U-235 Sound Engine

User Manual

Contents

Introduction

The U-235 SoundEngine(tm) is a software package intended for use by developers on the Atari Jaguar 64bit Multimedia System. It provides multi-voice sample playback using entirely DSP-based code; this frees up the other system CPUs for your game. Technology has been built into the SoundEngine(tm) to simplify its use, whilst providing flexibility and minimizing main system bus access.

Licensing

Definitions

- "The Software" refers to the U-235 SoundEngine, which is provided as an assembled binary object for use on the Atari Jaguar 64bit Multimedia System.
- "Raptor" refers to the Raptor Game Engine by Reboot (http://reboot.atari.org).
- "Author(s)" refers to group U-235 (http://www.u-235.co.uk).

Licence

This software is provided free of charge to anyone and everyone. U-235 accepts no responsibility for damage or loss by its use or misuse. U-235 grants you the right to use this software within your own works provided that:

- Clear identification of the use of this software is included within your own works, either by use of one of the approved logos provided, or textually.
- The identification of the use of this software must appear within the digital works in a manner that is visible to the end user and upon any physical packaging.
- The software may not be reverse engineered or modified without prior consent of the author(s).
- No source code forming any part of The Software is to be distributed without explicit permission from the author(s).

In other works

The software also forms part of Raptor, which is a licensed external work which has been approved by the authors to include and distribute the software under the terms of Raptor's licence agreement.

Bugs

Open and closed bugs for the SoundEngine are logged here:

http://www.u-235.co.uk/developer/sound-engine/bugs/

Features

- Entirely DSP based RISC core
- Kudos Ware Licensing model
- 8 voices with independent frequency playback
- Pad reading and parsing
- 16-bit Random number generator
- Per voice volume control
- Music and SFX master mix volumes
- Mono or Stereo playback for music or sound effects
- Support for 4-channel Tracker Modules
- High fidelity sample playback

Included Files

Contained within the SoundEngine package there are the following files

Name	Required by SE	Description
Manual.pdf		This file
Licence.txt		A text-based version of the U-235 Sound Engine licence
Alf.mod		An example sound tracker file. Written by Trash
Changelog.txt		Details of changes that have occurred to the U-235 SoundEngine package
DSP.OBJ	Yes	The binary object file that is the SoundEngine itself
JAGUAR.INC		Atari's Jaguar header file unmodified. This is included to aid in assembling the example code.
~~Joypad.s~~		~~Simple code to read the joypad in port 1 of the Jaguar~~ Joypad's are now read by the SoundEngine directly
Main.s		Source code for the main body of the demonstration code.
MAKEFILE		A Makefile to build the demonstration code using standard Atari build tools (MADMAC and ALN)
ORCH.SAM		An 8bit signed mono sample of an Orchestral Hit, used within the demonstration code.
Setup.s		Simple setup routines for initialising the Atari Jaguar hardware
U235se.inc	Yes	Include file containing equates and extern directives for variables and flags within the Sound Engine
Vbi.s		Source code for the VBI handler routines.

Contained within the subdirectory "logos" are the official U-235 SoundEngine graphics for use within your project and on its packaging.

Building the Demo

To help illustrate the operation of the U-235 SoundEngine a demonstration program is included. This will play the example module "`alf.mod`" and also allow for playing the sample "`orch.sam`" via various sound voices. Before this can be built however, additional tools are required that are not included within this package.

Prerequisites

Assembler	A suitable Jaguar assembler; the whole U-235 SoundEngine has been built with Reboot's RMAC assembler.
Linker	As SoundEngine is provided as a pre-assembled object, it is required that it must be linked into any project that intends to use it. U-235 makes use of Reboot's RLN linker.
Make	This is optional but highly recommended for any project that is complex. The use of make will greatly simplify the building process of the demonstration and also of any other project if used correctly.

(*RMAC and RLN can be downloaded from Reboot's website here:*
http://reboot.atari.org/new-reboot/rmacrln.html)

Using Make

A makefile is already included with the demonstration source; it assumes the use of Atari's own tools (MAC and ALN) and that these are located within the search path. If this is the case, running make should yield a complete working `MAIN.COF` file that can be loaded into Jaguar memory.

By hand

It is highly recommended that make is used. To assemble the demonstration without make follow these steps. (It is assumed that MADMAC and ALN are being used and are in the systems search path.

