Odin 13

“Nathalie Baye”

Vice snapshot with Vice palette

Made with the GIMP from a NB photo
and converted to C64 320x200
Hires Mode Bitmap
by Stefano Tognon
in 2008

“Tracker forever”
...

Team Ice **Free Software Group**

Gwin 13
version 1.00
28 June 2010

SIDin Contents

General Index

Editorials

Stefano Tognon <ice00@libero.it>

Hi, again.

It is long long time that this number is being writing. Delay after delay, but finally it is ready :)

You will see that the news section is very big this time as the previous number was released in middle 2008.

The time for this delay is that my activity were very busy in this periods and most of the work where putted into the programming of my personal music editor: JITT64.

This number, in fact, shows you all about JITT64.

In the first article we will see how JITT64 works from an user perspective, so after reading this you will be able to produce music with the tracker.

The second article, instead, shows how the player works by analyzing his implementation: there are lot of technical stuff in it.

For finish this editorial I like to give two words about Facebook. One night of many months ago I just see one name into a box that Facebook use for let you find other people that maybe have some of your interested or that you can know: Wally Beben!!

Yes, Wally Beben the famous SID musician that made epically music like “Dark Side” :)

He did not consider him so famous and he is out of scene for so long time right now, but it is always a pleasure to speak with this legend people, so thanks to Facebook for have proposed me to find him.

Bye
S.T.

News

Some various news of players, programs, and competitions:

- Goattracker at SF
- HVSC #49
- High Voltage SID Collection Search v1.4
- MSSIAH
- Iphone Sid Player
- Sidplayw2.5
- CGSC updates
- HVMEC 0.9
- HVSC #51
- HardSID Uno/UPlay
- ACID 64 Player 2.2-2.4
- Catweasel MK4*plus*
- C64.sk Sidcompo 8 Radiostream
- C64.sk Sidcompo 8
- GoatTracker v2.68/2.69
- XSidplay 2.0.3
- ACID 64 Player Pro v3.0.0-3.0.2
- HVSC #50
- HVSC #52
- HVSC #53

Goatracker at SF

Goattracker, the crossplatform C64 music editor, is now available even at Sourceforce: http://sourceforge.net/projects/goattracker2/

You can so check the developed version thrown svn access and can contribute to it using the features that Sourcefoge let you use.

Looking at the svn repository, we can find this interesting new document made by Simon Bennett about the tracker:

Track effects

000	Do nothing.
1ST	Slide up. (ST is an index in the speedtable, left and right columns combined.)
2ST	Slide down (as above).
3ST	Slide to note. As above, or ST = 00 slides instantly.
4ST	Vibrato. Left column of ST index is frequency, right is amplitude
5AD	Set attack/decay.
6SR	Set sustain/release.
7XY	Set waveform register to XY. Wavetable takes precedence.
8WT	Set wavetable index.
9PT	Set pulsetable index.
AFT	Set filtertable index.
BRM	Set resonance to R and channel bitmask to M.
CCO	Set filter cutoff to CO.
DXY	Set master volume to Y. If X is not zero, copies XY to timing mark location (player address + 3F)
EST	Global funk tempo. Shuffles between tempo specified in left and right bytes at speedtable index ST.
FXY	Set tempo. 03-7F sets global tempo. 83-FF sets channel tempo + 80. Tempos 00-01 use the funk tempo values set by the E command above.

Signed values

01 -> 7F	Up
FF -> 80	Down

Wavetable (left)

00	Null command.
01-0F	Delay step by 1-15 frames
E0-EF	Inaudible
F0-FE	Execute track effect 0-E with right side as data
FF	Jump to table pos on right side
	Values from here are bitmasks
x1	Gate and initiate attack/decay. (0 here initiates sustain/release.)
x2	Hardsync. Ch1 uses Ch3, Ch2 uses Ch1 and Ch3 uses Ch2
x4	Ringmod, channels as above
x8	Test bit. Resets oscillator
1x	Use triangle
2x	Use sawtooth
4x	Use pulsewave
8x	Use noise

Wavetable (right)

00-5F	Relative notes* upward
7F-60	Relative notes* downward
80	Unchanged note
81-DF	Absolute notes* C#0 to B-7

Relative notes (wavetable right)

Horizontal is octave shift, vertical is interval

	-3	-2	-1	+0	+1	+2	+3	+4	+5	+6	+7
r	-	68	74	00	0C	18	24	30	3C	48	54
b2	-	69	75	01	0D	19	25	31	3D	49	55
2	-	6A	76	02	0E	1A	26	32	3E	4A	56
b3	-	6B	77	03	0F	1B	27	33	3F	4B	57
3	60	6C	78	04	10	1C	28	34	40	4C	58
4	61	6D	79	05	11	1D	29	35	41	4D	59
b5	62	6E	7A	06	12	1E	2A	36	42	4E	5A
5	63	6F	7B	07	13	1F	2B	37	43	4F	5B
b6	64	70	7C	08	14	20	2C	38	44	50	5C
6	65	71	7D	09	15	21	2D	39	45	51	5D
b7	66	72	7E	0A	16	22	2E	3A	46	52	5E
7	67	73	7F	0B	17	23	2F	3B	47	53	5F

Absolute notes (wavetable right)

Horizontal is note, vertical is octave

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0		81	82	83	84	85	86	87	88	89	8A	8B
1	8C	8D	8E	8F	90	91	92	93	94	95	96	97
2	98	99	9A	9B	9C	9D	9E	9F	A0	A1	A2	A3
3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
4	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB
5	BC	BD	BE	BF	C0	C1	C2	C3	C4	C5	C6	C7
6	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3
7	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF

Chord spellings

	major	minor	dim	aug	sus4	dim7	7	m7	b5	#5	b9	9	#9	11	#11	b13	13
root	04 07	03 07	03 06	04 08	05 07	03 06 09	+0B	+0A	-07 +06	-07 +08	+0D	+0E	+0F	+11	+12	+14	+15
1st Inv	78 7B	77 7B	7A 7D	78 7C	79 7B	77 7A 7D	+7F	+7E	-7B +7A	-7B +7C							
2nd Inv	04 7B	03 7B	03 7D	04 7C	05 7B	03 7A 7D											
3rd						03 06 7D											

Filtertable

00	Set cutoff as right column.
01-7F	Filter modulation step. Time in left column, signed* extent and direction of modulation in right column.
80-F0	Filter configuration. Filter mode bitmask* in left column (filter can be in multiple modes); resonance is first value of right column and channel bitmask* is second value.
FF	Jump to index in right column. FF 00 means stop.

Bitmasks

Filter mode	
80	none
90	LP
A0	BP
B0	LP & BP
C0	HP
D0	LP & HP
E0	BP & HP
F0	all

Channel	
0	none
1	1
2	2
3	1 2
4	3
5	1 3
6	2 3
7	all

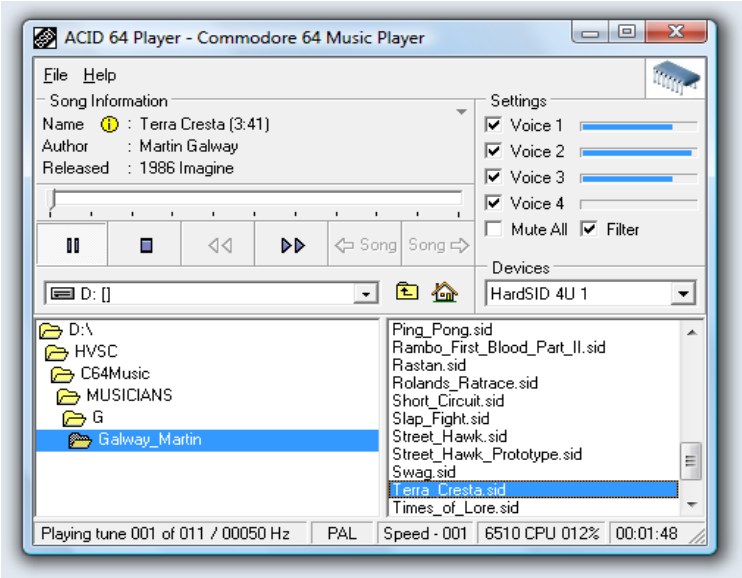
Pulsetable

01-7F	Pulse modulation step: time in left column; signed* speed in right.
8X-FX	Set pulse width. X is high value, right column is low value.
FF	Jump to index in right column. FF 00 stops the table.

Ravenspiral GoatTracker Command Chart Correct to GoatTracker v2.67. Part of the Ravenspiral collection. www.ravenspiral.com
GoatTracker is by Covert Bitops. For more info see http://sourceforge.net/projects/goattracker2

ACID 64 Player 2.2-2.4

The C64 Music Player for all HardSID devices and the Catweasel MK series was updated in June 2008.



What's new in version 2.2:

Improvements

- Improved response time on the HardSID 4U (new hardsid.dll required)
- Improved playback on multi core CPU systems during background playback on ISA/PCI cards
- Increased FIFO memory on Catweasel MK4 for better playback of digi tunes
- Song length database is now loaded when HVSC location is updated in preferences dialog

Fixes

- Clock is now displayed correctly during HardSID 4U playback
- Vista layout issues

What's new in version 2.3:

New

- Popup menu added to select sub tunes
- Device selection can be temporarily locked when automatic device selection is configured

Improvements

- Improved SID reset
- Display of track bar for sub tune selection changed
- Numeric keypad can now be used for some short cuts
- Added Voice 4 mute/undo mute short cut. See readme.txt file for all short cuts
- Relocation area is renamed to Free Pages and is showing auto detect memory ranges

Fixes

- Digi playback on ISA/PCI cards could cause timing problems on some systems
- Window size and position startup option is now shown correctly

What's new in version 2.3.1:

Improvements

- Button icons

Fixes

- Opening SID files via Internet Explorer is now possible. This makes it possible to play SID tunes via the online SID search of www.hardsid.com

What's new in version 2.4.0:

Improvements

- Emulation improvements
- STIL info can now be retrieved for a SID file if it is not played from the default HVSC location
- When HardSID 4U device is turned off while ACID 64 Player is running and turned on again, playback can continue after song restart
-

Download it from http://www.acid64.com/

HVSC #49

High Voltage SID Collection: Update #49
Date: November 09, 2008

Hello fellow lover of SID music! Nice you found some time to read through this script, to see what has been changed in the HVSC and for what reason. After this update, the collection should contain 36,081 SID files!

This update features (all approximates):

- ✗ 1059 new SIDs
- ✗ 54 fixed/better rips
- ✗ 2 fixes of PlaySID/Sidplay1 specific SIDs
- ✗ 10 repeats/bad rips eliminated
- ✗ 431 SID credit fixes
- ✗ 232 SID model/clock infos
- ✗ 9 tunes from */DEMOS/UNKNOWN/* identified
- ✗ 40 tunes moved out of */DEMOS/* to their composers' directories 14 tunes moved out of */GAMES/* to their composers' directories

Download HVSC Update #49 from the usual address: http://www.hvsc.c64.org/

Catweasel MK4plus

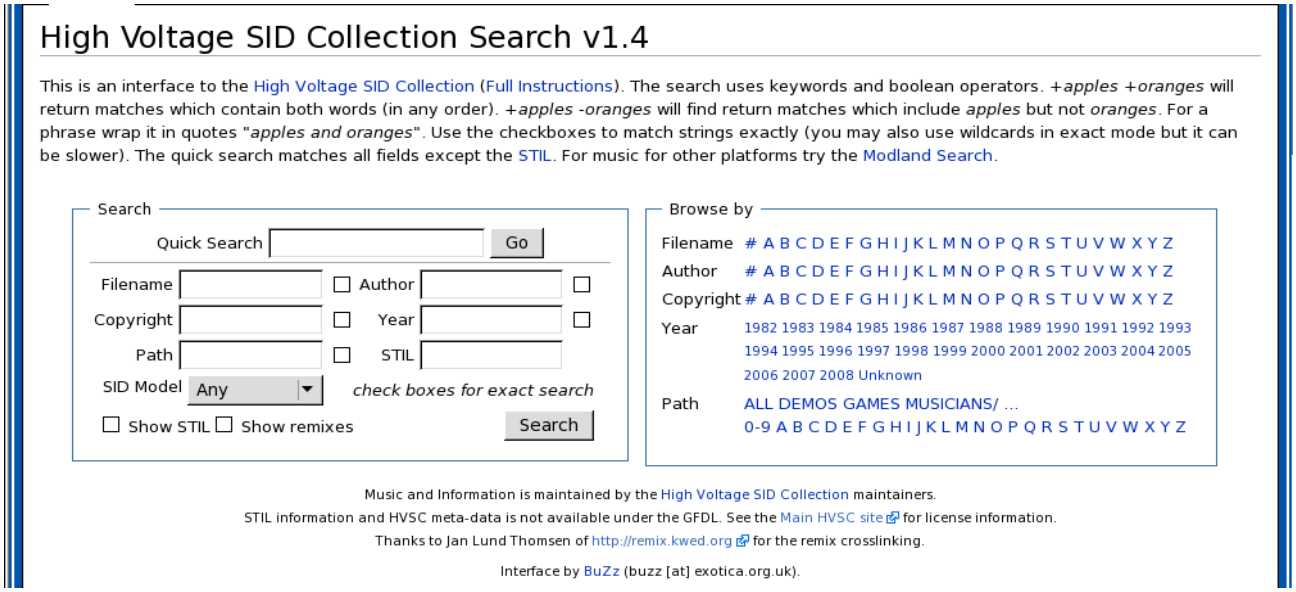
The Catweasel MK4*plus* replaces the Catweasel MK4, which is sold out and discontinued. The main changes are of cosmetic nature, but there have also been improvements made following customer feedback in some places.

The one easily visible change is that the new card is no longer low-profile PCI compliant. This feature of the old Catweasel MK4 was rarely used by customers, so it was decided to use the increased space for a better arrangement of the two SID sockets. These are more easily accessible now. Additional filters in the audio part are geared towards filtering noise from high-performance graphics cards and low-quality power supplies. Another novelty is the external audio jack and an angled internal audio connector for better accessibility.

Check: http://www.vesalia.de/e_catweaselmk4.htm

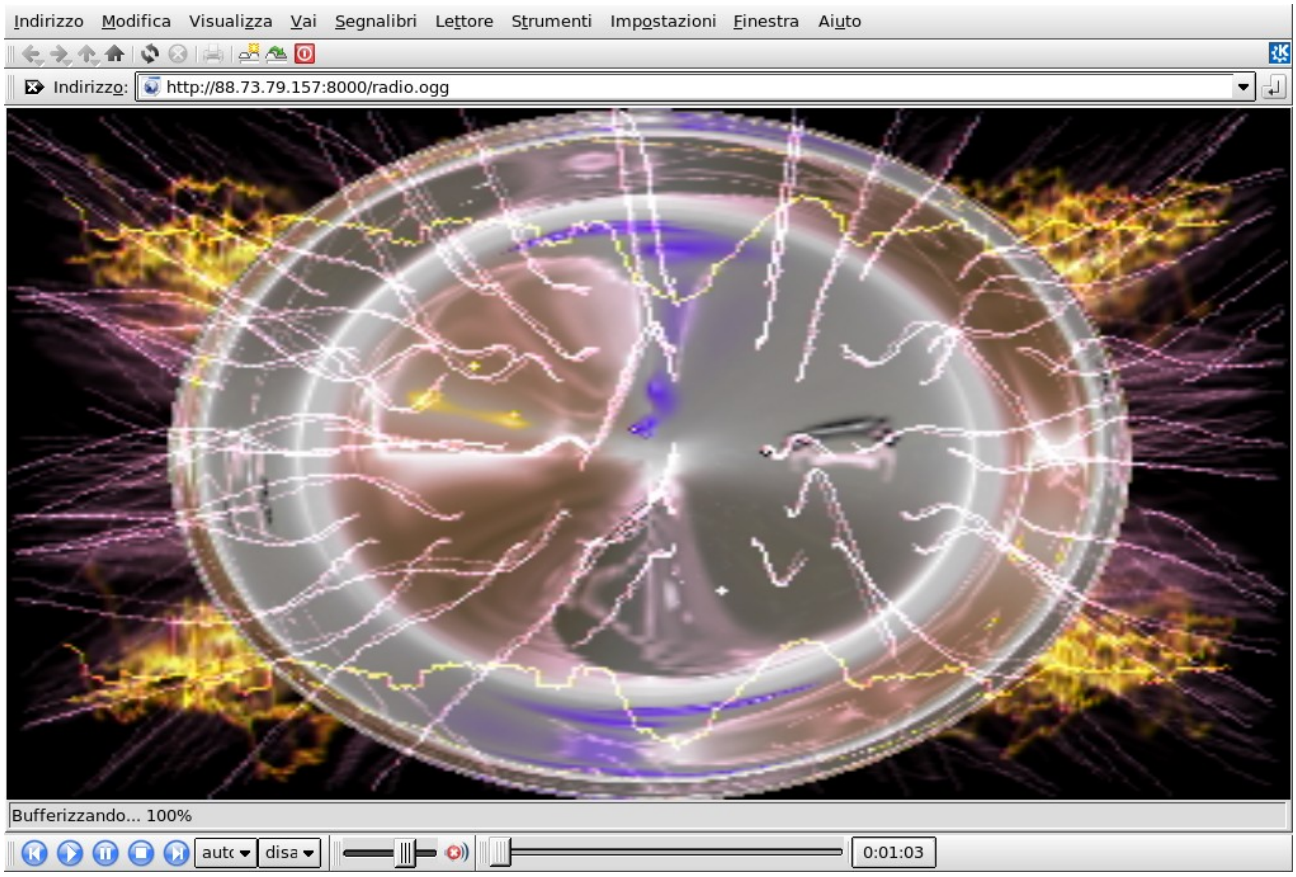
High Voltage SID Collection Search v1.4

A new online site for searching tunes in HVSC is available at:
http://www.exotica.org.uk/wiki/Special:HVSC



C64.sk Sidcompo 8 Radiostream

Within the start of voting phase of SidCompo 8, a new online radio stream was made available for listening all the compo tunes: http://88.73.79.157:8000/radio.ogg



MSSIAH

The MSSIAH is a MIDI cartridge for the Commodore 64.

