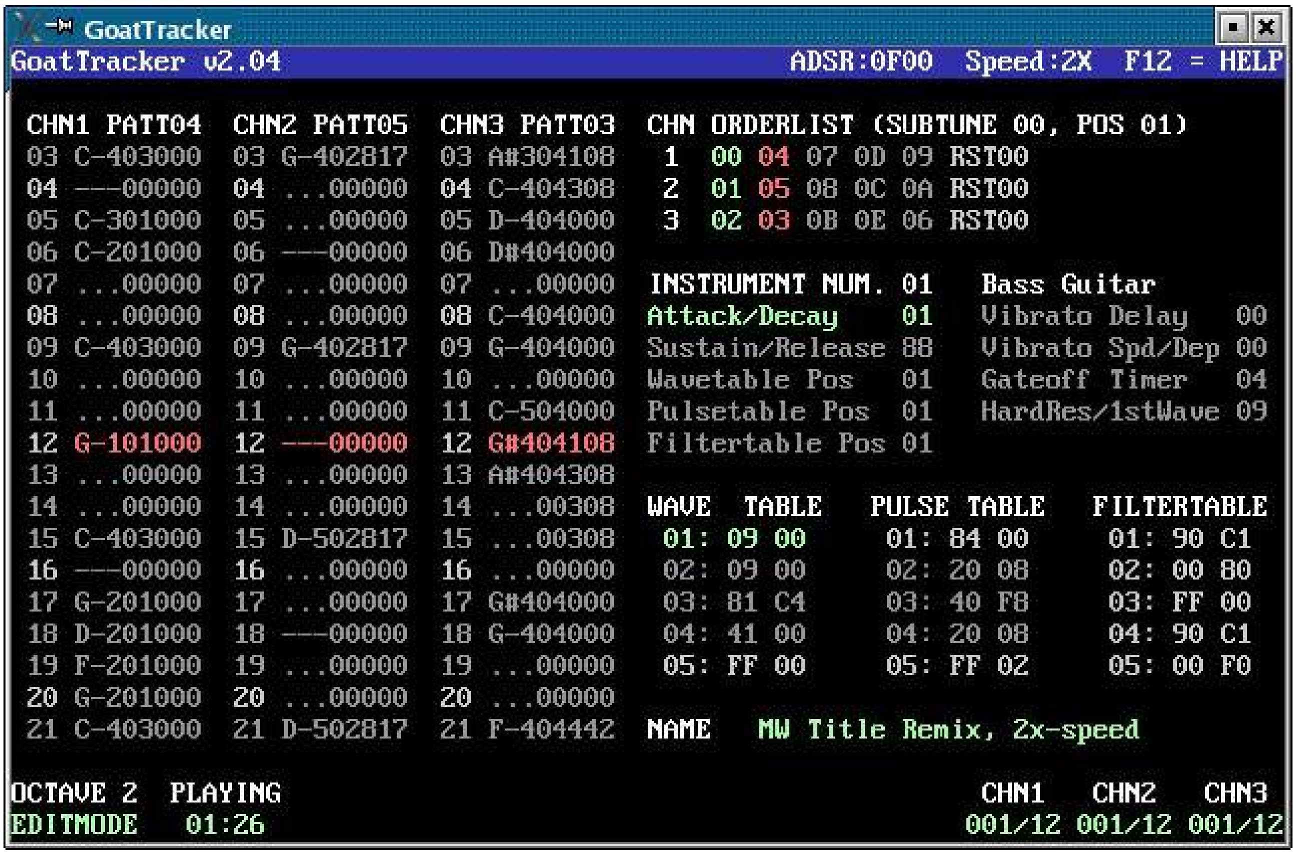
New versions still comes day by day with v2.1 released on 30 July 2005.

v2.0 Beta Original public release.

v2.0 RC1:
- Fixed crash in DMC-note entry mode.
- Fixed v1.xx import pulse conversion, a case where pulse startpos was lower than the pulse low limit was not converted right.
- Fixed instrument gatetimer becoming zero (can lead to tempo bugs) when cutting an instrument (SHIFT+X)
- Fixed order of SID writes to make editor & C64 playroutine behave as similarly as possible.
- Fixed removal of table-entries (when loading an instrument) in case another instrument uses

them too.

- Fixed toneportamento (3XY) with wavetable delays.
- Fixed funktempo in packed/relocated songs.
- Added SHIFT+DEL to delete an instrument + its tabledata.
- Added SHIFT+INS to insert rows on table first row without having to adjust tablepointers afterwards.
- Added SHIFT+W,S for octave transpose.
- Added SHIFT+1,2,3 to swap orderlists between channels.
- Added SHIFT+BACKSPACE to clear a whole pattern row.
- Added optimized playroutines, that will be used if all instruments have the same gateoff timer & 1st wave parameters.
- Added duplicate check for v1.xx pulse conversion.
- Added reSID interpolation option /I.
- Added hexadecimal pattern row display option /D.
- Added BETACONV-utility for conversion from beta format.
- Changed appearance of pattern special notes (rest, keyoff/on)
- Increased amount of instruments to 63 and changed song dataformat.
- v1.xx import converts arpeggios to instruments as long as there is room.
- Vibrato depth changed back to same as in v1.xx.
- Pulse modulation speeds are doubled.
- Upon startup, songdata erase, or importing v1.xx data, gatetimer will be set to 2 * multiplier as opposed to just 2.
- Optimized size & speed of all playroutines (initialization, checking for new note fetch, pulsetable execution).
- Wavetable delay or no-wavechange on first step of instrument is definitely unsupported now! Protection to allow this conflicted with 3XY command.



```
GoatTracker v2.04                               ADSR:0F00  Speed:2X  F12 = HELP

 CHN1 PATT04  CHN2 PATT05  CHN3 PATT03  CHN ORDERLIST (SUBTUNE 00, POS 01)
 03 C-403000  03 G-402817  03 A#304108   1  00 04 07 0D 09 RST00
 04 ---00000  04 ...00000  04 C-404308   2  01 05 08 0C 0A RST00
 05 C-301000  05 ...00000  05 D-404000   3  02 03 0B 0E 06 RST00
 06 C-201000  06 ---00000  06 D#404000
 07 ...00000  07 ...00000  07 ...00000  INSTRUMENT NUM. 01   Bass Guitar
 08 ...00000  08 ...00000  08 C-404000  Attack/Decay    01   Vibrato Delay   00
 09 C-403000  09 G-402817  09 G-404000  Sustain/Release 88   Vibrato Spd/Dep 00
 10 ...00000  10 ...00000  10 ...00000  Wavetable Pos   01   Gateoff Timer   04
 11 ...00000  11 ...00000  11 C-504000  Pulsetable Pos  01   HardRes/1stWave 09
 12 G-101000  12 ---00000  12 G#404108  Filtertable Pos 01
 13 ...00000  13 ...00000  13 A#404308
 14 ...00000  14 ...00000  14 ...00308  WAVE TABLE    PULSE TABLE    FILTERTABLE
 15 C-403000  15 D-502817  15 ...00308   01: 09 00     01: 84 00      01: 90 C1
 16 ---00000  16 ...00000  16 ...00000   02: 09 00     02: 20 08      02: 00 80
 17 G-201000  17 ...00000  17 G#404000   03: 81 C4     03: 40 F8      03: FF 00
 18 D-201000  18 ---00000  18 G-404000   04: 41 00     04: 20 08      04: 90 C1
 19 F-201000  19 ...00000  19 ...00000   05: FF 00     05: FF 02      05: 00 F0
 20 G-201000  20 ...00000  20 ...00000
 21 C-403000  21 D-502817  21 F-404442  NAME    MW Title Remix, 2x-speed

OCTAVE 2  PLAYING                                      CHN1    CHN2    CHN3
EDITMODE   01:26                                     001/12 001/12 001/12
```

v2.0 RC2:

- Fixed behaviour of + & - keys while editing instrument name.
- Added SID memory location parameter /L.
- reSID interpolation is on by default.

v2.0:

- Fixed pattern editor transpose functions.

v2.01:

- Added under-IO gamemusic playroutine.

v2.02:

- Added RETURN when instrument tableparameter (wave/pulse/filter) is 0 to get you to the first free table location. SHIFT+RETURN will additionally also set the instrument tableparameter.
- Added "set mastervolume"-jump to gamemusic routines.
- Improved sound effect handling, when an effect is interrupted by another (less silence between sounds).

v2.03:

Added execution of continuous commands & instrument vibrato during wavetable delay.

- Added wavetable right side value $80 to keep frequency unchanged (as a consequence, the note C-0 will cause the same effect).
- Added relocator optimizations: all unnecessary data is stripped when packing/relocating.
- Playroutines size-optimized.
- Explanation of how different gatetimer values can lead to playback going out of sync (section 3.3).
- reSID interpolation is no longer on by default :)

v2.04:

- Added a questionable 25Hz mode (/S0)
- Execution of commands on wavetable delay is completely reworked and more consistent now.
- Note C-0 is usable again.

v2.05

- Added instrument legato feature (Hardres/1stwave parameter 00).
- ADSR writes moved farther away from wave writes in the standard playroutine noteinit, lessening possibility of ADSR bugs.
- Song initialization and pulsetable execution speed/sizeoptimized.

v2.06

- Optimized playroutines for 3 bytes (No functionality change).
- Packer/relocator tolerates up to 256 bytes long patterns now.

v2.07

- Fixed transpose reset with F2/F3 song start (should not happen).
- Fixed varying tempo on channels with F2/F3 song start.
- Added warning screen to packer/relocator if table execution overflows.
- Added writing of PAL/NTSC and 6581/8580 flags according to PSID V2 NG specification.
- -mwindows added to linker options for no displaying of DOS prompt window (in win32 version).

v2.08

- Added finevibrato mode (/V command line option).
- Optimized playroutine sizes & speed.

v2.09

- Fixed max.pattern length in Clear Musicdata operation.
- Added Minimal playroutine.
- First wave value $80 also acts as a proper legato instrument now.
- Optimized playroutine sizes & speed.
- SID chip type & timing (PAL/NTSC) displayed on top row.

v2.1

- Added SHIFT+N for negating pulse/filter modulation speeds.
- Added no-funktempo & no-octave0 optimizations to playroutines (a total of 8 sub-types for each playroutine).
- Maximum speed is 16X now.
- Separate song/instrument/packed song directories are remembered during session.
- Pathname is displayed in the fileselector.
- Filters ** and *.* display all files in the fileselector.
- Song entered on commandline will be loaded at startup.
- Song filename currently being edited is shown in the titlebar.
- Currently edited song filename will be used as default in the
- "SAVE SONG" dialog.
- Instrument name will also be used as instrument filename as default in the "SAVE INSTRUMENT" dialog.
- UPX used for compression of win32 executables.
- Included a short reference of waveform bits (manual only).
- Configuration file has clearer structure.

