GSDim 10

"Nicole Eggert"

Vice snapshot with Vice palette

Made with the GIMP from a NE photo
and converted to C64 320x200
Hires Mode Bitmap
by Stefano Tognon
in 2006

"EVS memory"
...

Ice Team Free Software Group

SIDin 10
version 1.00
16 July 2006

SIDin 10
version 1.00
16 July 2006

SIDin Contents

General Index

Editorials

Stefano Tognon <ice00@libero.it>

Hi, again.

This is the 10° issue of the magazine.
I did not imagine that when I started to wrote SIDin, I will reach a so high number of issues.
I very hope to continue with this trend.

I report here a comment by Frantic about the previous number that corrects some my mistakes:

Some comments on my player was clearly wrong (the tune is long not because of filtersweeps but because of different combinations of the songsequences, and also, not mentioned explicitly but still, my player wouldn't benefit from being placed in ZP since the code to copy it would outnumber the bytes earned), but it's a nice effort and a nice issue in general. I like it! Player coding nerds unite! Don't think I have time to participate in Tiny Sid compo #2 though, which is too bad since that would give me the opportunity to demonstrate that almost the same player structure would indeed be capable of complex sounds too (although I didn't focus on that in the former compo entry tune).. ;)

Thanks for this.

In this number we see some entries from the inside of the Tiny Sid Compo 2, the others will be in the next number. I think that we need more musician/coder that are able to produce tiny music, as this little music are essential for having a full features minigame, so don't blame if in all numbers I present such mini players :)

In the end article, I show how I use the new Goattraker 2 for creating a cover of a tune. This was for me a way to learn the new features of the program. I so don't go to describe all the features of GT2, as the *Readme* is full completed of all you need for use the program, but all the process involved for produce the tune.

Bye
S.T.

News

Some various news of players, programs, and competitions:

- Goattracker 2.23/2.35
- JCH New Player 21.g4 Final
- Fap 0.2.3
- SID DUZZ' IT v2.0 beta
- SIDPLAY for Mac OS X 3.4
- HVSC Update #45
- JCH New Player 21.g5 Final
- Tiny Sid 2 Compo result
- Goattracker 2.4/2.5
- Big Sid Compo

Goattracker 2.23/2.35

The Cadaver PC music editor available at http://covertbitops.c64.org is available:

v2.23

- Fixed SHIFT+INS in wavetable (inserts $E9 $00 now as it should).
- Added SHIFT+U to unlock table scrolling.
- Added SHIFT+F5/F6 to change speed multiplier.

v2.24

- Added SHIFT+F7 to edit the hardrestart ADSR parameter.
- Added SHIFT+F8 to switch between 6581 and 8580 SID models.

v2.25

- Added detail to table overflow error message in relocator.
- Added stop of playback if a tablepointer gets pointed directly to a jump (would bug in packed/relocated tune).
- Added relocator error message for the above case.
- Added relocator error message for illegal song restart position.
- Not possible to save erroneous packed/relocated songs anymore (now that there is adequate error location).
- Added SHIFT+,. to move song startposition on all channels and restart last playmode

v2.26

- Fixed relocator complaining of FF parameter in speedtable.
- To increase stability, in multispeed mode relocator doesn't use the "same gatetimer/1st wave" optimization.
- Changed Shift+O to optimize the currently edited table, not just speedtable.
- Added an option to optimize everything with the SHIFT+ESC clear function. This clears unused patterns, instruments and table-entries.

v2.27

- Fixed optimize function with different pattern lengths.
- Added marking & copy/pasting orderlist data like patterns & tables

v2.28

- Added sort of mouse control.
- Sound reinitialization when changing multiplier or other parameters is now faster.

v2.29Beta

- Internal reorganization.
- New assembler-based relocator that leaves out unused portions of player code.
- Instead of choosing a player version, one can choose combinations of player features.
- Relocator does not save selfcontained duplicate parts of tables.
- Added keys 0-9 & A-Z in the fileselector to move to the corresponding part of the filelist (if

found).
- Added SHIFT+H to calculate left/right shifted "hifi" speedtable values, according to the frequency of a note.

v2.3
- Fixed songdata optimize function.
- Fixed relocator with no-hardrestart instrument in slot 1.
- Fixed relocator handling of effect C (Set cutoff).

v2.31
- Fixed effect copy/paste.
- Added possibility to disable pulseskipping with /O command line option. Warning: causes huge rastertime increase.
- Tempo 2 is allowed. See the tips section (3.6) for details.

v2.32
- Maximum pattern length increased to 128. Do not complain of "pattern too complex" errors :)
- Optimized playroutine init in case of only 1 subtune.

v2.33
- Fixed a bug where relocator was checking pattern 208 (illegal).
- Optimized playroutine in case there is no keyoff/keyon used.
- Optimized playroutine in case there are no pattern effects used.
- Optimized playroutine channel variables in case there is no transpose, repeat, wavedelay and pattern effects.
- If not using the last instrument (63), you can control song startup default tempo by using its Attack/Decay parameter.

v2.34
- Added relocator optimization for using 1 or 2 channels only. To enable, you must use only channels 1 or 1,2 and not be using sound effect support or zeropage ghostregs.
- Turning off buffered writes in relocator options works now even if sound effects or zeropage ghostregs are selected (they will be turned off too).

v2.35
- *Auto-creation of speedtable entries will always generate a new entry (except for song loading), duplicate values will be optimized by the relocator.*
- *Pressing SHIFT+ENTER over a speedtable-using pattern command while the parameter is zero, goes to the next empty speedtable entry and sets the command parameter to point to that entry.*

For version 2.25 it is available even for Mac OS X.

JCH New Player 21.g4 Final

Released in January 2006 the player coded by Laxity to use with JCH

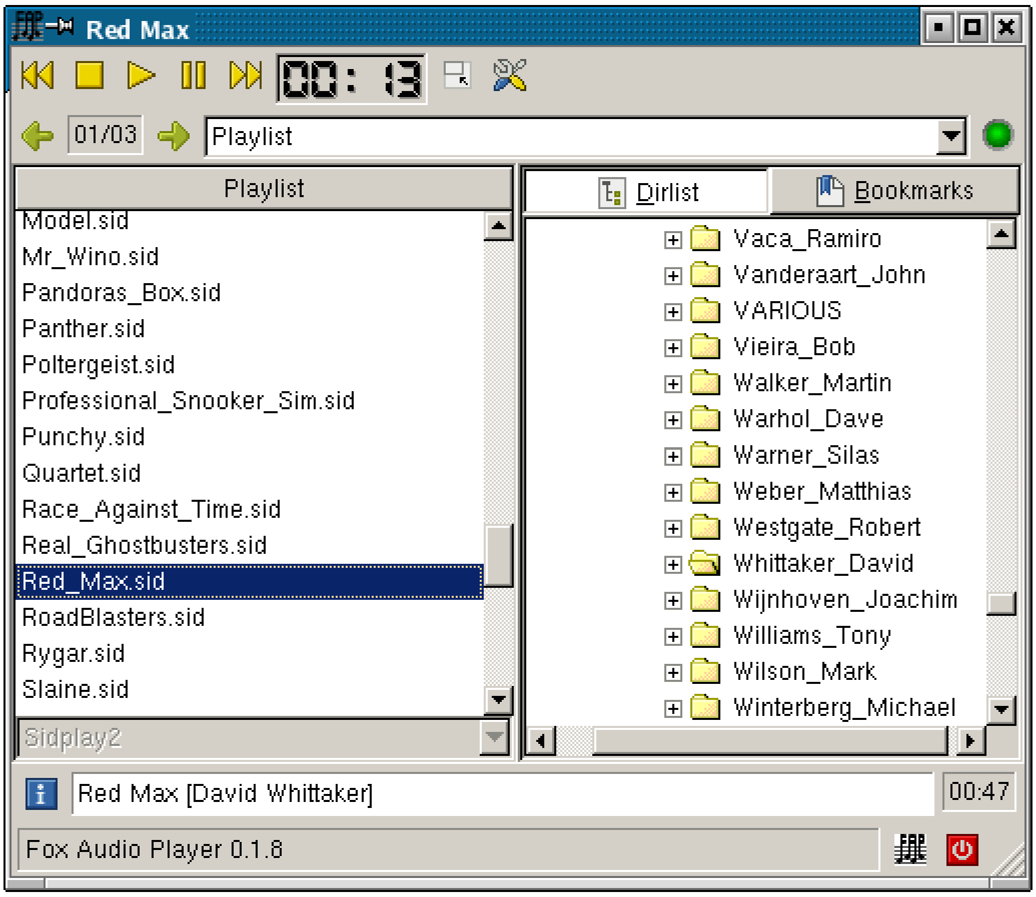
Updates in Final:
- Fixed a bug in the vibrato feel, that caused an overflow and made the note go out of tune..
- Fixed the synchronization (initialization) bug.

Download from:
http://noname.c64.org/csdb/getinternalfile.php/14838/NewPlayer v21.G4 Final.zip

Fap 0.2.3

Fap is a new simple and lightweight audio player by "Dr. Fiemost".

It features a dirlist view, where you can browse through your disk and bookmark directories in which you've stored your music, and a playlist view which is automatically filled with all the supported files contained in the selected directory



Red Max [David Whittaker] 00:47

Fox Audio Player 0.1.8

Formats currently supported:

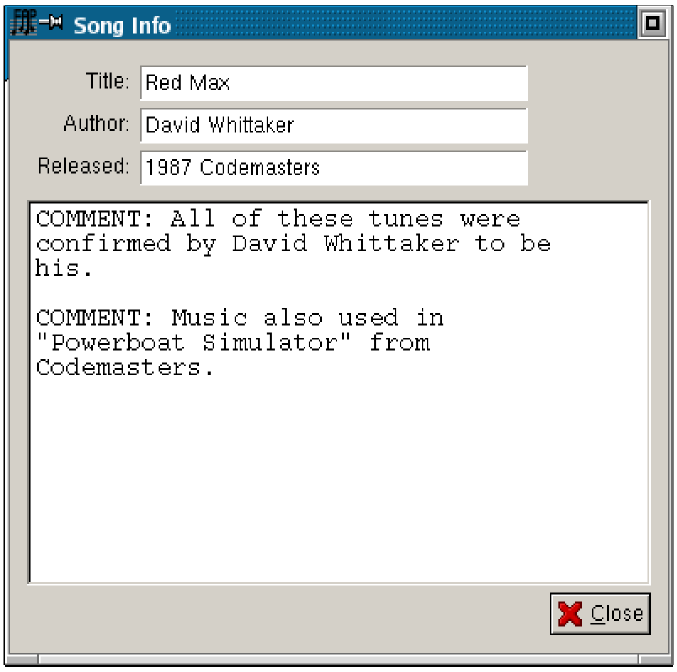
- modules: mod, s3m, xm, it
- compressed modules: xmz, itz, mo3
- sidplay: sid (includes songlength db support)
- ogg-vorbis:ogg
- others: wav, voc, au

Changelog:

0.1.0 2006-01-12
- first public release

0.1.1 2006-01-16
- fixed a couple of bugs

0.1.2 2006-01-23
- Some improvements with songlength db usage
- Use memory mapped files for OGGs



Title: Red Max
Author: David Whittaker
Released: 1987 Codemasters

```
COMMENT: All of these tunes were
confirmed by David Whittaker to be
his.

COMMENT: Music also used in
"Powerboat Simulator" from
Codemasters.
```

reading
- Modules can now play at 48KHz
- 8 bit output is disabled because it's broken
- Support for WAV, VOC and AU files

0.1.3 2006-01-28
- Some minor fixes and changes

0.1.4 2006-02-04
- Fixed some regressions introduced in last release
- Always start with ascending sorting
- Added a dialog for displaying song info

0.1.5 2006-02-22
- Experimental support for hardsid
- Now it's possible to disable SID filter
- Sort type is now saved again
- Various bugfixes

0.1.6 2006-03-01
- Doubleclick on a song start playing
- Display current and total subtunes
- Select between playlist and song mode: the first play all the songs in the playlist, the latter play only selected song
- Various bugfixes

0.1.7 2006-03-06
- Display OGG comments with correct codec (UTF8)
- Experimental support for mp3 using libmad
- Various bugfixes

0.1.8 2006-03-12
- STIL support
- Now it's possible to disable unwanted plugins at compile time
- Various bugfixes

0.1.9 2006-03-27
- If both "filename_PSID.sid" and "filename.sid" exists display only first
- Various bugfixes

0.1.10 2006-04-09
- Fixed a possible error with samplerate conversion
- Some minor fixes and changes

0.1.11 2006-04-23
- When a song ends and directory has been changed start playing new playlist
- * Added ffmpeg plugin with support for wma, ra, aac and other formats (need recent ffmpeg form cvs, enable with --enable-ffmpeg-plugin)
- Bookmarks are associated with plugins
- Added playlist support (pls, m3u)
- Sndfile plugin now detect song duration
- Formats supported by the sndfile plugin are automatically detected at runtime
- Fixed a memory leak
- Some minor fixes and changes

0.1.12 2006-05-10
- Fixed an error when configuring with --disable-src
- Some minor changes

0.2.0 2006-05-27
- Now it's possible to create and save custom playlist

0.2.1 2006-06-04
- Fixed opening of compressed modules when extension is not lower case
- Fixed comments for platform other than Unix/Linux
- Ogg and Mad backends can play 8 bit
- Some minor changes

0.2.2   2006-06-12

- Fixed config dialog: cancel didn't work as expected
- Fixed a bug when saving settings
- Added Game Music Emulator backend
- Fixed a bug with non-PSID file filtering

0.2.3   2006-07-15

- Added random sorting option to playlist
- Do not reopen STIL and songlength DB if HVSC path is not changed
- Added two options to the configure script:
    - --disable-psid-filter - disables non PSID filtering in Sidplay backend
    - --disable-noise-shaping - disables noise shaping in Mad backend
- Fixed opening files with spaces in filename in Modplug backend

Download at http://www.geocities.com/drfiemost/fap.html

SID DUZZ' IT v2.0 beta

SID DUZZ' IT is the C64 music editor by GRG/Shape. A public beta release was released into the begin of this year:

- Player sources have been rewritten and optimized.
- New filter effects and detuning in the player.
- Some editor routines have been fixed.
- Fixed two bugs in the dumper. (Tracks that was off was not dumped properly.)
- Load/Save/Dump and dos commands are new.
- Works nicely on CBM, CMD and IDE64 drives.