First assemble the `main.s` source file

```
rmac –fb –s main.s
```

This will produce the file `main.o` this object file now needs linking before it can be executed

```
rln –e –w –rq –o main –a 4000 x x main.o dsp.obj
```

Will combine the two object files `main.o` and `dsp.obj` and resolve all labels within the object files producing a linked binary `main.cof`.

Demonstration controls

Once loaded into an Atari Jaguar the demo will start playing the module. The screen will be blank. It is possible to interact with the demo using the keypad on the controller plugged into port 1 of the Jaguar. The controls are:

Button	Function
1	Start module playback
2	Stop module playback
3	Play sample on voice 4 at note 32
4	Play sample on voice 5 at note 37
5	Play sample on voice 6 at note 42
6	Play sample on voice 7 at note 47
7	Play sample on voices 0-7 at note 25 simultaneously

Using the sound engine

This section covers the process of utilising the sound engine within your own projects. The detail section will cover each aspect in detail.

Overview

Before the sound engine can be used it must first be loaded into the DSP RAM, the DSP must then be started with its program counter pointing at the start of DSP RAM. The sound engine will initialise the DSP timers and start waiting for jobs immediately.

With the engine running if there are any external (non-module) based samples that are to be used the pointer to the sample definition table should be set.

If a module is to be played it must first be processed by the `modinit` function (68K code at the time of writing), which will initialise the appropriate variables within the sound engine. Once the module has been initialised the module playback can be started by setting the playback flag appropriately.

Command lists can now be passed to the sound engine.

The SoundEngine

Throughout this section the example code provided with the sound engine will be used as the code base for the examples presented here. This should provide a useful reference.

Throughout this document it is assumed that the `JAGUAR.INC` and `u235se.inc` files are included in the project for the purposes of easy to read equates to Jaguar hardware addresses etc.

Required Files

There are certain key files that are needed to use the sound engine. Some will need to be included within the project's source and assembled whilst the main sound engine itself needs to be linked into the project.

U235se.inc	This file provides equates for configuration parameters and also .EXTERN directives for variables and labels within the Sound Engine itself. It should be included into a project in the same was as JAGUAR.INC is included.
DSP.OBJ	This is the actual DSP binary code that forms the sound engine. This needs to be linked into the project via the linker (ALN for example).

Initialisation

Before the sound engine can be used, the DSP must be made ready, the code copied into the DSP and the DSP started.

The sound engine can be copied into the DSP RAM with the following simple loop:

```
        move.w      #2048,d0
        lea         dspcode,a0
        move.l      #D_RAM,a1

.loop:
        move.l      (a0)+,(a1)+
        dbra.w      d0,.loop
```

This code will copy 8KB of data into the DSP RAM. It's not exactly refined but keeps things simple ☺

The Sound Engine defaults the master playback rate to 16kHz, if in your project you wish to use a different playback rate the rate and associated period can be changed. These parameters must be set before the DSP is initialised. The u235se.inc file includes a selection of parameters for this purpose, the sample code sets up the Sound Engine for 24kHz playback.

```
        move.l      #U235SE_24KHZ,U235SE_playback_rate
        move.l      #U235SE_24KHZ_PERIOD, U235SE_playback_period
```

As we are going to be using sound, it is a good idea to enable the sound output with the following:

```
move.w      #$100,JOYSTICK
```

All the code is now loaded in the DSP, so it can be started. The start-up first sets the DSP PC to the start of DSP RAM which will cause the sound engine initialisation code to run and setup the programmable timers within the DSP, and start waiting for work to do. Once running the sound engine will not access main RAM within the Jaguar until it is needed for a task.

```
move.l      #D_RAM,D_PC
move.l      #RISCGO,D_CTRL
```

The sound engine is now running, but has not yet been initialised with locations of sample banks etc.

Sample Bank Definition

The Sound Engine works with sample banks; these are simple data structures within RAM that contain all the information needed by the Sound Engine to play the various samples. It is possible to have as many sample banks as desired, the module player makes use of its own internal sample bank for the modules samples, if so desired this bank can be accessed via the label `U235SE_sample_tbl`, however it should be noted that the default playback frequency will not be set so a frequency must always be provided. Samples are numbered starting at 0.