CGSC v1.13

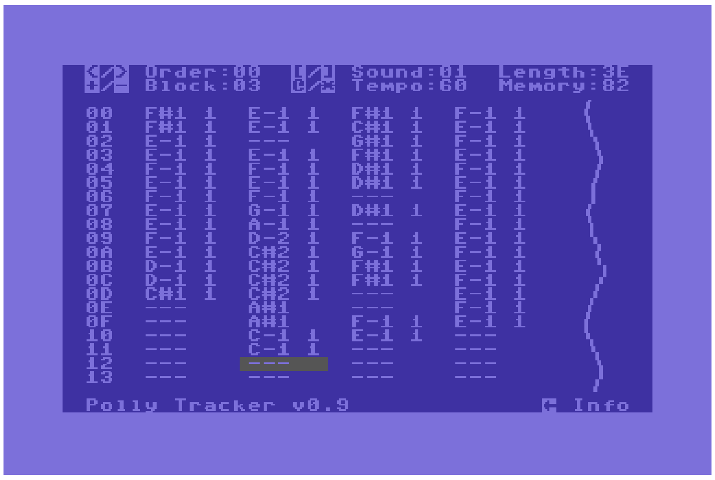
The new version of Compute's Gazette Sid Collection has been released in June 2005.

This updated contains 512 MUS, 138 STR & 200 WDS new files and 19 MUS & 1 WDS updated, for a total to 6507 MUS files, 1448 STR files and 1766 WDS files.

The vast majority of this update is due to the donation of 62 disk images by Troy Rutter (Datasid)

Download the update at http://www.btinternet.com/~pweighill/music/CGSC_v113_upd.zip

Polly Tracker v0.9 beta/v1.0



```
<[/]> Order:00  [/]  Sound:01  Length:3E
+/-   Block:03  C/   Tempo:60  Memory:82

00   F#1 1  E-1 1  F#1 1  F-1 1
01   F#1 1  E-1 1  C#1 1  E-1 1
02   E-1 1  ---    G#1 1  F-1 1
03   E-1 1  E-1 1  F#1 1  E-1 1
04   F-1 1  F-1 1  D#1 1  F-1 1
05   E-1 1  E-1 1  D#1 1  E-1 1
06   F-1 1  F-1 1  ---    F-1 1
07   E-1 1  G-1 1  D#1 1  E-1 1
08   E-1 1  A-1 1  ---    F-1 1
09   F-1 1  D-2 1  F-1 1  E-1 1
0A   E-1 1  C#2 1  G-1 1  F-1 1
0B   D-1 1  C#2 1  F#1 1  E-1 1
0C   D-1 1  C#2 1  F#1 1  F-1 1
0D   C#1 1  C#2 1  ---    E-1 1
0E   ---    A#1    ---    F-1 1
0F   ---    A#1    F-1 1  E-1 1
10   ---    C-1 1  E-1 1  ---
11   ---    C-1 1  ---    ---
12   ---    ---    ---    ---
13   ---    ---    ---    ---

Polly Tracker v0.9                  Info
```

Released on 13 July 2005 by Aleksi Eeben, Polly Tracker is a four channel sample-based music editor. Polly loads and plays 8-bit unsigned raw samples at 4-9 kHz:

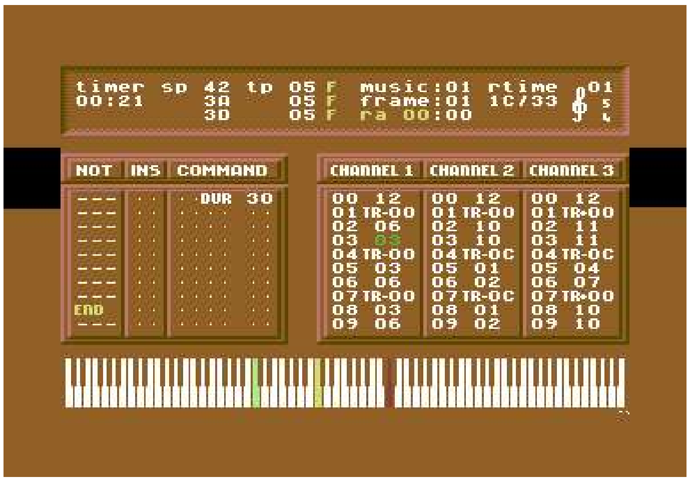
- 4 channels \o/
- 4-9 kHz sample rate on each channel (C-2 = 8000 Hz)
- 8-bit internal mixing, 4-bit output on stock C-64
- Dynamic mixing based on polling the hardware timers, no resampling

artifacts
- 48K reserved for sample data
- Loads 8-bit unsigned raw samples
- Edit options to adjust sample volume, trim sample end and octave upsample
- 6581/8580 ok, NTSC/PAL compatible and IDE64 friendly

- No SID voices used (except voice 3 output as sequencer sync :)
- Standalone player, module-to-executable and module-to-SID tools included
- 40 samples for instant joy

Download at http://pleco.mikrolahti.fi/~ae/download/polly09beta.zip
And at http://noname.c64.org/csdb/release/download.php?id=19810 the version 1.0

TFX version 2.99



```
timer  sp 42 tp 05 F  music:01 rtime   01
00:21     3A    05 F  frame:01 1C/33    5
          3D    05 F  ra 00:00          4

NOT  INS  COMMAND      CHANNEL 1  CHANNEL 2  CHANNEL 3
---  ..     DUR  30    00  12     00  12     00  12
---  ..   .... ..      01 TR-00   01 TR-00   01 TR-00
---  ..   .... ..      02  06     02  10     02  11
---  ..   .... ..      03  03     03  10     03  11
---  ..   .... ..      04 TR-00   04 TR-0C   04 TR-0C
---  ..   .... ..      05  03     05  01     05  04
---  ..   .... ..      06  06     06  02     06  07
---  ..   .... ..      07 TR-00   07 TR-0C   07 TR-00
END  ..   .... ..      08  03     08  01     08  10
---  ..   .... ..      09  06     09  02     09  10
```

Released on 15 uly 2005 by Unreal the new version of the music editor.

News and fixes:

- Intermittent pulse bug fixed. This specifically effects certain uses of dual filter tables in one specific stupid case.
- Length calculation hanging the editor really fixed this time, we think...
- The "note down" option (which controls whether you go onto the next line automatically when you enter notes) actually works again. Go figure.
- Added sac - set accurate - 16bit set command to the pulse/filter tables.
- Make player behave correctly when sliding and using hard note mode simeltaneously.
- Fixed 'set' bug in pulse tables. jaymz==moron.
- Added sound effect mode for games - call init+9 with the note in 'a', the channel*7 in 'x', the instrument in 'y', the minimum play time (i.e. the time that the main player will NOT be allowed to generate new notes over your effect) in init-1, and the hard restart time in init-2.
- Ccontrol the hard restart used for midi mode with shift-@ and shift-*, with a display in the top right corner. (which is also much more reliable, since it uses the game hooks)

Download at: http://noname.c64.org/csdb/release/download.php?id=19786

Audio::SID v3.10

On 18 July 2005 LaLa release the new verision of Perl module for Sid:

- Fixed MD5 calculation for RSIDs.
- The 'name' field is now called 'title' for consistency with HVSC and other SID-related programs. 'name' is still accepted for backwards compatibility.
- New method isRSID() that returns 'true' if the file is a RealSID, 'false' otherwise.

Downlaod from http://www.transbyte.org/SID/Audio-SID-3.10.tgz or http://www.cpan.org

Polly Tracker v1.1

Released on 19 July the version 1.1 of Polly Tracker:

- Fixed occasional crashes from keyboard activity during loading (oops)
- Fixed NTSC timing in executable player
- Memory copy in executable player no longer relies on end of basic text pointer ($2d/$2e), better compatibility with some crunchers
- Added another sample disk

Download at: http://noname.c64.org/csdb/release/download.php?id=19809

Goat Tracker 2 Music Compo

Goat Traker 2 Music Compo was performed in July by www.c64.sk, for celebrating the new editor of Cadaver.

Submission where anonymous and source of the tunes will be included as examples into the Goat Tracker release.

Here the result:

```
1: 223  I'll Be a Pimp in Cabrini Green (when I grow up)/Randall 2:40
2: 207  My Own Hyperspace/Jammer 4:13
3: 195  Sixpack Of Cola/No-XS
4: 191  Stargate/Nata/Samar 3:35
5: 187  On a sanction from CIA/Cadaver 2:05
6: 182  Ghost trackers/Hein 2:10
7: 136  Everlasting anoyance/Richard Bayliss 1:32
8: 100  Kikstart/Rafal 3:30
9:  73  Ledermaus 2:55

Inexplicable Obsession By Vengeance (cover of Vengeance)/CreaMD 1:50
```

Event: http://www.c64.sk/index.php?content=article.php&articleid=111&id=2791

Aleksi Eeben Interview!

by Stefano Tognon

This time I go to interview a coder and a musician that had created some useful tools for creating music, and one of this is very new...

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

I'm 29, currently living in Southern Finland, but strongly prefer North for it's nature, space and quietness. Sound designer by trade, employed and underpaid by a mobile phone company. I like jazz, but also Venetian Snares.

You had compose music and programming for many platforms (c64, amiga, pc, gameboy), so what is the one you find more interesting to work with?

I like VIC 20 the best, because it's the most limited and least explored so far. I have also studied Colecovision and Game Boy Advance a bit.

I have never liked PC's, except of course nowadays they can be used as professional sound workstations. Mac's are more sexy, but I'm somewhat comfortable with both.

John Player, the C64 editor you wrote, is one on the less rasterline consuming editor around. Can you say the story under the creation of this tool?

I wanted to compose some C64 sounds, but found all other editors too difficult to learn, so I wrote my own.

Low rastertime came as a bonus, as I wanted to keep it as simple as possible.

Polly Tracker is a 4 channels sample editor you release this month. Why had you create it?

It's a sort of expansion of the 3-channel 'mod' player I wrote for VIC 20 demo 'Back in the Good Old Days', released at Assembly 2004.

Can you describe in simple manner how it works (I know you use sid voice 3 for synchronization and 8 bit internally for sample manipulation)?

Instead of using interrupts I'm polling the timers in a busy loop, hence the name Polly. Whenever at least one of the channel pointers has moved to next sample, all four channels are mixed together and played out through '4-bit DAC' volume register $D418.

C64 has 4 timers. In Polly each timer controls the pitch of one channel.

Timer B hardware bug is confusing, and if it's not avoided, the sound channels relying on timer B of each CIA can't play a constant pitch. The aural output suggests that there's not enough CPU time, but that's a wrong conclusion.

There's a bug in the CIA chip logic, which clears the timer B interrupt flag if the interrupt register is read on the same cycle when timer B runs to zero. Workaround was to check timer B value before reading interrupt register, and spending one more cycle if a 'clash' was about to happen:

```
         cpy $dc06    ; y=6, check timer B value (ch2)
         beq .1       ; one extra cycle if 'clash'
     .1
         lda $dc0d    ; now it's safe to read interrupt flags (ch1&2)
```

Muted SID voice 3 plays a low frequency square wave which defines tempo. As you know the SID voice 3 waveform output can be read from SID register $D41B, so whenever this value has changed, it's time to step in the sequencer and read new note data.

Another way to mix sample channels would be using a fixed mixing rate, but then you need to run fractional bits in the waveform pointers, which is cumbersome with 8-bit processor. It's doable, but with 4 channels you cannot reach very high mixing rate on 1 MHz C64, the limit is probably around 6-7 kHz. Also, such mixer resampling produces quite nasty artifacts, especially when samples are skipped (channel rate higher than mix rate).

With timer polling you can reach 9 kHz sample rate on all four channels, and there is no extra artifacts from resampling, because samples are never skipped. Of course the samples are sometimes very slightly delayed, but you cannot bribe the timers, so the frequency output remains stable.