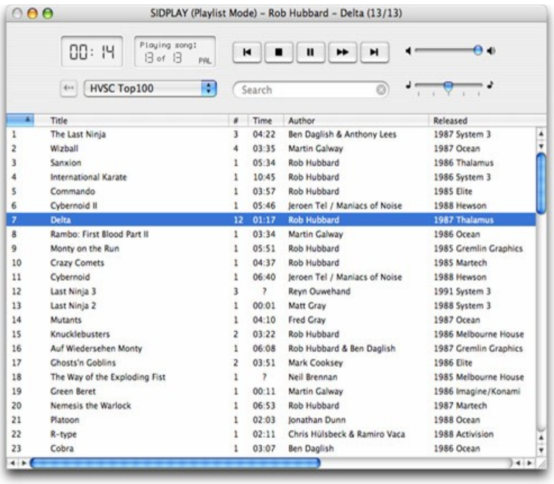
Download at http://home.eunet.no/~ggallefo/sdi/

SIDPLAY for MAC OS X 3.4

SIDPLAY for Mac OS X is a complete rewrite of SIDPLAY for Mac OS, which was originally released in June of 1996. Version 3.4 was released in March 2006

News in 3.4:

- SIDPLAY is a universal binary now and runs great on the new Intel-based Macs
- Added Spotlight plug-in and changed search feature to use Spotlight
- Added scrollbar for info window
- New anti-aliased oscilloscope rendering using Quartz 2D
- Update to latest libsidplay2 source
- Updated to *PSID64* *0.7*
- Fixed oversampling problem
- Fixed bug in PRG export if path to SIDPLAY application contains a space character
- Added songlength display for RSID tunes
- Added display of SID chip model settingFixed various major and minor bugs


SIDPLAY (Playlist Mode) – Rob Hubbard – Delta (13/13)
00: 14
Playing song: 13 of 13 PAL
HVSC Top100
Search

	Title	#	Time	Author	Released
1	The Last Ninja	3	04:22	Ben Daglish & Anthony Lees	1987 System 3
2	Wizball	4	03:35	Martin Galway	1987 Ocean
3	Sanxion	1	05:34	Rob Hubbard	1986 Thalamus
4	International Karate	1	10:45	Rob Hubbard	1986 System 3
5	Commando	1	03:57	Rob Hubbard	1985 Elite
6	Cybernoid II	1	05:46	Jeroen Tel / Maniacs of Noise	1988 Hewson
7	Delta	12	01:17	Rob Hubbard	1987 Thalamus
8	Rambo: First Blood Part II	1	03:34	Martin Galway	1986 Ocean
9	Monty on the Run	1	05:51	Rob Hubbard	1985 Gremlin Graphics
10	Crazy Comets	1	04:37	Rob Hubbard	1985 Martech
11	Cybernoid	1	06:40	Jeroen Tel / Maniacs of Noise	1988 Hewson
12	Last Ninja 3	3	?	Reyn Ouwehand	1991 System 3
13	Last Ninja 2	1	00:01	Matt Gray	1988 System 3
14	Mutants	1	04:10	Fred Gray	1987 Ocean
15	Knucklebusters	2	03:22	Rob Hubbard	1986 Melbourne House
16	Auf Wiedersehen Monty	1	06:08	Rob Hubbard & Ben Daglish	1987 Gremlin Graphics
17	Ghosts'n Goblins	2	03:51	Mark Cooksey	1986 Elite
18	The Way of the Exploding Fist	1	?	Neil Brennan	1985 Melbourne House
19	Green Beret	1	00:11	Martin Galway	1986 Imagine/Konami
20	Nemesis the Warlock	1	06:53	Rob Hubbard	1987 Martech
21	Platoon	1	02:03	Jonathan Dunn	1988 Ocean
22	R-type	1	02:11	Chris Hülsbeck & Ramiro Vaca	1988 Activision
23	Cobra	1	03:07	Ben Daglish	1986 Ocean

Download from: http://www.sidmusic.org/sidplay/mac/

HVSC Update #45

HVSC Update #45 was released on April at www.hvsc.c64.org

After this update, the collection should contain 32,000 SID files!

This update features (all approximates):

```
  663 new SIDs
 1004 fixed/better rips
   11 fixes of PlaySID/Sidplay1 specific SIDs
    8 repeats/bad rips eliminated
  466 SID credit fixes
  135 SID model/clock infos
   17 tunes from /DEMOS/UNKNOWN/ identified  :-)
   10 tunes moved out of /DEMOS/ to their composers' directories
    7 tunes moved out of /GAMES/ to their composers' directories
```

Main Composers featured in this update:

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

DRAX	Laxity	Barry Leitch
Jeroen Tel	Arthur Keleti (Arthur)	Dwayne Bakewell
Richard Bayliss	Bizet	Butka
Dalezy	Dr. Piotr	Hein Holt
Josstintimberlake	Harold Klink (NEW)	Klax
Monk (NEW)	Nata	Psycho
Randy	Reason	Rygar
Sidder	Skysurfer	Square (NEW)
TC	Van Styer	Watchman (NEW)
Wuiti	Xonic the Fox (NEW)	Yaemon
Yuro	Zabutom (NEW)	

JCH New Player 21.g5 Final

Released in May 2006 the player coded by Laxity to use with JCH

Updates in Final:

- Fixed the periodical retrig bug, in liveplay mode.
- Implemented stop ($fe in pattern list). Forgot that completely in pre .g5. DOH!

Download from:

http://noname.c64.org/csdb/getinternalfile.php/22927/NewPlayer v21.G5 Final.zip

Tiny Sid 2 Compo result

The second Tiny Sid Compo was concluded on 21 April 2006 with this result:

```
256B RESULT:
            Pos. Points    Title                          Author
             -----------------------------------------------------------------
              1     96     BLOCK ACID DUB                 Frantic
              2     94     XL5                            Stefano Tognon
              3     86     Miniature Rhapsody             Eric Odland
              4     55     Back To Basics                 Jaymz Julian

512B RESULT:
             Pos. Points    Title                          Author
             -----------------------------------------------------------------
              1    103     Plaster                        GRG
              2     92     RM                             Stefano Tognon
              3     64     Resolution                     Eric Odland
              4     57     Mini Digi Drum                 Leif Bloomquist
```

Download the result pack with comments from: http://digilander.iol.it/ice00/tsid/tinysid2

Goattracker 2.4/2.5

The editor has reach a major update as the internal file format is changed as new features was addes:

v2.4:
- Added one bit of accuracy to pulse modulation speed. A new song- format (identifier GTS4) reflects this, old songs will be converted upon loading.
- Added possibility to execute pattern commands from the wavetable.

v2.41
- Fixed player code assembly errors when using some pattern commands only from the wavetable.
- Fixed bug with command BXY when command AXY is not used.
- Fixed SHIFT+O removing table entries pointed to from wavetable pattern commands.

v2.42
- Fixed frequency table size determination.
- Added SHIFT+ENTER when parameter is zero to get you into an empty table entry for 8XY, 9XY and AXY pattern commands as well.
- Added adjustment of table views when changing instrument number in unlocked table scrolling mode.
- Added ENTER over nonzero instrument number (in patterns) to go editing that instrument.

v2.43
- Added duplicate pattern detection/removal to song optimize.
- Added pattern shrink/expand (SHIFT+O,P).
- Added pattern join/split (SHIFT+J,K).

v2.44
- Fixed editor playroutine to not execute tick n commands, when a command has already been executed through the wavetable.

v2.45
- Added possibility to disable realtime pattern command optimization at tick 0 (/Rxx command line option or click the "RO" text on the statusbar) for tempo-independent portamento & vibrato.
- To make room on the status bar for the realtime optimization status indicator, finevibrato & pulse optimization indicators have been shortened to "FV" and "PO".

v2.46

- Added transpose of speedtable portamento speeds (SHIFT+Q,A,W,S) SHIFT+W,S (octave transpose, multiply/divide by 2) also work in pulse & filtertable.
- When editing the wavetable, RETURN & SHIFT+RETURN work for wave-table executed commands as in patterns (go to table position pointed by command, create new speedtable entries)

v2.47

- Added experimental random delay to write of reSID register, in order to make potential ADSR-bugging on C64 audible. The amount of delay can be adjusted with /Z command line option.
- SHIFT+L in pulsetable creates repeated "set tempo" commands if speed is greater than 127.
- Unbuffered playroutine timing slightly changed.

v2.48

- Fixed playroutine to use configurable baseaddress again.

v2.49

- Added /G command line option, guard 1stwave/gatetimer parameters. When this is on (default), only silent waveforms can be entered as firstwave (only thing that makes sense) and editing gatetimer of one instrument affects all of them, as otherwise bugs in playback result easily. Use /G0 to turn off.

v2.5

- Fixed "optimize everything" removing the tempo setting in instrument 63.
- Fixed default tempo of multispeed tunes in packer/relocator.

Big Sid Compo

This is a tentative to produce some big sid tune where more composers will work onto the same peaces using the same editor. I try to make this in a compo like style, so maybe this could stimulate the composers.

Check at: http://digilander.iol.it/ice00/tsid/bigsid and look at the forum

Submission is open until 2 December 2006

Richard Bayliss Interview!

by Stefano Tognon

This time I go to interview Richard Bayliss that maybe you know for having played one of his innumerable number of games. Richard is very active and I think that in game music he is very talented.

The interview was made in this month.

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

I am 27 years old, will be 28 on Christmas Day. I work in a pharmaceutical distribution warehouse, picking and packing orders as well as maintain the warehouse, keeping it the way it should be. I don't mind the work that I do. I am a keen C64 programmer & musician (You might have already known before). I originally wrote games since 1991 using the Shoot 'Em Up Construction kit. It wasn't until 1994 I learned BASIC programming games and composing music, after purchasing a utilities tape from a Public Domain library and played around with the editors. It wasn't until 1999 I learned to code games in Turbo Assembler, then 2003 (or was it 2004) was where I started Cross-Assembly :o)

Maybe you are the unique scener around today that can release a game a week? (even if some people blame for the not optimal gameplay of many of your past games, but music is always good). Have you some games now in develop?

Yes, one of which (I'm hoping to get finished before Christmas 2006, although there is a dealy possible) is called Sub Hunter. This game is based on the Vic 20 version of Sub Hunt by Mastertronic software, but is more up to date with parallax scrolling, a few other sub games. Secondly is a co-op game production with The Art Ravers, called "Crash Course", a kind of maze game, where you are playing against an opponent and have to drive around the maze, collecting items to boost your score on over 20 different courses. This sounds interesting.

From time to time you release a Music collection onto disk. Have you already planned a new one in the future?

Ah, you're on about my DMC albums, etc. Yes, I am hoping to do a few more music collections in the future, but I have not confirmed when.

What are the music editors that you use to composer sid tunes and why have you choose them?

At first it used to be the Music Assembler and Music Mixer Module 5 (Formly known as Voicetracker), because it was small in size and fun to compose. I tried Future Composer V4, it wasn't that easy to use. I then moved on to DMC V2.0 which was a whole lot easier to use, as the utility disk had example tunes. I needed to play around with the example tunes before I was able to create my own instruments. The composer which I used the most were between DMC V4.0 and DMC V5.0. At first I originally had a version of DMC V4.0B by Graffity/Panic Design, found on the Magna Musica utility disk. Sadly this version had a bug in the tune packer. So it wasnt until 2001 I had found a 100% working version of this composer on the Internet (Between 1995-2000 I had no Internet access at home at the time).

I got in contact with a few people and one person in particular had introduced me to DMC V5.0. I had found this to be one of the most complex version of DMC at the time, so I had to experiment a lot with the features that were included in the DMC music composer. Occasionally, but not all the time, I compose on Goat Tracker music editor, but don't feel comfortable with the sounds that are used in this program. It does not beat the traditional real C64 old/new SID chip.

Are there some particular features that you want to find in an editor that are missing from to-day ones?

Goat Tracker lacks the *real C64* SID environment, which I think would be great if it could emulate exactly the same thing. Also with this composer, I miss the good old DMC V5 environment, for the sector editor, where you could try and control filtered sounds to go up or down like in DMC V5 itself. It is not a bad composer though.

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

Real machine. Emulators cannot beat the real C64, because of its capabilities to sound and vision handling. With an emulator, like VICE it does not sync sound correctly when playing C64 games. For example a game written in SEUCK. Press fire to shoot something, your weapon appears but the sound comes after the weapon move slightly.