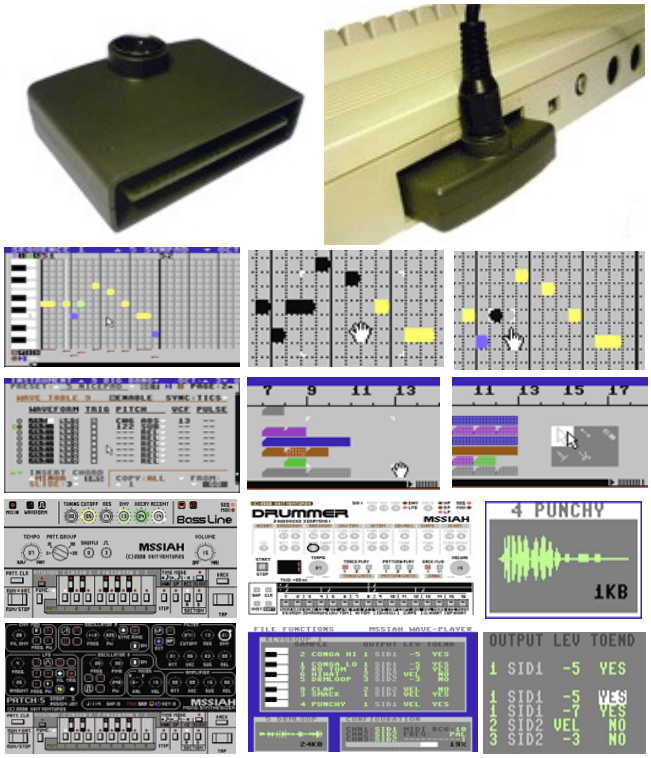
It contains a suite of music applications that starts instantly as you insert the cartridge and start up the computer.

With these applications you can play the C64's audio circuit (SID) via MIDI or standalone with the internal sequencers.

MSSIAH is short for MIDI SID Software Integrated Applications and Hardware and means that the cartridge contains both software and hardware to midify the C64. Since they are integrated you won't need hard-to-get MIDI peripherals to hook it up to your synthesizer or sequencer.

Just plug in a MIDI cable and off you go!

http://www.8bitventures.com/mssiah/



C64.sk Sidcompo 8

This year was the Last SidCompo (the 8 of the series) that www.c64.sk organized. Next year there will be some different kind of compo...

Here the result:

```
01. Eskimonika by Stellan Andersson (Dane)                              (852 PTS)
02. Intrinsic by Conrad/Viruz/Samar/Onslaught (Owen Crowley)            (850.5 PTS)
03. A Liquor Store Anthem by Randall                                    (787.5 PTS)
04. Love Land by Steven Diemer (A-Man/Xenon)                            (781 PTS)
05. Two Minute Jam by Josep Barwick (Stainless Steel)                   (675 PTS)
06. Stay Chill by Marcin Majdzik (PSycHo)                               (641 PTS)
07. Rocco Siffredi Invades 1541-II by Kamil Wolnikowski (Jammer)        (638 PTS)
08. Stretch Marks by Hein Holt (Hein/Vision)                            (627 PTS)
09. Christ 69 Electroclash Deluxe by Arman Behdad (Intensity)           (615 PTS)
10. Wander Fool by Vincent Merken (_V_)                                 (563 PTS)
11. Johnny Rocket by Uneksija (Antti Pitkämäki)                         (534 PTS)
12. Back to Planet:dATA by LordNikon/Dekadence                          (513 PTS)
13. fuckyou.progressivedata.fuckme by Sascha Zeidler (Linus)            (508 PTS)
14. Pixel Hell Level 9 by Hlkon Repstad (Archmage of Instinct)          (498 PTS)
15. Drunken Ninja Dance by Rambones                                     (454 PTS)
16. See You Later Oscilator by Kristian Myklebust (kribust)             (451.5 PTS)
17. Elegy by Peter Bergstrand                                           (392 PTS)
18. Levitation by Henne / The Dreams                                    (385 PTS)
19. Disco Dream by Richard Bayliss                                      (380 PTS)
20. Mustelid by Hukka/Dekadence                                         (359.5 PTS)
21. Ninja Life by G-Fellow / CiViTaS (Gerhard Flagge)                   (353 PTS)
22. Upgrade by Dennis Hildingsson (Rusty46)                             (286 PTS)
```

iPhone Sid Player

In the App Store for the iPhone there is available a Sid Player for very low cost.

From http://iphone.vanille.de/sidplayer/ we can get more informations:

“Sid Player brings you the sound of the Commodore C64 to your iPhone and the iPod Touch. Enjoy game classics such as 'Commando', 'Arkanoid', 'The last V8' or listen to the music of 'Rob Hubbard', 'Martin Galway', and many others.

Sid Player gives you access to the High Voltage Sid Collection (HVSC) consisting of over 33.000 songs from 1125 authors. Thanks to the unique sound aesthetics, Sid music is now considered an art of its own and there's a constant flow of new creations.

A Sid file occupies only few kbytes, thus you can quickly download C64 music via EDGE or UMTS and listen for hours. The search function enables you to find your favourites in a blink.”

GoatTracker v2.68/2.69

The new versions of Goattracker2 is available at Sourceforge:

v2.68:

- Fixed set tempo -command overwriting frequencytable in 1 or 2 channel modes.
- Fixed sound uninit crash with multicore processors (?)
- SID register write order tweaked to resemble JCH NewPlayer 21.
- Unbuffered playroutine optimized & modified to resemble buffered mode timing more.
- New reSID-fp engine (with distortion & nonlinearity) from Antti
- Lankila integrated. Activated with /I command line option parameters 2 & 3.
- Command quick reference by Simon Bennett included.

V2.69:

- Fixed click bug in reSID audio output.
- Newest reSID-fp code integrated.
- reSID-fp filter parameters adjustable from the configuration file.

Sidplayw2.5

This is just an update of the old Sidplay2/windows. Some facts:

- Contains the latest (v32) filter distortion emu by Antti Lankila
- Uses a lot of raster time (cpu). 35-45% cpu on my P4 2,4 GHz.
- Looks just as boring as the old Sidplay2.
- Works in Wine!
- Sounds really great ;-)

Download at: http://noname.c64.org/csdb/release/?id=76056

XSidplay 2.0.3

It is available from March 2009 the new version of XSidplay2 (the sid player for Linux): http://sourceforge.net/projects/xsidplay2/

Changes:

- Allow to compile in Windows with QT3
- Remove LCD display bug when changing audio
- Add experimental support for ALSA sound library
- Add low level setting in audio
- Add clock frequency string (useful if ANY or UNKNOWN field is set)
- Add multispeed indication (need to patch libsidplay2)

CGSC updates

The Compute's Gazette Sid Collection maintained by Peter Weighill has made some little update in March 2009, a big one in May 2009 (26% of increase) and one in May of this year.

Now the collection contains: 8395 MUS files, 1966 STR files and 2631 WDS files

Download from: http://www.c64music.co.uk/

ACID 64 Player Pro v3.0.0-3.0.2

In May 2009, the ACID 64 Player has released a important version of the program, followed after by other updates.

New (3.0.0):

- Fast incremental SID file search
  on title, author, year and publisher fields

- Seek through SID tunes via slider bar
- Digital clock
- Scroll wheel support when hovering over grids and list boxes
- Anonymous usage statistics

Improvements (3.0.0):

- Tree view of folders
- SidID search in properties menu is done in background now to access dialog faster
- Emulation improvements
- Keyboard navigation improvements
- Many small improvements/fixes

Fixes (3.0.1)

- On some machines the index file wasn't created correctly (re-indexing is required)
- Indexing of folder with a few SIDs is fixed
- Seek speed isn't dependent anymore on fast forward limitation
- STIL and SLDB info are now working for files outside HVSC
- Minimal horizontal width isn't locked to previous width anymore

Improvements (3.0.1)

- STIL window can always be opened now and doesn't close automatically anymore
- Improved handling of SLDB entries
- Digital clock color changed
- Changed a few keyboard short cuts
- A few other small improvements/fixes

Fixes (3.0.2)

- No error anymore when last directory was a root folder
- Corrected memory bank setting for PSID tunes
- Resizing window was corrupting active row of search grids
- Minimal width is now set correctly

Improvements (3.0.2)

- Last played file is remembered now when ACID 64 is started
- When Folders tab is clicked, the current selected file will always be visible
- Special characters Ø and ø can now be searched by o or oe (re-indexing required)
- Pressing escape in search boxes will now select the text instead of clearing the input
- Selecting a search filter will change the color of the filter box
- Copy filename strips HVSCRoot automatically
- Seeking (dragging thumb) can now be cancelled by pressing escape key or by pressing right mouse button
- Scrolling search results will now update rows immediately without releasing the thumb
- Clicking on a folder name in Folders tab will expand the folder
- Changed "Browse" tab name to "Folders"
-

Download at: http://www.acid64.com/

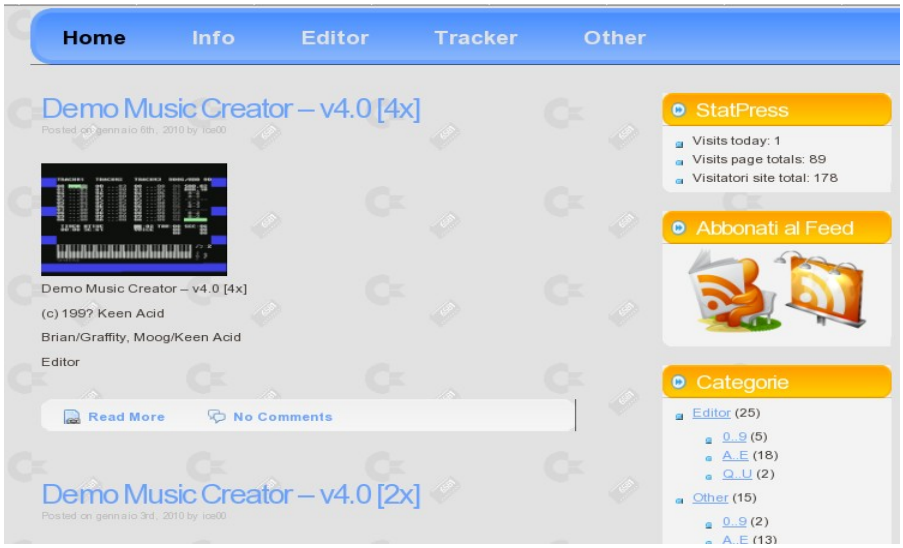
HVMEC 0.9

The High Voltage Music Engine Collection ha released a new version at:
http://digilander.libero.it/ice00/tsid/hvmec.html

The collection now contains:

- 140 editors
- 84 trackers
- 70 others program

A new Wordpress version of the collection will be created at this location:
http://hvmec.hellospace.net/blog



HVSC #50

Date: May 01, 2009
After this update, the collection should contain 36,000 SID files!
This update features (all approximates):

```
678 new SIDs
59 fixed/better rips
573 PlaySID/Sidplay1 specific SIDs eliminated
189 repeats/bad rips eliminated
482 SID credit fixes
135 SID model/clock infos
15 tunes from /DEMOS/UNKNOWN/ identified
10 tunes from /GAMES/ identified
36 tunes moved out of /DEMOS/ to their composers' directories
11 tunes moved out of /GAMES/ to their composers' directories
```

As you can see, a great amount of files were eliminated in this update.
We decided to remove all the old _PSID files that have a RSID equivalent.
Those _PSID files were only hacks meant for digitized sounds to be played in Sidplay1, but not on real C64. The _PSID files will be eliminated in every future releases as soon as new RSID versions of the same .sid files are added. For those still needing the _PSID files we have put them in a separate archive: http://hvsc.c64.org/Downloads/C64MUSIC_PlaySID.rar
Download HVSC Update #50 from the usual address: http://www.hvsc.c64.org/

HVSC #51

Date: August 22, 2009

After this update, the collection should contain 36,937 SID files!
This update features (all approximates):

```
950 new SIDs
93 fixed/better rips
6 PlaySID/Sidplay1 specific SIDs eliminated
14 repeats/bad rips eliminated
320 SID credit fixes
159 SID model/clock infos
8 tunes from /DEMOS/UNKNOWN/ identified
6 tunes from /GAMES/ identified
38 tunes moved out of /DEMOS/ to their composers' directories
14 tunes moved out of /GAMES/ to their composers' directories
```

Download HVSC Update #51 from the usual address: http://www.hvsc.c64.org/

HVSC #52

High Voltage SID Collection Update 52
Date: December 24, 2009

Hello fellow lover of SID music!
After this update, the collection should contain 37,801 SID files!

This update features (all approximates):

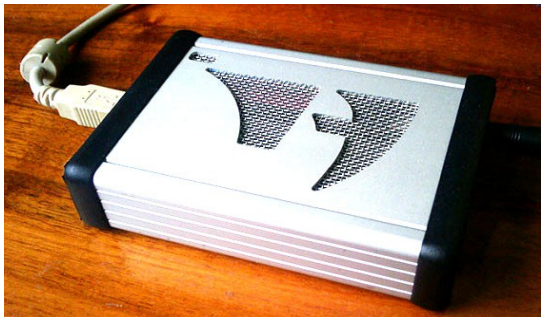
```
874 new SIDs
65 fixed/better rips
0 PlaySID/Sidplay1 specific SIDs eliminated
12 repeats/bad rips eliminated
470 SID credit fixes
174 SID model/clock infos
17 tunes from /DEMOS/UNKNOWN/ identified
2 tunes from /GAMES/ identified
25 tunes moved out of /DEMOS/ to their composers' directories
4 tunes moved out of /GAMES/ to their composers' directories
```

HardSID Uno/UPlay

Two new Hardsid cards are available:
http://www.hardsid.com/

- HardSID Uno
- HardSID UPlay

Here the features for all the cards:



HardSID Uno:Enjoy over 36000 wonderful C64 tunes in high-quality played back on a real SID chip!

- Ideal for SID Players, C64 Emulators, SID Trackers
- USB connection (compatible with both 2.0 & 1.1)
- No power supply required! Use it with a notebook just anywhere!
- Supported by the HardSID 4U Winamp Plugin! (seeking, sub-tune handling, IR remote controlling)
- Superior sound quality (..it is a HardSID!)
- Support for one SID chip of any version (old: 6581, new: 8580/6582)
- Updateable firmware over USB
- Drivers for Windows XP/Vista/Win7
- Cycle-accurate playback of your favorite SID tunes
- Digitized sound + high-speed playback with low CPU utilization
- Seamless playback of .sid tunes while you work on your PC
- One analog SID sound output (jack)
- USB connector for connecting the unit to a desktop PC or a Notebook

HardSID UPlay: Do you need both old and new SID versions for playback? You need the Hard-SID UPlay then!

- You can install two SIDs in it: one old 6581 + one new 8580/6582
- Switch between the two SIDs simply from the player software
- Ideal for SID Players, C64 Emulators, SID Trackers
- USB connection (compatible with both 2.0 & 1.1)
- No power supply required! Use it with a notebook just anywhere!
- Supported by the HardSID 4U Winamp Plugin! (seeking, sub-tune handling, IR remote controlling)
- Superior sound quality (..it is a HardSID!)
- Updateable firmware over USB
- Drivers for Windows XP/Vista/Win7
- Cycle-accurate playback of your favorite SID tunes
- Digitized sound + high-speed playback with low CPU utilization
- Seamless playback of .sid tunes while you work on your PC
- One analog SID sound output: Outputs the sound of the actually selected SID chip (jack)
- USB connector for connecting the unit to a desktop PC or a Notebook

HVSC #53

Date: June 25, 2010: After this update, the collection should contain 38,714 SID files!