IMPORTANT: All sample banks MUST be double phrase aligned (DPHRASE)

The sample table is a simple data structure which, in the example code, is as follows:

```
.DPHRASE
    ; Sample #1
        dc.l  sample1          ; base address of sample in RAM
        dc.l  sample1_end      ; End of sample
        dc.l  0                ; RBASE
        dc.l  0                ; REND
        dc.l  64               ; <null word> | Fine tune | Volume
        dc.l  2990          ; Default play freq (only used in SFX)
                            ; 24 bytes per sample.  744 bytes total
```

The elements of this data structure are:

BASE	The location in RAM or ROM of the sample to be played. All samples must be word-aligned.
END	This is the address in memory of the end of the sample. Note: The end MUST be greater than the sample base or the sound engine will simply assume it has already reached the end of the sample and play nothing.
RBASE	Repeat Offset address - this address is only for use for samples that loop. If a sample is to loop then the RBASE value is the address in memory of where the sample loop should start between BASE and END. This is an absolute address and not a relative one.
REND	Repeat End address - once a sample has looped past its original end and is in a looping state, the value in REND will be used to determine the end of the sample, at which point it will repeat again until stopped by another means (voice stop or new sample on that voice).
SETTINGS	Each sample has a default playback volume and, if being used via note, calls a fine tune setting. These two values are defined in this long. The lowest byte defines the volume, values of 0-64 are permitted, and the byte before this defines the fine tune value (0-15)
FREQ	The default playback frequency for the sample in Hertz (Hz)

Before a sample is played from the sample bank the Sound Engine must be told to use the specific sample bank. This is achieved by using the "Set Sample Bank" Sound Engine command within the playlist. A sample bank must always be defined as there is no guarantee that one has been set.

Adjusting the volume

Within the final mixing stages of the SoundEngine the voices 0-3 and 4-7 are treated independently. Voices 0-3 are grouped as music voices (they are used by the mod playback routines) and voices 4-7 are grouped together as sound effects (or SFX). This grouping allows for adjustment of the overall music or sound effect volumes. These adjustments will be applied to all 4 voices in the group and after the volume adjustments of the channels themselves.

The volume range is 0-63 as per the individual voices themselves. Adjusting the volume is simply a case of writing the desired volume for that group to the appropriate variable within the Sound Engine. The defined equates for these are:

Volume group	Equate
Music Volume (voices 0-3)	`U235SE_music_vol`
Sound Effects Volume (voices 4-7)	`U235SE_sfx_vol`

Note: As these variables exist within DSP memory, writes to them made by the 68000 CPU must be a long write.

Mixing

The SoundEngine will mix the music and sound effects in either mono or stereo independent of one another. In Stereo mode the music will honour the original left and right configurations of the Amiga module format voice 0 & 3 left, 1 & 2 right, for sound effects voices 4 & 6 are left 5 & 7 are right. By default the SoundEngine is configured to play the music in mono mode (ensuring all instruments play out of both speakers) whilst sound effects maintain stereo.

To set the configuration of this the variable `U235SE_playbackmode` is used. Setting bit 0 enables stereo Music, Setting bit 1 enables Stereo sound effects.

Command List

The SoundEngine operates entirely independently of the rest of the Jaguar system. It is a completely self-contained system, instructions are passed to the SoundEngine in a similar way to how the Jaguar's Object Processor operates. A simple list of commands are passed to the SoundEngine, which then works its way through them in turn. Unlike the Object Processor a command list is only needed when there is a change to the current state of the SoundEngine. The frequency that new lists can be provided to the SoundEngine is up to the programmer, lists are processed fairly quickly from the point of being issued, and there is no upper limit to the number of events that can be presented on the list, only the limit of time to process. If the SoundEngine does not have a list to process it will simply continue doing what it is currently doing.

A list is made up of 32bit long instructions, the end of the list is indicated by a terminator command. The list is not changed by processing, so it can be re-issued as many times as required.

A sound engine instruction is a single 32bit long; this is built up of four parts.

32 bit long			
Data Section Y (16bit)	Data Section X (8bit)	Voice # (4bit)	Command (4 bit)

The use of the data sections is command dependant (see the table in the Sound Engine commands section of this manual)

Constructing an entry in a list can be achieved with:

```
lea        playlist,a0

move.l     #sample_bank,d0  ; Put address of sample bank in d0
and.l      #$fffffff0,d0    ; mask off the low nibble
or.l       #$b,d0           ; add Set Sample Bank instruction
move.l     d0,(a0)+         ; Write Set bank to playlist

move.l     #$11940054,(a0)+
move.l     #0,(a0)
```

In this example the first command sets the sample bank to be used for following instructions within the play list, the next command ($11940054) instructs the sound engine to play sample 0 on voice 5 at frequency 4500Hz ($1194 is 4500 in hex), this is command 4.