6581 vs 8580 chip: any (musical) preference?

I like the new SID (8580), because of the features with the sound filters. Especially when it comes to using cool composers like DMC V5.

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

The music I wrote for Fake Game. It was that rubbish, but I only composed it as a little joke for the title screen. It made me laugh after I composed it. I suppose I really should have preserved it for an entry for the 2006 Crap Game compo Block Defenz, but instead I ***programmed*** the music with the Turbo Assembler. :)

Who are your best sid authors?

The best SID authors? In the past, Martin Galway did amazing ambient tunes, like Insects in Space and Parallax. I loved both of those tunes, not to mention the title music for Miami Vice. Secondly has to be Rob Hubbard, one of the greatest legends. I loved his work on Mega Apocalypse. Awesome sounds, but his best work had to be Lightforce. Thirdly Jeroen Tel of Maniacs of Noise. He done some brilliant tunes for games between 1988 and 1990. Finally Edwin Van Santen (Who sadly died earlier this year). He created excellent tunes for 20CC, I remember a disk which had the entire 20CC music collection between 1988-1991, featuring many of his great tracks. I'm also going to be working on a tribute demo to EVS, which will feature some of his best tunes, and messages on scroll text from various people of the scene today.

The best scene musicians of today has to be Drax/MON, Intensity/Cosine, Fanta, Laxity and a few others.

What are the best sids ever in your opinion?

It is hard to say as there are loads of great SIDs stored on the HVSC. Except that, I wish people would cut down on the hardcore drum 'n bass style music, as to my opinion Drum 'n Bass is not music at all - even the real drum 'n bass, but rubbish in general. SID has not been built for awful (what people call) music like this. Of course, it is up to the SID musician themselves :)

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!

I'd like to say a big thank you to all of those who are reading this article right now. I am impressed that although the C64 is over 22 years old, people are still supporting this great machine, like Cronosoft, the CSDB and many other C64 enthusiasts. This impresses me a lot as I still like using the C64 today, and still keep many of the old C64 classics.

Secondly I'd like to say how awful it has been to hear about the death of EVS/20CC, in May 2006 and I would like to pay tribute to the great scene C64 musician. To show respect, I'll be releasing a demo by the end of July 2006 featuring a selection of his great tunes, and a few messages from other sceners, as well as myself (Possibly with a note file included). EVS's great tunes will be remembered.

Tiny Sid 2 (part 1)

by Stefano Tognon <ice00@libero.it>

In this first part I go to show 4 entries of the Tiny Sid 2 Compo, two 256 bytes entries and two 512 bytes entries. The other 4 will be in next number.

The tunes can be downloaded at: http://digilander.iol.it/ice00/tsid/tinysid2/tinysid2_res.zip

XL5 (256b)

As I like the Jeff “X-Large” series of tunes, I have choose the number five for trying to cover it in 256 bytes. The tune is about 0:36 seconds long and is quite simple, but with a good result that you will listen for some minutes for sure.

A look to the sid2mid output, gives and idea of the tune:

```
          Voice 1                     Voice 2                     Voice 3
Time      Note  Freq  PW  WF ADSR VL  Note  Freq  PW  WF ADSR VL  Note  Freq  PW  WF ADSR VL   Filter
=============================================================================================================
00:00.04  >G#1<   51 2048 40 00c9 --  >C-1<   32     0 10 0069 --  >C-1<   32     0 10 0029 --  L__ 1__ 380 f
00:00.04   +++    51 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 2d0 f
00:00.05   +++    51 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 260 f
00:00.06   +++    51 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 220 f
00:00.06   +++    51 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 1f0 f
00:00.07   +++    51 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 1c0 f
00:00.07   +++    51 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 190 f
00:00.08   +++    51 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 160 f
00:00.09  >G#2<  103 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 440 f
00:00.09   +++   103 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 390 f
00:00.10   +++   103 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 320 f
00:00.10   +++   103 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 2e0 f
00:00.11   +++   103 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 2b0 f
00:00.12   +++   103 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 280 f
00:00.12   +++   103 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 250 f
00:00.13   +++   103 2048 40 00c9 --   +++    32     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 220 f
00:00.13  >G#3<  207 2048 40 00c9 --  >G#1<   51     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 500 0
00:00.14   +++   207 2048 40 00c9 --   +++    51     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 450 0
00:00.15   +++   207 2048 40 00c9 --   +++    51     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 3e0 0
00:00.15   +++   207 2048 40 00c9 --   +++    51     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 3a0 0
00:00.16   +++   207 2048 40 00c9 --   +++    51     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 370 0
00:00.16   +++   207 2048 40 00c9 --   +++    51     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 340 0
00:00.17   +++   207 2048 40 00c9 --   +++    51     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 310 0
00:00.18   +++   207 2048 40 00c9 --   +++    51     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 2e0 0
00:00.18  >G#1<   51 2048 40 00c9 --  >G#2<  103     0 10 0069 --   +++    32     0 10 0029 --  L__ 1__ 380 f
```

The voice 2 and 3 plays the same pattern of voice 1, but with a delay of 2 and 4 notes. Else, it uses a low sustain, so there is a sort of low echo. All notes are equal length and instrument are fixed (with the exception of filter)

Those points are very important: we have only one track to play for each voice, without the problem of different note length and changing of instruments. However, the implementation of the filter (that change each tick and use different resonance) is out of question for a 256 byte tunes with all the note we have to play.

I agree that the filter is what that give to this tune his style, but the result without it is acceptable having in mind the 256 bytes restriction.

When I take all the notes for imagine how the track/pattern should be written, I had the first delusion: the used notes are 17 and so I cannot use nibble to store them. However, I find that the tune could be represented by pattern and this means a little of compression that can be achieved.

Here you can see the final pattern disposition (the first I wrote where a little different but more natural to use):

```
[1] G#1 G#2 G#3
[1] G#1 G#2 G#3
[1] G#1 G#2 G#3
[1] G#1 G#2 G#3
[1] G#1 G#2 G#3
[2] F#1
[1] G#1 G#2 G#3
[1] G#1 G#2 G#3
[1] G#1 G#2 G#3
[1] G#1 G#2 G#3
[1] G#1 G#2 G#3
[2] F#1
[3] G#1 G#2 D#4 G#1 G#2 D-4 G#1 G#2 C-4 G#1
[4] G#2 A#3 G#1 G#3 G#2 A#3
[3] G#1 G#2 D#4 G#1 G#2 D-4 G#1 G#2 C-4 G#1
[5] D-4 D#4 G#2 C-4 G#2 G#3
++
[3] G#1 G#2 D#4 G#1 G#2 D-4 G#1 G#2 C-4 G#1
[4] G#2 A#3 G#1 G#3 G#2 A#3
[3] G#1 G#2 D#4 G#1 G#2 D-4 G#1 G#2 C-4 G#1
[6] D-4 D#4 G#2 G-4 G#2 G#4
[3] G#1 G#2 D#4 G#1 G#2 D-4 G#1 G#2 C-4 G#1
[4] G#2 A#3 G#1 G#3 G#2 A#3
[3] G#1 G#2 D#4 G#1 G#2 D-4 G#1 G#2 C-4 G#1
[5] D-4 D#4 G#2 C-4 G#2 G#3
++
[3] G#1 G#2 D#4  G#1 G#2 D-4 G#1 G#2 C-4 G#1
[4] G#2 A#3 G#1 G#3 G#2 A#3
[3] G#1 G#2 D#4  G#1 G#2 D-4 G#1 G#2 C-4 G#1
[7] D-4 D#4 G#2 G#4 G#2 A#4
++
[8] F-1 F-2 G#4 F-1 F-2 G-4 F-1 D#4 F-2 C-4 F-1 D#4 F-4 F-1 C-4 D#4 F#1 F#2 F#4 F#1 F#2 F-4
[9] F-1
[10]C#4 F#2 G#3
[9] F-1
[11]A#3 F#2 C#4 F#1 G#3
[8] F-1 F-2 G#4 F-1 F-2 G-4 F-1 D#4 F-2 C-4 F-1 D#4 F-4 F-1 C-4 D#4 F#1 F#2 F#4 F#1 F#2 F-4
[2] F#1
[10]C#4 F#2 G#3

[2] F#1
[11]A#3 F#2 C#4 F#1 G#3
```

I so wrote the tune that use the above information and I had it reach 309 bytes. No, we are too much out.

However, adding the filter and have a 512 bytes entry should be a good solution, but I want to optimize ever better that better. I so change little things at each step and save the source of modification (that you find inside the released tune) and finally I could stay into 256 bytes.

Here the optimization used:

- I switch the basic startup SYS instruction by loading directly the tune at address $0326. This seems to be the better way to made the tune running: you save 8 bytes.
- I used to synchronize the tune with raster position (interrupt is disable with SEI), then cycle to the 3 voices (I start from the first and then go to the others). Using instead the Frantic technique that use SBX undocumented instruction (and cycles from the 3° voices to the 1°) we can save another 4 bytes
- The pattern 1 was long 15 notes, but I divide into only 3 notes. I obtain to save of 4 bytes. One more byte I save using INX for passing from 0 to 1 in an initialization
- 2 more bytes were saved using the LDY absolute,X instruction in two points
- The best reduction is of 15 bytes and come out from having the pattern/track pointer already set to 0. I have carefully look at address page 0 to find 6 location that are in sequence of 0,+7,+7 and are empty. I have compare them after loading a program and at startup of the machine. Is not granted that if you made other things with the C64 before loading the tune they are still 0.
- Other 3 bytes were saved porting the initialization of control register of the voices inside the

main cycle (and remembering that $11 is for voice 2 and 3 and $41 for voice 1)

- Sustain/Release of voice are: $C9, $69 and $29. You should note that release is never activated into the tune, so we can look only to attack. The idea is to use $30, $60 and $90, as $60=$30<<1 and $90=$60<<1. So 1 bytes are saved using the shifts.
- Other 4 bytes are saved by using a zero page location that is already set to 1 at startup. We can use even other little above value, as this is only the delay for starting to play the first note. Fortunately one location was found.
- 8 bytes were saved by this idea: we have that one pattern of note is terminated by a 0 value. In some cases, one pattern is followed only by another one, like: 4 and 3. If we remove the end of pattern 4, pattern 3 will be executed as it come sequentially in the source.
- Last bytes were saved by removing a redundant instruction and by moving the init of volume in the end of cycle