This update features (all approximates):

```
1045 new SIDs
897 fixed/better rips
20 PlaySID/Sidplay1 specific SIDs eliminated
133 repeats/bad rips eliminated
1076 SID credit fixes
81 SID model/clock infos
23 tunes from /DEMOS/UNKNOWN/ identified
10 tunes from /GAMES/ identified
81 tunes moved out of /DEMOS/ to their composers' directories
11 tunes moved out of /GAMES/ to their composers' directories
```

TSM & Freedom Interview!

by Stefano Tognon

As this number comes out many times after the previous issue, I decided so to add many interview to compensate for this ;)

I finally got two interviews from TSM and Freedom, two composers that take place to the develop of Pushover game. The interview is made as “double”: (almost) same questions to all. This is a kind of interview that is quite popular in Italian Television :)

I had used the [/] in the question to separate what is changed if the questions where a little different.

Hello Carmine [/] Freedom, could you introduce yourself and tell us what you do in real life?

TSM: Hello everyone. My name is Carmine Migliaccio and I was born in Napoli, Italia, 30 years ago. I was a computer programmer, but resigned from my job. Now I am looking for a new job and I hope to find one that has little or nothing to do with computers.

Freedom: Hi, I am from Italy. My job deals with mechanics.

When did you start to use a C64 and why did you start to compose music using a so old chip?

TSM: My parents bought me a Commodore 64 in 1988. It was my main computer till 1992, when I got an Amiga 500. I used my C64 mainly to play games and, from time to time, I wrote some simple Basic programs. I also made some experiments with the SID in Basic, but at that time I didn’t know anything about Machine Language or music editors.
My first serious attempts at making music with the C64 came much later: around 2003! That’s why I don’t consider myself a scener nor a musician. Anyway I started to make music with the SID because I can’t play any conventional music instrument and I wanted to play one without the hassle of practicing for years.
I had spent long time creating music modules on PC with FastTracker II but I always ended up using the same few samples for anything and that was starting to bore me. I tried to create new samples and I also found huge sample libraries on the Web, but the whole thing simply didn’t stimulate me anymore.
The sound of the SID is the thing that most amazed me when I got my C64 back in the day and I still think that sound is the field where the C64 is most powerful. So, sooner or later, I had to do something with it.

Freedom: I started using a C64 when I was a kid, in late 1992. It was given to me as a gift for my birthday. I immediately started to code with BASIC and did some little experiments with music, but nothing worth noticing. I composed my first tune only in 2005, using Odin Tracker on VICE emulator.
I think I am in SID music because of nostalgia of the great fun I had with the Commodore 64 in my childhood. I remember I used to listen to loading and game tunes very often, without being that much interested on the game itself.
I do love SID music because of the 3 channel limit of the SID. You should be able to make the listener believe that there are more than just 3 voices... and that's a really interesting challenge which requires quite a different approach of that used in common music.

What editors have you used for composing music and what made you choose them?

TSM: Before 2003, I had made some attempts at learning some editors, but I found each of them to be too complicated for me. Then, I found CyberTracker by Cyberbrain. It was very similar to FastTracker II, the famous MS-DOS tracker that I knew very well. So, learning to compose music with CyberTracker was just a matter of understanding how the SID worked. I already knew most of it, so I could make my first tunes very soon. There was a big problem, though: the packer was (and still is) in early beta stage. As a result, packed songs were too big and used too much rastertime. Furthermore, due to a bug, some songs could not be packed at all. This meant that music made with CyberTracker was almost unusable for demos, games, etc. so I stopped using it.
Later, I wanted to take a look at C64 machine language and, as an exercise, came up with my own music routine. It's the simplest routine one could imagine, with no effects whatsoever, no hardrestart routine and no memory optimizations. It works fairly well and uses a laughable amount of rastertime. I made 2 songs with it (bath.sid and willow.sid), then I lost interest.
The turning point came in 2006. Ian Coog, Roberto and Pippo79 were making a simple little game called "Pick Up Sticks" and asked me to make a tune for it. I made it in CyberTracker, but it was too big, so I decided to give GoatTracker a chance. About two weeks later, I had finished the new version of the tune. Using GoatTracker was a real pleasure for me, because it has mouse support and an interface very similar to that of FastTracker II. It is thought in a very rational way.

Freedom: I started with Odin Tracker but immediately quit using it because of its strange way of handling filters. Also, Odin Tracker routines are not very efficient and so too much rasterlines are required. Then I switched to Goat Tracker which is a very easy to use cross-platform SID tracker.
In fact, now I always use Goattracker. It is easy, it is powerful, very efficient and it comes with a good documentation.

Are there any other music editors that you would like to use in the future?

TSM: GoatTracker is pretty much everything I need. Its only flaw is that it's a cross-tool. If there was an editor designed for the 80-column mode of the C128, with 1351 mouse support and Fast-Tracker-like interface, I would certainly love it.

Freedom: Maybe I would like to use a native tracker for C64 like JCH. But Goattracker just suits my needs.

Duration based editor (like DMC, ...) vs tracker style editor (like JCH, ...): what is your opinion about the two types of editors?

TSM: Well, I believe the answer is easily predictable by now. I honestly don't understand duration based editors, so I have no opinion about them. I really love tracker style editors, because they match the way I think music myself.

Freedom: I absolutely prefer tracker style editors.

Have you ever thought to compose music using samples for having more that 3 channels playing into a song?

TSM: I did think of it, but never tried.

Freedom: I actually prefer to force the 3 analog channels to create the illusion of more than 3 channels (at least, I try it, of course). However, I did a 4 channel sid tune using Pollytracker, but I didn't release it.

You had produced [/] are producing some music stuff for the Pushover game (a game converted from Amiga to C64, not yet released). Have you anything to tell us about this work?

TSM: Well, it was an exciting task indeed. Pippo79 and Raffox were (and still are) doing a very professional job. I tried to get the songs as close as possible to the Amiga originals. I made some of them from scratch and some by using format conversion tools. In the latter case, IIRC, I used some tool to convert the MED originals to MOD. I performed some preliminary adjustments on the MOD files, then I converted them to GoatTracker and added instruments and final corrections. The title tune is made this way and I think it's the only tune of mine that will appear in the final version of the game. When I gave up for personal issues, Freedom took over as musician. This game is gonna be something special.

Freedom: I have been doing many tunes for this game. I had a stop but I am going to continue. This game is a really, really nice project in my opinion. Trying to port those Amiga mods is quite fun I must say.

Have you any important music/project planned to realize in future?

TSM: No I have not. I do have one unreleased tune, though.

Freedom: No, nothing at the moment.

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

TSM: They both have their pros and cons. I prefer the real thing, because I like to see old electronics still functioning. I also like to put my hands in the hardware and mess with it :)

Freedom: I use emulators because I don't have enough room for original hardware. However, I think the best thing is to use both of them. For many things, emulator are far more practical, but if you want to watch a demo I think the real machine is the next best thing around.

6581 vs 8580 chip: any (musical) preference?

TSM: Well, I prefer the 8580. It's more versatile as it allows more combined waveforms. It is also very consistent, as every 8580 sounds the same, unlike the 6581. Inconsistency isn't the only problem with 6581's: they are very fragile. They often fail and sometimes they only work partially. This is awful, because a user may think he has a working SID and curse the author of a tune for the horrible sounds coming from the speakers.

Freedom: I prefer 8580 chip. Way too open filters but it features cleaner waveforms and it has less noise. For instance, when you change filter type you don't get an unwanted noise as noticeable as the one you hear in 6581. This makes it possible to change filter type on an instrument while a note is being played and you get more sophisticated sounds. Also, I do love $51 waveform on 8580.

What is the worst and the better sid you composed?

TSM: Worst tune: "Q-Game".
Best tune: don't know, maybe "Vieni avanti cretino", but it's a cover.

Freedom: Considering released stuff, my worst sid maybe is Exploring New Worlds. My better one is probably Dreamlights, a cover of a tune by Chris Huelsbeck originally intented to be used in Turrican.

Who are your best sid authors?

TSM: The first names that come to mind are Ben Daglish, Jeroen Tel, Rob Hubbard, Tim Follin, David Whittaker and Chris Huelsbeck.

Freedom: Musicians of commercial games: Chris Huelsbeck, Jeroen Tel and Rob Hubbard. Talking about sceners: Dane, Drax, Jeff, Linus.

What are the best sids ever in your opinion?

TSM: Krakout's music, of course. I also like Ghouls'n'Ghosts' title tune very much.

Freedom: Talking about scene music: I really love Arctic Circles by Dane. Dazzler by Mitch and Dane. Special Agent Rocco Montefiori by Linus is also a very good sid in my opinion. I love Ode to C64 by Jeff too.
From videogames, I really like ACE 2 by Rob Hubbard, Cybernoid II by Jeroen Tel... The list would be endless I think.

Finally, many thanks for the time you gave for this interview, and now would you say something else to the our readers?

TSM: Thank you for the opportunity. What can I say to our readers? Enjoy your life, love each other and play Pushover when it's out! Byez!!

Freedom: I wish I had enough time to enjoy C64 music as much as I want... I hope I'll be able to release new stuff in the near future. Bye!

JITT64 Tracker

by Stefano Tognon <ice00@libero.it>

JITT64 (Java Ice Team Tracker) is my C64 cross-platform music tracker. It is written in Java for being portable and use JSidPlay library for sound reproduction.

It is born with the idea to have a very freedom instrument implementation, using full features tables. Essentially you can have an instrument that use up to 2KB of data for its definition as the tables are not shared between instruments (only the packer will apply optimization in tables if possible for reduce space usage).

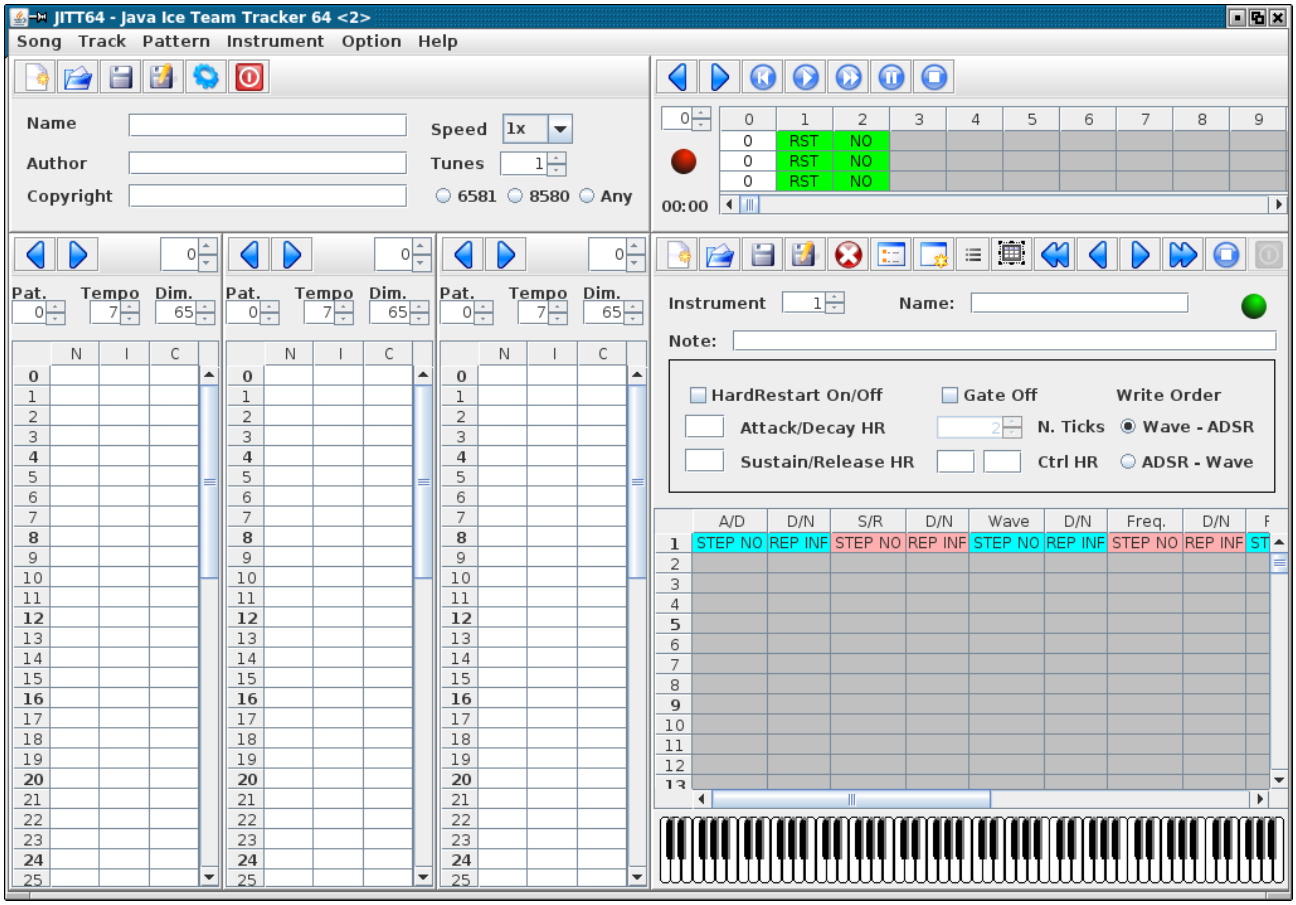
Due to this, the player is very rastertime consuming, but maybe you could be able to write even up to 4X tune with the editor.

In this article we will see how the editor works and in the next we will look even from the inside of his implementation, but remember that the program has an hypertext help that describe the program in all of his aspect.

Main screen

The application is composed by a big screen that is divided into 4 regions:

- song panel
- track panel
- pattern panel
- instrument panel



The song panel is where you can enter some information about the tunes you are composing:

- Name (of the songs)
- Author
- Copyright
- Speed of tunes (all tunes and voices have the same speed)
- Number of tunes
- Type of sid chip to use for these tunes

The track panel is divided into 3 rows (one for each voices) and contains the patterns and commands to execute for the song you are editing. It has even buttons for playback reproduction of the tune and a semaphore that indicate if the tune is ready to be played.

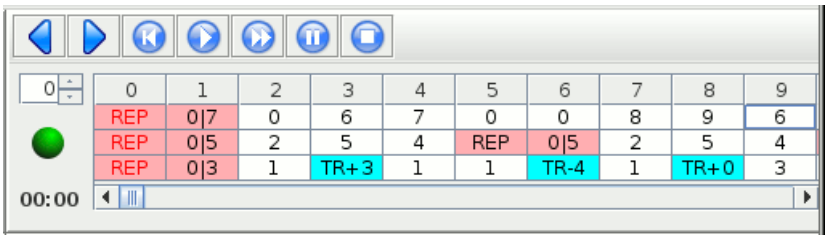
The pattern panel is divided into 3 sub-column regions: one for each voices and then you can insert the event (note/instrument/command) for each ticks. The tempo and dimension of the pattern can be selected independent from each one.

The instrument panel is composed by a upper region where you can insert some informations:

- Name (of instrument)
- Note (a comment about an instrument)
- Hard-restart/restart of note information

It has even a middle part where there is a big tables of values (that give the timbre of the instrument) and in a low part that is a piano roll, useful for testing the instrument.

Tracks

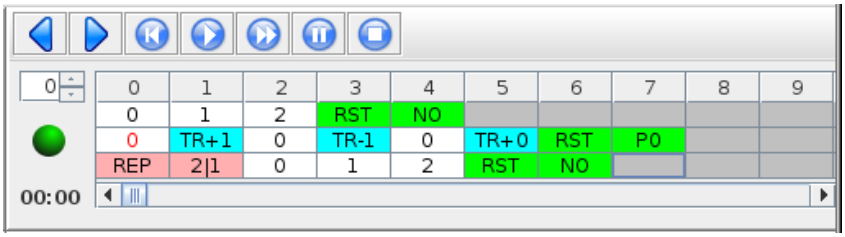


The tracks that compose a tune is entered into the above panel. The upper row is for voice 1, the middle for voice 2 and the lower row for voice 3.