Do you think that you will add more features to this editor? Some people will find interesting to could use voice 1 and 2 in common sid way, for example.

Polly player is not very suitable for this, it would be better to write a new player which plays 3 SID voices and 2-4 sampled voices. Samples must be driven with interrupts then, but maybe it's possible to have two mixing modes - Polly-like busy loop mixing when SID part is not using CPU and then switching to slower interrupt mixing while running the SID voice code (at raster interrupt signal). The interrupts could be optimized by not using x or y register in the SID voice code. Fixed mixing rate for sample channels could be tried too.

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

Always test on the real machine, but emulators are great for cross-development.

For sound, reSID is quite accurate except for the filter. This could be improved a lot by trying different cliptables before the filter input.

6581 vs 8580 chip: any (musical) preference?

6581, definately. It's the SID sound for me.

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

I like them all. But it's kind of silly that HVSC seems to hold every
single 10 second test song I've made for for example John Player.

Who are your best sid authors?

TBB, AMJ, GRG, Goto80, Reed, Abaddon, Agemixer, Ed, Dane

What are the best sids ever in your opinion?

Unsound Minds tunes by AMJ. Very good.

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!

Every SID composer out there, start learning an instrument so we can play when everyone's 50.

Download the latest version of Polly Tracker:
http://pleco.mikrolahti.fi/~ae/download/pollytracker.zip

Tiny Sid Compo 512b entries

by Stefano Tognon <ice00@libero.it>

In this article I go to show and comment the entries of Tiny Sid Compo for the 512b category. This was the list of entries:

1. 15BB
2. Crue Gurl
3. Twone Five
4. Empty

15BB

15BB is my entry for the 512byte size category. The tune is born with the porpoise of 99% cover the Mike's “Bat of Basses” first seconds of the tune.

This tune is one of the my most listened sid ever. I like the initial sound, and I would like to listen to it for more time, so this is the reason 15BB was realized.

The first step I made was to look at the sound generation and notes used by the tune. The best tool for this is sid2midi program.

Look at the first seconds:

```
     Voice 1                     Voice 2                     Voice 3
Ti  Note  Freq  PW  WF ADSR VL  Note  Freq  PW  WF ADSR VL  Note  Freq  PW  WF ADSR VL  Filter
=====================================================================================================
00   ---     0    0 00 0000 --   ---     0    0 00 0000 --   ---     0    0 00 0000 --  ___ ___   0 0
02   ---     0    0 00 0000 --   ---     0    0 00 0000 --   ---     0    0 00 0000 --  L__ ___   0 0
04   ---     0    0 00 0000 --   ---     0    0 00 0000 --   ---     0    0 00 0000 --  L__ ___   0 0
06  >G#3<  207 2048 00 0000 --  >G#4<  415 4080 50 0000 --   ---     0    0 00 0000 --  L__ 1__ 180 f
08   ---   207 2064 00 0291 --   +++   415   48 50 0938 --   ---     0    0 00 0000 --  L__ 1__ 1a8 f
10   +++   207 2080 40 0291 --   +++   415  112 50 0938 --   ---     0    0 00 0000 --  L__ 1__ 1d0 f
12   +++   103 2096 40 0291 --   +++   415  176 50 0938 --   ---     0    0 00 0000 --  L__ 1__ 1f8 f
14   +++   103 2112 40 0291 --   +++   415  240 50 0938 --   ---     0    0 00 0000 --  L__ 1__ 220 f
16   +++   103 2128 40 0291 --   +++   415  304 50 0938 --   ---     0    0 00 0000 --  L__ 1__ 1f8 f
18   +++   103 2144 40 0291 --   +++   415  240 50 0938 --   ---     0    0 00 0000 --  L__ 1__ 1d0 f
20   +++   103 2160 40 0291 --   +++   415  176 50 0938 --   ---     0    0 00 0000 --  L__ 1__ 1a8 f
22   +++   103 2176 40 0291 --   +++   413  112 50 0938 --   ---     0    0 00 0000 --  L__ 1__ 180 f
24   +++   103 2192 40 0291 --   +++   411   48 50 0938 --   ---     0    0 00 0000 --  L__ 1__ 158 f
26   +++   103 2208 40 0291 --   +++   413 4080 50 0938 --   ---     0    0 00 0000 --  L__ 1__ 130 f
28   +++   103 2224 40 0291 --   +++   415   48 50 0938 --   ---     0    0 00 0000 --  L__ 1__ 108 f
30   +++   103 2240 40 0291 --   +++   417  112 50 0938 --  >D#4<  311 4080 50 0000 --  L__ 1__  e0 f
32   +++   103 2256 40 0291 --   +++   415  176 50 0938 --   +++   311   48 50 0938 --  L__ 1__  b8 f
34   +++   103 2272 40 0291 --   +++   413  240 50 0938 --   +++   311  112 50 0938 --  L__ 1__  90 f
36   +++   103 2288 40 0291 --   +++   411  304 50 0938 --   +++   311  176 50 0938 --  L__ 1__  68 f
38   +++   103 2304 40 0291 --   +++   413  240 50 0938 --   +++   311  240 50 0938 --  L__ 1__  40 f
40   +++   103 2320 40 0291 --   +++   415  176 50 0938 --   +++   311  304 50 0938 --  L__ 1__  18 f
42   +++   103 2336 40 0291 --   +++   417  112 50 0938 --   +++   311  240 50 0938 --  L__ 1__  18 f
44   +++   103 2352 40 0291 --   +++   415   48 50 0938 --   +++   311  176 50 0938 --  L__ 1__  18 f
46   +++   103 2368 40 0291 --   +++   413 4080 50 0938 --   +++   309  112 50 0938 --  L__ 1__  18 f
48   +++   103 2384 40 0291 --   +++   411   48 50 0938 --   +++   307   48 50 0938 --  L__ 1__  18 f
50   +++   102 2400 40 0291 --   +++   413  112 50 0938 --   +++   309 4080 50 0938 --  L__ 1__  18 f
52   +++   101 2416 40 0291 --   +++   413  112 50 0938 --   +++   311   48 50 0938 --  L__ 1__  18 f
54   +++   101 2432 40 0291 --  >C-4<  261 4080 50 0938 --   +++   313  112 50 0938 --  L__ 1__  18 f
56   +++   101 2448 40 0291 --   +++   261   48 50 0938 --   +++   311  176 50 0938 --  L__ 1__  18 f
58   +++   102 2464 40 0291 --   +++   261  112 50 0938 --   +++   309  240 50 0938 --  L__ 1__  18 f
60   +++   103 2480 40 0291 --   +++   261  176 50 0938 --   +++   307  304 50 0938 --  L__ 1__  18 f
62   +++   104 2496 40 0291 --   +++   261  240 50 0938 --   +++   309  240 50 0938 --  L__ 1__  18 f
64   +++   105 2512 40 0291 --   +++   261  304 50 0938 --   +++   311  176 50 0938 --  L__ 1__  18 f
66   +++   104 2528 40 0291 --   +++   261  240 50 0938 --   +++   313  112 50 0938 --  L__ 1__  18 f
68   +++   103 2544 40 0291 --   +++   261  176 50 0938 --   +++   311   48 50 0938 --  L__ 1__  18 f
70   +++   102 2560 40 0291 --   +++   259  112 50 0938 --   +++   309 4080 50 0938 --  L__ 1__  18 f
72   +++   101 2576 40 0291 --   +++   257   48 50 0938 --   +++   307   48 50 0938 --  L__ 1__  18 f
74   +++   101 2592 40 0291 --   +++   259 4080 50 0938 --   +++   309  112 50 0938 --  L__ 1__  18 f
76   +++   101 2608 40 0291 --   +++   261   48 50 0938 --   +++   309  112 50 0938 --  L__ 1__  18 f
78   +++   102 2592 40 0291 --   +++   263  112 50 0938 --  >G#4<  415 4080 50 0938 --  L__ 1__  18 f
```

We can see that voice 2 and 3 implements the same instrument:

- $51 waveform
- A little vibrato in frequency
- Pulse modulation with a cycle of 10 values. The pulse is reset at each new note.
- Each note has the same duration and a note of voice 3 start in middle of voice 2 note. This means that at the beginning voice 3 is mute for some ticks.

Instead voice 1:

- $41 waveform
- Here there is even a vibrato in frequency, but the most evident effect is that at start of note, the frequency is double for 3 ticks.
- Pulse modulation: here the cycle is long: it starts from 2048 and goes to 2608 and forward with step of 16. The value is reset at each new note.
- There is a low pass filter in this voice: the cut off frequency start from a value, than goes up and after release to a minimum value.
- Note duration are not constant for this voice

I have implemented the stuff according to the above points, using some differences where was necessary for reducing the code:

- No vibrato for the voices. The only frequency effect taken is the double frequency at beginning of note for voice 1. This is very important, as the timbre of instrument is given even from this effect. Its implementation is discussed later.
- The steps pulse for voice 1 uses other values as this let some code optimization. In original it goes up and down from $0800 to $A030. Here he goes from $07F0 to $A000, because we can use the high byte for known if the limit is reached. Testing a byte instead of a word is much simple.

At this point we can look at the source.

In the initialization part:

- the IRQ pointer is set
- patterns pointers position are reset
- duration of notes are reset (for voice 3 it is set with some delay, so the first time voice 3 is muted as in original)

In the IRQ routine, at each ticks, there is a part that generate the timbre of the instrument, and then, only 1 time onto 7 there is the note management.

In the timbre part, the code look at this:

- Generate the pulse step of voice 1. The code use step that is viewed as complement two number to add to current pulse.
- Generate the step for filter cut-off frequency: simple we look at a table of values, so increment current index until the max table size, and put the value of table into filter frequency
- Decrement a counter, and if zero, put the right saved note frequency for voice 1 (remember that for 3 ticks, the beginning of voice 1 note use double frequency)
- The pulse for voice 2 and 3 are calculated using an index to a table of values. To the index is added a direction flag value (+1, or -1) according with the direction we are looking to the table

Before looking to the last part of the code, we discuss how notes and durations are represented.
As for voice 2 and 3 the duration is the same and only for voice 1 the durations vary, a good solution is to use a special instruction that set the duration, while the other values are notes (only 10 notes are used).
So we have:

- Negative values: the low nibble give the duration (half of the one needed)
- Positive values: the low nibble give the note index (frequencies are in a table)
- Zero value: end of pattern.

in the note part, the code look so like this:

- The duration of the note is decremented and if zero the other part is executed
- Increment the pattern index and read note/duration from pattern
- If there is a new note, reset the gate of voice and put note frequency to sid registers.
- In case of voice 1, store the frequency and double it before putting to sid registers (this is for make the timbre we see before) and reset cut off-frequency of filter
- Put the pulse waveform, control, ADSR of sid for this voice