(*A complete list of SoundEngine commands can be found in the Appendices, page 16*)

With the list completed (ensure there is a terminator at the end of every list!), the address of the head of the list is written to the SoundEngine variable `U235SE_sfxplaylist_ptr`, the SoundEngine will then process this list.

```
    lea        U235SE_sfxplaylist_ptr,a1
    move.l     #playlist,d0
    move.l     d0,(a1)
```

Note: The module player communicates with the SoundEngine in exactly the same manner, it produces playback command lists and passes these to the SoundEngine. To remove the risk of conflicts, it uses a second internal command list hook to the SoundEngine.

SoundEngine Commands
Following are details of each individual SoundEngine command. The number in parenthesis represents the command number.

Stop Voice (1)
Resets the specified voice to no sample playing. This is a complete reset of the voice and removes all details of current playing sample.

Play Sample (2)

Play the sample number indicated in the Data X field from the current sample bank at the default frequency for that sample. This command will interrupt any already playing sample on the voice.

Set Volume (3)

Set or adjust the volume of the sample playing on the specified channel. Either a specific volume can be specified in the range if 0-63 in the Data X portion of the command, Data Y must be 0 in this case.

Or a volume adjust is required, then a signed volume change value should be specified in Data Y, this will override any specific volume setting specified in Data X at the same time.

Play Sample @ frequency (4)

Play the specified sample number in Data X from the current sample bank at the specified frequency in Data Y. The sample frequency must be specified in Hertz (Hz), so for 16kHz the value of 16000 should be used. This command will interrupt any already playing sample on the voice.

Adjust playback rate (7)

Sets the frequency of the voice to the frequency specified in Data Y. This will immediately jump the playing sample on that voice to the new frequency. It will not restart the sample, only change the playback frequency.

Reset Volume (8)

Resets the volume of the voice to the default value specified for the currently playing sample on that voice.

Play from offset (9)

Restart the currently playing sample from the offset specified in Data Y and Data X combined. For this command Data Y & X are to be treated as a single element, with Data Y being the most significant portion of the variable. The playback frequency and volume of the sample will not be altered, just the current playback position of the sample changed to be that of the offset specified.

Set Sample Bank (11)

Set the address in memory of the Sample Bank table to be used for commands following this entry in the command list. The address must be Double phrase aligned (DPHRASE) aligned (bits 0-3 all set to 0). Data Y, X and the voice number portions of this command are all used for the data.

It is entirely valid to have multiple sample banks used within a single command list, each time a sample bank is set it only has effect for the commands following.

Note: Good practice is to ALWAYS set the sound bank at the start of the list before any other commands.

Tracker Modules

Module Initialisation

Before a module can be played it must first be initialised. This requires that the variable `U235SE_moduleaddr` is set to point to the address in RAM or ROM that the module exists in.
Note: <u>Modules must be long aligned</u>.

With this variable set, the initialisation routine can be called

```
jsr        modinit
```

The module is now ready to play.

Module Playback

Playback of an initialised module is extremely simple. The variable within the sound engine to control module playback takes 1 of 3 values or for readability using one of the predefined equates.

Value	Equate	Definition
0	U235SE_NOMOD	No playback of module, samples initiated by the module will continue to play.
1	U235SE_PLAYMONO	Play module ~~mono. All 4 voices of the module will play from both left and right channels.~~
2	U235SE_PLAYSTEREO	Play module ~~stereo. Two voices of the module will play through each channel. This is how the module would sound on an Amiga.~~

NOTE: Mixing of the module is now controlled via the U235SE_playbackmode variable, please see the section 'Mixing' on page 10.

This can be seen in the example code

```
move.l      #1, U235SE_playmod

; or

move.l      #U235SE_PLAYMONO, U235SE_playmod
```

IMPORTANT: Remember, whenever writing to DSP RAM with the 68k, ALWAYS write 32bit longs or strange things can happen!

Module Position Adjustment

If it is required to reset a module back to the start (perhaps playing a new module), or adjust other playback attributes of a module, it is possible to manually edit the variables used by the module player. At present there is no API call to do this.