In a cell of the table you can enter 4 type of commands (available with the right click of mouse):

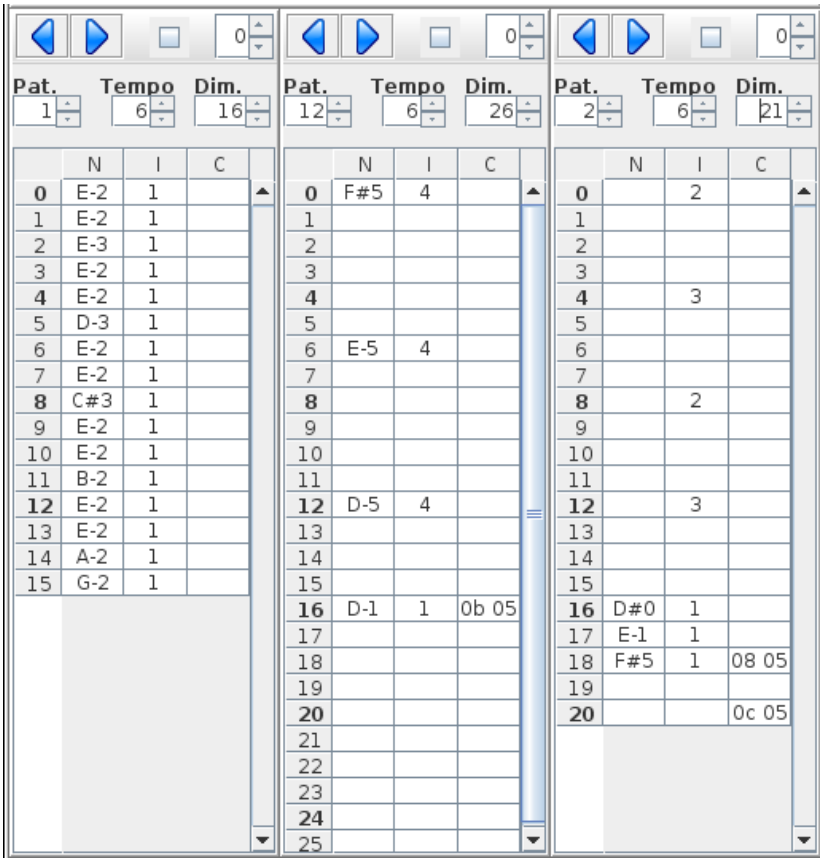
Value	***Description***
0..222	This is the number of pattern to play
REP	Repetition command. This command is for repeat a number of time the next commands. Unlike usual command, in JITT64, you can specify a sequence of commands to repeat. Dimension is 16, while the number of repeat is from 2 to 17
TR	Transpose command. Transpose can goes from -15 to + 15 steps. When you set a transpose, all the patterns that come after are transposed of that quantity.
RST	Restart command. This is always the last command in the track and it is for inserting the point (index in table) where we want to have the tune to restart. We can even let the tune finish without restarting.

The best way to see the commands in action is with some examples:



In the first voice, the pattern 0, 1 and 2 are play in sequences, then voice stops to play. In the second voice, pattern 0 is play at expected rate, than it is played 1 half step over, and then 1 half step below. This sequence restarts forever. In the last voice, the pattern 0, 1, and 2 are played two times in sequence, then the sound for this voice ends.

Pattern



The pattern panel is composed by 3 equals sub-panel: one for each voice.

In JITT64 I choose to have that each pattern has his tempo and his dimension. Although the dimension is quite common to be different for different patterns, the tempo is usually set by command.

With the JITT64's option you can set the tempo and pattern dimension to use by default.

In the sub-panel you have the splitter (and buttons) for selecting the pattern to edit, the splitter for selecting the octave to use for keyboard notes insertion and the checkbox for mute/unmute the voice when playing.

In the pattern table there are tree columns:

1. Notes
2. Instruments
3. Commands (with parameters)

Using right click you can open the menu to insert the notes/instrument/command to play at that time tick. It is also possible to insert notes with keyboard using Protracker or DMC mode.

With the same mouse key it is also possible to cut/copy/paste/clear selection from one pattern to another and load/save the pattern from/to a file

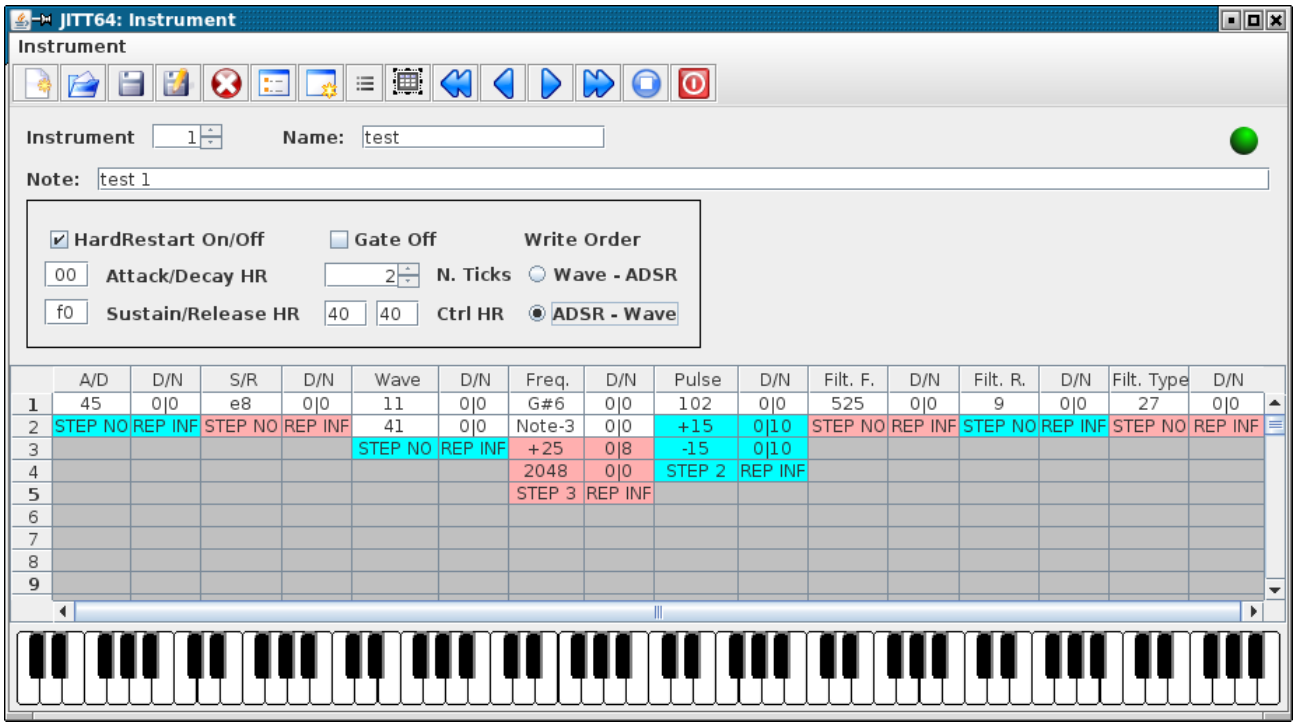
In JITT64 now it is implemented 16 commands, but more are planned to be added in future versions:

N	*Command*	*Description*
00	Stop Command	If parameter is not zero, this apply a stop action for the bit-field commands: ● bit 0: stop Arpeggio ● bit 1: stop Portamento up ● bit 2: stop Portamento dn ● bit 3: stop Tine Portamento ● bit 4: stop Vibrato ● bit 5: stop Pulse up ● bit 6: stop Pulse dn
01	Set Tempo	Set the tempo to use from this position in pattern. There is even the possibility to use the global (the one specify into the pattern spinner)
02	Set Attack/Decay	Set the attack/decay to use for the current voice
03	Set Sustain/Release	Set the sustain/release to use for the current voice
04	Set Volume	Set the volume of the tune. Volume starts at 15 at the initialization of tune.
05	Set Arpeggio	Set the arpeggio. There is a speed to choose for the arpeggio and two tone to add to base note and second note.
06	Set Portamento up	Set the amount (*2) of frequency to add for portamento up effect
07	Set Portamento dn	Set the amount (*2) of frequency to subtract for portamento down effect
08	Set Tone Portamento	Set the amount (*2) of frequency to add/subtract for a tone portamento effect.
09	Set Vibrato	Set the speed and the amount (*16) of frequency variations for a vibrato effect
0A	Set Pulse Slide up	Set the amount of variations for a slide up of duty cycle
0B	Set Pulse Slide dn	Set the amount of variations for a slide down of duty cycle
0C	Set Auto Fade out	Automatically fade out the volume with the given delay
0D	Set Filter Type	Set the filter type (high, middle, low) and voice where to use
0E	Set Filter Resonance	Set the filter resonance to use
0F	Set Filter Cut off	Set the filter cut off to use
10	Set Gate Sustain/Release	Set the sustain/release to use for current voice, but with gate release (you can go up in volume level)

Future command that will be implemented are:

- **Hardrestart command**: This will trigger an hard restart in next note even if instrument did not have it
- **Pointer command to instrument** This is for activate an effect in an instrument at level pattern. Note that this already works using instrument, but is not so intuitive
- **Hifi portamento** A portamento independent from octave/note
- **Hifi vibrato** A vibrato independent from octave/note
- **Vibrato slider** Automatically increase of frequency in vibrato

Instruments



The instrument window is the most complex of JITT64 as it is for creating the timbre of instruments. It has a upper part with some information about the kind of restart of note to use, a middle part with tables of values and a low part with a piano roll (for test the instrument, but it is not jet activated into the program).

Looking for the restart of note, there are tree kind of methods you can choose:

- **Never use an action**
  If you did not check the *Hard Restart* and *Gate off* fields, no action is taken when a new note start (so, all is left to you for preventing ADSR bugs).
- **Gate off**
  If you check the *Flag off* field, then the gate bit of sid register is released before the end of the note. How many ticks before this happen is given by the number you inserted into *N. ticks*. Read the notes below for more information about this.
- **Hard Restart**
  If you check the *Hard Restart* field, then those actions are taken before the end of the note (even here using the *N. ticks* for the timing):
  - **Attack/Decay HR:** this value is put as Attack/Decay when is time to start the HR
  - **Sustain/Release HR:** this value is put as Sustain/Release when is time to start the HR
  - **First Control HR:** this value is put as Control when is time to start the HR
  - **Second Control HR:** this value is put as Control when there is the last tick before the new note comes

  So, you can set all the values you want for having the HR.

The number of ticks you enter into the field have meaning that depend by the timing scale you choose into pattern. Here some examples:

- Pattern Timing 5, Number of ticks 2: effective ticks are 2
- Pattern Timing 5, Number of ticks 4: effective ticks are 4
- Pattern Timing 3, Number of ticks 4: effective ticks are 3

This is for taking easier to test for *Hard Restart* when a pattern is finish and a new one (that could or not containing a new note) is reached.

The last choice you have (but that affect all the actions for this instrument) is the order you want that sid registers will be written:

- **Wave – ADSR:** Wave register is writing before Attack/Decay/Sustain/Release
- **ADSR – Wave:** Attack/Decay/Sustain/Release are writing before the Wave register

This things if for letting you obtain good hardrestart (some one need accurate write timing to be achieved).

If you are wondering how to obtain and hard restart that use the test bit, you have two ways:

- Use Hard Restart features, setting AD/SR to use into HR, then make the first control register with the gate bit released. The second one need to be $09 (test bit + gate on). Use at least 2 ticks and set write order as ADSR-Wave

- Use Hard Restart features, setting AD/SR to use into HR, then make the first and second control register with the gate bit released. Use at least only one ticks. Then into the fist row of instrument table, set ADSR as you want for your instrument and the first wave as $09 (test bit + gate on). Set write order as ADSR-Wave. After this you will start the wave you need for your instrument.

The main part where you create the timbre of the instrument is the big middle table. It has 127 rows of commands and couples of columns for specific aspect of sid sound. Couple of column are differentiated by having two different colors. You can add a new cell using the *Ins* key and remove one using the *Canc* key. With the right click of the mouse in a column, you open a sub menu for enter values.

- **D/N Delay/Number of Repeat** (sometimes called D/R)
  This is common to all the couple of columns. With the first number you specify how long will take the delay between two commands to be executed. The second number specifies how many times the command will be executed in sequences. Example:
  - 0|0 the most used: no delay and no repeat (the command is executed into one tick)
  - 2|0 Command has a delay of 2 ticks after his execution (and no repeat, so next command will start after 3 ticks)
  - 2|3 Command has a delay of 2 ticks after his execution, but then it is repeated for 3 times. So the next command starts after (2+1)*(3+1)=12 completed ticks.
- **AD Attack/Decay**
  Here you can specify the attack/decay to use for the instrument. Values goes from $00 to $FF and are to be insert in hex.
- **SR Sustain/Release**
  Here you can specify the sustain/release to use for the instrument. Values goes from $00 to $FF and are to be insert in hex.
- **Wave**
  Here you can specify the wave (control register) to use for the instrument. Values goes from $00 to $FF and are to be insert in hex.
- **Frequency**
  Control the frequency (notes) to put into sid registers. The right-click menu contains four options:
  - **Insert absolute note**
    The table that is opened contains all the 8-octave notes you can use. Them are hard-coded into the player so you cannot change the frequency for one note, with the exception that you can choose from 4 kind of tables in Option that differs only by the middle A4 note: 424, 434, 440, 442Hz.
  - **Insert relative note**
    In the table that is opened there are relative values to add or subtract from the

note the instruments was activated by the pattern that is using the instrument. It is always related to notes in the pattern, for example if you use an absolute note of the case above, this is not the note used for adding or subtracting the value. Those values are hard-coded into the player and cannot be varied. They varied from 0 to 31 so about +/- 2,5 octave from the note of the pattern. The value of Note+0 is to use if you want to maintain the actual note played by the pattern.

- **Insert relative frequency**
  In the table that is opened you can choose relative values of frequencies to add or subtract from current frequencies the sid was used. Value of +/- 0 can be used for taking the actual frequency the sid is working. This command can be used for creating vibrato or portamento at the instrument level. The values to add/subtract go from 0 to 32768 and you can change them using the *Tables view icon/menu command*.
- **Insert absolute frequency**
  In the table that is opened you can choose fixed values of frequencies to put directly into sid register. The 0 value is useful if you want to play silent. The values go from 0 to 32768 and you can change them using the *Tables view icon/menu command*.

- **Pulse**
  Control the pulse generation for $41 waveform.
  The right-click menu contains two options:
  - **Insert fixed pulse value**
    With this it is open a sub-table where you can choose the fixed values (from 0..2048 that you can change using the *Tables view icon/menu command*) that are to be putted into pulse registers of sid.
  - **Insert relative pulse value**
    With this it is open a sub-table where you can choose the relative (positive or negative) value to add/subtract from current value of the pulse registers. Values can be varied using the *Tables view icon/menu command*
- **Filter Cut-off**
  Set the filter cut-off frequency.
  The right-click menu contains two options:
  - **Insert relative frequency** With this it is open a sub-table where you can choose the relative cut-off frequency to add/subtract from the current value. You can change the values using the *Tables view icon/menu command*.
  - **Insert absolute frequency** Whit this it is open a sub-table where you can choose the absolute frequency to put into sid registers. Values can be varied using the *Tables view icon/menu command*
- **Filter Resonance**
  Set the filter resonance value.
  The right-click menu contains one option:
  - **Insert value** With this it is open a sub-table where you can choose:
    - Fixed value of resonance to put to sid register
    - Value to add to current resonance
    - Value to subtract to current resonance

  The values go from 0 to 15 and are hard-coded into the player, so they cannot be changed.
- **Filter type**
  Set the filter type to use and the voices where apply filter.
  The right-click menu contains one option:
  - **Insert type**
    In the dialog that it is opened containing checkboxes, you can choose:
    - High pass filter
    - Band pass filter
    - Low pass filter
    - Filter active in voice 3
    - Filter active in voice 2