15BB Code

```
; 15BB
; 512b SID music
; This is a 99% cover of first seconds of
; "Bat of Basses" of Mike
; I like that tune that is one of my most listened sid ever. Thanks Mike!!!
; I put some effort in recreating the same instrument set.
; There are some differences, but the result is quite similar.

  processor 6502

   org 2049

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

  .org 2061

; real tune:
; voice 1:
;  PW goes from $0800 to $A030 (up/down) with step of $10. $0800 is reset at new note

; cover:
; voice 1:
;  PW goes from $07F0 to $A000 with step of $10

; no vibrato in all voices


delay =   $80        ; actual delay
pwLo1 =   $81        ; wave low byte for voice 1
pwHi1 =   $48        ; wave high byte for voice 1
step  =   $83        ; step for wave effect
point =   $84        ; +$85 pointer

fInd  =   $86        ; filter index
toUseLo = $87        ; freq lo to use for voice 1
toUseHi = $88        ; freq hi to use for voice 1
indVF   = $89        ; index for voice frequency effect

duration = $47       ; 4E, 55  duration of this note
rel_dur  = $43       ; 4A, 51  duration to reload
patt     = $44       ; 4B, 52  index to current pattern
pIndex   = $45       ; 4C, 53  pulse index
pDir     = $46       ; 4D, 54  pulse dir

RDELAY = 7        ; reload delay

; note declaration
DUR = $80    ; duration command

F2  = 1
G2  = 2
Gd2 = 3
Ad2 = 4
C4  = 5
D4  = 6
Dd4 = 7
F4  = 8
G4  = 9
Gd4 = 10

      lda  #$10
      sta  step
      ldx  #0                 ; reset pattern
      stx  patt
      stx  patt+7
      stx  patt+14
      inx
      stx  delay              ; initial delay to virtually 0
      stx  duration
      stx  duration+7
      inx                     ; voice 3 start after the others
      inx
      stx  duration+14

      lda  #$1F
      sta  $D418              ; volume max, low filter

      lda  #$F1
      sta  $D417              ; max resonance, filter voice 1

      sei
      lda  #<irq
      sta  $0314
      lda  #>irq
      sta  $0315
      cli
      rts
```

Initialization: pattern, duration (voice 3 is in delay), irq

```
freqLo:
  .byte 207       ; F2
  .byte 133       ; G2
  .byte 232       ; G#2
  .byte 193       ; A#2
  .byte 103       ; C4
  .byte 137       ; D4
  .byte 178       ; D#4
  .byte 59        ; F4
  .byte 20        ; G4
  .byte 160       ; G#4

freqHi:
  .byte 5         ; F2
  .byte 6         ; G2
  .byte 6         ; G#2
  .byte 7         ; A#2
  .byte 17        ; C4
  .byte 19        ; D4
  .byte 20        ; D#4
  .byte 23        ; F4
  .byte 26        ; G4
  .byte 27        ; G#4
```

Table of frequency (low/high) for the 10 different notes used

```
FT_DIM = $10

filter:
  ;.byte $30
  .byte $35, $3A, $3F, $44, $3F, $3A, $35
  .byte $30, $2B, $26, $21, $1C, $17, $12, $0D
  .byte $08, $03

pwLo:
  .byte $F0, $30, $70, $B0, $F0, $30
pwHi:
  .byte $0F, $00, $00, $00, $00, $01

patLo:  .byte #<(pat01-1)
patHi:  .byte #>(pat01-1)
AD:     .byte $02
SR:     .byte $91
ctrl:   .byte $41
; ctrl2:  .byte $40
PWL:    .byte $00
PWH:    .byte $08

  .byte #<(pat02-1)
  .byte #>(pat02-1)
  .byte $09
  .byte $38
  .byte $51
  ; .byte $50
  .byte $F0
  .byte $0F

  .byte #<(pat03-1)
  .byte #>(pat03-1)
  .byte $09
  .byte $38
  .byte $51
  ; .byte $50
  .byte $F0
  .byte $0F
```

filter and pulse tables, instruments values definition

```
; pattern
pat01:
  .byte DUR+14, Gd2, DUR+1, G2, F2, DUR+16, G2, DUR+11, Gd2,
  .byte DUR+3, Ad2, DUR+1, Gd2, G2, DUR+14, F2, DUR+1, G2, F2, 0

pat02:
  .byte DUR+2
  .byte Gd4, C4, Dd4, Gd4
  .byte Gd4, C4, Dd4, Gd4
  .byte G4,  C4, Dd4, G4
  .byte G4,  C4, Dd4, G4
  .byte Gd4, C4, Dd4, Gd4
  .byte Gd4, C4, Dd4, Gd4
  .byte F4,  C4, Dd4, F4
  .byte F4,  C4, Dd4, F4, 0
```

Music pattern for each voices

```
pat03:
  .byte DUR+2
  .byte Dd4, Gd4, C4, Dd4
  .byte Dd4, Gd4, C4, Dd4
  .byte Dd4, G4,  C4, Dd4
  .byte Dd4, G4,  C4, Dd4
  .byte Dd4, Gd4, C4, Dd4
  .byte Dd4, Gd4, C4, Dd4
  .byte Dd4, F4,  C4, Dd4
  .byte Dd4, F4,  C4, Dd4, 0


irq:
                                 ; make timbre of instruments
timbre:
                                 ; make timbre of voice 1: wave
      lda  pwLo1
      adc  step
      sta  pwLo1
      lda  pwHi1
      adc  #0                    ; add/sub step to pulse
      sta  pwHi1
      cmp  #$07
      beq  invert
      cmp  #$0A
      bne  setPulse
invert:                          ; invert the step direction
      lda  step
      eor  #$FF
      sec
      adc  #00
      sta  step

setPulse:                        ; set pulse for voice 1
      lda  pwLo1
      sta  $D402
      lda  pwHi1
      sta  $D403
                                 ; make timbre of voice 1: filter
      ldy  fInd
      cpy  #FT_DIM
      beq  skipF
      iny
      sty  fInd
skipF:                           ; read freq. from table and put to sid
      lda  filter,y
      sta  $D416

      dec  indVF                 ; dec freq effect for voice 1
      bne  skipVF
                                 ; put the right frequency
      lda  toUseLo
      sta  $D400
      lda  toUseHi
      sta  $D401

skipVF:
                                 ; make voice 2 and 3 pulse timbre
      ldx  #7
loopV3:

      lda  pIndex,x
      clc
      adc  pDir,x
      sta  pIndex,x
      tay
      beq  invertD
      cmp  #6
      bne  pPulse

invertD:                         ; invert direction
      lda  pDir,x
      eor  #$FF
      sec
      adc  #00
      sta  pDir,x

pPulse:                            ; put pulse
      lda  pwLo,y
      sta  $D402,x
      lda  pwHi,y
      sta  $D403,x

      jsr  incVoice
      bcc  loopV3

      dec  delay
      beq  music
      jmp  exit

music:
      lda  #RDELAY
      sta  delay

      ldx  #$0                   ; index of first voice
loopV:

      ldy  duration,x
      dey
      tya
      sta  duration,x
      beq  cont1
      jmp  nextV

cont1:
      lda  patt,x                ; load pattern index of this voice
nextIndY:
      tay
nextInd:
      iny                        ; increment index (so pattern must start one position before)
      tya
      sta  patt,x
                                 ; set pattern address
      lda  patLo,x
      sta  point
      lda  patHi,x
      sta  point+1

      lda  (point),y             ; read current note/duration
      beq  endPat
      bpl  newNote
                                 ; this is a duration
      asl                        ; double the duration, but delay is half as needed
      sta  rel_dur,x             ; store duration to reload

      jmp  nextInd

endPat:
      lda  #$00
      sta  patt,x                ; reset pattern index
      beq  nextIndY

newNote:
      tay

      ;lda  ctrl2,x
      lda  #00                   ; reset gate
      sta  $D404,x
      sta  pIndex,x              ; pulse index to 0

      cpx  #0
      beq  specV1
                                 ; put right frequency
      lda  freqLo-1,y
      sta  $D400,x
      lda  freqHi-1,y
      sta  $D401,x
      bne  contV
                                 ; special action for voice 1
specV1:
      lda  freqLo-1,y
      sta  toUseLo
      lda  freqHi-1,y
      sta  toUseHi
                                 ; double the frequency to use
      lda  toUseLo
      adc  toUseLo
      sta  $D400,x
      lda  toUseHi
      adc  toUseHi
      sta  $D401,x
                                 ; put initial filter cut-off frequency
      lda  #$00
      sta  $D415
      sta  fInd

      lda  #$30
      sta  $D416

      lda  #2
      sta  indVF                 ; set index for voice frequency effect

contV:
      lda  rel_dur,x             ; reset duration to reload value
      sta  duration,x

      lda  PWL,x
      sta  $D402,x               ; set pulse low
      lda  PWH,x
      sta  $D403,x               ; set pulse hi
      sta  pwHi1,x

      lda  ctrl,x
      sta  $D404,x
      lda  AD,x
      sta  $D405,x
      lda  SR,x
      sta  $D406,x

      lda  #1
      sta  pDir,x                ; direction up

nextV:
      jsr  incVoice
      bcs  exit
      jmp  loopV
exit:
      jmp  $ea31

; increment voice index and test for last voice
incVoice:
      txa
      clc
      adc  #7
      tax
      cmp  #$11
      rts
```

Crue Gurl

"Crue Gurl" is a 512byte entry by "A Life in Hell". As you can see from the source, the code has some features that are controlled by flags during compilation: you can so add or remove some part of the player as you need. You can so enable only the features you really need for your tune, saving so some spaces. In the code, I put the unused part in blue, so it is more easy to look at the instructions flow.

We will see in the next number the 256byte version of this tune that use the same base code player.

The best mode to understand the player is to give some hint about his work:

- The tune is composed by 3 tracks (one for each voices): *orderList1*, *orderList2*, *orderList3*
- $FF is the value that means that the track is finished and must be repeated
- Each other values are pointer to pattern of music data. A pointer in 16 bit address memory should use 2 bytes, but here we have only one byte for the pointer. What is done is to store the relative position to the first pattern (*pat1*) pointer. So *pat2-pat1* is the offset of *pat2* from *pat1*.
  So the instruction to read the pattern from actual track index position is:
  ```
          olsm      lda orderList1,y
                    tay
                    lda (zp1),y
  ```
  Where *zp1* is a zero memory page address that point to *pat1*
  Using 1 byte instead of 2 for each pointer is a clear way for save lot of space
- Into the patterns there are coded the notes to play. A $FF value means that the pattern is finished
- A 0 value means to not change actual sid parameters, and to reload the duration of the note
- Other values are note frequency to use. If *useTranspose* is 0, them are in this form:
  $XY $0Y=High Frequency of note, $X0 low frequency of note
  So the engine can use up to 4 octave, even if you have not full control into frequency.
  For example $86 is a note that is a G2 (~98 Hz)
  After the frequency is set, $41 is put as waveform of voices.
- Every time the duration has the values: 5 then 13, then repeat the sequence (*useSpeedEor* flag)
- For voice 2, there is a special note and control values used: the read value is an index to both wave forms table and chords table to use. Chords simple contain the high value of frequency to use.
- The player use hard restart
- Sustain/Release is always used as $e9 (no Attack/Decay)
- Pulse width goes from a min value to a max value with a given speed (for voice 1 and 3)
- A filter with incremented (high byte) of cut off frequency is added to one voice

Finally the player is synchronized with a loop into raster position $80 of VIC II (Interrupt is previously disable with a SEI instruction) and this is another best way to obtain IRQ timing for the player using few bytes of code.