It is recommended that these variables are not adjusted whilst a module is actually playing, the changes could be ignored or cause the player to do undesirable things (like consume all the available bus bandwidth).

To start a new module, it is recommended that any playing module is stopped, then prior to running `modinit` that 0 is written to the first 4 longs at the address `U235SE_modregdump.`

The following table describes the use of each of these longs, the hex value is the offset from `U235SE_modregdump.`

Offset	Purpose	Description
$0	Ticks	The number of timer ticks that have passed. Used to calculate when to advance a division
$4	Count	The number of ticks per division. This is also pointed to by the variable `U235SE_modspeed`
$8	Div	The current pattern division
$C	Position	The current position within the module.

Random Number Generator

As of version 0.20 the Sound Engine now features a 16-bit pseudo random number generator (RNG). If the Sound Engine is loaded and the DSP running the RNG will continue to generate random numbers. This has no impact on the Sound Engine or module playback and runs in the free time the DSP has between processing sample data.

To access the output from the RNG simply perform a long read from the label `U235SE_rng`, for example:

```
move.l      U235SE_rng,d0
```

Will populate d0 with a 16-bit random number. Note: a long read is required due to the architecture of the Jaguar bus, also the 17th bit will also toggle, so it could be considered to be a 17-bit random number should that be of use.

Joypad's

As of version 0.21 the SoundEngine automatically polls and interprets the Jaguar's joystick ports. This allows the user to easily determine the state of the Jaguar's pad 1 and pad 2 with very simple code. The current state of the pad is accessed by performing a long read (32 bit) of the address `U235SE_pad1` and `U235SE_pad2` respectively. Doing so will yield a bitmap indicating the state of each button of the Jaguar pad.

To aid with code readability equates have been defined within the SoundEngine include file (u235se.inc) for each of the buttons on the pad. A table of these equates can be found on page 18. Equates have been provided for testing for a specific bit as well as an integer value.

For example:

```
move.l      U235SE_PAD1,d0   ; Get pad 1 status
and.l       #U235SE_BUT_STAR|U235SE_BUT_HASH,d0
cmpi.l      #U235SE_BUT_STAR|U235SE_BUT_HASH,d0
beq         Reset
```

or, using bit tests:

```
move.l      U235SE_PAD1,d0
btst        #U235SE_BBUT_STAR,d0
beq         .noreset
btst        #U235SE_BBUT_HASH,d0
beq         .noreset
; RESET if we get here!
```

Appendices

Sound Engine commands

All commands to the sound engine are 32bit long. They are composed as per the following table:

32 bit long			
Data Section Y (16bit)	Data Section X (8bit)	Voice # (4bit)	Command (4 bit)

The following table lists all the commands that are available, and a brief description of their function.

Cmnd #	Data Y	Data X	Name	Description
0	-	-	Terminator	Indicates the end of the command list. This is required at the end of every list.
1	-	-	Stop Voice	Ceases all sample playback on the specified voice.
2	-	Sample number	Play Sample	Plays a sample on the specified voice at the default playback frequency (FREQ) for that sample.
3	Volume adjust amount, signed (Optional)	Volume	Set Volume	Either set or adjust the volume of the specified voice.
4	Playback Frequency in Hz	Sample number	Play Sample @ frequency	Play a sample but at the playback frequency specified in the command.
5	~~New playback rate~~	~~Volume ($FF = no change)~~	~~Modify Channel~~	~~Make a change to a voice's settings, change the playback rate and optionally volume.~~ No longer implemented, please use commands 3 & 7
6				Unused
7	New playback frequency in Hz	-	Adjust playback rate	Change the playback frequency of the specified voice.
8	-	-	Reset Volume	Restore the volume of channel to the currently playing sample's default volume.
9	Offset of sample		Play from offset	Play sample on voice but start the sample playback at the given offset.
10				Unused
11	DPhrase aligned address of sample bank to use for following commands		Set Sample Bank	Sets the sample bank that the Sound Engine is to use, only parameter is the command nibble, all other bits are for the address as this is a global command and not voice specific.

Tracker Module Effects support

The following table details each of the sound tracker effects and their current level of support within the sound engine.