- Filter active in voice 1

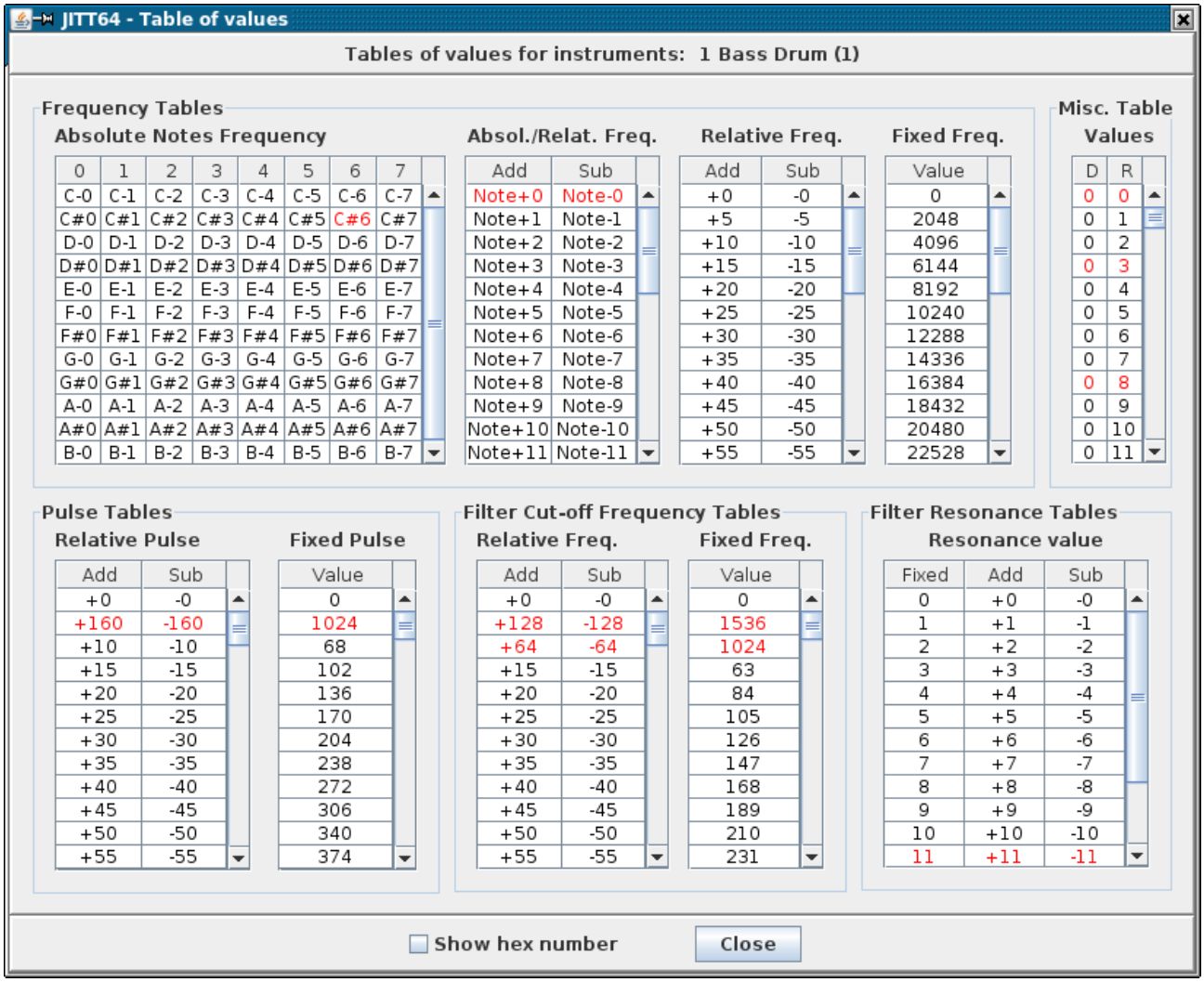
- **Common part**
  At each right-click in a column different by *D/N*, the first two option available are:
  - **Set step to here**
    It takes the position where you right-click the one for repeating the sequence of commands where the last is executed. You will see that all the commands that repeats are now colored.
  - **Set no step**
    It remove the position for repeating sequence, and so when the last command is executed, no one for this column will be performed by the player, until the instrument restart by tracker command.

  If the step is defined, the number of repeat can be chosen by right-clicking the last row of *D/N* column. You have a fixed table with values from 1 to 255 and the *inf* one. With *inf* you specify that the repeat is forever.

Table of Values

For changing the values you see into sub-menu, you have to open the *Table of values for instruments*.

In this big screen are reported all the used tables for one instrument. Red values are the one you have used into the instrument definition, the Blacks are the ones not yet used.



Filling this tables of values are the most tedious work for you (and as these tables are not shared, this operation is to be done for each instrument), but this is what we have to pay for having freedom in implementing one instrument.

The tables are pre-filled with some custom values and in future I will add some configuration wizard that will help you filling it. Else, I will make that you can vary one value when you are into instrument main table, without need to open this screen.

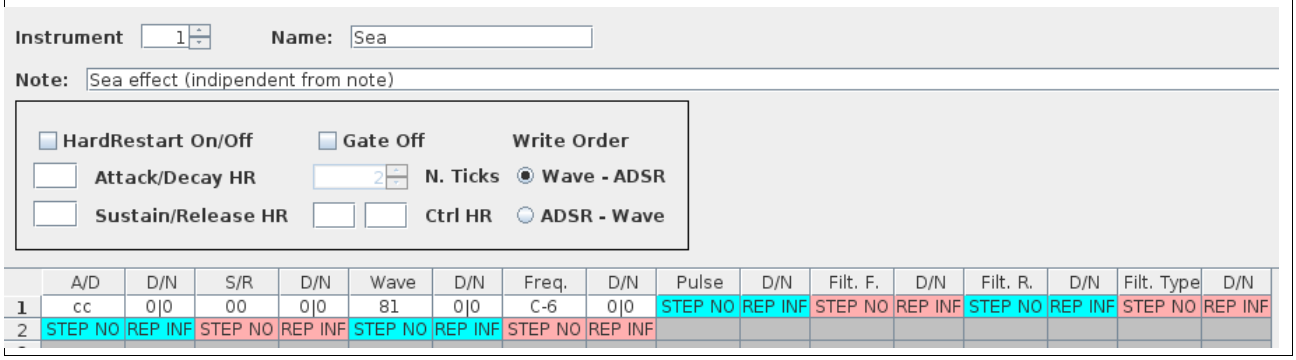
Take present that each of this table can be loaded/saved in/from file, so you can create one table library to use.

Instrument examples

At this point I like to show some example of instrument implementation, so you can figure better how this task look like in JITT64.

SEA effect

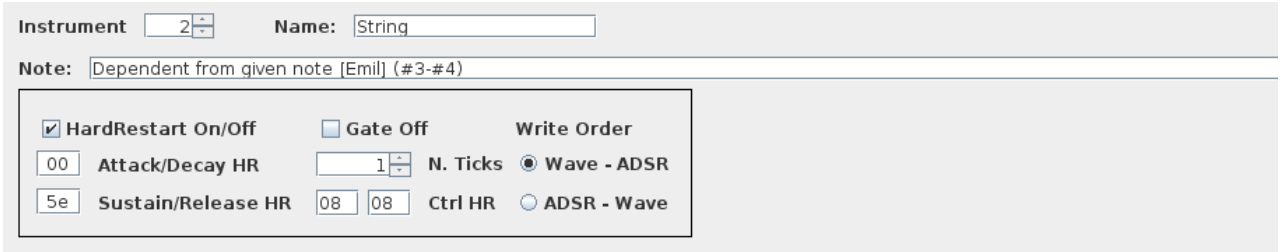
A simple sea effect. It not used any hard-restart of note: only AD/SR and fixed wave/frequency are used. The sea is obtained from the use of noise with very long Attack/Decay without a Sustain level left in volume.



	A/D	D/N	S/R	D/N	Wave	D/N	Freq.	D/N	Pulse	D/N	Filt. F.	D/N	Filt. R.	D/N	Filt. Type	D/N
1	cc	0|0	00	0|0	81	0|0	C-6	0|0	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF
2	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF								

Emil's String

This is an example of a String instrument from Emil music. First of all it use a sort of arpeggio for giving more tones to the voice, else it gives a big “vibrato” to the pulse of rectangular waveform. Even if gate of note is released, the slow release time is compensated by an hardrestart of note with only one tick of delay.



	A/D	D/N	S/R	D/N	Wave	D/N	Freq.	D/N	Pulse	D/N	Filt. F.	D/N	Filt. R.	D/N	Filt. Type	D/N
1	00	0|0	5e	0|0	41	2|0	Note+0	1|0	1024	0|0	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF
2	STEP NO	REP INF	STEP NO	REP INF	40	0|0	Note+5	1|0	+64	0|2						
3					STEP NO	REP INF	Note+8	1|0	+64	0|25						
4							Note+12	1|0	-64	0|25						
5							STEP 1	REP INF	STEP 3	REP INF						

Oriental flute

This is an oriental flute with an high pitch at beginning and a incremental vibrato in the main part. It needs an hard-restart for a best note starting.

Instrument 1 Name: Oriental Flute

Note: Depend from given note (best as for octave 4)

HardRestart On/Off | Gate Off | Write Order

0f Attack/Decay HR | 2 N. Ticks | Wave - ADSR

66 Sustain/Release HR | 10 01 Ctrl HR | ADSR - Wave

	A/D	D/N	S/R	D/N	Wave	D/N	Freq.	D/N	Pulse	D/N	Filt. F.	D/N	Filt. R.	D/N	Filt. Type	D/N
1	0f	0|0	66	0|0	11	0|0	Note+3	0|0	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF
2	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF	Note+2	0|0								
3							Note+1	0|0								
4							Note+0	0|26								
5							+2	0|0								
6							+4	0|0								
7							+6	0|0								
8							+8	0|0								
9							+10	0|0								
10							-12	0|0								
11							-14	0|0								
12							-16	0|0								
13							-18	0|0								
14							-20	0|0								
15							+22	0|0								
16							+24	0|0								
17							+26	0|0								
18							+28	0|0								
19							+30	0|0								
20							-32	0|0								
21							-34	0|0								
22							-36	0|0								
23							-38	0|0								
24							-40	0|0								
25							+42	0|0								
26							+44	0|0								
27							+46	0|0								
28							+48	0|0								
29							+50	0|0								
30							-52	0|0								
31							-54	0|0								
32							-56	0|3								
33							+56	0|5								
34							-56	0|5								
35							STEP 33	REP INF								

Bass Drum

A bass drum with fixed notes and lot use of noise waveform

Instrument 1 Name: Bass Drum (3)

Note: Fixed note, no filter

HardRestart On/Off | Gate Off | Write Order

Attack/Decay HR | 2 N. Ticks | Wave - ADSR

Sustain/Release HR | Ctrl HR | ADSR - Wave

	A/D	D/N	S/R	D/N	Wave	D/N	Freq.	D/N	Pulse	D/N	Filt. F.	D/N	Filt. R.	D/N	Filt. Type	D/N
1	00	0|0	e8	0|0	11	0|0	C-0	0|0	2048	0|0	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF
2	STEP NO	REP INF	STEP NO	REP INF	81	0|0	B-4	0|0	STEP NO	REP INF						
3					41	0|0	G#3	0|0								
4					40	0|0	F#3	0|0								
5					80	4|0	F#5	0|0								
6					10	0|0	D#4	0|0								
7					STEP NO	REP INF	B-4	0|0								
8							B-3	0|0								
9							F#5	0|0								
10							G-3	0|0								
11							STEP NO	REP INF								

Bass drum

Another bass drum that use more low pitch notes and less noise

Instrument 1 Name: Bass Drum (2)

Note: Fixed note, no filter

HardRestart On/Off | Gate Off | Write Order

Attack/Decay HR | N. Ticks 2 | Wave - ADSR

Sustain/Release HR | Ctrl HR | ADSR - Wave

	A/D	D/N	S/R	D/N	Wave	D/N	Freq.	D/N	Pulse	D/N	Filt. F.	D/N	Filt. R.	D/N	Filt. Type	D/N
1	00	0|0	e8	0|0	11	0|0	C-0	0|0	2048	0|0	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF
2	STEP NO	REP INF	STEP NO	REP INF	81	0|0	F#5	0|0	STEP NO	REP INF						
3					41	0|0	D-3	0|0								
4					40	0|12	B-2	0|0								
5					10	0|0	F#2	0|0								
6					STEP NO	REP INF	D-2	0|0								
7							B-1	0|0								
8							F#1	1|0								
9							B-0	1|0								
10							C-0	1|0								
11							B-0	0|0								
12							F#1	0|0								
13							STEP NO	REP INF								

Filtered bass drum

A filtered bass drum that depends from pattern note in all unless the fixed drum effect. There is a “portamento” onto the pulse of rectangular waveform and the filter dynamically changes the cut off frequency.

Instrument 3 Name: Bass Drum (1)

Note: Fot notes of octave #2-3 (use filter in voice 3)

HardRestart On/Off | Gate Off | Write Order

Attack/Decay HR | N. Ticks 2 | Wave - ADSR

Sustain/Release HR | Ctrl HR | ADSR - Wave

	A/D	D/N	S/R	D/N	Wave	D/N	Freq.	D/N	Pulse	D/N	Filt. F.	D/N	Filt. R.	D/N	Filt. Type	D/N
1	00	0|0	ec	0|0	41	0|0	Note+0	0|0	1024	0|0	0	0|0	11	0|0	14	0|0
2	STEP NO	REP INF	STEP NO	REP INF	81	0|0	C#6	0|0	+160	0|0	1536	0|0	STEP NO	REP INF	STEP NO	REP INF
3					40	0|0	Note+0	0|0	STEP 2	REP INF	-128	0|3				
4					STEP NO	REP INF	STEP NO	REP INF			-64	0|8				
5											STEP NO	REP INF				

Matt Gray's bass

A bass from Matt Gray music. There is a vibrato in pulse width.

Instrument 4 Name: Bass

Note: Dependent from given note [Matt Grey] (#1)

HardRestart On/Off | Gate Off | Write Order

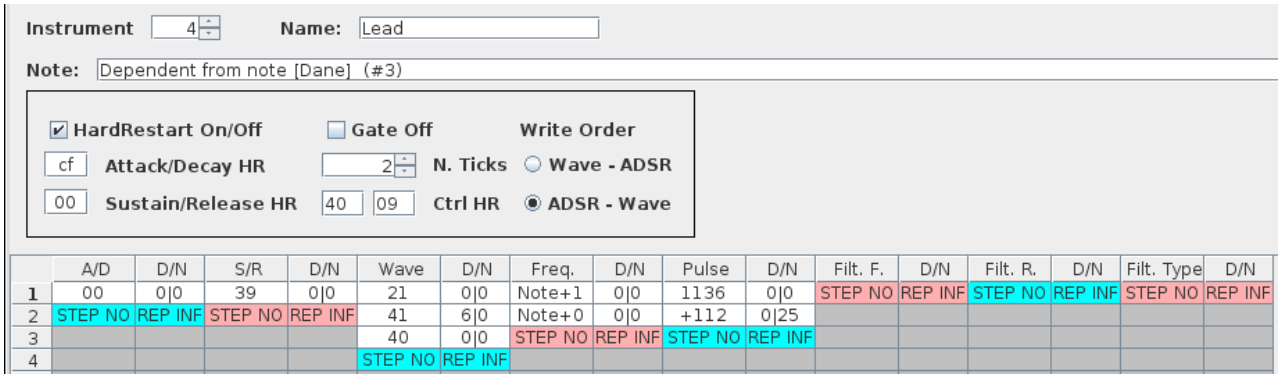
Attack/Decay HR | N. Ticks 2 | Wave - ADSR

Sustain/Release HR | Ctrl HR | ADSR - Wave

	A/D	D/N	S/R	D/N	Wave	D/N	Freq.	D/N	Pulse	D/N	Filt. F.	D/N	Filt. R.	D/N	Filt. Type	D/N
1	00	0|0	cd	0|0	41	0|0	Note+0	0|0	368	0|0	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF
2	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF	+32	0|37						
3									-32	0|34						
4									+32	0|34						
5									STEP 3	REP INF						

Dane's lead

An example of Dane's short lead sound. It uses hardrestart of note and a “portamento” in pulse



Instrument 4 Name: Lead

Note: Dependent from note [Dane] (#3)

☑ HardRestart On/Off ☐ Gate Off Write Order

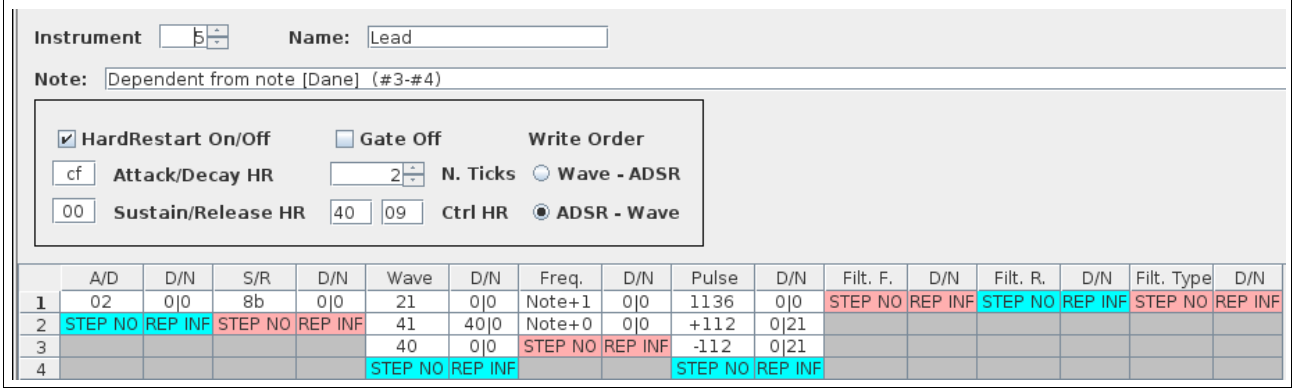
cf Attack/Decay HR 2 N. Ticks ○ Wave - ADSR