Crue Gurl Code

```
; Tiny Player v0.4
; 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 frequancy 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
useRealPulse = 0
lotsOfZpage = 1
useTranspose = 0
tranposeUp = 0
defaultSpeed = 5
gateEnd = 3
speedEor = 8
useSpeedEor = 1
noFilter =  0
filterSweep = 1
filterReset = 0
stupidlyCompact = 1
rlines = 1
defaultSr = $e9
pwmMin = 3
pwmMax = 6
pwmSpeed = 3               ; must be (n^2)-1!!!!

.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
.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 = 1
             lda #$0f
             sta sid_ctrl
.else
             lda #$1f
             sta sid_ctrl
.if filterReset = 0
             lda #$f4
             sta $d417
.endif
             lda #$40
             sta $d416
.endif
```

```
             ; set channel #1 adsr now!
.if stupidlyCompact = 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 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
             lda #<orderList1
             sta olsm+1
             ;lda #>orderList1
             ;sta olsm+2
             jsr playPlayer

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

             ; 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

             ; now finish the channels!
             ldx #7
             lda #<orderList2
             sta olsm+1
             ;lda #>orderList2
             ;sta olsm+2
             jsr playPlayer

             ldx #14
             lda #<orderList3
             sta olsm+1
             ;lda #>orderList3
             ;sta olsm+2
.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


             ; 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
             cpx #7
             beq notC1upd
norestartchan
             cmp #3
             bne noGateReset
             ;bne owt2
             lda #$40
             sta sid_v0_ctrl,x
noGateReset
             ;jmp owt2
             ; not restarting - ensure filter is set!
.if filterReset = 1
             cpx #0
             bne nofiltr4
             lda #$f4
             sta $d417
nofiltr4
.endif
             bne owt2                  ;branch always
notC1upd
             cmp #3
             bne ng2
             lda #$fe
             ;sta gater,x
             sta gater+1
ng2
             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
gater        and #$ff
             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

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

newNote
             ; get current byte
             lda (zp1),y

             beq out

             cmp #$ff
             bne valid
             ldy orderPos,x
             iny
xxx          sty orderPos,x
olsm         lda orderList1,y
             bpl noreset
             ldy #0
             beq xxx                   ;branch always
noreset
             sta pos,x
             tay
             bpl newNote               ;branch always
valid
             cpx #7
             bne notChannel2
             ; channel 2 is the hard one, actually!
             tay
             lda chords-1,y
             sta sid_v0_freq_hi,x
             lda wave-1,y
             sta sid_v0_ctrl,x
             iny
             sty chordPtr
             lda #$ff
             ;sta gater,x
             sta gater+1
             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
             lda #$41
             sta sid_v0_ctrl,x
out
             lda curDur,x
             sta durTable,x
.if useSpeedEor = 1
             eor #speedEor
             sta curDur,x
.endif
             inc pos,x
             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


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

wave
             .byte $21, $21, $21, $00, $01          ; 6
             .byte $ff, $41, $40, $80, $40, $80, $00, $0b
             .byte $21, $21, $21, $00, $0e          ; 6
             .byte $81, $41, $41, $41, $11, $08, $00, $19
chords
             .byte $28, $2f, $3c, $00, $01          ; 6
             .byte $81, $0b, $0b, $b5, $0a, $ff, $00, $0b
             .byte $28, $2d, $3c, $00, $0e          ; $11
             .byte $ff, $08, $06, $03, $09, $09, $00, $19


pat1
      .byte $28, $00, $14, $28, $14, $24, $22, $14, $28, $00, $0f, $14, $28, $00, $0f, $14, $ff
pat2
      .byte $14, $00, $01, $01, $14, $06, $01, $01
      .byte $14, $00, $14, $01, $14, $06, $01, $06, $ff
pat3
.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
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

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
.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
```

Twone Five

"Twone Five" is the tune by Laxity and it was done with TASM assembler inside the C64. Here the source is extracted from the program itself.

The player starts by "disabling" the IRQ. This is achieve by using a RTI as IRQ routine. Note that it goes to use hardware IRQ vector, not the software, so no kernel IRQ routine is called (and so we can say that IRQ is disable). Basic and Kernel Roms are disable too.

Then the player creates the complete notes frequency using a little table of 24 bytes, and goes to synchronize with VIC II raster position $50 for simulate a IRQ call. The total space used for generating the frequency table is of 82 bytes instead of 188 bytes for a typical one.

The player manages all tree sid voices and the tune is composed by a pointer to 3 sequences (pattern): seqlo, seqhi.
A pattern of values has this meaning:

Range	*Description*
$00..$7E	Note
$7F	End mark
$80..$BF	Duration
$C0..$FF	Wave pointer

so, for example the first sequence:

```
seq01    .byte $83,$ca,$24,$df,$30
         .byte $c4,$3c,$df,$30
         .byte $7f
```

become:

```
duration=3
instrument=Kick
note=C3
instrument=thing
note=C4
instrument=Snare
note=C5
instrument=thing
note=C4
end of sequence
```

One instrument is so based onto 2 wave tables: wavetable1 and wavetable2

The first contains the control value to use for the voice, the other the relative (value positive) or absolute (value negative) note to use. If relative, it is add to current note, if absolute it is the value to use. A $00 value end the instrument.

Let we look at some example:

```
Bass:
.byte $81,$81,$41,$00
.byte $c0,$c0,$00,$00
```

There is 2 ticks with noise and E5 note, follow by rectangular waveform and relative note 0 to add.

```
Snare:
.byte $81,$81,$11,$40,$80,$00
.byte $d0,$d0,$b2,$ac,$ca,$00
```

A classical snare implementation with noise, triangular voices, follow by wave with gate bit released.

I think that if you had composed with some real editor around, you are familiar with this kind of instrument implementation.

The other values that are used for the instruments are:

- Attack/Decay: fixed for all sid voices
- Sustain/Release: each voice has his value
- Hardrestart of note: hardrestart is always active for each note (and start two frame before the end of note)
- The pitch for each voice (when using rectangular waveform) is different: the high value comes from pulsehi and low value is incremented by the value of pulseadd at each sid out.

Twone Five Code

```
;---------------------------------------
;TPlayer 01.g0
;By Laxity of Vibrants/Manicas of Noise
;---------------------------------------
;Coded on 25th of March 2005
;---------------------------------------
;Contains tune: Five Twone
;---------------------------------------
;Sparecly documented, I know. The player
;is extreemly simple but still handles
;wave tables for instrument. AD is fixed
;for all channels, SR is set per channel
;
;Hard restart is always enabled.
;
;No vibrato, no slide, no tie notes.
;
;Only one sequence per channel.
;
;Channel two has a transpose table for
;the bass ostinate.
;---------------------------------------
zp       = $02
;---------------------------------------
clrbegin = $04
;---------------------------------------
pulselo  = clrbegin

speedcnt = pulselo+3
wavepoi  = speedcnt+3
wavecur  = wavepoi+3
durcnt   = wavecur+3
dur      = durcnt+3
seqofs   = dur+3

note     = seqofs+3
notetp   = note+3
noteout  = notetp+3

shd404   = noteout+3
shd405   = shd404+3
shd406   = shd405+3

tpcnt    = shd406+3
;---------------------------------------
clrend   = tpcnt+1
;---------------------------------------
         *= $080e
;---------------------------------------
; Mask out the basic and kernal rom and
; setup the irq vector to point to an
; "RTI" instruction so the program won't
; crash.
;---------------------------------------
```

```
start    sei
         lda #$35  ;Mask out basic and
         sta $01   ;kernal rom.
         lda #<int ;Set up the irq-vec
         sta $fffe ;to avoid the program
         lda #>int ;crashing.
         sta $ffff
;---------------------------------------
; Calculate the frequency table
;---------------------------------------
         lda #6
         sta zp

         lda #<freqtab+2
         sta zp+3
         lda #>freqtab+2
         sta zp+4
```

Generates table of frequency (low/high) at run time

```
frq03
         ldy #0
frq02
         lda frqsrc,y
         sta zp+5
         lda frqsrc+1,y
         sta zp+6

         ldx #0
frq01
         cpx zp
         beq skipshift

         lsr zp+6
         ror zp+5
         inx
         bne frq01
skipshift
         lda zp+5
         sta (zp+3),y
         iny
         lda zp+6
         sta (zp+3),y

         iny
         cpy #12*2
         bne frq02

         dec zp
         bmi frqend

         clc
         lda zp+3
         adc #12*2
         sta zp+3
         bcc frq03
         inc zp+4
         jmp frq03
frqend
;---------------------------------------
; Clear player variables
;---------------------------------------
init     ldx #clrend-clrbegin
         lda #0
tinit01
         sta clrbegin,x
         dex
         bpl tinit01
;---------------------------------------
; Loop point of program
;---------------------------------------
loop
         lda $d012
         cmp #$50
         bne loop
         inc $d020
;---------------------------------------
; Play a frame of music. (Player starts
; here.
;---------------------------------------
play     lda #$0f
         sta $d418

         dec speedcnt
         bpl speedok
;---------------------------------------
         lda #2
         sta speedcnt
;---------------------------------------
speedok
         ldx #2
tnextvoice
         lda speedcnt
         bne updateout
```

```
updatecnt
         dec durcnt,x
         bpl updateout
;---------------------------------------
; Read and parse sequence data.
;---------------------------------------
updateseq
         ldy seqofs,x
         lda seqlo,x
         sta zp
         lda seqhi,x
         sta zp+1
         dey
readagain
         iny
         lda (zp),y    ;Read seq byte
         bpl notdur    ;<$80 / note
         and #$7f
         cmp #$40
         bpl setwave   ;<$c0 / duration
         sta dur,x
         bmi readagain
setwave
         and #$3f      ;>=$c0 / wavepoi.
         sta wavepoi,x
         bpl readagain
notdur
         cmp #$7f      ;$7f / end mark
         bne notend
         cpx #1        ;apply transpose
         bne notp      ;to channel 1
         inc tpcnt
         lda tpcnt
         and #7
         tay
         lda tp,y
         sta notetp,x
notp
         ldy #$ff
         bmi readagain
notend
         sta note,x    ;set note and
         lda dur,x     ;reset duration
         sta durcnt,x  ;counter
         iny
         tya           ;store sequence
         sta seqofs,x  ;pointer

         lda wavepoi,x ;reset wave
         sta wavecur,x ;pointer
;---------------------------------------
; Update output.
;---------------------------------------
updateout
         ldy voice,x  ;Get voice offset
         lda note,x
         beq hard
         lda durcnt,x
         bne nothard
         lda speedcnt
         cmp #0
         beq nothard
```