Well I hope that this description is useful for learning some optimization techniques

```
; XL5
; 256b SID music
; by Stefano Tognon 2006
; This is a cover of Jeff "X-Large 5"

; step1: 309 bytes (normal code of the tune)
; step2: 301 bytes (use $0326 start up)
; step3: 297 bytes (use Frantic sbx/sync thecnique)
; step4: 292 bytes (rearrange pat1 and inx)
; step4: 290 bytes (use ldy directly)
; step5: 277 bytes (use 0 value already setted)
; step6: 274 bytes (use common set of control register+fix Frantic code) +1 must be added
; step7: 273 bytes (use different SR initialization)
; step8: 269 bytes (use location defined to 1 at sturtup)
; step9: 265 bytes (rearrange patterns removing END)
; stepA: 263 bytes (new rearrange patterns removing END)
; stepB: 261 bytes (removing other 0 bytes)
; stepC: 259 bytes (remove redondant instruction)
; stepD: 256 bytes (move volume init, lost first note, but nop not needed)

  processor 6502

curTrack = $57    ; this and +7 and +14 are 0 at sturtup of after loading
curPat   = $58    ; this and +7 and +14 are 0 at sturtup of after loading
duration = $2B    ; init at 1
DUR = 8

   ;org 2049
   ;.byte $0b,$08,$e8,$03,$9e,"2061",0,0,0
   ;.org 2061

  .org $0326

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

  .org $032A

      sei                     ; disable interrupt

      ;lda  #$0F
      ;sta  $D418              ; volume max
      ;lda  #$11               ; triangle waveform
      ;sta  $D40B              ; voice 2
      ;sta  $D412              ; voice 3
      ;lda  #$41
      ;sta  $D404              ; voice 1
      ;lda  #$C9
      ;sta  $D406              ; SR voice 1
      ;lda  #$69
      ;sta  $D40d              ; SR voice 2
      ;lda  #$29
      ;sta  $d414              ; SR voice 3

      lda  #$30
      sta  $d414              ; SR voice 3
      asl
      sta  $D40d              ; SR voice 2
      asl
      sta  $D406              ; SR voice 1

      lda  #$08
      sta  $D403              ; wave high

      ;ldx  #$00
      ;stx  curTrack
      ;stx  curPat
      ;stx  curTrack+7
      ;stx  curPat+7
      ;stx  curTrack+14
      ;stx  curPat+14
      ; inx
      ;ldx  #1
      ;stx  duration
      lda  #1+DUR+DUR
      sta  duration+7
      lda  #1+DUR+DUR+DUR+DUR
      sta  duration+14

sync:
     ;lda  #$ff
loopSync:
     cpx  $D012
     bne  loopSync

     ;ldx  #0                  ; start from first voice
      ldx  #14                 ; start from last voice
loopVoice:
      dec  duration,x
      bne  nextVoice
      lda  #DUR
      sta  duration,x

      lda  #$11
      cpx  #0
      bne  skip
      lda  #$41
skip:
      sta  $D404,x            ; control register of voice

nextNote:
                              ; read next note
      ldy  curPat,x
      lda  pat,y
      beq  nextTrack
      inc  curPat,x           ; go to next note
                              ; put note y to sid x
      tay

      lda  freqLo-1,y
      sta  $D400,x
      lda  freqHi-1,y
      sta  $D401,x

; increment voice index and test for last voice
nextVoice:
     ; nop  ; without this it goes out of sync!
      ;txa
      ;clc
      ;adc  #7
      ;tax
      ;cmp  #$11
      ;bcs  sync
      ;bcc  loopVoice

      lda  #$ff
      sta  $D418
      .byte $cb,7             ; sbx #7
      bpl  loopVoice
      bmi  sync

nextTrack:
      inc  curTrack,x
      ldy  curTrack,x
      lda  track,y
      bmi  resetTrack
      sta  curPat,x
      bpl  nextNote
resetTrack:
      lda  #0
      sta  curTrack,x
      ;sta  curPat,x
      beq  nextNote

; note declaration
END = 0
F1  = 1
Fd1 = 2
Gd1 = 3
F2  = 4
Fd2 = 5
Gd2 = 6
Gd3 = 7
Ad3 = 8
C4  = 9
```

```
Cd4 = 10
D4  = 11
Dd4 = 12
F4  = 13
Fd4 = 14
G4  = 15
Gd4 = 16
Ad4 = 17

freqLo:
  .byte 231 ;F1
  .byte  20 ;F#1
  .byte 116 ;G#1
  .byte 207 ;F2
  .byte  39 ;F#2
  .byte 232 ;G#2
  .byte 208 ;G#3
  .byte 129 ;A#3
  .byte 103 ;C4
  .byte 112 ;C#4
  .byte 137 ;D4
  .byte 178 ;D#4
  .byte  59 ;F4
  .byte 157 ;F#4
  .byte  20 ;G4
  .byte 160 ;G#4
  .byte   3 ;A#4

freqHi:
  .byte 2  ;F1
  .byte 3  ;F#1
  .byte 3  ;G#1
  .byte 5  ;F2
  .byte 6  ;F#2
  .byte 6  ;G#2
  .byte 13 ;G#3
  .byte 15 ;A#3
  .byte 17 ;C4
  .byte 18 ;C#4
  .byte 19 ;D4
  .byte 20 ;D#4
  .byte 23 ;F4
  .byte 24 ;F#4
  .byte 26 ;G4
  .byte 27 ;G#4
  .byte 31 ;A#4

pat:
pat1:
  .byte Gd1, Gd2, Gd3, END
pat2:
  .byte Fd1, END
pat4:
  .byte Gd2, Ad3, Gd1, Gd3, Gd2, Ad3;, END
pat3:
  .byte Gd1, Gd2, Dd4, Gd1, Gd2, D4, Gd1, Gd2, C4, Gd1, END
pat5:
  .byte D4, Dd4, Gd2, C4, Gd2, Gd3, END
pat6:
  .byte D4, Dd4, Gd2, G4, Gd2, Gd4, END
pat7:
  .byte D4, Dd4, Gd2, Gd4, Gd2, Ad4;, END
pat8:
  .byte F1, F2, Gd4, F1, F2, G4, F1, Dd4, F2, C4, F1, Dd4, F4, F1, C4, Dd4, Fd1, Fd2, Fd4, Fd1, Fd2, F4, END
pat9:
  .byte F1, END
pat10:
  .byte Cd4, Fd2, Gd3, END
pat11:
  .byte Ad3, Fd2, Cd4, Fd1, Gd3; , END next byte is already 0

track:
  .byte (pat1-pat), (pat1-pat), (pat1-pat), (pat1-pat)
  .byte (pat1-pat), (pat2-pat), (pat1-pat), (pat1-pat)
  .byte (pat1-pat), (pat2-pat), (pat1-pat), (pat2-pat)
  .byte (pat3-pat), (pat4-pat); (pat3-pat)
  .byte (pat5-pat)
  .byte (pat3-pat), (pat4-pat);, (pat3-pat)
  .byte (pat6-pat)
  .byte (pat3-pat), (pat4-pat);, (pat3-pat)
  .byte (pat5-pat), (pat3-pat)
  .byte (pat4-pat);, (pat3-pat)
  .byte (pat7-pat)
  ;.byte (pat8-pat),
  .byte  (pat9-pat), (pat10-pat), (pat9-pat)
  .byte (pat11-pat), (pat8-pat), (pat2-pat), (pat10-pat)
  .byte (pat2-pat), (pat11-pat), $FF
```

Mini Digi Drum (512b)

		1/4 note	1/8 note	1/16 note				
	128	64	32	16				
	7	6	5	4				
Speed	Instrument	Duration Bits				Byte	Sound	Bar
12	0	1			dc.b	76	bump	1
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	
12	0		1		dc.b	44	da	
12	0		1		dc.b	44	da	
12	0	1			dc.b	76	bump	2
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	
12	0	1			dc.b	76	bump	
1	0		1		dc.b	33	bip	3
1	0		1		dc.b	33	bip	
1	0		1		dc.b	33	bip	
1	0		1		dc.b	33	bip	
5	0		1		dc.b	37	bup	
5	0		1		dc.b	37	bup	
5	0		1		dc.b	37	bup	
5	0		1		dc.b	37	bup	
12	0	1			dc.b	76	bump	4
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	
12	0		1		dc.b	44	da	
12	0		1		dc.b	44	da	
12	0	1			dc.b	76	bump	5
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	6
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	
12	0	1			dc.b	76	bump	
15	1		1		dc.b	175	pshhh	7
4	1		1		dc.b	164	tsk	
12	0		1		dc.b	44	bump	
4	1		1		dc.b	164	tsk	
15	1		1		dc.b	175	pshhh	
4	1		1		dc.b	164	tsk	
12	0		1		dc.b	44	bump	
4	1		1		dc.b	164	tsk	
15	1	1			dc.b	207	pshhh	8
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	
12	0	1			dc.b	76	bump	
15	1		1		dc.b	175	pshhh	9
4	1		1		dc.b	164	tsk	
12	0		1		dc.b	44	bump	
4	1		1		dc.b	164	tsk	
15	1		1		dc.b	175	pshhh	
4	1		1		dc.b	164	tsk	
12	0		1		dc.b	44	bump	
4	1		1		dc.b	164	tsk	
15	1	1			dc.b	207	pshhh	10
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	
12	0	1			dc.b	76	bump	
15	1		1		dc.b	175	pshhh	11
4	1		1		dc.b	164	tsk	
12	0		1		dc.b	44	bump	
4	1		1		dc.b	164	tsk	
15	1		1		dc.b	175	pshhh	
4	1		1		dc.b	164	tsk	
12	0		1		dc.b	44	bump	
4	1		1		dc.b	164	tsk	
15	1	1			dc.b	207	pshhh	12
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	
12	0	1			dc.b	76	bump	
15	1		1		dc.b	175	pshhh	13
4	1		1		dc.b	164	tsk	
12	0		1		dc.b	44	bump	
4	1		1		dc.b	164	tsk	
15	1		1		dc.b	175	pshhh	
4	1		1		dc.b	164	tsk	
12	0		1		dc.b	44	bump	
4	1		1		dc.b	164	tsk	
12	0	1			dc.b	76	bump	14
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	
12	0		1		dc.b	44	da	
12	0		1		dc.b	44	da	
12	0	1			dc.b	76	bump	15
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	16
12	0	1			dc.b	76	bump	
15	1	1			dc.b	207	pshhh	
12	0	1			dc.b	76	bump	
15	1		1		dc.b	175	pshhh	17
4	1		1		dc.b	164	tsk	
12	0		1		dc.b	44	bump	
4	1		1		dc.b	164	tsk	
15	1		1		dc.b	175	pshhh	
4	1		1		dc.b	164	tsk	
12	0		1		dc.b	44	bump	
4	1		1		dc.b	164	tsk	
4	1	1			dc.b	196	tsk	18
4	1	1			dc.b	196	tsk	
4	1			1	dc.b	148	tsk	
4	1			1	dc.b	148	tsk	
4	1			1	dc.b	148	tsk	
4	1			1	dc.b	148	tsk	
12	0		1		dc.b	44	bump	
12	0		1		dc.b	44	bump	
15	1	1			dc.b	207	pshhh	19
15	1	1			dc.b	207	pshhh	
15	1		1		dc.b	175	pshhh	
15	1		1		dc.b	175	pshhh	
15	1		1		dc.b	175	pshhh	
15	1		1		dc.b	175	pshhh	

This is the 512 bytes tune by Leif Bloomquist, and it is quite unusual. In fact it is a sampled based tune!

The tune use two instruments: kick and hi-hat.

In this left table you will see the structure of the tune: the instrument to play, the duration of the note and the speed for each bars.

Changing the speed of the sample, will produce different sound that is wrote in the *Sound* column.

All those information are packet into a byte:

- Bit 7: instrument to use
- Bits 6,5,4: duration
- Bits 3,2,1,0: speed

The samples for the two instruments are packed together in one byte:

- high nibble: kick
- low nibble: hihat

and they eat half of the space available for the tune.

The last words to say about this player is that synchronization is made by counting the current clock time in jiffies.

```
; "Midi Digi Drums" By Leif Bloomquist   Schema/AIC  leif@schemafactor.com
;
; Entry into the TinySID2 Compo 2006  (512 byte category)

;"Vars"

;$fb - placeholder for jiffy value when done current note
;$fc - holds location in track

;$fd - always 0
;$fe - points to current instrument - either $09 (kick) or $0a (hihat)

; ================ Setup ======================
  processor 6502 ;C64

  org $0801,00

basic  ;"SYS2061"
        dc.b $0c,$08,$64,$0c,$9e,$32,$30
        dc.b $36,$31,$00,$00,$00

  ;Digibooster
  lda #$ff
  sta $d406
  sta $d406+7
  sta $d406+14
  lda #$49
  sta $d404
  sta $d404+7
  sta $d404+14

  ;Fill in the "513th" byte - last byte of sample
  lda #$14
  sta $09FF

  ldx #$00
  stx $fd
  stx $d011  ; Blank Screen

  ; Split up track
split
  lda sampleraw,x       ; Grab the packed byte
  lsr sampleraw,x       ; Rotate original right
  lsr sampleraw,x       ; Rotate original right
  lsr sampleraw,x       ; Rotate original right
  lsr sampleraw,x       ; Rotate original right - high nybble should be low nybble now
  and #$0f              ; Strip high bits out of accumulator
  sta sampleraw+$100,x  ; Stash original low nybble in next page
  inx
  bne split

; Reset to beginning of track
init
  ldx #$00
  stx $fc

; ================ Main Loop ======================

repeat
  ; 1. Get a byte from the "track" and determine note duration.
  ldy $fc
  lda track,y

  beq init ; A zero byte means the end of the track.  Start from the beginning again.

  ; Mask out the duration bits
  and #$70

  ;Divide by 4 and store in the code ahead
  lsr
  lsr
  sta duration+1

  ; Figure out the jiffy count at the end of this duration
  lda $a2  ; Current clock time in jiffies

duration
  adc #16  ; Ignore carry, we want wraparound  (#16 gets overwritten)
  sta $fb

  ; 2. Select instrument.
  ; Default to kick
  lda #>sampleraw
  sta $fe

  ; Get the original byte again
  lda track,y

  ; Determine instrument through hi bit
  bmi go    ; Leave as kick
  inc $fe   ; Move to next page where hi-hat samples are

  ; 3. Determine the playback speed
go
  and #$0f  ; Mask the playback speed
  sta samplespeed+1   ; Store in code ahead

  ; 4. Play the sample.
  jsr play

  ; 5. Wait for timeout to expire
  ldy $fb
rest
  cpy $a2
  bne rest

  ; Move to next entry in track
  inc $fc
  jmp repeat

; ================ Play Instrument =====================
play
  ldy #$00
playloop
   lda ($fd),y
   sta $d418
   sta $d020

samplespeed
   ldx #12               ;some delay value -  set from track (#12 gets overwritten)
playdelay
   dex
   bne playdelay

   iny
   bne playloop          ; Samples are exactly 255 bytes long, so we're done when back to 0

kickdone
   rts

; ================ Track =====================
track  ;(Copy/paste from Excel)

   dc.b 76
   dc.b 76
   dc.b 207
   dc.b 44
   dc.b 44
   dc.b 76
   dc.b 76
   dc.b 207
   dc.b 76
   dc.b 33
   dc.b 33
   dc.b 33
   dc.b 33
   dc.b 37
   dc.b 37
   dc.b 37
   dc.b 37
   dc.b 76
   dc.b 76
   dc.b 207
   dc.b 44
   dc.b 44
   dc.b 76
   dc.b 76
   dc.b 207
   dc.b 76
   dc.b 207
   dc.b 76
   dc.b 207
   dc.b 76
   dc.b 175
   dc.b 164
   dc.b 44
   dc.b 164
   dc.b 175
   dc.b 164
   dc.b 44
   dc.b 164
   dc.b 207
   dc.b 76
   dc.b 207
   dc.b 76
   dc.b 175
   dc.b 164
   dc.b 44
   dc.b 164
   dc.b 175
   dc.b 164
   dc.b 44
   dc.b 164
   dc.b 207
   dc.b 76
   dc.b 207
   dc.b 76
   dc.b 175
   dc.b 164
   dc.b 44
   dc.b 164
   dc.b 175
   dc.b 164
   dc.b 44
   dc.b 164
   dc.b 207
   dc.b 76
   dc.b 207
   dc.b 76
   dc.b 175
   dc.b 164
   dc.b 44
   dc.b 164
   dc.b 175
   dc.b 164
   dc.b 44
   dc.b 164
   dc.b 76
   dc.b 76
   dc.b 207
   dc.b 44
   dc.b 44
   dc.b 76
   dc.b 76
   dc.b 207
   dc.b 76
   dc.b 207
   dc.b 76
   dc.b 207
   dc.b 76
   dc.b 175
   dc.b 164
   dc.b 44
   dc.b 164
   dc.b 175
   dc.b 164
   dc.b 44
   dc.b 164
   dc.b 196
   dc.b 196
   dc.b 148
   dc.b 148
   dc.b 148
   dc.b 148
   dc.b 44
   dc.b 44
   dc.b 207
   dc.b 207
   dc.b 175
   dc.b 175
   dc.b 175
   dc.b 175

  dc.b 0  ; EOT

; ================ Packed Sample ======================
  org $0900
sampleraw
  incbin "samples-packed.bin"
```

RM (512b)

RM is my cover of David Whittaker “Red Max”. This is a repetitive tune with lot of interesting sounds, so maybe it is a little difficult to fit in so little space: a good challenge for cover.