00 Sustain/Release HR 40 09 Ctrl HR ◉ ADSR - Wave

	A/D	D/N	S/R	D/N	Wave	D/N	Freq.	D/N	Pulse	D/N	Filt. F.	D/N	Filt. R.	D/N	Filt. Type	D/N
1	00	0|0	39	0|0	21	0|0	Note+1	0|0	1136	0|0	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF
2	STEP NO	REP INF	STEP NO	REP INF	41	6|0	Note+0	0|0	+112	0|25						
3					40	0|0	STEP NO	REP INF	STEP NO	REP INF						
4					STEP NO	REP INF										

Dane's lead

Another example of Dane's short lead sound. It uses hardrestart of note and a “vibrato” in pulse



Instrument 5 Name: Lead

Note: Dependent from note [Dane] (#3-#4)

☑ HardRestart On/Off ☐ Gate Off Write Order

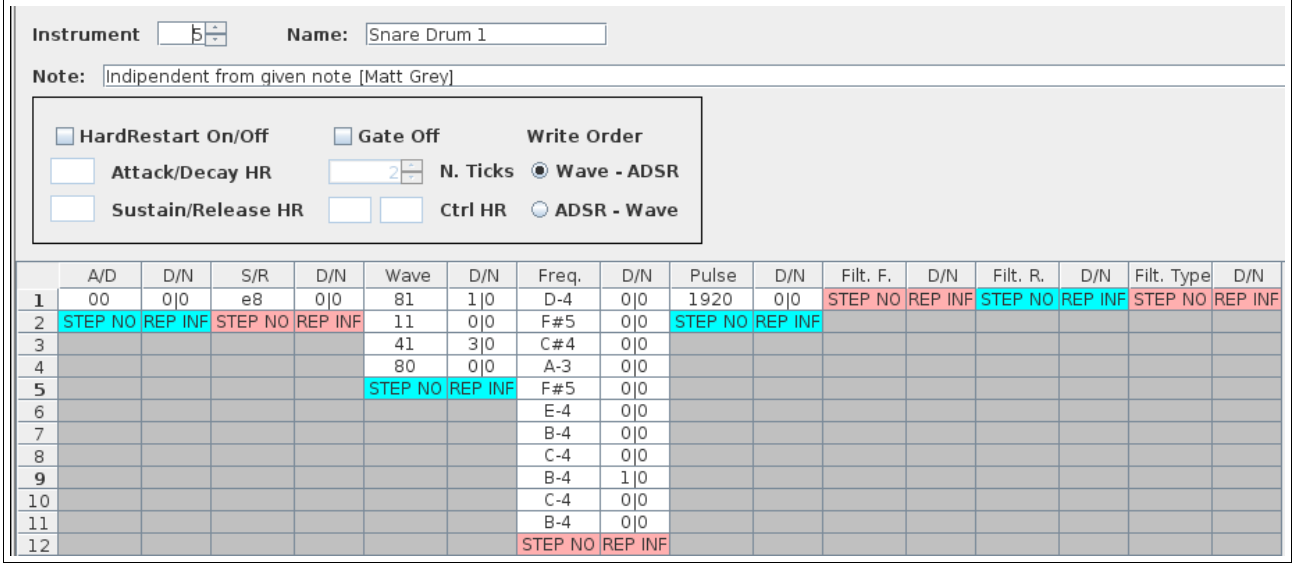
cf Attack/Decay HR 2 N. Ticks ○ Wave - ADSR

00 Sustain/Release HR 40 09 Ctrl HR ◉ ADSR - Wave

	A/D	D/N	S/R	D/N	Wave	D/N	Freq.	D/N	Pulse	D/N	Filt. F.	D/N	Filt. R.	D/N	Filt. Type	D/N
1	02	0|0	8b	0|0	21	0|0	Note+1	0|0	1136	0|0	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF
2	STEP NO	REP INF	STEP NO	REP INF	41	40|0	Note+0	0|0	+112	0|21						
3					40	0|0	STEP NO	REP INF	-112	0|21						
4					STEP NO	REP INF			STEP NO	REP INF						

Matt Gray's snare drum

An example of Matt Gray's snare drum. It uses fixed notes and noise waveform.



Instrument 5 Name: Snare Drum 1

Note: Indipendent from given note [Matt Grey]

☐ HardRestart On/Off ☐ Gate Off Write Order

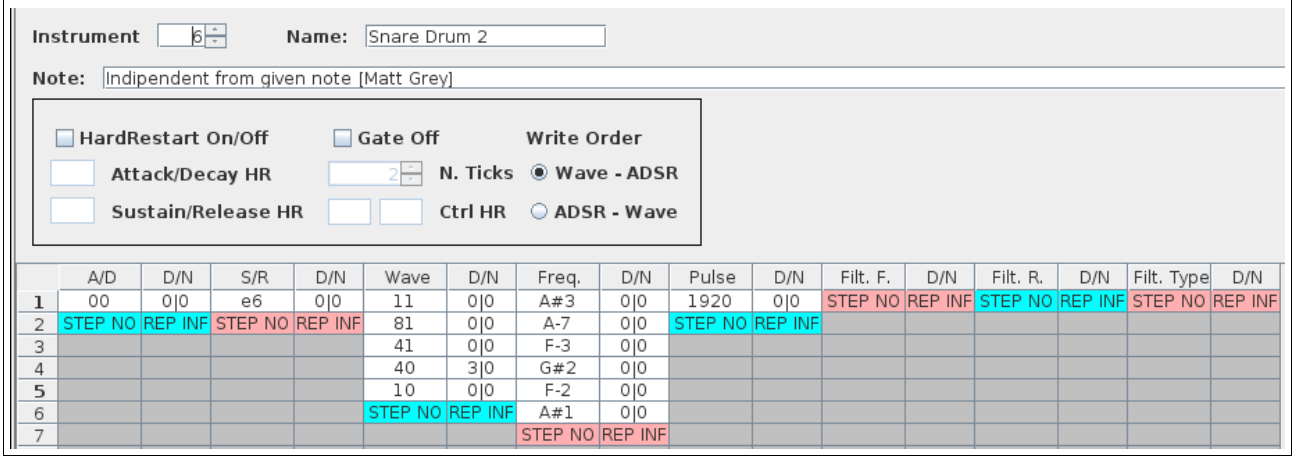
Attack/Decay HR 2 N. Ticks ◉ Wave - ADSR

Sustain/Release HR Ctrl HR ○ ADSR - Wave

	A/D	D/N	S/R	D/N	Wave	D/N	Freq.	D/N	Pulse	D/N	Filt. F.	D/N	Filt. R.	D/N	Filt. Type	D/N
1	00	0|0	e8	0|0	81	1|0	D-4	0|0	1920	0|0	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF
2	STEP NO	REP INF	STEP NO	REP INF	11	0|0	F#5	0|0	STEP NO	REP INF						
3					41	3|0	C#4	0|0								
4					80	0|0	A-3	0|0								
5					STEP NO	REP INF	F#5	0|0								
6							E-4	0|0								
7							B-4	0|0								
8							C-4	0|0								
9							B-4	1|0								
10							C-4	0|0								
11							B-4	0|0								
12							STEP NO	REP INF								

Matt Gray's snare drum

Another example of Matt Gray's snare drum. It uses fixed notes and it is less hard of the previous.



	A/D	D/N	S/R	D/N	Wave	D/N	Freq.	D/N	Pulse	D/N	Filt. F.	D/N	Filt. R.	D/N	Filt. Type	D/N
1	00	0|0	e6	0|0	11	0|0	A#3	0|0	1920	0|0	STEP NO	REP INF	STEP NO	REP INF	STEP NO	REP INF
2	STEP NO	REP INF	STEP NO	REP INF	81	0|0	A-7	0|0	STEP NO	REP INF						
3					41	0|0	F-3	0|0								
4					40	3|0	G#2	0|0								
5					10	0|0	F-2	0|0								
6					STEP NO	REP INF	A#1	0|0								
7							STEP NO	REP INF								

Conclusion

What you read here is just an introduction about the use of JITT64. As JITT64 is a work in progress project, you will find that the actual developed version has more improvements in lot of points.

For example it has a conditional compilation that remove the code that in not used in player, for saving rastrer time. Else it has the support for MIDI music keyboard for simplifying the use of piano roll (you can even insert notes in pattern with your MIDI keyboard). Finally there is a Goattracker2 instrument import for let you use even instrument you make for Goattracker2.

For long term, it is planner do add even a digi track to the player, but now it is too early for discuss about this...

Check the program here: http://sourceforge.net/projects/jitt64/

Inside JITT64

by Stefano Tognon <ice00@libero.it>

With this other article onto JITT64, I want to go inside it and show many of low level details about the player. It is better that you read the previous article to have a background about the tracker, before read this one.

Init &Play (IRQ) routine

Each player has almost the same structure:

- There is a initialization routine that set up the player for playing a given tune.
- There is the play routine that must be called by an IRQ event each frame (or at a given slice of scheduled time) for generating the sound.

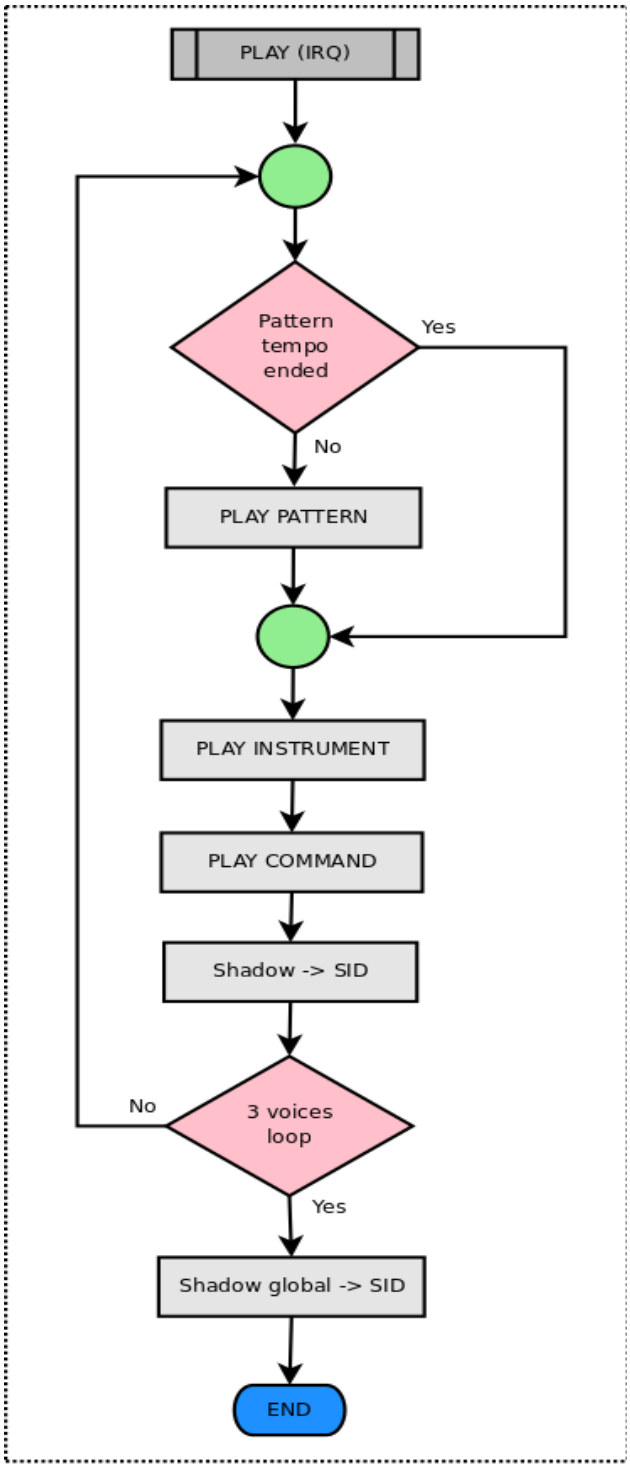
In JITT64 the initialization routines will clear all the used memory locations of the player and starts to read the first pattern of the track of the given tune. The later was a choice for reducing the code that is used inside the IRQ play routine. The initialization even set the volume to maximum level.

The play routine starts with a loop of instructions that are repeated to all the tree voices (from voice 3 to voice 1). The porpoise of that instructions is to generate the sid values that are to be put into the chip register, according to the tracker rules. In the player there are many variables called shadow_xx (where xx is one abbreviation for the sid registers, like FH, FL, and so on) that will store the values that are to be putted into the sid. Only at the end of player calculation those values are putted into the sid chip (look at the Shadow -> SID block) all in one passed, so there is not distortion in sound generation for that voice.

At the end of the main loop, there is the last block of instructions (Shadow global -> SID block) that will put the latest 4 registers of the sid chip that are common to all the 3 voices (like filter and volume).

The instructions that generate the sound for a sid voice can be grouped into:

- Play Pattern: decode and execute one row in pattern
- Play Instrument: play the instrument core
- Play Command: play one command of the pattern



Before analyzing all the blocks in more details there are some words to say about them:

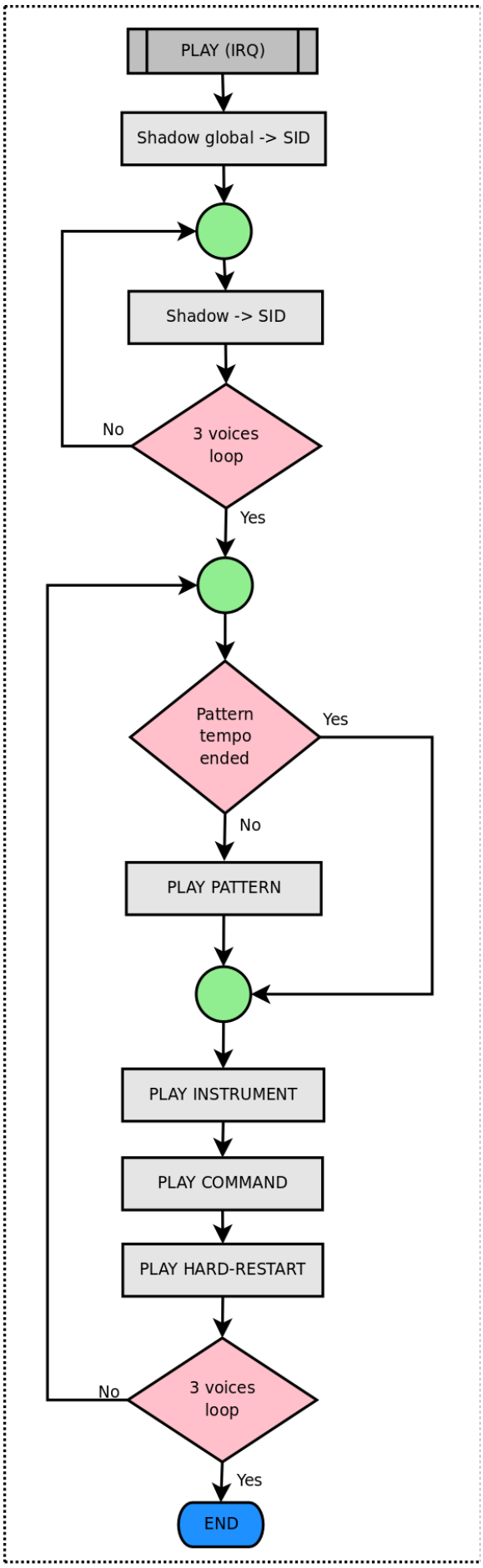
- Play Pattern is executed only when the tempo associated to the pattern is over. So, if you have a tempo of 7 set into a pattern row, the routine is executed only after 7 IRQ calling. In fact, this routine has to decode the notes/instruments/command of one row pattern and activate them.
- Play Instruments and Play Command are executed at each IRQ calling as they have to generate the instrument timbre and the commands execution flow of the music.
- It is very important that Play Pattern is executed before the others blocks, otherwise there will be a delay of one frame when a new note/instrument/command is to be activated, and this will cause some sound problems (as some part of the player continues with the old setting, and some part will start with the new one).