Hardrestart
(AD, SR and control)

```
hard
         lda #$0f
         sta $d405,y
         lda #$00
         sta $d406,y
         sta shd404,x
         beq sidout

nothard  lda #$03
         sta $d405,y
         lda sr,x
         sta $d406,y

         ldy wavecur,x
         lda wavetab1,y
         beq sidout
         sta shd404,x
         lda wavetab2,y
         bmi noteabs
         clc
         adc note,x
         adc notetp,x
noteabs
         asl a
         sta noteout,x

         inc wavecur,x
sidout
         ldy noteout,x
         lda freqtab,y
         pha
         lda freqtab+1,y
         ldy voice,x
         sta $d401,y
         pla
         sta $d400,y
         lda shd404,x
         sta $d404,y
         lda pulsehi,x
         sta $d403,y
         lda pulselo,x
         clc
         adc pulseadd,x
         sta pulselo,x
         sta $d402,y

         dex
         bmi tend
         jmp tnextvoice
tend
         dec $d020
         jmp loop
int      rti
;---------------------------------------
voice    .byte 0,7,14

sr       .byte $e6,$d6,$e7

pulsehi  .byte $08,$01,$04
pulseadd .byte $00,$10,$3d

seqlo    .byte <seq01,<seq02,<seq03
seqhi    .byte >seq01,>seq02,>seq03
;---------------------------------------
wavetab1 ; 00 - Bass
         .byte $81,$81,$41,$00

         ; 04 - Snare
         .byte $81,$81,$11,$40,$80,$00

         ; 0a - Kick
         .byte $09,$81,$11,$10,$10,$10
         .byte $10,$00

         ; 12 - Crystal sound
         .byte $41,$41,$11,$42,$20,$20
         .byte $40,$00

         ; 1a - Sine plug
         .byte $81,$81,$15,$10,$00

         ; 1f - thing
         .byte $81,$81,$12,$00

wavetab2 ; 00 - Bass
         .byte $c0,$c0,$00,$00

         ; 04 - Snare
         .byte $d0,$d0,$b2,$ac,$ca,$00

         ; 0a - Kick
         .byte $d2,$d2,$ae,$a6,$a2,$9e
         .byte $96,$00

         ; 12 - Crystal sound
         .byte $0c,$0c,$00,$00,$0c,$0c
         .byte $00,$00

         ; 1a - Sine plug
         .byte $d0,$d0,$00,$00,$00

         ; 1f - thing
         .byte $c0,$c6,$00,$00
;---------------------------------------
tp       .byte $00,$00,$00,$00
         .byte $05,$07,$00,$00
;---------------------------------------
; Music data
;---------------------------------------
         ; Drums beat
seq01    .byte $83,$ca,$24,$df,$30
         .byte $c4,$3c,$df,$30
         .byte $7f

         ; Bass ostinat
seq02    .byte $83,$c0,$18,$da,$3c,$3a
         .byte $c0,$16,$da,$3c,$3a
         .byte $c0,$18,$24
         .byte $7f
         ; Crappy melody line
seq03    .byte $bf,$d2

         .byte $30,$30,$30,$30

         .byte $8b
         .byte $43,$3f,$87,$3c
         .byte $8b
         .byte $3a,$3c,$87,$43
         .byte $8b
         .byte $3a,$39,$83,$3a
         .byte $a3
         .byte $3f
         .byte $9f,$30,$8f,$30
         .byte $87,$3c,$83,$3a,$3c
         .byte $bf,$30

         .byte $7f
;---------------------------------------
frqsrc
         .byte $a0,$45,$b8,$49,$20,$4e;6
         .byte $bc,$52,$ac,$57,$e4,$5c
         .byte $70,$62,$4c,$68,$84,$6e
         .byte $18,$75,$10,$7c,$70,$83
;---------------------------------------
freqtab
```

Empty

"Empty" is the winner 512b tune by 4Mat. The source code is here reformatted using an indentation for better view of it.

The first instruction of the player is to disable interrupt with a SEI instruction. This is off course the best way to disable interrupt using only 1 byte of code.

Even in this player, the frequencies table is generated at runtime. In this case only 12 initials bytes are used and the code is more compacted: 58 total bytes for generating the table.

The player then synchronized with raster position of VIC II using low value compared to actual Accumulator register. You can see that first time A=0, but in all the followed times, it is equal to Sustain/Release of voice 3 (this is the last byte putter to Sid register by the last procedure running before going to synchronized again). The minimum value is SR is $6E and the maximum is $EC for the given instrument, so this value will occurs only one time (low of raster position) in a frame (however voice 3 use SR of $6F).

The player is then divided into two part: playframe and update: the first manages tempo, note duration and reading of new data from patterns, the seconds manages instruments and sound generation.

But let we see how the song is managed by the player.

The songstart contains 3 offsets to the songdata area: one for each voice to play. The meaning of the bytes in this area is:

Value	*Description*
=< $7f	Position in pattern data to play from
$80-$8f	Transpose value for this channel
$ff	End of song data

So, 128 bytes is the maximum number of bytes that pattdata (the area with pattern to play) can have.

Value	*Description*
=< $7f	Note value. (if zero leave the previous note playing)
$80-$8f	Set Note length
$e0-$ef	Set Instrument number
$ff	End pattern

So, let we see the voice 2 definition, as an example:

```
songstart -> $0e

songdata -> $13,$ff

pattdata -> $e2,$55,$e5,$30,$35,$3a,$41,$35,$3a,$29,$ff
```

The player has so 16 max instruments and in this song it uses 7 of them.

Instruments are defined by:

- Sustain/Release (no Attack/Decay used): instadsr
- Pulse width to add at each frame: this value is added to low of Pulse: instadd
- Pulse width high byte to set (low 4 bits of instpuls)
- Index loop position to sequence of values for the instrument (high 4 bits of instpuls)

For each instrument there is a max of 3 sequences of values to put at each ticks: the index loop position (from 1 to 3) says on where to loop.

The values are:

- waveform values to use (e.g. rectangular, triangular, ...)
- instrument note pitches

This is a coded bytes:

Value	***Description***
=<$ef	Value to set in pitch high byte. (low byte isn't cleared)
=<$f0-$ff	Arpeggio value, or if $f0 play pattern data note as it is.

For example, instrument 0 has:

```
instdata   .byte $f3,$f7,$f0
instwave   .byte $41,$41,$41
instpuls   .byte $15
```

So this is a classical arpeggio with 3 values.

Empty Code

```
; "Empty" : 512 byte tune by 4mat
; for the Tiny Sid Competition.
;
; For PAL machines, tested on 6581 chip.
;
; Source compiled with C64ASM.

*= $0801

;;;; Variables:
;;;; Some of this data isn't used now as I removed features from the player.

voicedata = $1000
data1 = voicedata+$15
data2 = data1+$15
notehi = data2+$80
notelo = notehi+$80

;;;; Basic Loader

.byte <start-1,>start,$9c,$ad,$9e,$32,$30,$36,$31,$00,$00,$00

;;;; Generate frequency table & setup player.

start sei

      ldx #$00
freqloop
      lda #$01
      pha
      txa
      tay
freqset
      pla
      sta notehi+$01,y
      asl
      pha
      lda freqsource,x
      sta notelo+$01,y
      rol
      sta freqsource,x
      bcc notinc
      pla
      adc #$00
      pha
notinc
      lda #$00
      sta voicedata,y
      clc
      tya
      adc #$0c
      tay
      bpl freqset
      inx
      cpx #$0c
      bne freqloop

      ldy #$0e
      sty $d418
      ldx #$02
startup
      lda songstart,x
      sta data2+$05,y
      lda #$0e
      sta data1+$05,y
      tya
      sbc #$07
      tay
      dex
      bne startup

;;;; Loop forever playing the tune...

wait
     cmp $d012
     bne wait
     jsr playframe
     bpl wait

;;;; Player Loop.

;;;; The 2nd half of the player (update) is above the first (playframe)
;;;; to avoid branch out of range errors.

update
     ldy data1,x

; pulsewidth modulation.

     lda voicedata+$02,x
     adc instadd,y
     cmp voicedata+$02,x
     sta voicedata+$02,x
     bcs notaddp
     inc voicedata+$03,x

; load variables with note & instrument data.

notaddp
     tya
     asl
     asl
     adc data1+$01,x
     sbc data1,x
     tay
     lda instwave,y
     sta voicedata+$04,x
     lda instdata,y
     cmp #$f0
     bcc justset
     and #$0f
     adc data1+$03,x
     adc data2+$02,x
     tay
     lda notelo,y
     sta voicedata,x
     lda notehi,y
justset
     sta voicedata+$01,x

; check position in instrument table and loop if required.

ignore
     inc data1+$01,x
     lda data1+$01,x
     and #$03
     bne resetinst
     ldy data1,x
     lda instpuls,y
     lsr
     lsr
     lsr
     lsr
resetinst
     sta data1+$01,x

; write data to sid chip.

notnewnote
      ldy #$07
playsid
      lda voicedata,x
      sta $d400,x
      inx
      dey
      bne playsid

      cpx #$15
      bne loopplay
      rts

playframe
      ldx #$00

; check if tempo counter < 0 and reset tempo, then check if
; note length < 0 and get more pattern data if it is.

loopplay
      dec data2+$04,x
      bpl update

      lda #$06 ; Tune Tempo
      sta data2+$04,x

      dec data2,x
      bpl update

; get pattern data.

getmore
      inc data1+$04,x
getmore2
      ldy data1+$04,x
      lda data1+$06,x
      sta data2,x
      lda pattdata,y
      beq notnewnote
      bmi other
      sta data1+$03,x
      lda #$00
      sta data1+$01,x
      ldy data1,x
      lda instpuls,y
      sta voicedata+$03,x
      lda instadsr,y
      sta voicedata+$06,x
      lda #$09
      sta voicedata+$04,x
      bpl ignore
other cmp #$ff
      beq songadd
      cmp #$df
      and #$0f
      bcc lengthset
      sta data1,x
      bpl getmore
lengthset
      sta data1+$06,x
      bpl getmore

; get song data.

songadd
      inc data1+$05,x
songadd2
      ldy data1+$05,x
      lda songdata,y
      bmi other2
      sta data1+$04,x
      bpl getmore2
other2
      cmp #$ff
      beq restart
      and #$0f
      sta data2+$02,x
      bpl songadd
restart
```

```
      lda data2+$05,x
      sta data1+$05,x
      bpl songadd2

; instrument note pitches, format is:
; =<$ef     - Value to set in pitch high byte. (low byte isn't cleared).
; =<$f0-$ff - Arpeggio value, or if $f0 play pattern data note as it is.

instdata .byte $f3,$f7,$f0
         .byte $f4,$f7,$f0
         .byte $10,$af,$06
         .byte $f0,$f0,$f0
         .byte $14,$0c,$e0
         .byte $fc,$ef,$f0
         .byte $fc,$fc,$f0

; instrument waveform values
; I only have one counter for both data & waveform to save memory, so
; any instrument that loops can't use release value in this player. :(

instwave .byte $41,$41,$41
         .byte $41,$41,$41
         .byte $41,$81,$40
         .byte $41,$41,$40
         .byte $41,$41,$80
         .byte $11,$81,$40
         .byte $11,$21,$40

; instrument Sustain/Release settings:

instadsr .byte $6f,$6f,$95,$ec,$a9,$79,$6e

; low 4-bits are pulsewidth hibyte, high 4-bits are instrument loop position:

instpuls .byte $15,$18,$38,$30,$38,$3b,$36

; pulsewidth value to add each frame:

instadd  .byte $1c,$0c,$00,$15,$00,$74,$a5

; Pattern data format:
; =< $7f  - Note value. (if zero leave the previous note playing)
; $80-$8f - Set Note length.
; $e0-$ef - Set Instrument number.
; $ff     - End pattern.

pattdata

         ; reset data

              .byte $00,$ff

         ; bassline

         .byte $81,$e3,$11,$1d,$82,$e4,$55,$82,$e3,$11,$81,$1d
         .byte $e4,$55,$e3,$18,$ff

         ; bassdrum and accompany voice

         .byte $e2,$55,$e5,$30,$35,$3a,$41,$35,$3a,$29,$ff

         ; arpeggio

         .byte $8f,$e0,$35,$e1,$33,$31,$00,$e0,$35,$e1,$33,$e0,$2e,$00,$ff

         ; riff

         .byte $e6,$82,$38,$37,$87,$ff
         .byte $33,$80,$2e,$2c,$ff
         .byte $31,$8f,$00,$81,$00,$ff
         .byte $8f,$2e,$00,$ff

; Song data format:
; =< $7f  - Position in pattern data to play from.
; $80-$8f - Tranpose value for this channel.
; I removed the 'repeat pattern' command and made the maximum
; patterndata 128 bytes to save some memory.

songdata

         ; channel 1

         .byte $f0,$02,$02,$f5,$02,$02,$f8,$02,$fa,$02,$f1,$02,$02,$ff

         ; channel 2

         .byte $13,$ff

         ; channel 3

         .byte $1e,$1e
         .byte $2d,$33,$2d,$33,$2d,$38
         .byte $2d,$33,$2d,$33,$3e
         .byte $2d,$f7,$33,$f0,$2d,$f9,$33,$f0,$2d,$38
         .byte $2d,$33,$2d,$33,$3e,$ff

; start positions in the songdata for each channel.

songstart .byte $00,$0e,$10

; source values used to calculate frequency table.

freqsource  .byte 12,28,45,62,81,102,123,145,169,195,221,250
```

Conclusion

By looking at the sources of those 512 bytes entries I hope you have seen some interesting peaces of music code and learn some music programming techniques.