Effect #	Effect HEX	Name	Status	Notes
0	0	Arpeggio	Un-Supported	
1	1	Slide up	Supported	
2	2	Slide down	Supported	
3	3	Slide to note	Supported	
4	4	Vibrato	Supported	
5	5	Slide to note & Volume slide	Un-Supported	
6	6	Vibrato & Volume slide	Supported	
7	7	Tremolo	Un-Supported	
9	9	Set sample offset	Supported	
10	A	Volume Slide	Supported	
11	B	Position Jump	Supported	
12	C	Set Volume	Supported	
13	D	Pattern Break	Supported	
14 0	E0	Set filter	Un-Supported	This is likely to never be implemented
14 1	E1	Fine slide up	Supported	
14 2	E2	Fine slide down	Supported	
14 3	E3	Set glissando	Un-Supported	
14 4	E4	Set vibrato waveform	Un-Supported	
14 5	E5	Set finetune value	Un-Supported	
14 6	E6	Loop Pattern	Un-Supported	
14 7	E7	Set Tremolo waveform	Un-Supported	
14 9	E9	Retrigger Sample	Un-Supported	
14 10	EA	Fine volume slide up	Supported	
14 11	EB	Fine volume slide down	Supported	
14 12	EC	Cut Sample	Un-Supported	
14 13	ED	Delay sample	Un-Supported	
14 14	EE	Delay pattern	Un-Supported	
14 15	EF	Invert Loop	Un-Supported	
15	F	Set speed	Supported	

NB: Unless otherwise stated "Un-Supported" effects will be implemented over time.

Joypad Equates

This table illustrates the defined equates for determining the state of the joypad

Action	Integer Equate	Bit Equate
D-pad Up	`U235SE_BUT_UP / U235SE_BUT_U`	`U235SE_BBUT_UP / U235SE_BBUT_U`
D-pad Down	`U235SE_BUT_DOWN / U235SE_BUT_D`	`U235SE_BBUT_DOWN / U235SE_BBUT_D`
D-pad Left	`U235SE_BUT_LEFT / U235SE_BUT_L`	`U235SE_BBUT_LEFT / U235SE_BBUT_L`
D-pad Right	`U235SE_BUT_RIGHT / U235SE_BUT_R`	`U235SE_BBUT_RIGHT / U235SE_BBUT_R`
A	`U235SE_BUT_A`	`U235SE_BBUT_A`
B	`U235SE_BUT_B`	`U235SE_BBUT_B`
C	`U235SE_BUT_C`	`U235SE_BBUT_C`
#	`U235SE_BUT_HASH`	`U235SE_BBUT_HASH`
*	`U235SE_BUT_STAR`	`U235SE_BBUT_STAR`
0	`U235SE_BUT_0`	`U235SE_BBUT_0`
1	`U235SE_BUT_1`	`U235SE_BBUT_1`
2	`U235SE_BUT_2`	`U235SE_BBUT_2`
3	`U235SE_BUT_3`	`U235SE_BBUT_3`
4	`U235SE_BUT_4`	`U235SE_BBUT_4`
5	`U235SE_BUT_5`	`U235SE_BBUT_5`
6	`U235SE_BUT_6`	`U235SE_BBUT_6`
7	`U235SE_BUT_7`	`U235SE_BBUT_7`
8	`U235SE_BUT_8`	`U235SE_BBUT_8`
9	`U235SE_BUT_9`	`U235SE_BBUT_9`
Pause	`U235SE_BUT_PAUSE`	`U235SE_BBUT_PAUSE`
Option	`U235SE_BUT_OPTION`	`U235SE_BBUT_OPTION`

Credits

Code & Docs: LinkoVitch
Testing: sh3
Additional Testing: Cyrano Jones, Matmook
Logo artwork: sh3
Proof Reading: Mug UK, GazTee, Cyrano Jones, sh3

Greetings and thanks

In no special order:

U-235

GazTee & sh3 (thanks guys, without you chaps this wouldn't be possible -Link)

Reboot

Special hellos to Cyrano Jones, RemoWilliams, Sauron, MSG & ggn

Jagware

Special thanks to ZeroSquare for all the Jaguar help, GT-Turbo,
SCPCD, Matmook

AtariAge, Shamus, 505, Tyr of the Arcana, OMF, GroovyBee, AtariOwl, StarCat, Thorn, BMX, Mr & Mrs Atari,
Nick Harlow, Stone, Partycle, Mug UK, DrTypo