Even if the proposed structure of IRQ seems very linear, it suffers of some timing problems and it was changed to a similar form that is showed in the next diagram. The problem with the initial used engines is that the 3 voices are not outputted at the same time as the Play Pattern, Play Instruments, and Play Command will take some variables time in execution and so the 3 voices will starts play with many cycles of delay (and this can cause some hard-restart of note problem too). In fact, one of the missing block in first diagram is Play Hard-Restart that are to be executed after all the other blocks, as if an hard-restart of note condition is reached, it had to take control on sid output over the others commands.

The new structures is formed by:

- Shadow register are outputted the same time at the start of irq routine: this granted that at every frames the sid registers are updated all together
- The execution of player behaviors is after having updating the sid registers, so there is no problem in how many cycles this will take

The only side effect of this new structures is that sound output is always one frame later his calculation. This could not be a right thinks if you are coding a demo where sound is to be synchronized with graphics, but JITT64 is born for write only music, so this is not a problem.



Play Pattern

Play Pattern (pl_play_pattern) has to decode the next pattern value of note/instrument/command/parameter so it is divided into 4 parts. In order to follow it, we need to know that some variables store the action to give (at a bit level):

BIT	*actCommand2*	*activeCommand*	*actCommand3*
1	AD	Arpeggio	Gate SR
2	SR	Port. Up	
3	Key ON	Port. Dn	
4	Key OFF	Tone-Port.	
5	REST	Vibrato	
6	Filter Type	Slide Up	
7	Filter Resonance	Slide Dn	
8	Filter Cut Off		

Play Pattern is so divided into this 4 parts:

1. decode note (or command for note)
2. decode instrument to use
3. decode command to perform
4. decode parameters to use for command

The first block is code as this:

```
;================================
; play the pattern
;================================
pl_play_pattern:
      lda  #0
      sta  actCommand2,x          ; clear actual command 2

      lda  pattTempo,x            ; restore tempo with saved one
      sta  pattDelay,x

      inc  pattIndex,x            ; increment pattern index
      ldy  pattIndex,x            ; read index

      TBLR pattPoint1             ; read the value of note

      cmp  #PAT_END               ; end of pattern reached ?
      beq  nextPattInTrack

      cmp  #PAT_NULL              ; null value ?
      beq  executeB2

      cmp  #PAT_KON               ; key on ?
      bne  testKOff

      lda  #4
      sta  actCommand2,x          ; put key on command
      jmp  executeB2              ; execute byte 2

testKOff:
      cmp  #PAT_KOFF              ; key off ?
      bne  testRest

      lda  #8
      sta  actCommand2,x          ; put key off command
      jmp  executeB2              ; execute byte 2

testRest:
      cmp  #PAT_REST
      bne  isNote

      lda  #$10
```

```
      sta  actCommand2,x          ; put rest command
      jmp  executeB2              ; execute byte 2

isNote:
      clc
      adc  trackTransp,x          ; add the transpose for this pattern
      sta  actNote,x              ; store the value in actual note
      tay
      lda  frequencyLo,y
      sta  shadow_FL,x

      lda  frequencyHi,y
      sta  shadow_FH,x

      lda  #0
      sta  activeCommand,x        ; stop all pattern command
      jmp  executeB2              ; execute byte 2

exitPP:
      rts

nextPattInTrack:
      jsr  pl_read_track

      lda  stopTrack,x            ; if no more track, exit
      bne  exitPP

      jmp  pl_play_pattern
```

You can so see that it is test for:

1. Pattern end
2. Pattern null
3. Key on
4. Key off
5. Rest
6. Note

and then take the appropriate action.

The second block is very tiny:

```
;================================
; execute byte 2 (instruction)
;================================
executeB2:
      ldy  pattIndex,x            ; read index (again)
      TBLR pattPoint2             ; read the value of instrument
      beq  executeB3              ; skip if zero

      jsr  pl_new_instr           ; set up a new instrument
```

It just initialize instrument, if instrument is used.
In block 3, commands are decoded and if needed parameters are decoded too.

```
;================================
; execute B3 (command)
;================================
executeB3:
      ldy  pattIndex,x            ; read index (again)
      TBLR pattPoint3             ; read the value of command
      sta  actCommand,x

      ;ldy  pattIndex,x            ; read index (again)
      TBLR pattPoint4             ; read the value of param
      sta  actParam,x

                                  ; check for all possible commands and initialize them
      ldy  actCommand,x           ; read command in y
      cpy  #CMD_STOP              ; test for stop command
      bne  chk_next0

      cmp  #0
      beq  exitPP

      eor  #$FF
      and  activeCommand,x
      sta  activeCommand,x
      rts
```

```
;=======
; Set Tempo
;=======
chk_next0:
      cpy  #CMD_TEMPO            ; test for set tempo
      bne  chk_next1

      cmp  #0
      beq  setNT

      sta  pattTempo,x           ; store new pattern tempo to use
      sta  pattDelay,x
      rts

setNT:
      ldy  #0
      TBLR pattPoint1            ; read pattern value
      sta  pattTempo,x           ; set the pattern tempo for the pattern
      sta  pattDelay,x
      rts

;=======
; Set AD
;=======

chk_next1:
      cpy  #CMD_AD               ; test for set atack/decay
      bne  chk_next2

      sta  cmdADSR,x             ; store in command AD reg.

      lda  #1                    ; set for a AD in next command play routine
      sta  actCommand2,x
      rts

;=======
; Set SR
;=======

chk_next2:
      cpy  #CMD_SR               ; test for sustain/release
      bne  chk_next3

      sta  cmdADSR,x             ; store in command SR reg.

      lda  #2                    ; set for a SR in next command play routine
      sta  actCommand2,x
      rts

;========
; Set Volume
;========

chk_next3:
      cpy #CMD_VOL               ; test for volume
      bne chk_next4

      sta  shadow_VOL            ; store in shadow volume reg.
      rts

;========
; Set Arpeggio
;========

chk_next4:
      cpy  #CMD_ARP              ; test for arpeggio
      bne  chk_next5

      sta  actParam,x

      rol                        ; isolate the speed
      rol
      rol
      and  #$03
      sta  speedArpeggio,x
      sta  speedRelArp,x

      lda  actParam,x
      lsr
      lsr
      lsr
      and  #$07
      clc
      adc  actNote,x
      sta  note1Arpeggio,x       ; store first arpeggio note

      lda  actParam,x
      and  #$07
```

```
      clc
      adc  note1Arpeggio,x
      sta  note2Arpeggio,x        ; store second arpeggio note

      lda  #$01
      ora  activeCommand,x        ; activate arpeggio effect
      sta  activeCommand,x

      lda  #0
      sta  posArpeggio,x
      rts

;========
; Set Port. up
;========

chk_next5:
      cpy  #CMD_PUP               ; test for portamento up
      bne  chk_next6

      sta  freqPortUp,x           ; store freq. for portamento up

      lda  #$02
      ora  activeCommand,x        ; activate portamento up
      sta  activeCommand,x
      rts

;========
; Set Port. dn
;========

chk_next6:
      cpy  #CMD_PDN               ; test for portamento dn
      bne  chk_next7

      sta  freqPortDn,x           ; store freq. for portamento dn

      lda  #$04
      ora  activeCommand,x        ; activate portamento dn
      sta  activeCommand,x
      rts

;========
; Set Tone Port.
;========

chk_next7:
      cpy  #CMD_TPO               ; test for tone portamento
      bne  chk_next8

      sta  freqTonePort,x         ; store freq. for tone portamento

      lda  #$08
      ora  activeCommand,x        ; activate toneportamento
      sta  activeCommand,x

      lda  #00                    ; suppose to have a up direction
      sta  toneDir,x

      lda  shadow_FH,x            ; store tone frequency to reach
      sta  tone_FH,x
      lda  shadow_FL,x
      sta  tone_FL,x

      lda  copy_FL,x              ; store the copy as the freq. to start from
      sta  shadow_FL,x
      lda  copy_FH,x
      sta  shadow_FH,x

      cmp  tone_FH,x              ; test the high freq. of toneport. for dir.
      beq  testLow
      bcc  chk_next8
      inc  toneDir,x              ; invert direction
      bne  chk_next8

testLow:
      lda  copy_FL,x              ; test the low freq. of toneport. for dir.
      cmp  tone_FL,x
      bcc  chk_next8
      inc  toneDir,x              ; inver direction

;========
; Set Vibrato
;========

chk_next8:
      cpy  #CMD_VIB               ; test for vibrato
      bne  chk_next9
```

```
      pha
      and  #$0F
      sta  vibSpeed,x             ; store vibrato speed
      pla
      and  #$F0
      clc
      adc  #$10
      sta  vibFreq,x              ; store vibrato frequency
      inc  vibSpeed,x             ; increment speed (0 is end mark)

      lda  vibSpeed,x
      sta  vibSpeed2,x            ; load second speed
      lda  #00
      sta  vibSpeed1,x            ; reset first speed

      lda  #$10
      ora  activeCommand,x        ; activate vibrato
      sta  activeCommand,x
      rts

;========
; Set Slide Up
;========

chk_next9:
      cpy  #CMD_SUP               ; test for slide up
      bne  chk_next10

      sta  slideUp,x              ; store slide up value

      lda  #$20
      ora  activeCommand,x        ; activate slide up
      sta  activeCommand,x
      rts

;========
; Set Slide Down
;========

chk_next10:
      cpy  #CMD_SDN               ; test for slide down
      bne  chk_next11

      sta  slideDn,x              ; store slide down value

      lda  #$40
      ora  activeCommand,x        ; activate slide down
      sta  activeCommand,x
      rts

;========
; Set Auto Fade out
;========

chk_next11:
      cpy  #CMD_AFO               ; test for auto fade out
      bne  chk_next12

      sta  autoFadeOut,x          ; store auto fade out value
      sta  actualFadeOut,x        ; store auto fade out value
      rts

;========
; Set Filter type
;========

chk_next12:
      cpy  #CMD_FTY               ; test for filter type
      bne  chk_next13

      sta  cmdFilter,x            ; store filter params
      lda  #$20
      sta  actCommand2,x
      rts

;========
; Set Filter resonance
;========

chk_next13:
      cpy  #CMD_FRE               ; test for filter res.
      bne  chk_next14

      sta  cmdFilter,x            ; store filter resonance
      lda  #$40
      sta  actCommand2,x
      rts
```

```
;========
; Set Filter cut off
;========

chk_next14:
      cpy  #CMD_FCU              ; test for filter cut off
      bne  chk_next15

      sta  cmdFilter,x           ; store filter resonance
      lda  #$80
      sta  actCommand2,x
      rts

;=======
; Set Gate SR
;=======

chk_next15:
      cpy  #CMD_GSR              ; test for gate sustain/release
      bne  exitExecute

      sta  cmdADSR,x             ; store in command SR reg.

      lda  #1                    ; set for a Gate SR in next command play routine
      sta  actCommand3,x

exitExecute:
      rts
```

Instruments definition

As JITT64 can handle up to 255 instruments, the declaration of it into the code is actuating by using macro code:

```
; Instrument definition (filled according to actual number of instruments)

 .mac ins_instr
 .if (NUM_INSTR> ({1}-1))
instr{1}:
  .byte INSTR_HR_{1}
  .byte INSTR_AD_{1}
  .byte INSTR_SR_{1}
  .byte INSTR_CTRL1_{1}
  .byte INSTR_CTRL2_{1}
  .byte <instrAD_{1}         , >instrAD_{1}
  .byte <instrSR_{1}         , >instrSR_{1}
  .byte <instrWave_{1}       , >instrWave_{1}
  .byte <instrFreq_{1}       , >instrFreq_{1}
  .byte <instrPulse_{1}      , >instrPulse_{1}
  .byte <instrFilter_{1}     , >instrFilter_{1}
  .byte <instrRes_{1}        , >instrRes_{1}
  .byte <instrType_{1}       , >instrType_{1}
  .byte <instrFixFreq_{1}    , >instrFixFreq_{1}
  .byte <instrRelFreq_{1}    , >instrRelFreq_{1}
  .byte <instrFixPulse_{1}   , >instrFixPulse_{1}
  .byte <instrRelPulse_{1}   , >instrRelPulse_{1}
  .byte <instrRelFilter_{1}  , >instrRelFilter_{1}
  .byte <instrFixFilter_{1}  , >instrFixFilter_{1}
  .byte <instrDelay_{1}      , >instrDelay_{1}
  .byte <instrRepeat_{1}     , >instrRepeat_{1}
 .endif
 .endm
```

The macro needs the number of instrument to define as a parameter and will declare all the pointers to the used table of values. It is the packer that will add the tables of instruments at run-time when you pack the tune. Only the max number of instrument used (NUM_INSTR) is passed by the packer, so there is conditional code that test it.

When a new instrument is used, those values are copied to many variables:

```
instrPtrAD_L:    .byte $00  ; (low)  pointer to instrument table AD
instrPtrAD_H:    .byte $00  ; (high) pointer to instrument table AD
```

```
instrPtrSR_L:   .byte $00  ; (low)  pointer to instrument table SR
instrPtrSR_H:   .byte $00  ; (high) pointer to instrument table SR
instrPtrWave_L: .byte $00  ; (low)  pointer to instrument table Wave
instrPtrWave_H:   .byte $00  ; (high) pointer to instrument table Wave
instrPtrFreq_L:   .byte $00  ; (low)  pointer to instrument table Freq
instrPtrFreq_H:   .byte $00  ; (high) pointer to instrument table Freq
instrPtrPulse_L:  .byte $00  ; (low)  pointer to instrument table Pulse
instrPtrPulse_H:  .byte $00  ; (high) pointer to instrument table Pulse
instrPtrFilter_L: .byte $00  ; (low)  pointer to instrument table Filter
instrPtrFilter_H: .byte $00  ; (high) pointer to instrument table Filter
instrPtrRes_L:     .byte $00  ; (low)  pointer to instrument table Res
instrPtrRes_H:     .byte $00  ; (high) pointer to instrument table Res
instrPtrType_L:    .byte $00  ; (low)  pointer to instrument table Type
instrPtrType_H:    .byte $00  ; (high) pointer to instrument table Type
instrPtrFixFreq_L: .byte $00  ; (low)  pointer to instrument table Fix Freq
instrPtrFixFreq_H: .byte $00  ; (high) pointer to instrument table Fix Freq
instrPtrRelFreq_L: .byte $00  ; (low)  pointer to instrument table Rel Freq
instrPtrRelFreq_H:   .byte $00  ; (high) pointer to instrument table Rel Freq
instrPtrFixPulse_L:  .byte $00  ; (low)  pointer to instrument table Fix Pulse
instrPtrFixPulse_H:  .byte $00  ; (high) pointer to instrument table Fix Pulse
instrPtrRelPulse_L:  .byte $00  ; (low)  pointer to instrument table Rel Pulse
instrPtrRelPulse_H:  .byte $00  ; (high) pointer to instrument table Rel Pulse
instrPtrRelFilter_L: .byte $00  ; (low)  pointer to instrument table Rel Filter
instrPtrRelFilter_H: .byte $00  ; (high) pointer to instrument table Rel Filter
instrPtrFixFilter_L: .byte $00  ; (low)  pointer to instrument table Fix Filter
instrPtrFixFilter_H: .byte $00  ; (high) pointer to instrument table Fix Filter
instrPtrDelay_L:     .byte $00  ; (low)  pointer to instrument table Delay
instrPtrDelay_H:     .byte $00  ; (high) pointer to instrument table Delay
instrPtrRepeat_L:    .byte $00  ; (low)  pointer to instrument table Repeat
instrPtrRepeat_H:    .byte $00  ; (high) pointer to instrument table Repeat
```

Having this structured of pointers for instrument tables, a new macro is used when it is needed to read one value from the table:

```
;================================
; Read the value of passed table
; pointer of voice x with the y
; index
;================================
 .MAC TBLR
      lda  {1}_L,x
      sta  ADDR_LOW
      lda  {1}_H,x
      sta  ADDR_HIGH
      lda  (ADDR_LOW),y
 .ENDM
```

So, for example:

```
TBLR instrPtrPulse
```

it will read one byte from the instrument pulse table at position given by of the y register of the voice given by x register.