At the beginning, the voice 3 performs an arpeggio with 4 notes that it is combined with a long pulse modulation that give the characteristic fabulous sound. Voice 1 uses 3 instruments for having a drum sound using an octave arpeggio. Inside the tune there is long sound with decreasing portamento.

In totally it is used 8 different instruments that are based onto ADSR and waveform, so we have to manage for sure the AD, SR and control register for one instrument. We start using a fixed pulse modulation for all the voice.

Looking at the notes, we see that there are lot of different notes used, so it is not convenient to store only the high/low frequency of the used notes, as for sure generating them at runtime will take less space.

Actually the best runtime generation that I see so far is with about 52 bytes, based onto a 11 bytes of frequency table. But when I reverse engineering Modulus tune, I remember that Ivan generates always his notes frequency based onto 22 values, and he not store the values into a table.

For madding this, he stored the note as: octave/base note

then he takes the frequency from the table (that are for the highest octave) and down sample until the octave he has to produce. I so take the same idea, but using 11 bytes for the lower octave frequency (that has the high byte always at 1) and over sampling until the octave to produce. The result is about a 41 bytes procedure (I use an ASR undocumented instruction for a AND/SHR instructions inside it).

Looking now at the structure of the song, we see that voice 1 plays a fixed pattern of 10 notes/durations/instruments and voice 2 plays 16 notes, but each pattern has a transposition from the precedent.

Voice 3 has different patterns, but with some exception of equal pattern and pattern transposition. This make voice 3 a little more complicated.

Now the idea is simple: memorize the transposition that are to used for each track when the pattern of note is over. For voice 3, we even memorize the transposition of the pattern, and even the pattern to use for that transposition.

All is now good, but not the duration/base note used for memorize one note, as making a transposition is not like to add the transpose value to the actual note to play. This is a right reason for abort the used note representation and to go to generate the frequency with the more expansive 52 bytes routines, as for sure the transpose is just an ADC instead of a lot of base-12 addictions.

No. No so fast. Maybe the transpositions are apply to certain notes only and some accurate analysis could help us to use this system with the ADC for transposition at the same time.

Well, to make it short, it is possible and using only 2 additional bytes, and this is lot better than the standard method. However I think it is a little complicated to show how this come from (I have made lot of rearranging for reach this point), but I hope that you understand the idea below this by looking at this final description.

1.	C	C		
2.	C#	D		
3.	D	E		
4.	D#	D#	Voice 3: E F C A# G G#	0, +12
5.	E	F		
6.	F	G	Voice 2: A#	0, +2, +4, +7
7.	F#	A		
8.	G	G#	Voice 1: F C G G#	0, +2, +4, +9
9.	A	C#		
10.	A#	F#		
11.	B	D		
12.		G		
13.		A		
14.		.		
15.		.		

In the right part above we see all the notes (without octave) used by the 3 voices and the relative arpeggio values to add to each note. In the left part we see the common notes sequence from 0 to 11. Notes from 12 are like the others, but 1 octave above. Here you see even the base notes stored into the nibble: you see that the order is particular. Else notes at position 12 and 13 are the same of position 6 and 7. Remember that note is stored as octave/base note.

Now, take those values for arpeggio:

Voice 1:	0, +1, +12, +13
Voice 2:	0, +8, +9, +12
Voice 3:	0, +16

You will see that a simple addiction will perform the right note steps.

At this point we can describe how a pattern is made:

Byte	***Description***
1000.xxxx	xxxx= instrument to use
11nn.nnnn	nnnnnn= note length
0ccc.nnnn	ccc=octave nnnn=base note

Positive values are notes in the form octave/base note, while negative values are instruments or note length according with bit 6 value.

Now that all seems perfectly, we have to make the arpeggio in voice 3 at the beginning (first 4 bars of song) and in the middle of song, as this is a sound that must be replicated. Well, this maybe late me think to rewrite all the stuff not using octave/base note, as now it is not possible to make the arpeggio by simple adding a value.

However, now we have save lot of bytes and maybe we could find a way to make the arpeggio in another way. In fact, the arpeggio must be generated at each frame.

I so try to make arpeggio by adding a total amount of frequency to add at the last note played on the voice 3 when it plays note of octave 6 (that fortunately is when arpeggio is used). Arpeggio steps are taken from a table.

Unfortunately this code goes out of space, so I remove some value of arpeggio steps and even I manage only high frequency for reducing space. The final arpeggio (that it is different from the original) come out after several try into changing the values until I find a sound that was acceptable.

```
; RM
; 512b SID music
; by Stefano Tognon 2006
; This is a cover of David W. "Red Max"

; For having all the high/low frequency of notes, I use the inverse
; thecnique of Ivan Del Duca (he starts from octave 8 and go down,
; I start from octave 0 and goes up, saving 12 bytes for high values
; as they are to take all = 1)
; The base routine take about 41 bytes, some overhead is due to the use
; of transpositions of note

; note sequence are choosed for having a transpose simple by adding the value
; to octave/note. Fortunately only 2 bytes are needed for this

; optimization:
;  -3 make pat4 followed by pat5
;  -2 move trPos before trans1, removing two 0
;  -1 use ASR for AND and LSR
;  -6 use ADSR in same byte
;  -1 skip a zero byte as next is 0
;  -2 use bpl instead of jmp
;  -5 do not init an already 0 value
;  -2 put right value inside trPos and trPos+1
;  all the other just removing arpeggio routine and modify until a good sound is reached

; devel note: AD SR could be combined as low/high nibble are in right position

; later fix: remove a typing error that I see while I was trying to create the psid version

 .processor 6502

actFreqLow  = $10
actFreqHigh = $11


curTrack = $57   ; this and +7 and +14 are 0 at sturtup of after loading current track/transposition position
curPat   = $58   ; this and +7 and +14 are 0 at sturtup of after loading

temp2    = $2A
delay    = $2B   ; init at 1
temp     = $2C
duration = $2D   ; actual duration +0 +7 +14
storeDur = $2E   ; stored duration to use +0 +7 +14
instr    = $2F   ; instrument number to use +0 +7 +14
transp   = $30   ; transposition for this pattern +0 +7 +14
note     = $31   ; note +0 +7 +14
DUR =  12;48

  .org $0326

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

  .org $032A

      sei                     ; disable interrupt

      lda  #(trans1-trans)       ; read initial track position
      sta  curTrack
;  lda #$8
  ;sta  $d402
  sta  $d403
  ;sta  $d402+7
  sta  $d403+7
  ;sta  $d402+14
  sta  $d403+14

      lda  #(trans2-trans)        ; read initial track position
      sta  curTrack+7
                                ; value to set is 0
      ;lda  trPos+3              ; read initial track position
      ;sta  curTrack+14
  lda #1
  sta  duration
  sta  duration+7
  sta duration+14

sync:
     ;lda  #$ff
loopSync:
     cpx  $D012
     bne  loopSync

     ldx #14
     lda  note,x                 ; read voice 3 note
     pha

     jsr  noteFreq

     ldy  delay
     ;lda  delay
     ;lsr
     ;and  #$03
     ;tay
     lda  arpeggio,y
     tay

contArpeggio:
     ;cpy  #0
     beq  skipArpeggio
     lda  actFreqLow
     adc  #$24
     sta  actFreqLow
     lda  actFreqHigh
     adc  #$06          ; 04

     sta  actFreqHigh
     dey
     bpl  contArpeggio
skipArpeggio:
     pla
     cmp  #$60
     bmi  ggg

     jsr  putFreq
ggg:
      ldy  #DUR
      dec  delay
      bne  sync
      sty  delay

      ;ldx  #14              ; start from last voice
loopVoice:
      dec  duration,x
      bne  nextVoice
      lda  storeDur,x
      sta  duration,x

nextNote:
      lda #$40
      sta $D404,x
                             ; read next command
      ldy  curPat,x
      lda  pat,y
      beq  nextTrack

      bpl  putNote
      cmp  #$C0               ; len/instr?
      bcc  newInst

      and  #$3F               ; store duration
      sta  storeDur,x
      sta  duration,x

      inc  curPat,x
      bne  nextNote

newInst:
      and  #$0F
      sta  instr,x            ; store instrument to use
      inc  curPat,x
      bne  nextNote


putNote:
      sta  note,x             ; used for voice 3 arpeggio
      jsr  noteFreq

      jsr  putFreq

      ldy  instr,x             ; instrument number
      lda  control,y
      sta  $D404,x            ; put control sid value
      lda  adsr,y
      pha
      and  #$0F
      sta  $D405,x            ; put attack/decay
      pla
      and  #$f0
      sta  $D406,x            ; put sustain/release
      inc  curPat,x
```

```
; increment voice index and test for last voice
nextVoice:

      lda  #$7f
      sta  $D418
 ;jmp sync
      .byte $cb,7            ; sbx #7
      bpl  loopVoice
      ;bmi  sync
     jmp  sync


nextTrack:
      ldy  curTrack,x
      lda  trans,y
      sta  transp,x              ; store transposition for this pattern
      bmi  resetTrack

      cpx  #7
      beq  tr2
      bpl  tr3
      lda  #(pat0-pat)           ; fix pattern for voice 1

setPat:
      sta  curPat,x              ; store current pattern
      inc  curTrack,x
      bpl  nextNote
resetTrack:
      txa
      lsr
      lsr
      tay
      lda  trPos,y               ; read initial track position
      sta  curTrack,x
      ;sta  curPat,x
      bpl  nextNote

tr2:                             ; fix pattern for voice 2
      lda  #(pat1-pat)
      bpl  setPat

tr3:                             ; voice 3 use different patterns
      lda  track,y
      bne  setPat


noteFreq:
      stx  temp
; set frequency A=octave/note
      clc
      adc  transp,x
      pha

      and  #$0F
      tax
      lda  freqSource,x
      sta  actFreqLow
      lda  #1
      sta  actFreqHigh

      pla
      ;and  #$F0                    ; take the octave of note
      ;lsr
      .byte $4b, $F0               ; ASR #$0F
      lsr
      lsr
      lsr
      tay
      beq  freqOk
calcFreq:
      asl  actFreqLow
      rol  actFreqHigh
      dey
      bne  calcFreq
freqOk:
      ldx  temp
      rts

putFreq:
      lda  actFreqLow
      sta  $D400,x
      lda  actFreqHigh
      sta  $D401,x
      rts

fC0  = 12
fC_0 = 28
```

```
fD0  = 45
fD_0 = 62
fE0  = 81
fF0  = 102
fF_0 = 123
fG0  = 145
fG_0 = 169
fA0  = 195
fA_0 = 221
fB0  = 250


; source values used to calculate frequency table.   (taken from 4mat)
freqSource  .byte fC0, fD0, fE0, fD_0, fF0, fG0, fA0, fG_0, fA_0, fC_0, fF_0, fB0,   fG0, fA0


; instrument
control:                    ; $41
  .byte $41, $11, $81, $41, $21, $41, $41, $41, $41
; SR/AD
adsr:
  .byte $4A, $09, $09, $08, $30, $48, $49, $6a, $8C

; voice 1:
I1 = 0  ; 41 0a40
I2 = 1  ; 11 0900
I3 = 2  ; 81 0900

; voice 2
I4 = 3  ; 41 0800

; voice 3
I5 = 4  ; 41 0030
I6 = 5  ; 41 0840
I7 = 6  ; 41 0940
I8 = 7  ; 41 0a60
I9 = 8  ; 41 0c80

; length:
; 1 2 3 4 6 7 16 17
L1  = 1
L2  = 2
L3  = 3
L4  = 4
L6  = 6
L7  = 7
L16 = 16
L17 = 17
L32 = 32

INS = $80        ; instrument
LEN = $C0        ; length

C0  = 0
D0  = 1
E0  = 2
D_0 = 3
F0  = 4
G0  = 5
A0  = 6
G_0 = 7
A_0 = 8
C_0 = 9
F_0 = 10
B0  = 11


; notes:
C1  = $10 + C0  ; 12
F1  = $10 + F0  ; 17
A_1 = $10 + A_0 ; 22
C2  = $20 + C0  ; 24
A_2 = $20 + A_0 ; 34
C3  = $30 + C0  ; 36
D3  = $30 + D0  ; 38
E3  = $30 + E0  ; 40
F3  = $30 + F0  ; 41
G3  = $30 + G0  ; 43
G_3 = $30 + G_0 ; 44
A3  = $30 + A0  ; 45
A_3 = $30 + A_0 ; 46
C4  = $40 + C0  ; 48
D4  = $40 + D0  ; 50
E4  = $40 + E0  ; 52
G4  = $40 + G0  ; 55
C5  = $50 + C0  ; 60
G5  = $50 + G0  ; 67
G_5 = $50 + G_0 ; 68
C6  = $60 + C0  ; 72
D6  = $60 + D0  ; 74
```

```
F6  = $60 + F0  ; 77

pat:
arpeggio:
  .byte $00  ; this must be zero as all pattern start from here (already init at 0)
  .byte 3, 7;, 12

pat0:
  .byte INS+I1, LEN+L2, F1
  .byte INS+I2, LEN+L1, G_5, C6
  .byte INS+I3, LEN+L2, C6
  .byte INS+I1,         C2, F1
  .byte INS+I2, LEN+L1, G_5, G5
  .byte INS+I3, LEN+L2, C5
  .byte INS+I1,         C1, 0

pat1:
  .byte INS+I4, LEN+L1
  .byte A_1, A_1, A_1, A_2, A_1, A_1, A_1, A_1
  .byte A_1, A_1, A_1, A_2, A_1, A_1, A_2, A_2, 0

pat2:
  .byte INS+I5, LEN+L16, F6, F6, C6, C6, F6, F6, C6, C6, 0

pat3:
  .byte INS+I6, LEN+L1, E3, F3
  .byte INS+I7, LEN+L2, F3, F3, F3
  .byte         LEN+L6, F3
  .byte         LEN+L2, C3
  .byte INS+I6, LEN+L1, F3, C4
  .byte INS+I7, LEN+L2, C4
  .byte         LEN+L3, C4
  .byte         LEN+L1, C3
  .byte INS+I6,         C4, C4
  .byte INS+I7, LEN+L2, C4, A_3
  .byte         LEN+L1, G_3
  .byte INS+I8, LEN+L17,G3
  .byte INS+I6, LEN+L1, G3, E3, C3, C4, G3, E3, E4, C4, G3, G4, E4, C4, C5, G4, E4, C4, 0

pat4:
  .byte INS+I9, LEN+L16, D3, A_2;, 0

pat5:
  .byte INS+I6, LEN+L1, A3, A3
  .byte INS+I7, LEN+L2, G3, A3
  .byte         LEN+L1, G3
  .byte         LEN+L2, A3
  .byte         LEN+L3, D4
  .byte         LEN+L4, D6
  .byte INS+I6, LEN+L1, F3, F3
  .byte INS+I7, LEN+L2, E3, F3
  .byte         LEN+L1, E3
  .byte         LEN+L3, F3
  .byte         LEN+L2, A3, F3, E3, 0

pat6:
  .byte INS+I9, LEN+L32, E3
  ;, 0   next byte is 0, so skip

; track1 transpose
trans:
; track3 transpose
trans3:
  .byte 0, $10, 0, 0, 0
        ;0,
  .byte 0, $FF

trPos:
  .byte (trans1-trans), (trans2-trans)
  ;, 0, 0        taken below

; track1 transpose
trans1:
  .byte 0, 0, 12, 12, 0, 0, 12, 12
  .byte 12, 12, 1, 1, 12, 12, 1, 1
  .byte 13, 13, 0, 0, 13, 13, 12, 12
  .byte $FF

; track2 transpose
trans2:
  .byte 12, 12, 8, 8, 12, 12, 8, 8
  .byte 12, 12, 8, 8, 12, 12, 8, 8
  .byte 9, 9, 0, 0, 9, 9, 8, 8
  .byte $FF

track:
  .byte (pat2-pat), (pat3-pat), (pat3-pat), (pat5-pat), (pat4-pat)
        ;, (pat5-pat),
  .byte (pat6-pat)
```

