

U-235 Sound Engine



User Manual

Contents

Introduction	3
Licensing.....	3
Features	3
Included Files	4
Building the Demo	5
Prerequisites	5
Using Make	5
By hand	5
Demonstration controls.....	6
Using the sound engine	7
Overview	7
Detail.....	7
Required Files.....	7
Initialisation.....	8
Sample Bank Definition.....	9
Module Initialisation	10
Module Playback.....	10
Module Position Adjustment	11
Sample Playback	12
Random Number Generator	14
Appendices.....	15
Sound Engine commands.....	15
Tracker Module Effects support	16
Credits	17
Greetings and thanks	17

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
- 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
- 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	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
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.
Sequence.s		Source code for the module initialisation routines as used by the SoundEngine This is now included within the DSP.OBJ file.
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 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.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.

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.

U235se.inc	This file provides equates for configuration parameters and also <code>.EXTERN</code> directives for variables and labels within the Sound Engine itself. It should be included into a project in the same way as <code>JAGUAR.INC</code> 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. This also simplifies the playback of a sample as it can simply be referred to by number. 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.

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: Modules must be long aligned.

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	<code>U235SE_NOMOD</code>	No playback of module, samples initiated by the module will continue to play.
1	<code>U235SE_PLAYMONO</code>	Play module mono. All 4 voices of the module will play from both left and right channels.
2	<code>U235SE_PLAYSTEREO</code>	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, 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 <code>U235SE_modspeed</code>
\$8	Div	The current pattern division
\$C	Position	The current position within the module.

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 # (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)

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     #sample_bank,d0 ; Put address of sample bank in d0
and.l     #$ffffff0,d0     ; mask off the low nibble
or.l      #$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 (\$11940044) instructs the sound engine to play sample 0 on voice 5 at frequency 4500Hz (\$1194 is 4500 in hex), this is command 4.

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

```
lea        playlist,a0

move.l     #sample_bank,d0 ; Put address of sample bank in d0
and.l     #$ffffff0,d0    ; mask off the low nibble
or.l      #$b,d0         ; add Set Sample Bank instruction
move.l     d0,(a0)+       ; Write Set bank to playlist

move.l     #$11940004,(a0)+
move.l     #$11940014,(a0)+
move.l     #$11940024,(a0)+
move.l     #$11940034,(a0)+
move.l     #$11940044,(a0)+
move.l     #$11940054,(a0)+
move.l     #$11940064,(a0)+
move.l     #$11940074,(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        U235SE_sfxplaylist_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.

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.

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.

Command #	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.
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	Un-Supported	
14 2	E2	Fine slide down	Un-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.

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, mucho chocs coming your way
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