Play Instrument

The main block about play instrument in the diagram is mapped to one routine in the player code: pl_intr_core

```
;================================
; Player instrument core: called
; at each ticks. It executes all
; the tables core
;================================
pl_intr_core:
      jsr  pl_instr_core_AD
      jsr  pl_instr_core_SR
      jsr  pl_instr_core_Wave
```

```
      jsr  pl_instr_core_Freq
      jsr  pl_instr_core_Pulse
      jsr  pl_instr_core_Filter
      jsr  pl_instr_core_Res
      jsr  pl_instr_core_Type
      rts
```

Essentially it calls the execution of all the tables that compose an instrument: AD, SR, Wave, Frequency, Pulse, Filter, Resonance and Type.

```
;==================================
; Attack/Decay instruction core
;==================================
pl_instr_core_AD:
      pl_instr_core_ AD, 1

;==================================
; Sustain/Release instruction core
;==================================
pl_instr_core_SR:
      pl_instr_core_ SR, 2

;==================================
; Wave instruction core
;==================================
pl_instr_core_Wave:
      pl_instr_core_ Wave, 3

;==================================
; Freq instruction core
;==================================
pl_instr_core_Freq:
      pl_instr_core_ Freq, 4

;==================================
; Pulse instruction core
;==================================
pl_instr_core_Pulse:
      pl_instr_core_ Pulse, 5

;==================================
; Filter instruction core
;==================================
pl_instr_core_Filter:
      pl_instr_core_ Filter, 6

;==================================
; Resonance instruction core
;==================================
pl_instr_core_Res:
      pl_instr_core_ Res, 7

;==================================
; Type instruction core
;==================================
pl_instr_core_Type:
      pl_instr_core_ Type, 8
```

All this is done by a big macro (pl_instr_core_) that receive two parameters: the kind of table to use and a number for let implementing conditional code based onto table. Inside it, another macro (OUTV) is used for output the value according to the actual table:

```
;==================================
; Macro for instruction core
;==================================
 .mac pl_instr_core_
      ldy  actIndex_{1},x         ; is first time with the instrument?
      bne  cont_IC_{1}

      TBLR instrPtr{1}            ; read values
      sta  dimension_{1},x        ; save the dimension of table

      iny
```

```
      lda  (ADDR_LOW),y          ; read value
      sta  allowRepeat_{1},x     ; save repeat information
      iny
      lda  (ADDR_LOW),y          ; read value
      sta  stepPos_{1},x         ; save step position

      iny
      tya
      sta  actIndex_{1},x        ; store actual index

      lda  dimension_{1},x       ; if table is empty, exit
      beq  exit_IC_{1}

      lda  (ADDR_LOW),y          ; read values
start_IC_{1}:
      OUTV {1}, {2}              ; out value

ecount_IC_{1}:
      iny
      lda  (ADDR_LOW),y          ; read delay repeat from table
      tay
      TBLR instrPtrDelay         ; read delay from table
      sta  actDelay_{1},x        ; store in actual delay
      TBLR instrPtrRepeat        ; read repeat from table
      sta  actRepeat_{1},x       ; store actual repeat

exit_IC_{1}:
      rts                        ; exit

cont_IC_{1}:
      lda  dimension_{1},x       ; if table is empty, exit
      beq  exit_IC_{1}

      lda  actDelay_{1},x        ; read actual delay
      beq  cont1_IC_{1}          ; if zero go away with repeat

      dec  actDelay_{1},x        ; decrement actual delay
      rts                        ; exit

cont1_IC_{1}:
      lda  actRepeat_{1},x       ; read actual repeat value
      beq  cont2_IC_{1}          ; if zero go away with next index in table

      TBLR instrPtr{1}           ; read values
      OUTV {1}, {2}              ; out value

      iny
      lda  (ADDR_LOW),y          ; read delay repeat from table
      tay
      TBLR instrPtrDelay         ; read delay from table
      sta  actDelay_{1},x        ; store in actual delay
      dec  actRepeat_{1},x       ; decrement actual repeat
      rts                        ; exit

cont2_IC_{1}:
      lda  actIndex_{1},x        ; test if actual position is over dimension
      cmp  dimension_{1},x
      bcs  cont3_IC_{1}

      inc  actIndex_{1},x        ; increment actual index
      inc  actIndex_{1},x        ; increment actual index
      ldy  actIndex_{1},x
      TBLR instrPtr{1}           ; read values
      OUTV {1}, {2}              ; out value
      jmp  ecount_IC_{1}

cont3_IC_{1}:
      lda  stepPos_{1},x         ; test if there is no repeat
      beq  exit2_IC_{1}

      ldy  allowRepeat_{1},x     ; test if infinite repeat
      bne  cont4_IC_{1}

econt2_IC_{1}:
      sta  actIndex_{1},x        ; store stepPos in actual position
      tay
      TBLR instrPtr{1}           ; read values
      jmp  start_IC_{1}          ; and restart the core
```

```
cont4_IC_{1}:
      lda  actNumRepeat_{1},x     ; test if actual repeat is equal to max nun of repeat
      cmp  allowRepeat_{1},x
      beq  exit2_IC_{1}

      inc  actNumRepeat_{1},x     ; increment actual number of repeat
      lda  stepPos_{1},x          ; and use a new step position
      bne  econt2_IC_{1}

exit2_IC_{1}:
      rts
 .endm


;==================================
; Out the value to shadow reg.
; according to the actual type of
; action (A=read value from table)
;==================================
 .mac OUTV
par SET {2}
 .if par=1                        ; AD
      sta  shadow_AD,x            ; out value
 .endif
 .if par=2                        ; SR
      sta  shadow_SR,x            ; out value
 .endif
 .if par=3                        ; Wave
      beq  .skip_CTRL             ; skip if zero
      sta  shadow_CTRL,x
.skip_CTRL:
 .endif
 .if par=4                        ; Freq
      sty  TEMP                   ; preserve y index reg
      jsr  putFreq
      ldy  TEMP
 .endif
 .if par=5                        ; Pulse
      sty  TEMP                   ; preserve y index reg
      jsr  putPulse
      ldy  TEMP
 .endif
 .if par=6                        ; Filter Freq
      sty  TEMP                   ; preserve y index reg
      jsr  putFilterFreq
      ldy  TEMP
 .endif
 .if par=7                        ; Filter Resonance
      sty  TEMP                   ; preserve y index reg
      jsr  putFilterRes
      ldy  TEMP
 .endif
 .if par=8                        ; Filter Type
      sta  shadow_TYPE,x          ; out value
 .endif
 .endm
```

Play Command

Play Command routine is a conditional big test that, if command bits are set, will perform that command.

```
;==================================
; Play the command of pattern
;==================================
pl_play_command:

;========
; Arpeggio
;========
      lda  activeCommand,x
      and  #$01                   ; arpeggio command ?
      beq  skip_com1
```

```
      lda  posArpeggio,x          ; put the right note according to position
      cmp  #1
      bne  skipPos1

      ldy  note1Arpeggio,x        ; use middle note
      jmp  putFr

skipPos1:
      bcs  skipPos0

      ldy  actNote,x             ; use main note
      jmp  putFr

skipPos0:
      ldy  note2Arpeggio,x       ; use last note

putFr:                           ; put frequency
      lda  frequencyLo,y
      sta  shadow_FL,x

      lda  frequencyHi,y
      sta  shadow_FH,x

      dec  speedArpeggio,x
      bpl  skipInc

      lda  speedRelArp,x          ; reload speed
      sta  speedArpeggio,x

      inc  posArpeggio,x          ; go to next position
      lda  posArpeggio,x
      cmp  #3                     ; over the limit ?
      bne  skipInc

      lda  #0                     ; yes, restore to thestart
      sta  posArpeggio,x

skipInc:

;========
; Do portamento up
;========

skip_com1:
      lda  activeCommand,x
      and  #$02                   ; portamento up command ?
      beq  skip_com2

      ldy  #PORT_MULT             ; value is *PORT_MULT
loopSubPortUp:

      lda  shadow_FL,x            ; read pulse low
      clc
      adc  freqPortUp,x           ; add port. up value
      sta  shadow_FL,x
      bcc  skip_com2
      inc  shadow_FH,x            ; fix to high

      dey
      bne  loopSubPortUp

;========
; Do portamento dn
;========

skip_com2:
      lda  activeCommand,x
      and  #$04                   ; portamento dn command ?
      beq  skip_com3

      ldy  #PORT_MULT             ; value is *PORT_MULT
loopSubPortDn:

      lda  shadow_FL,x            ; read pulse low
      sec
      sbc  freqPortDn,x           ; sub prot. dn value
      sta  shadow_FL,x
      bcs  skip_com3
```

```
      dec  shadow_FH,x            ; fix to high

      dey
      bne  loopSubPortDn

;========
; tone-portamento
;========

skip_com3:
      lda  activeCommand,x
      and  #$08                   ; tone-portamento command ?
      beq  skip_com4

      lda  toneDir,x              ; read portamento direction
      bne  negPort

      ldy  #PORT_MULT             ; value is *PORT_MULT
loopAddPort:
      lda  shadow_FL,x            ; read actual low freq.
      clc
      adc  freqTonePort,x         ; add the tone port freq. value
      sta  shadow_FL,x
      bcc  skipTPU
      inc  shadow_FH,x            ; fix high of frequency
skipTPU:
      dey
      bne  loopAddPort

      lda  shadow_FH,x            ; test if high freq is reached
      cmp  tone_FH,x
      beq  testLUP                ; is to test low ?
      bcc  skip_com4
      bcs  stopTone

testLUP:
      lda  shadow_FL,x            ; test if low freq is reached
      cmp  tone_FL,x
      bcc  skip_com4

stopTone:
      lda  tone_FH,x              ; copy final value to freq.
      sta  shadow_FH,x
      lda  tone_FL,x
      sta  shadow_FL,x

      lda  activeCommand,x        ; stop tone portamento command
      and  #$F7
      sta  activeCommand,x
      jmp  skip_com4

negPort:
      ldy  #PORT_MULT             ; value is *PORT_MULT
loopSubPort:
      lda  shadow_FL,x            ; read actual low freq.
      sec
      sbc  freqTonePort,x         ; sub the tone port freq. value
      sta  shadow_FL,x
      bcs  skipTPD
      dec  shadow_FH,x            ; fix high of frequency
skipTPD:
      dey
      bne  loopSubPort

      lda  shadow_FH,x            ; test if high freq is reached
      cmp  tone_FH,x
      beq  testLDN                ; is to test low ?
      bcs  skip_com4
      bcc  stopTone

testLDN:
      lda  shadow_FL,x            ; test if low freq is reached
      cmp  tone_FL,x
      bcs  skip_com4
      bcc  stopTone

;========
; Do vibrato
```

```
;========

skip_com4:
      lda  activeCommand,x
      and  #$10                  ; vibrato command ?
      beq  skip_com5

      lda  vibSpeed1,x           ; test first speed value
      beq  testSp2

      dec  vibSpeed1,x           ; decrement speed counter
      jmp  goDir

testSp2:
      lda  vibSpeed2,x           ; test second speed value
      beq  invDir

      dec  vibSpeed2,x           ; decrement speed counter
      jmp  goDir

invDir:
      lda  vibSpeed,x            ; reload speed
      sta  vibSpeed1,x
      sta  vibSpeed2,x
      lda  vibDir,x              ; invert direction
      eor  #$01
      sta  vibDir,x

goDir:
      lda  vibDir,x              ; load vibrato direction
      bne  invertDir

      lda  shadow_FL,x           ; read actual low freq.
      clc
      adc  vibFreq,x             ; add vibrato value
      sta  shadow_FL,x
      bcc  skipVU
      inc  shadow_FH,x           ; fix for high freq.
skipVU:

      jmp  skip_com5

invertDir:
      lda  shadow_FL,x           ; read actual low freq.
      sec
      sbc  vibFreq,x             ; sub vibrato value
      sta  shadow_FL,x
      bcs  skipVD
      dec  shadow_FH,x           ; fix for high freq.
skipVD:

;========
; Do slide up
;========

skip_com5:
      lda  activeCommand,x
      and  #$20                  ; slide up command ?
      beq  skip_com6

      lda  shadow_PL,x           ; read pulse low
      clc
      adc  slideUp,x             ; add slide up value
      sta  shadow_PL,x
      bcc  skip_com6
      inc  shadow_PH,x           ; fix to high

;========
; Do slide down
;========

skip_com6:
      lda  activeCommand,x
      and  #$40                  ; slide dn command ?
      beq  skip_com7

      lda  shadow_PL,x           ; read pulse low
      sec
```

```
      sbc  slideDn,x             ; sub slide dn value
      sta  shadow_PL,x
      bcs  skip_com7
      dec  shadow_PH,x           ; fix to high

;========
; Do fade out
;========

skip_com7:

; do special command
      lda  autoFadeOut,x         ; read auto fade out
      beq  oneShotCmd

      lda  shadow_VOL            ; check actual volume
      beq  exitAFO

      dec  actualFadeOut,x       ; dec actual fade out counter
      bne  oneShotCmd

      dec  shadow_VOL            ; decrement volume
      lda  autoFadeOut,x         ; read auto fade out
      sta  actualFadeOut,x       ; set actual fade out
      bne  oneShotCmd

exitAFO:
      sta  autoFadeOut,x         ; clear auto fade out

; do one shot command
oneShotCmd:

;=======================
; AD command
;=======================

      lda  actCommand2,x
      ;beq  skip_3com1            ; exit if not more commands
      and  #1                    ; AD ?
      beq  skip_2com1

      lda  cmdADSR,x
      sta  shadow_AD,x           ; store AD

      lda  actCommand2,x
      and  #$F7
      sta  actCommand2,x         ; clear command flag

;=======================
; SR command
;=======================

skip_2com1:
      lda  actCommand2,x
      and  #2
      beq  skip_2com2

      lda  cmdADSR,x
      sta  shadow_SR,x           ; store SR
      lda  #$00
      sta  actCommand2,x         ; clear command flag

;=======================
; Key on command
;=======================

skip_2com2:
      lda  actCommand2,x
      and  #4
      beq  skip_2com3

      lda  shadow_CTRL,x         ; put key on
      ora  #$01
      sta  shadow_CTRL,x

;=======================
; Key off command
;=======================
```

```
skip_2com3:
      lda  actCommand2,x
      and  #8
      beq  skip_2com4

      lda  shadow_CTRL,x          ; put key off
      and  #$FE
      sta  shadow_CTRL,x

;=======================
; Rest command
;=======================

skip_2com4:
      lda  actCommand2,x
      and  #$10
      beq  skip_2com5

      lda  #0                     ; put frequency to zero
      sta  shadow_FL,x
      sta  shadow_FH,x

;========================
; Filter type command
;========================

skip_2com5:
      lda  actCommand2,x
      and  #$20
      beq  skip_2com6

      lda  cmdFilter,x             ; put filter type
      sta  shadow_TYPE

      lda  #$00
      sta  actCommand2,x          ; clear command flag

;========================
; Filter resonance
;========================

skip_2com6:
      lda  actCommand2,x
      and  #$40
      beq  skip_2com7

      lda  cmdFilter,x             ; put filter res.
      sta  shadow_RES

      lda  #$00
      sta  actCommand2,x          ; clear command flag

;========================
; Filter cut off
;========================

skip_2com7:
      lda  actCommand2,x
      and  #$80
      beq  skip_3com1

      lda  cmdFilter,x             ; put filter cut
      sta  shadow_FCH

      lda  #$00
      sta  actCommand2,x          ; clear command flag

;=======================
; SR command
;=======================

skip_3com1:
      lda  actCommand3,x
      and  #1
      beq  skip_3com2

      lda  cmdADSR,x
```

```
      sta   shadow_SR,x            ; store SR

skip_3com2:
      rts
```

Play Hard-Restart

Play hard restart is the last block in diagram and is for making the hard restart of note if this event is triggered.

```
;================================
; play a HR if this is the case
;================================
pl_play_hr:
      lda   hrActive,x             ; HR activated into this row?
      beq   skipPlayer

      lda   instr_HR,x             ; read HR of instrument
      and   #$0F
      sta   TEMP                   ; isolate the ticks
      inc   TEMP                   ; we check for minor of this

      ldy   pattDelay,x            ; read actual remaining delay ticks
      beq   latest                 ; last ticks?
      cpy   TEMP
      bpl   exitPHR                ; no time for HR

      lda   hrActive,x
      bmi   fullHR

      lda   shadow_CTRL,x          ; put gate off
      and   #$FE
      sta   shadow_CTRL,x
      rts

fullHR:                            ; activate the full HR
      lda   instr_AD,x
      sta   shadow_AD,x
      lda   instr_SR,x
      sta   shadow_SR,x
      lda   instr_CTRL1,x
      sta   shadow_CTRL,x
      rts

latest:                            ; latest cicle before new note
      lda   instr_HR,x
      and   #$80                   ; HR?
      beq   exitPHR                ; no, gate off, so skip

      lda   instr_CTRL2,x
      sta   shadow_CTRL,x          ; put control 2 to did

exitPHR:
      rts
```

Conclusion

Even if the actual code of JITT64 is changed as it has conditional compilation in it for removing unused code of your tune, the structure is not changed, so you will have no difficult to look at the JITT64 ver 1.03 source code.

As you have seen, the use of so freedom into instrument implementation have make the code to manage it very complex, and so raster time usage can be high. However, due to the high use of macro into the code, if I find some better way to make the same operation (maybe using undocumented instructions), then I just need to modify a macro for having all the code ready for being more quick in execution.

Sin 13 end