Miniature Rhapsody (256b)

This is the 256 bytes entries by Eric Odland, and how you can hear it is almost based onto arpeggio.

Let we see some points about the player:

- It initializes IRQ by starting at $0326 (the short way around), disabling interrupt with SEI, and then synchronized to raster position using 70 tests passages.
- Frequency table is built at startup using 12 values, then fixed values of Sustain/Release are set for voice 1 and 2, and Attack/Decay for voice 3.
- A low pass filter is apply in all the 3 voices.
- Voice 2 will play the notes of voice 1 but with a frame of delay.
- Voice 1 plays the arpeggio, while voice 3 the bass instrument.

It is very interesting to see how data to play is represented into this player:

Value (xxyyzzzz)	***Description***
xx	Arpeggio (0-3) that are taken from a table: 1 4 3 - maj7 3rd inversion 2 3 2 - 4 root 3 4 3 - min7 root min2 root
yy	Bass (0-3) that are taken from a table that contains the 3 values of note to play in each case
zzzz	Root note (1-F) for the arpeggio

The tune is so composed only from a sequence of bytes in the above form (each notes have the same duration), that are repeated as soon as the 00 value is reached.

Here the source:

```
; Miniature Rhapsody
; By Eric Odland
;
; Written for "Tiny SID 2" - Online SID Music-Programming Competition
; Platform:Commodore 64
;
; (c) 2006 Stone Monkey Software.
;
; File: mr.S
; Created on 3/22/06.
;
; NOTES:
;        Idea to use undocumented opcode SAX borrowed from Stefano Tognon's x15


        processor 6502


; ************ VARIABLES ***********
        SEG.U Data
        ; Zero Page variables.
        org $E0
voice1Note                      dc.b 0
        org $E7
voice2Note                      dc.b 0
        org $EE
voice3Note                      dc.b 0
```

```
        org $FA
pattern1Position        dc.b 0
arpeggioIndex           dc.b 0
arpeggioVector          dc.b 0
        ; arpeggio data
        org $C000
arpeggioData    ds.b 17,0



        SEG header
        org $0326

        dc.b $2A, $03
        dc.b $ED, $F6

; ****************    CODE   *****************
        SEG code
        org $032A

        sei

SetupFrequencyTable:
        ; Setup frequency table
        ldy #11
        lda #1
.1      sta freq_hi,y
        dey
        bpl .1
        ldy #0
.2      lda freq_lo,y
        asl
        sta freq_lo+12,y
        lda freq_hi,y
        rol
        sta freq_hi+12,y
        iny
        bne .2
        tya


Start:
        sta pattern1Position
        ; Initialize SID
        lda #$9D        ; voice1 sustain/release
        sta $D406
        lsr                     ; voice2 sustain/release
        sta $D40D
        sta $D413       ; voice3 attack/decay

        ; Init player
        bne GetNextArpeggio

UpdateFrame             subroutine
        ; update voice 1
        ldx arpeggioIndex
        lda arpeggioData,x
        sta voice1Note

        ldx #14
.nextVoice:
        lda voice1Note,x
        tay
        lda freq_lo,y
        sta $D400,x
        lda freq_hi,y
        sta $D401,x
        lda #$21
        sta $D404,x
        lda #$FF
        dc.b $CB, 7                     ; SAX #7 : X = (A & X)-7
        bpl .nextVoice

        lda arpeggioVector      ; arpeggioVector = -arpeggioVector
        ldx arpeggioIndex
        cpx #16
        bne .1
        eor #%11111110
        sta arpeggioVector
.1      clc
        adc arpeggioIndex
        sta arpeggioIndex

        bne endGetNextArpeggio
                inc pattern1Position

;--------- GETNEXTARPEGGIO ---------
GetNextArpeggio subroutine
                ldy pattern1Position
                lda PatternTable,y
                beq Start
        ; fill arpeggioData
        pha
        and #$0F
        clc
        adc #31
        sta arpeggioData        ; store first note in arpeggio
        pla

        lsr
        lsr
        lsr
        lsr

        pha             ; arpeggio/bass note values
        and #%00000011  ;extract bass note
        tay
        lda BassNote,y
        sta voice3Note

        pla             ; extract arpeggio value
        lsr
        lsr

        tay
        lda ArpeggioTable,y

        ldy #0
        ldx #3

.1      rol
        rol
        pha
        rol
        and #%00000011
        sec
        adc arpeggioData,y
        iny
        sta arpeggioData,y

        ; setup filter
        ; Put this here instead of at initizalization to save space
        lda SIDsnapshot-1,x
        sta $D416-1,x

        pla
        dex
        bne .1

.2      iny
        lda arpeggioData-4,y
        clc
        adc #12
        sta arpeggioData,y
        cpy #16
        bne .2

        stx arpeggioIndex
        ; arpeggioVector = 1
        inx
        stx arpeggioVector

        lda #$20
        sta $D412
;------- END GETNEXTARPEGGIO ----------
endGetNextArpeggio:
        ldx #70
.wait   lda $D012
        bne .wait
        dex
        bne .wait

        lda voice1Note
        sta voice2Note

        jmp UpdateFrame



;**************** DATA ****************
;
; Pattern data = xxyyzzzz
;               xx = arpeggio 0-3
;               yy = bass 0-3
;               zzzz = rootNote 1-F
; $00 = end of pattern

PatternTable:
        ;dc.b $75, $75, 0
        dc.b $95, $86, $E7, $8B ; part 1
        dc.b $D5, $22, $BC, $BB ; part 2
        dc.b $29, $89, $29, $8E ; part 3
        dc.b $1E, $9C, $73, $73 ; part 4
        dc.b $00

ArpeggioTable:
        dc.b %00111000          ; 1 4 3 - maj7 3rd inversion
        dc.b %01100100          ; 2 3 2 - 4 root
        dc.b %10111000          ; 3 4 3 - min7 root
        dc.b %11010000          ; min2 root

BassNote:
        dc.b 21, 24, 26, 0

SIDsnapshot:
        dc.b  $70, $8F, $1F                     ; filter        cutoff_hi, res/filt, mode/vol

freq_hi = . + $100
freq_lo:
freqMult        dc.b $0C, $1C, $2D, $3F, $52, $66, $7B, $92, $AA, $C3, $DE, $FA
```

Conclusion

For seeing the winners of the compo analyzed, you must attend the next number, but I think that with these entries you have already lot of material available for starting to think how to build your mini tunes, maybe for next compo or minigame.

Goattracker 2

by Stefano Tognon <ice00@libero.it>

In this article I will go to look at the new version of Cadaver's Goattracker pc music editor. You already know that version 2 is now tables based and so it can compete with the best C64 music tools available.

I think that the best mode to discuss this tool is by creating a tune with it. I so realize a cover of “What is Love” of Haddaway. This is a tune that I like very much.

Start

The first step was to setting up the Goattracker in Linux (as no executable for it is released): it is just to run *make* into the *src* directory, but after having already compiled the BCE library and put it in the *linux* directory, otherwise the compilation did not finish.

Now I went to search some midi version of the tune to cover: I download many versions, that differs only for the kind of midi-mapped instrument used. But finally I simple choose the Karaoke one, as I like to see the words of the song in it.

For extracting the notes from the midi, I had used the last version of *Rosegarden*. You find it at www.sf.net and I have compiled it directly from the source using the last tarball available.

Rosegarden is a midi/notation sequencer that features:

- Matrix editor
- Matrix drum editor
- Notation editor
- Midi event editor

For my porpoise I had used the matrix/drum editors using a 1/16 rhythm scale.

If you want to test the midi instrument used into the midi tune, for then going to reproduce that sound into the sid chip, the best way is to have *timidity* running as *alsa* server in Rosegarden:

```
timidity -Os -iA -B2,8 -s 16k
```

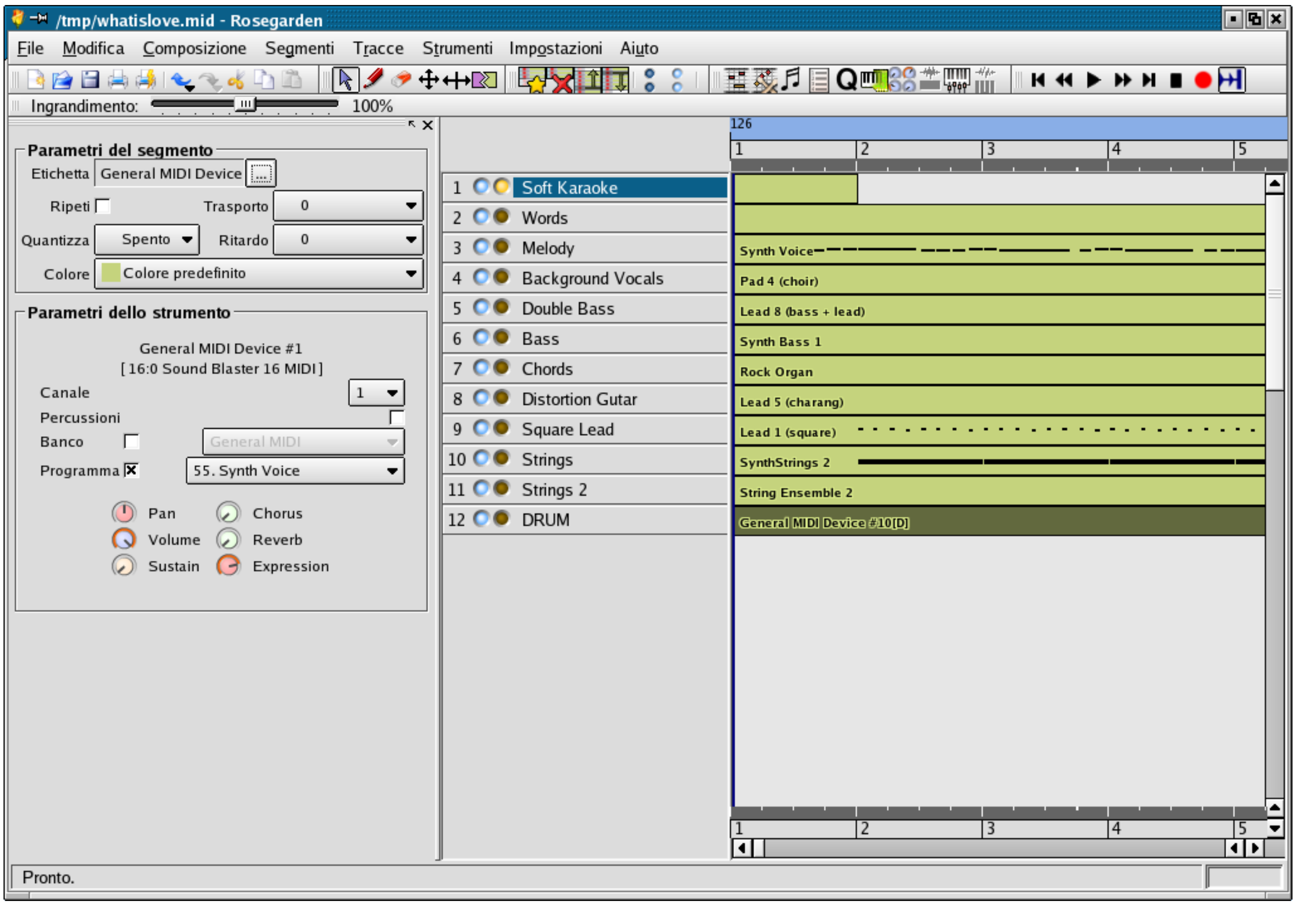
The -s option it to limit the cpu usage as in low system it can be too hard.