Maybe it is more simple to understand a 512 bytes engine, as it could have a more linear structure over a 256 bytes one. This is the reason why I start in this number to show the category with this size, instead of the more little.

So, next time we will see the 256 bytes entries, where the code will be more intricate...

Catweasel Mk4

by Stefano Tognon <ice00@libero.it>

The Catweasel Mk4 is a new PCI card produced by Individual Computers that support 2 sid chips.
I always had consider the HardSid card too much limited because it only supports one sid chip and the HardSid 4 a dream card with the possibility to use 4 sid chips. The Mk4 with 2 sid chips can be a good solution, as we can add a 6581 and a 8580 in the same card and so listen to all the tunes with the right chip.

The card is however a controller that let you to read lot of Amiga and C64 floppy disks and support even the real digital joystick. However I'm only interesting to sid stuff of this card at the moment.

In this article I will describe my experience with the MK4

Hardware

I bought the card thrown “Soft3” (a local Italian Amiga store) using the online shop.

That was very easy: fill the online form for getting in contact with the seller, then I choose the payment method that I like (all this went with emails). So, ordered in Monday, payed on Tuesday with credit card, sent to me in Wednesday using a typical post service, the card arrived on Tuesday of next week (there were 3 holiday days in that week).

As you can see in those photos, the pack is very well assorted for avoiding travel damage. Unfortunately it is known that Italian post service pack is not so careful into manage correctly the packs, so this is a good solution.

Opened the protective “air-bag” material and remove the pack paper, finally we can see the real content of the pack: a cardboard box with the same layout of Individual Computers web site.

When I opened the box, I found:

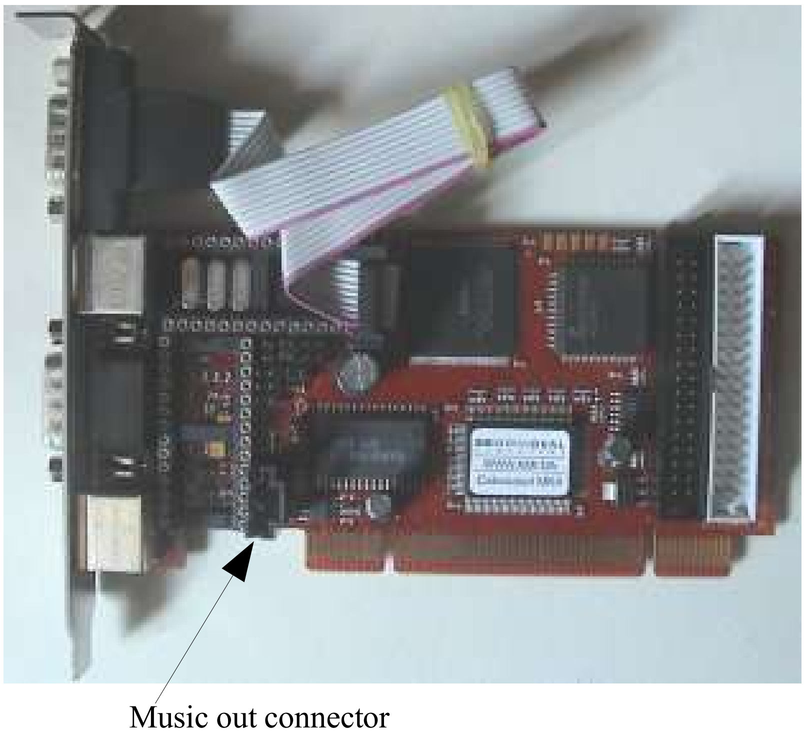
- The seller payment receipt
- Some jumpers
- A music cable like the CDROM one but with more in/out different terminations
- 3 IDE like flat cable for use with the floppies drives
- 2 metal brackets to use with some low-profile cases
- 4 pages of short manual (in English and German)
- A CDROM with the Windows driver and some documentation (like real photos of jumper setting for 6581/8580 chips)
- The card placed into a anti-static plastic envelope (that was not closed)

Now we can look at the card layout:
the card is very little and you can see lot of components (like oscillators) before the space where there are the 2 sid sockets.

The two sid sockets have gold metal contacts and so are very good.

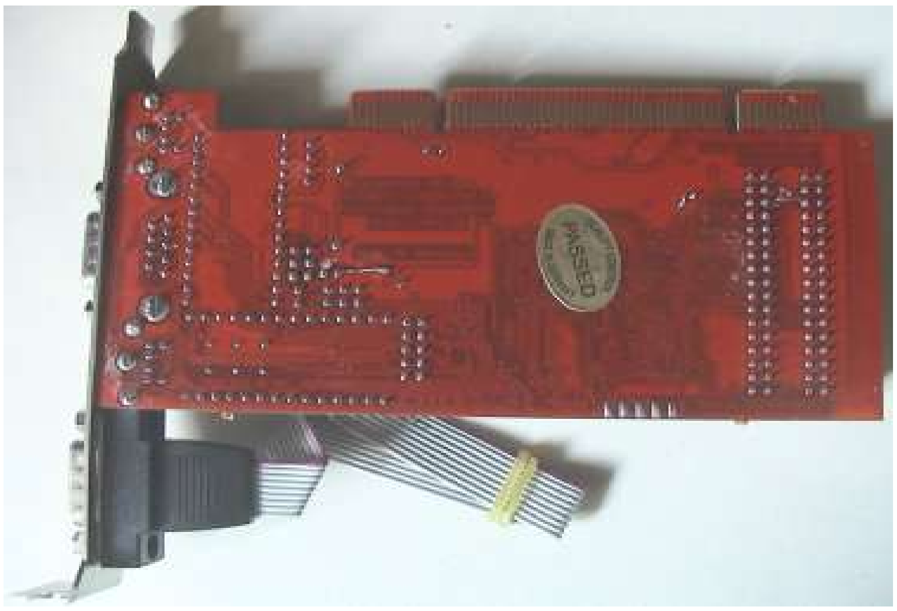
At this point for connecting the CDROM like cable into the sound card you probably should have the manual of the card for pin connection of his input.

For example my ISA Sound Blaster 16 Vibra has a CDROM input cable that cannot be connected with the given cable.



Instead the AUX in of the card can be connected with all the 3 little type of connectors the given cable have, but only one give the sound to the card.

Just try and error, or look at the manual the signals that the connector want onto his pins.



Now if we look at the back side of the card, I see that into the card is right visible a cut wire onto the board.

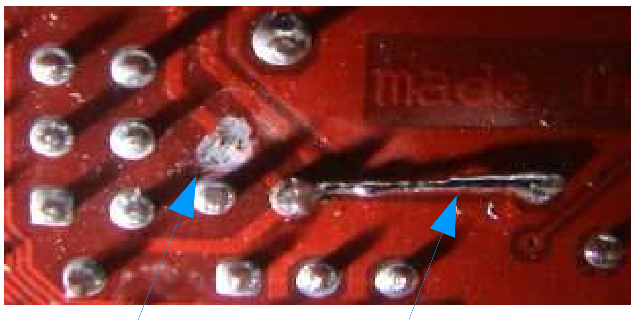
This look strange (look at the in-depth photo), but as the card have a quality test passed stamp, may be this could be a fix after the board production.

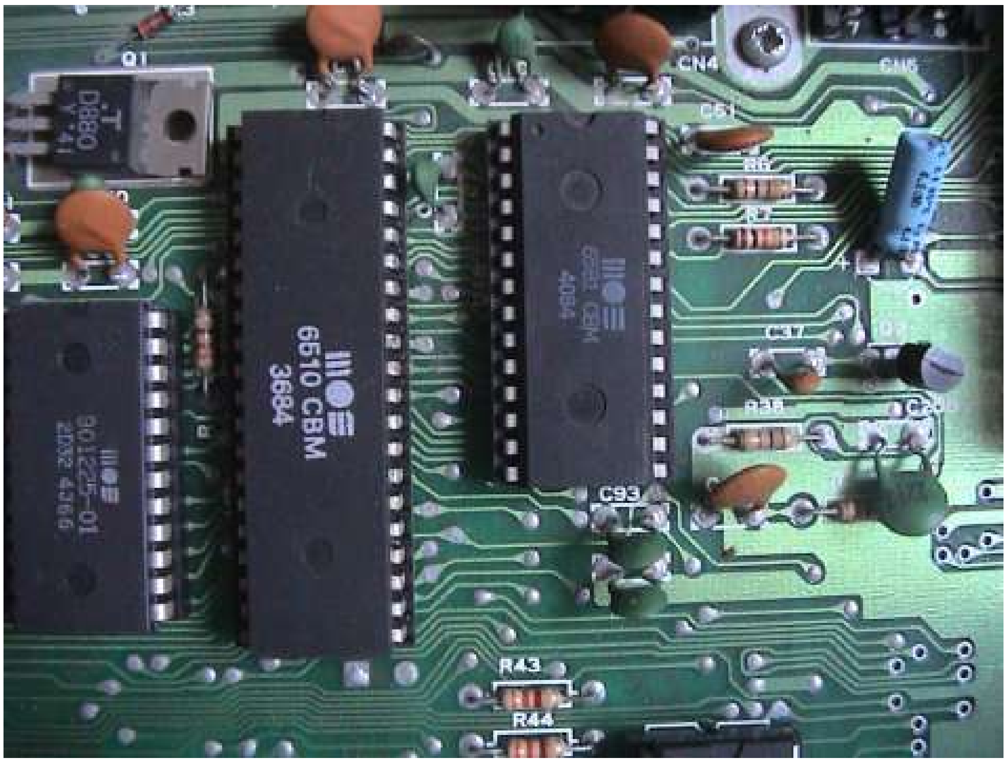
We can even see an external wire near this point, maybe the rest of the fix.

