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.

Features

- 8 voices with independent frequency playback
- 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
- Entirely DSP based RISC core
- High fidelity sample playback
- Kudos Ware Licensing model

Included Files

Contained within the SoundEngine package there are the following files

Name	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.O	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
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.
PERIOD14.S	Period timing table for 14.4kHz, used by the SoundEngine
Sequence.s	Source code for the module initialisation routines as used by the SoundEngine
Setup.s	Simple setup routines for initialising the Atari Jaguar hardware
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 Atari's own MADMAC 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 Atari's own ALN 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.

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

```
Mac -fb -s main.s
```

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

```
Aln -e -g -w -rq -o main -a 4000 x x main.o dsp.o
```

Will combine the two object files `main.o` and `dsp.o` 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.

Detail

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` file is 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.

PERIODxx.S	This file provides a lookup table for translating the playback frequencies used within tracker modules to values that can be handled by the sound engine. This file must be included within the project source. The numerical value in the file name indicates the playback frequency the lookup tables have been computed for - each playback frequency has different values.
SEQUENCE.S	This file contains the 68000 source code that is used to initialise the sound engine for the module to be played. It must be included within the project.
DSP.O	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). It actively uses the contents of the PERIODxx.S file hence the requirement for that file to be included in the project.

Initialisation

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

First of all, the I2C timer needs to be set; this is achieved with the following code:

```
move.l      #19,SCLK
move.l      #$15,SMODE
```

Now 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 ☺

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.

External Sample Initialisation

Samples not included in the module must be defined in a sample table. This table provides all the information about the sample in much the same way as a tracker module does; this also simplifies the playback of a sample as it can simply be referred to by number. Samples not located within the module (even if no module is loaded) start at number 32 (the first 31 sample slots being reserved for those contained within a module).

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

```
; Sample #1
      dc.l  sample1          ; base address of sample in RAM
      dc.l  sample1_end      ; End of sample
      dc.l  0                ; ROFF
      dc.l  0                ; RLEN
      dc.l  64               ; <null word> | Fine tune | Volume
      dc.l  2990       ; Default play rate (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 ROFF 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)
PERIOD	The default playback period for the sample. This 'period' is actually a ratio value between the output frequency and the sample playback frequency. It can be computed with the following formula $$\frac{fp}{fm} * 4096$$ Where *fp* is the sample playback frequency and *fm* is the sound engine's master playback frequency.

With all of the non-module samples defined, the sound engine needs to know the location in RAM/ROM of this table; this is set by the following code:

```
.EXTERN ptr_sample_bank
lea         ptr_sample_bank,a0
move.l      #sample_bank,(a0)
```

The non-module sample can now be used via the sound engine.

Module Initialisation

Before a module can be played it must first be initialised. This requires that the variable `moduleaddy` is set to point to the address in RAM or ROM that the module exists in. Note: Modules must be word aligned.

In the example code the address of the module is already set in the `sequence.s` file at the variable definition:

```
moduleaddy:      dc.l  module
```

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

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

This can be seen in the example code

```
move.l      #1,playmod
```

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

Sample Playback

Playback of individual samples with the sound engine works in a way similar to that of the Jaguar's object processor. A list of instructions are created; the sound engine is then pointed at this list and processes the instructions.

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 (4 bit)	Command (4 bit)

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

After the list of commands, there must be a terminator. This is a long word containing only '0' (zero)

Constructing an entry in a list can be achieved with:

```
lea         playlist,a0
move.l      #$00202046,(a0)+
move.l      #0,(a0)
```

In this example the command instructs the sound engine to play sample 32 (the 1st non module sample in the external bank) on voice 4 at note 32, this is command 6.

If there are multiple commands to be issued then they are added to the list before it is sent:

```
lea         playlist,a0
move.l      #$192006,(a0)+
move.l      #$192016,(a0)+
move.l      #$192026,(a0)+
move.l      #$192036,(a0)+
move.l      #$192046,(a0)+
move.l      #$192056,(a0)+
move.l      #$192066,(a0)+
move.l      #$192076,(a0)+
move.l      #0,(a0)
```

Here the sample will be played on all 8 voices simultaneously.

Simply creating the list will not cause the sound engine to act on the commands. Activating the list of commands is achieved by passing the address of the (terminated) list to the sound engine.

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

As soon as the playlist address is set, the sound engine will process the list. The sound engine will not modify the list in anyway (unlike the Jaguar's object processor), so a list can be reused if needed.

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 (4 bit)	Command (4 bit)

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

Command #	Data X	Data Y	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 rate for that sample.
3	Volume	Volume adjust amount, signed (Optional)	Set Volume	Either set or adjust the volume of the specified voice.
4	Sample number	Playback rate	Play Sample @ rate	Play a sample but at the playback rate specified in the command.
5	Volume ($FF = no change)	New playback rate	Modify Channel	Make a change to a voice's settings, change the playback rate and optionally volume.
6	Sample Number	**High byte**: Volume ($ff no change) / **Low byte**: Note number.	Play Note	Play sample on the voice using the playback rate for the specified note number. (See Note Lookup table)
7	Signed amount to adjust period by		Adjust playback rate	Change the playback rate 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.

Note lookup table

The following table gives the relationship between a note number as used within the sound engine, its matching Amiga Tracker module 'period' value and the actual sample playback frequency. The playback frequency will vary very slightly due to rounding errors generated within the byte-skipping code, but these should not be audible to the human ear.

Note Number	Module Period	Playback Frequency (Hz)
0	1712	2071.78
1	1616	2194.86
2	1525	2325.83
3	1440	2463.12
4	1357	2613.78
5	1281	2768.85
6	1209	2933.74
7	1141	3108.58
8	1077	3293.31
9	1017	3487.61
10	961	3690.84
11	907	3910.58
12	856	4143.57
13	808	4389.72
14	762	4654.72
15	720	4926.24
16	678	5231.41
17	640	5542.02
18	604	5872.34
19	570	6222.62
20	538	6592.74
21	508	6982.08
22	480	7389.36
23	453	7829.79
24	428	8287.14
25	404	8779.44
26	381	9309.44
27	360	9852.49
28	339	10462.82
29	320	11084.05
30	302	11744.69
31	285	12445.25
32	269	13185.48
33	254	13964.15
34	240	14778.73
35	226	15694.23
36	214	16574.28
37	202	17558.89
38	190	18667.87
39	180	19704.97
40	170	20864.09
41	160	22168.09
42	151	23489.37
43	143	24803.46
44	135	26273.30
45	127	27928.31
46	120	29557.46
47	113	31388.45
48	107	33148.55
49	101	35117.77
50	95	37335.74
51	90	39409.94
52	85	41728.18
53	80	44336.19
54	76	46669.67
55	71	49956.27
56	67	52938.73
57	64	55420.23
58	60	59114.92
59	57	62226.23

Credits

Code & Docs:	LinkoVitch
Testing:	sh3
Additional Testing:	Cyrano Jones
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, mucho chocs coming your way

SCPCD, Matmook

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