I also use to disable in runtime some voices of the midi track while running for better going into choose the 3 tracks that I can fit into the sid.

There are 10 music tracks into the midi (as you can see from the main Rosegarden page), but essentially I have choose to have:

- Drum
- Bass
- Melody / Lead/ Chords

for the 3 sid tracks.


/tmp/whatislove.mid - Rosegarden
File Modifica Composizione Segmenti Tracce Strumenti Impostazioni Aiuto
Ingrandimento: 100%
Pronto.

Parametri del segmento

Etichetta: General MIDI Device
Ripeti ☐ Trasporto: 0
Quantizza: Spento Ritardo: 0
Colore: Colore predefinito

Parametri dello strumento

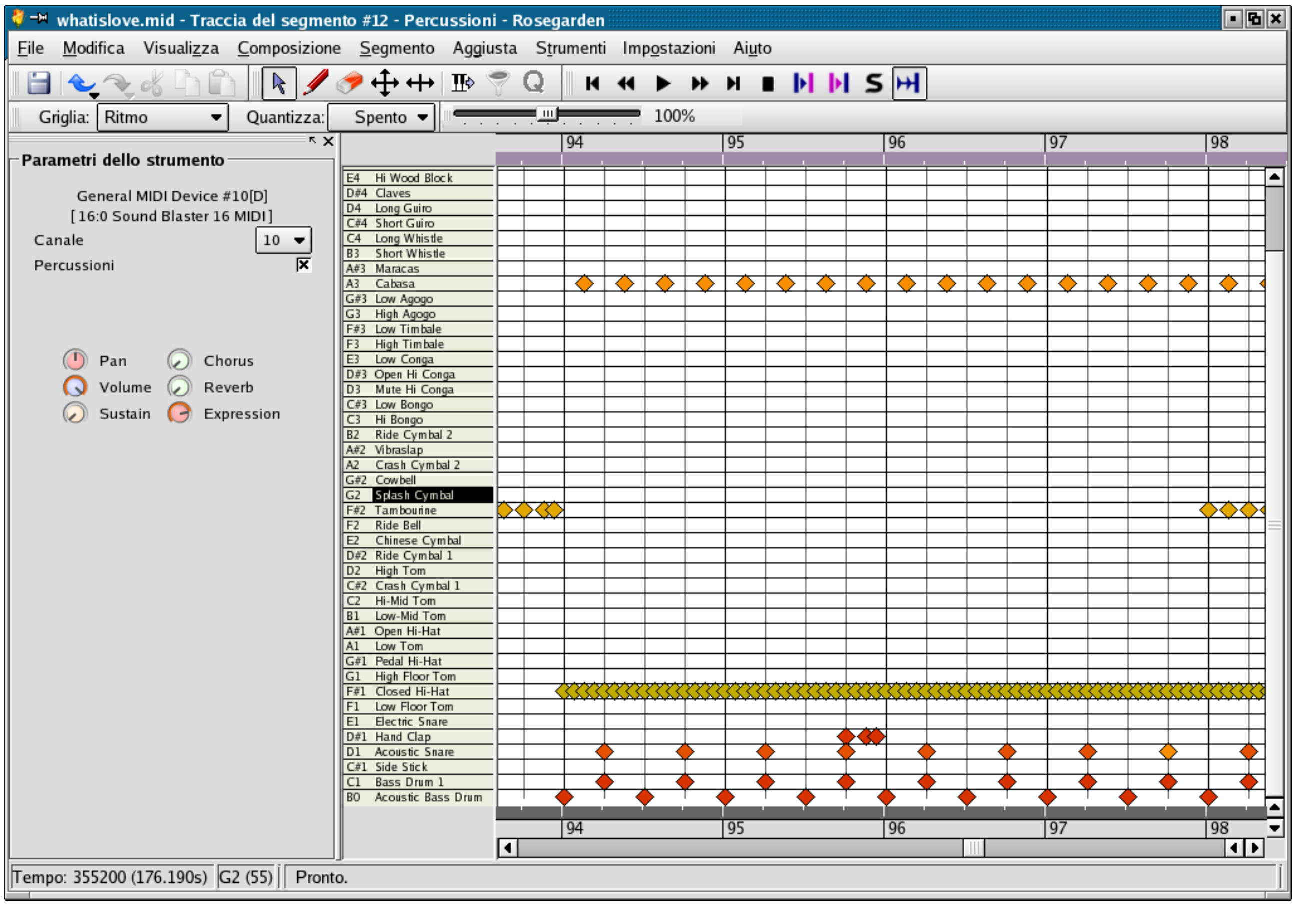
General MIDI Device #1
[16:0 Sound Blaster 16 MIDI]

Canale: 1
Percussioni ☐
Banco ☐ General MIDI
Programma ☒ 55. Synth Voice

Pan, Chorus, Volume, Reverb, Sustain, Expression

	Traccia	Segmento
1	Soft Karaoke	
2	Words	
3	Melody	Synth Voice
4	Background Vocals	Pad 4 (choir)
5	Double Bass	Lead 8 (bass + lead)
6	Bass	Synth Bass 1
7	Chords	Rock Organ
8	Distortion Gutar	Lead 5 (charang)
9	Square Lead	Lead 1 (square)
10	Strings	SynthStrings 2
11	Strings 2	String Ensemble 2
12	DRUM	General MIDI Device #10[D]

Rosegarden main page


whatislove.mid - Traccia del segmento #12 - Percussioni - Rosegarden
File Modifica Visualizza Composizione Segmento Aggiusta Strumenti Impostazioni Aiuto
Griglia: Ritmo Quantizza: Spento 100%
94 95 96 97 98

Parametri dello strumento

General MIDI Device #10[D]
[16:0 Sound Blaster 16 MIDI]

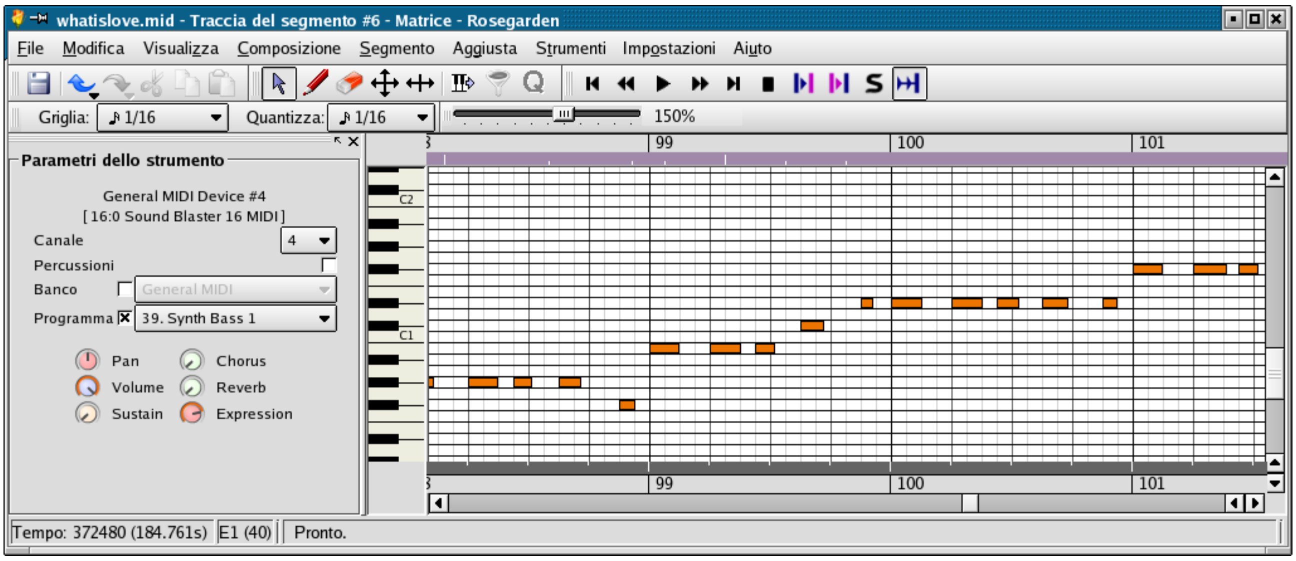
Canale: 10
Percussioni ☒

Pan, Chorus, Volume, Reverb, Sustain, Expression

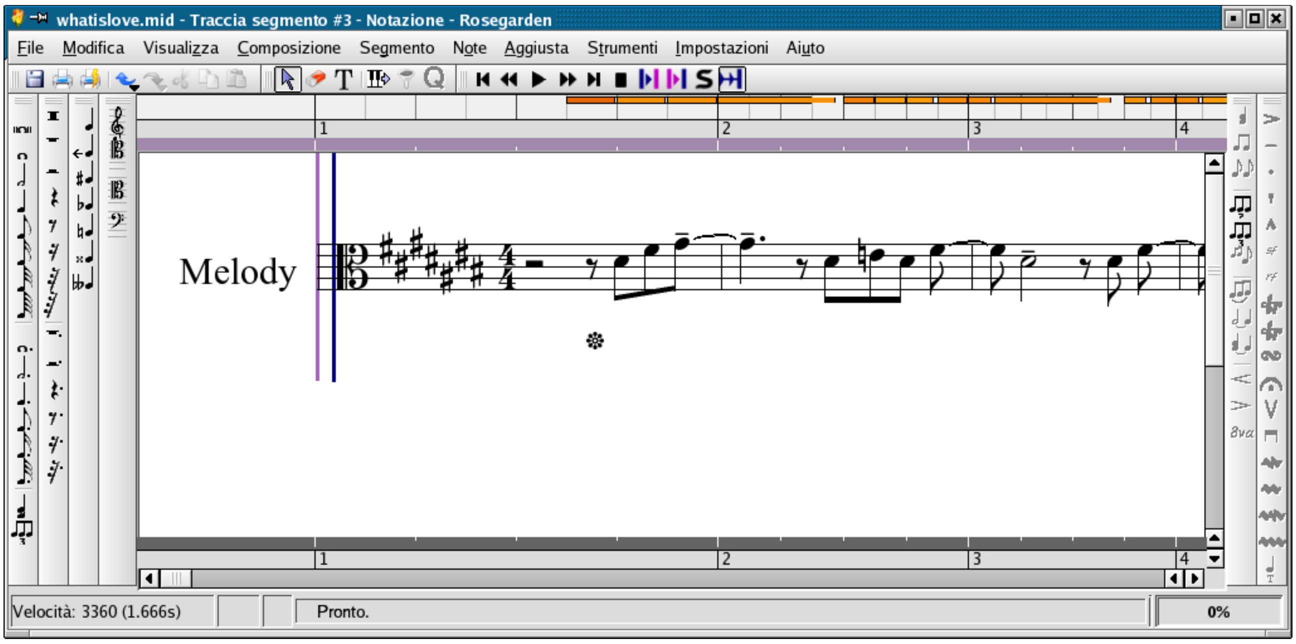
E4	Hi Wood Block
D#4	Claves
D4	Long Guiro
C#4	Short Guiro
C4	Long Whistle
B3	Short Whistle
A#3	Maracas
A3	Cabasa
G#3	Low Agogo
G3	High Agogo
F#3	Low Timbale
F3	High Timbale
E3	Low Conga
D#3	Open Hi Conga
D3	Mute Hi Conga
C#3	Low Bongo
C3	Hi Bongo
B2	Ride Cymbal 2
A#2	Vibraslap
A2	Crash Cymbal 2
G#2	Cowbell
G2	Splash Cymbal
F#2	Tambourine
F2	Ride Bell
E2	Chinese Cymbal
D#2	Ride Cymbal 1
D2	High Tom
C#2	Crash Cymbal 1
C2	Hi-Mid Tom
B1	Low-Mid Tom
A#1	Open Hi-Hat
A1	Low Tom
G#1	Pedal Hi-Hat
G1	High Floor Tom
F#1	Closed Hi-Hat
F1	Low Floor Tom
E1	Electric Snare
D#1	Hand Clap
D1	Acoustic Snare
C#1	Side Stick
C1	Bass Drum 1
B0	Acoustic Bass Drum

Tempo: 355200 (176.190s) | G2 (55) | Pronto.