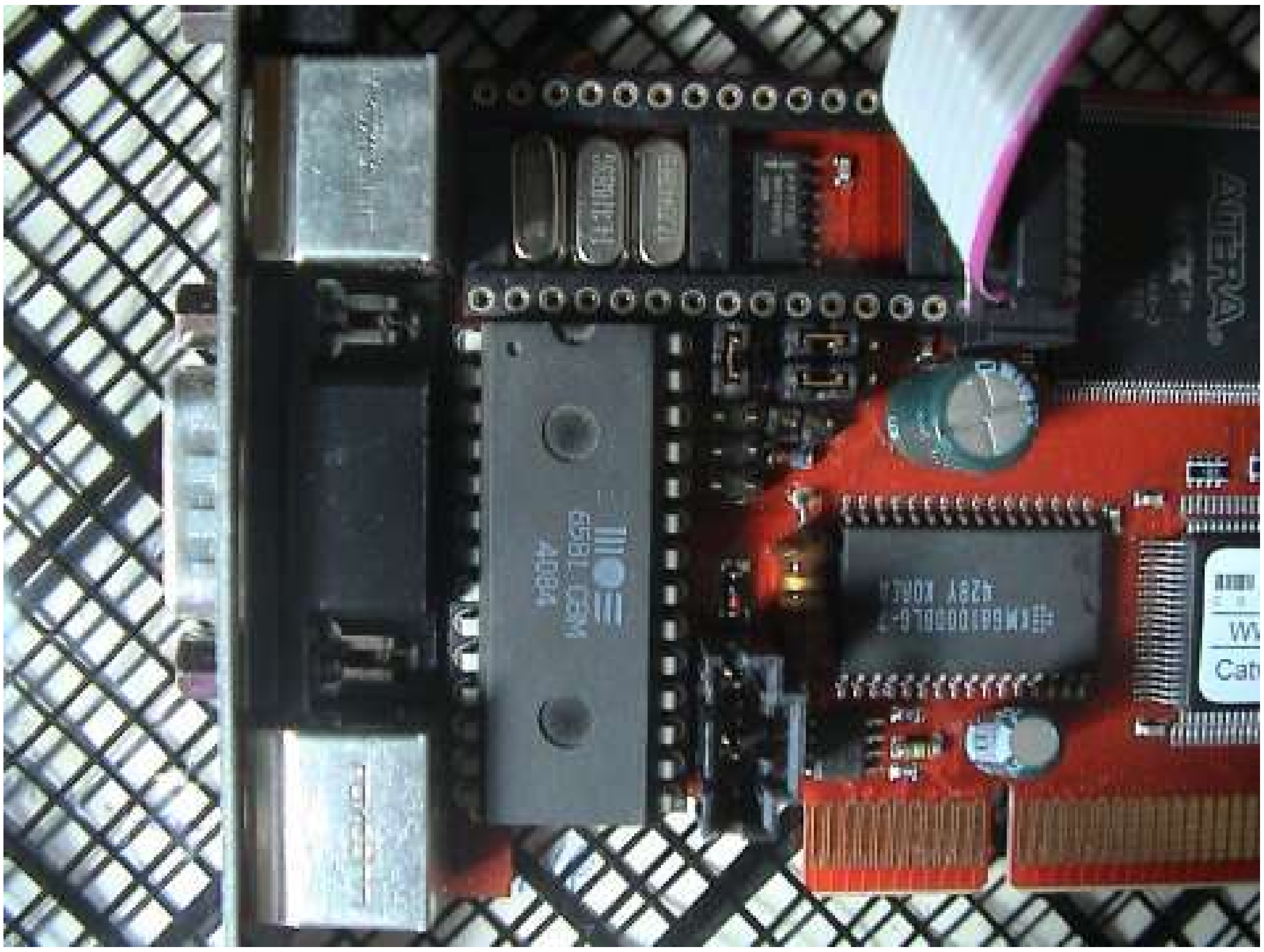
At this point I must put a sid chip to the board.

For that I take a dead C64 (not working for a dead PLA) and remove the 6581 sid chip in it.

You can see the chip in the photo below:







I so put the chip into the socket that is along side the connectors and put the jumper for 6581 (and 8580 for the other socket for future insertion of that chip), and put the card into PCI socket, connecting even the sound cable to the sound card.

Software

Now that the card is mounted into the PC's PCI socket, I have only to make it run. So I asked Simon White what to do for using his driver and having his experimental vice patch for making MK4 goes with his driver.

He tell me to download the CVS driver at http://hardsid.sf.net using the *experimental_cmk4* branch and the latest CVS version of libsidplay2 (with hardsid builder).

Unfortunately I download the *HEAD* branch for mistake: the differences is that *HEAD* did not use the hardware buffer of the card, instead the other use this features. At the moment that I test this, only Linux driver were able to take advantage of this hardware buffer.

I so compile the kernel driver simple by run make into hardsid directory and then make install as root user.

I also create:

```
[root@localhost ice]# mknod /dev/sid0 c 60 0
[root@localhost ice]# mknod /dev/sid1 c 60 1
[root@localhost ice]# mknod /dev/sid2 c 60 2
[root@localhost ice]# mknod /dev/sid3 c 60 3
```

```
[root@localhost ice]# mknod /dev/sid4 c 60 4
[root@localhost ice]# mknod /dev/sid5 c 60 4
[root@localhost ice]# mknod /dev/sid5 c 60 5
[root@localhost ice]# mknod /dev/sid6 c 60 6
[root@localhost ice]# mknod /dev/sid7 c 60 7
[root@localhost ice]# mknod /dev/sid8 c 60 8
[root@localhost ice]# mknod /dev/sid9 c 60 9
[root@localhost ice]# mknod /dev/sid10 c 60 10
[root@localhost ice]# mknod /dev/sid11 c 60 11
[root@localhost ice]# mknod /dev/sid12 c 60 12
[root@localhost ice]# mknod /dev/sid13 c 60 13
[root@localhost ice]# mknod /dev/sid14 c 60 14
[root@localhost ice]# mknod /dev/sid15 c 60 15
[root@localhost ice]# ln -s /dev/sid0 /dev/sid
```

because the readme say that. However those steps are no longer true with udev devices that comes with kernel 2.6, as /dev/sid is automatically created by the driver (the readme was not updated for this).

At this point you can found:

1. /dev/sid
2. /proc/hardsid
3. /etc/init.rd/hardsid

The first is the sid device, the second is a process that give information about the card, while the last is the script that start/stop/install the kernel driver.

As soon as this was installed, I switch off the power, mounted the card and start Linux again.

After the boot, Fedora Core 3 will recognize the card as an ISDN pci modem, but I not let him install that driver. At this point I go to see the kernel message:

```
May  1 20:49:21 localhost kernel: HardSID Driver v0.17-dev
May  1 20:49:22 localhost kernel: hardsid @ 0xe800: firmware 'cmk4.fw' not available or load failed
May  1 20:49:22 localhost kernel: hardsid: probe of 0000:00:0f.0 failed with error -2
```

Ops, the driver did not found the firmware. As you can see looking at the driver source, the firmware of the card must be putted into the car at start up (this is quite common this day for card).

I then see that in Fedora, the firmware must be into:

```
          /lib/firmware
```

instead of the position Mandrake look for. I just copy manually the firmware into the right position and restart the kernel driver:

```
May  1 20:50:17 localhost kernel: HardSID Driver v0.17-dev
May  1 20:50:19 localhost kernel: Catweasel MK4 card with 6581 as chip 0 detected @ 0xe800
May  1 20:50:19 localhost kernel: Catweasel MK4 card with 6581 as chip 1 detected @ 0xe800
```

Ok, now the driver see my 6581 chip mounted!

If you run:

```
[root@localhost ice]# more /proc/hardsid
Catweasel MK4 configured for port 0xe800 (firmware 0.029)
SID type 6581
Catweasel MK4 configured for port 0xe800 (firmware 0.029)
SID type 6581
```

we obtain that information asking the process. However, during music reproduction, this process

gives more information about the operation being done.
Now its time to compile sidplay2: this is simple: going into each downloaded module and give:

```
        ./bootstrap
        ./configure
        ./make
```

and then from root:

```
        ./make install
```

After some minutes I try running the sidplay2: it recognize the card as :

```
[root@localhost sidplay]# ./src/sidplay2 -h
Syntax: ./src/sidplay2 [-<option>...] <datafile>
Options:
 --help|-h    display this screen
 --help-debug debug help menu
 -b<num>      set start time in [m:]s format (default 0)
 -f<num>      set frequency in Hz (default: 44100)
 -fd          force dual sid environment
 -fs          force samples to a channel (default: uses sid)
 -nf[filter]  no/new SID filter emulation
 -ns[0|1]     (no) MOS 8580 waveforms (default: from tune or cfg)
 -o<l|s>      looping and/or single track
 -o<num>      start track (default: preset)
 -O<num>      optimisation level, max is 1 (default: 1)
 -p<num>      set bit precision for samples. (default: 16)
 -s[l|r]      stereo sid support or [left/right] channel only
 -t<num>      set play length in [m:]s format (0 is endless)
 -<v|q>       verbose or quiet (no time display) output
 -v[p|n][f]   set VIC PAL/NTSC clock speed (default: defined by song)
              Use 'f' to force the clock by preventing speed fixing
 -w[name]     create wav file (default: <datafile>[n].wav)
 --hardsid    enable hardsid support

Home Page: http://sidplay2.sourceforge.net/
```

If no card is detected, the –hardsid option is not showed.

I than run sidplay2 with a tune, but I see that the clock goes fastest: 50 second of music played in 8 seconds. And that no sound is listen from the card!!

I try Goattracker (that support MK3 card): it detects the card, but no sound is given.

At this point I go to investigate and:

- Found that my CD in line was muted (I never test it before, as I listen only sid music and the only 2 CDs I listen are the “Galway Project” that I listen only in my old 2.4 kernel where the line was not muted). But this not change the thing.
- Change the given cable that connect the MK4 card with the Sound Card with a CDROM one that was new. But this not change the thing.
- Change the 6581 chip with the one I have into my C128 and that for sure it sounds perfectly. But this not change the thing.

I so contact Simon and now I see I was using the wrong suggested driver: I so download the experimental branch and the new CVS *HEAD* version (that was modified).

I try the new *HEAD*, but here as soon as sidplay2 start to play, the Linux kernel is frozen and I must reset the pc (ops, where is the reset??? From when I did not use Windows anymore, that keys is not used...)

I then install the experimental branch and now I can listen the sound form the sid chip!! However the speed is always hypersonic, but now I'm thinking that the card is working and maybe there are some problems into the driver.

I so apply the patch to vice and compile it (the first time it not adds the hardsid support, so I must switch on manually after). If I ran x64 with a test tune prg, now the speed is corrected, but the sound is not good:

- The volume is extremely low (it must be putted at the maximum level to listen to something)
- The notes seems to by played not so in sync
- Sometimes the notes seems muted when they have to play
- If you move a windows into the desktop the sound goes killed or goes very slow

Help, now it is not good: I see that thing occurs to Windows users too and so it may be related with the card. I so contact Individual computer for help.

Conclusion

Unfortunately after 3 months I have not got yet an answer from Individual Computer, so I cannot know if my card is one of the few early realized that was not tested for DC-DC problem (if DC-DC component is not working fine, users had report my same music problems with the card).

As if the card is damaged I can suspect that Sid chip could be damaged (I not tested if now the chips work correctly in the C64) by using with it, so for the moment I leave the card not connected to the PC :(

I hope to have soon answers and so test again the card and listen to Sid sound.

The positive thing that I see is that Linux driver are well done (for example the driver automatically choose the right sid chip -if available into the card- that is specified into the sid chip model inside the tune and so all the software that use those driver, have this features already active) as they are based onto hardsid ones that are tested by long time.

Last thing: if you have a MK4 card, go to read what is write into the bottom of it ... interesting...

SIDin 8 end