Drum matrix editor


whatislove.mid - Traccia del segmento #6 - Matrice - Rosegarden
File Modifica Visualizza Composizione Segmento Aggiusta Strumenti Impostazioni Aiuto
Griglia: 1/16 Quantizza: 1/16 150%
Parametri dello strumento
General MIDI Device #4
[16:0 Sound Blaster 16 MIDI]
Canale 4
Percussioni
Banco General MIDI
Programma 39. Synth Bass 1
Pan Chorus
Volume Reverb
Sustain Expression
99 100 101
C2
C1
Tempo: 372480 (184.761s) E1 (40) Pronto.

Matrix editor


whatislove.mid - Traccia segmento #3 - Notazione - Rosegarden
File Modifica Visualizza Composizione Segmento Note Aggiusta Strumenti Impostazioni Aiuto
Melody
1 2 3 4
Velocità: 3360 (1.666s) Pronto. 0%

Notation editor

For my own use I make a table (build while creating the tune) in a spreadsheet that contains the Goattracker patterns with the used instruments for each voice. I put even some colors that give better looks to the global music (one color for one type of instrument).

This was for me the best way to remember each patterns into goattracker and even to control that filter may stay always into voice 3

You will see that the first patter is very short: this is a good characteristic of GT that allow you to set the length of each pattern independently.

In the tune, I use the filter in voice 3 for making the bass sound better. In some points I have to reproduce the lead or synth voice instead of bass: in that case I have to make a special instrument that stop filter execution as GT2 left the filter set up from the previous instrument (the bass).

I have also use a lot the pattern function Rxx that repeat xx time the pattern that follows. GT2 permit this instruction and even to transpose a pattern.

Bat.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
V1	08	00				01 Cabana				02 Cabana+Tamburine+Hihat				03 C.+T.+H.+Snare+Drum			
V2	08	07 Lead				07 Lead				0D Chords				0D Chords			
V3	0A	0B Synth Voice				0C Synth Voice				0F Bass			Synt	Synt	0F Bass		Synt

	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
V1	03 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum				04 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum			
V2	07 Lead				12 Synth Voice				13 Synth Voice				0B Synth Voice			
V3	Synt	10 Bass			11 Bass				11 Bass				11 Bass			

	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
V1	03 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum				05 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum			
V2	0C Synth Voice				14 Guitar + vocals				14 Guitar + vocals				0D Chords			
V3	11 Bass				11 Bass				11 Bass				11 Bass			

	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65
V1	05 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum				06 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum			
V2	0D Chords				12 Synth Voice				13 Synth Voice				0B Synth Voice			
V3	11 Bass				11 Bass				15 Bass				21 Lead			

	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81
V1	03 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum				05 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum			
V2	0C Synth Voice				14 Guitar + vocals				14 Guitar + vocals				0D Chords			
V3	21 Lead				11 Bass				16 Bass			Synt	Synt	17 Bass		Synt

	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97
V1	05 C.+T.+H.+Snare+Drum				00				1A Tamburine				1B C.+T.+H.+S +D +HC			
V2	0D Chords				00				00				Guitar	1D Synt voice		
V3	Synt	18 Bass		Synt	0C Synth Voice				11 Bass				19 Bass			

	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113
V1	03 C.+T.+H.+Snare+Drum				20 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum			
V2	1F Synth Voice				1E Synth Voice				0B Synth Voice				0C Synth Voice			
V3	21 Lead				22 Lead			Bass	21 Lead				21 Lead			

	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129
V1	03 C.+T.+H.+Snare+Drum				05 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum			
V2	14 Guitar + vocals				14 Guitar + vocals				07 Lead				07 Lead			
V3	11 Bass				16 Bass			Synt	23 Synth Voice				0C Synth Voice			

	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145
V1	03 C.+T.+H.+Snare+Drum				05 C.+T.+H.+Snare+Drum				03 C.+T.+H.+Snare+Drum				28 C.+T.+H.+Snare+Drum			
V2	Guitar	25 Synth Voice			Guitar	25 Synth Voice			07 Lead				07 Lead			
V3	11 Bass				11 Bass				24 Synth Voice				26 Synth Voice			

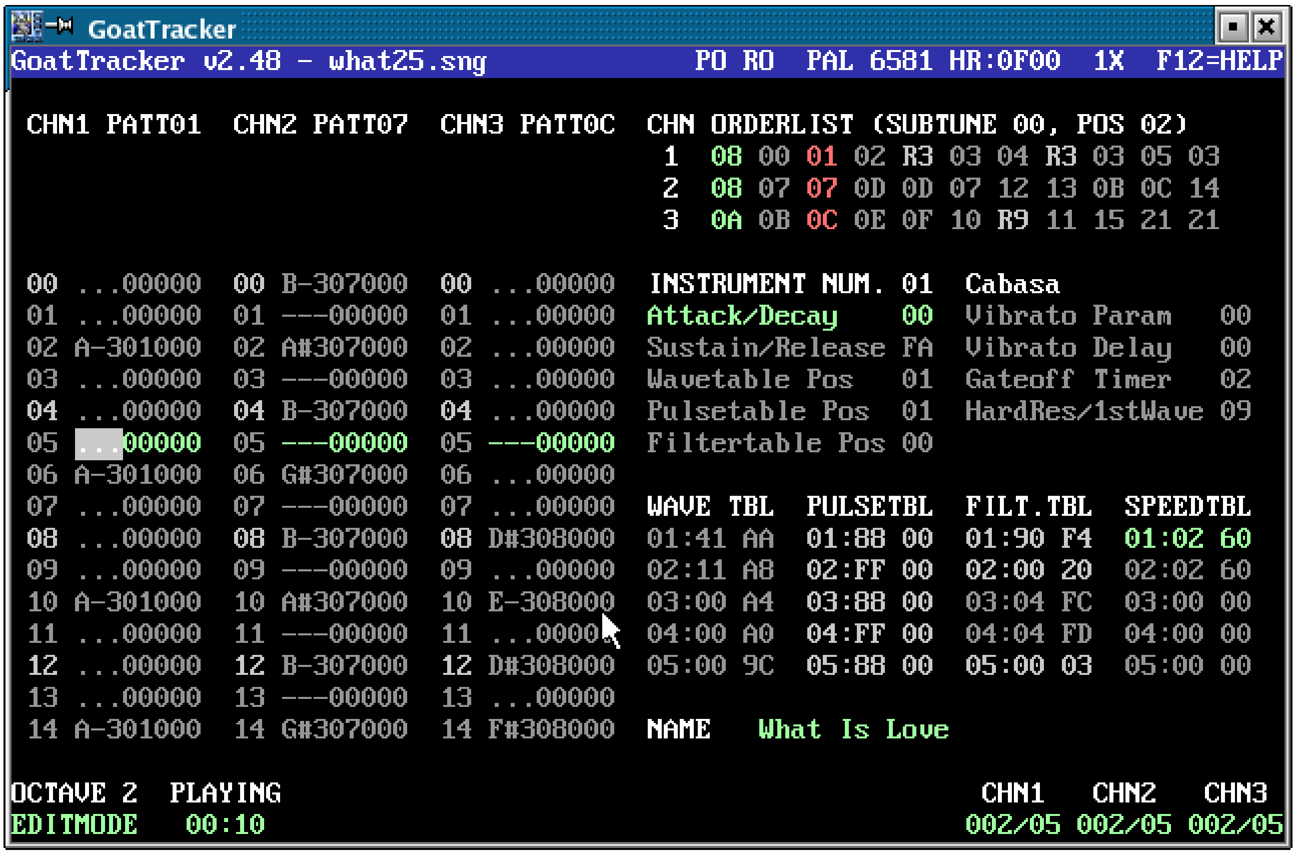
	146	147	148	149
V1	00			
V2	00			
V3	🟩	27 Synth Voice		

Patterns and instruments

One of the points that was not good in GT2 was the impossibility to change the ADSR of a instrument into the wavetable. This is very useful for having two instruments in sequence to reproduce (like a Drum + Bass) that may have different ADSR for better result.

But this was in version 2.35 (that I start to use): in next version the function was implemented letting a pattern command to be executed from wavetable.

Improvements



```
GoatTracker
GoatTracker v2.48 - what25.sng              PO RO  PAL 6581 HR:0F00  1X  F12=HELP

 CHN1 PATT01  CHN2 PATT07  CHN3 PATT0C  CHN ORDERLIST (SUBTUNE 00, POS 02)
                                         1  08 00 01 02 R3 03 04 R3 03 05 03
                                         2  08 07 07 0D 0D 07 12 13 0B 0C 14
                                         3  0A 0B 0C 0E 0F 10 R9 11 15 21 21

 00 ...00000  00 B-307000  00 ...00000  INSTRUMENT NUM. 01  Cabasa
 01 ...00000  01 ---00000  01 ...00000  Attack/Decay    00  Vibrato Param   00
 02 A-301000  02 A#307000  02 ...00000  Sustain/Release FA  Vibrato Delay   00
 03 ...00000  03 ---00000  03 ...00000  Wavetable Pos   01  Gateoff Timer   02
 04 ...00000  04 B-307000  04 ...00000  Pulsetable Pos  01  HardRes/1stWave 09
 05 ...00000  05 ---00000  05 ---00000  Filtertable Pos 00
 06 A-301000  06 G#307000  06 ...00000
 07 ...00000  07 ---00000  07 ...00000  WAVE TBL  PULSETBL  FILT.TBL  SPEEDTBL
 08 ...00000  08 B-307000  08 D#308000  01:41 AA  01:88 00  01:90 F4  01:02 60
 09 ...00000  09 ---00000  09 ...00000  02:11 A8  02:FF 00  02:00 20  02:02 60
 10 A-301000  10 A#307000  10 E-308000  03:00 A4  03:88 00  03:04 FC  03:00 00
 11 ...00000  11 ---00000  11 ...00000  04:00 A0  04:FF 00  04:04 FD  04:00 00
 12 ...00000  12 B-307000  12 D#308000  05:00 9C  05:88 00  05:00 03  05:00 00
 13 ...00000  13 ---00000  13 ...00000
 14 A-301000  14 G#307000  14 F#308000  NAME   What Is Love

OCTAVE 2  PLAYING                                        CHN1    CHN2    CHN3
EDITMODE   00:10                                       002/05  002/05  002/05
```

Before speaking about some improvements that I see for the editor, one point that now is perfect is the delay between a note played and the time you listen it in the PC. With GT in Linux there was a delay too high and you cannot follow a pattern being played. In GT2 all is in realtime.

I don't know if this depends by the editor itself or simple by improved sound libraries in linux, but this is essential for a good use of the editor.

Now the points I see that could be better, but not essential, to have:

- You can start to play the song from the current start pattern position: with this function you can, for example, listen carefully only the end part of a tune, without to start always from the beginning. Using some keys you can move the starting pattern into the editor. The main problem here is if you use the Rxx repeat command, as GT2 move command moves the position of the pattern of one voice for all the Rxx in one step: this means that after this you have the relative position of the tree voices that are no more synchronized, and so you will listen an out of tune song. This is not a big problem as you can simple don't use the Rxx command or you may use it only when the tune is finished.
- When you edit a pattern where you use lot of different instruments at each ticks, it could be very difficult to be sure to have insert the right instrument and in this case having different colors for different instruments could give an eyes help. That means, for example that you should have a color picker into instrument editor, and then that instrument will be showed with that color into the pattern. Maybe the use of more color should be used even in other cases, like in tables, but they are not essential (it's like to have a programming editor where you have code completion or not: you can program anyway, but in the first case you have more help into your work).
- MK4 did not sound onto it. Even if MK4 (linux) driver has some similitude with MK3, maybe

the 2 sids support into MK4 has break the sound if it is generated in the mode GT2 manage the virtual drive port. However GT2 did not support yet MK4, so if it works this will be only a side effect of MK3 support. This is actually the features I miss, as I have to save the tune as Sid and going to listen it into MK4 with a sidplayer for having the real chip sound.

- The save function is very dangerous: it is very easy to overwrite an existing song file as it not asks for overwriting a already present file (this come out when you press F11 instead of F10 for loading another tune)
- The Undo for commands is a features that in a PC based program could be implemented easily and that can be useful in some cases (sometimes I have to reload a previous version of the tune as when I listen that the last steps I made was not what I want, trying to going back to the last operations could not so simple).

Conclusion

I have found the tables (wavetable, pulsetable and filtertable) very powerful as now you can have very complex instruments that you can control accurately. The other point that GT2 has over other editors, is that his user/control interface is very intuitive and you have all under you keys (or even under the mouse, as now you can even use it). As soon as MK4 will work onto it, I'm sure to have all I need from the editor.

The other winning point of GT2 is the accurate documentation that follow the program (and even in the F12 help screen) as all the features are describe in all details.

Relative on my cover, I can only says that maybe I use very strong drums and a sort of old-school sound in some part. But as I like it, I think that my task is completed for me.

You can download the tune here:

http://digilander.iol.it/ice00/download/what.zip

SIDin 10 end