Acorn 16-bit Audio Card User Guide

Table of Contents

Introduction I

Unpacking 2

Installing the hardware 2

The software 4

The Programmer's Interface 9

16-bit stereo sound support for RISC OS
24

New sound system SWI calls 25

New CMOS RAM sound configuration options
39

Introduction

The Acorn 16-bit Audio Card is an audio upgrade designed for the Acorn Risc PC computer: It replaces the existing 8-bit logarithmic sound system with a 16-bit linear system. It consists of a simple hardware upgrade that fits onto the motherboard with the following software support:

- Emulation of the existing sound system with improved output quality
- A sample player application to play digital audio files supporting a wide range of sample types and rates
- Sound modules to provide 16-bit sound output from Acorn Replay.

Unpacking

Your Acorn 16-bit Audio Card pack should include

1 the hardware card

2 a floppy disc

3 this manual

4 a registration card:

If any of these is missing, please contact your supplier: Please fill in the registration card and return it to the address on the card:

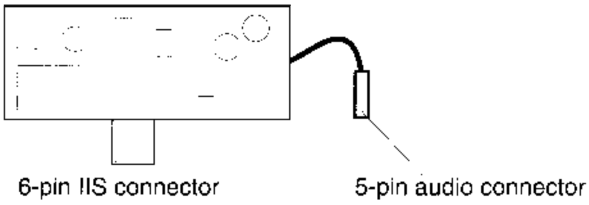
Installing the hardware

Switch off the computer and unplug the mains lead:

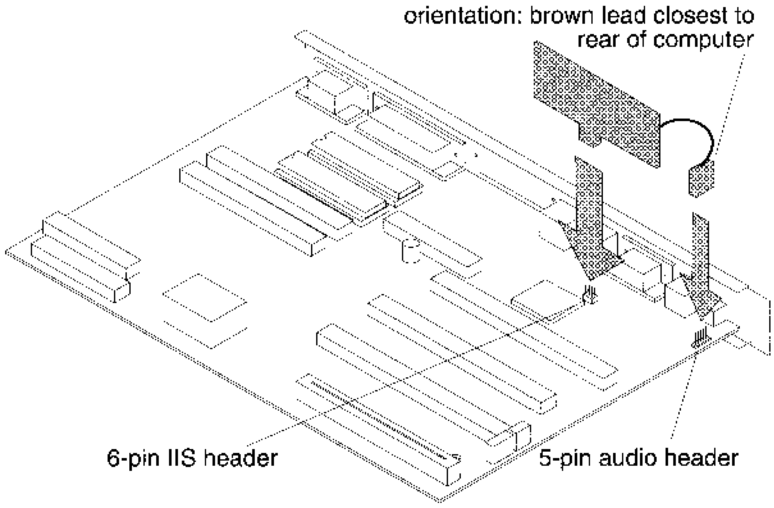
Open up the machine following the instructions given in your computer's *Welcome Guide:*

Remove any expansion cards that may be fitted to the computer:

The Acorn I6-bit Audio Card has two connectors as follows:



The headers can be found on the Risc PC circuit board as follows:



On some early models of the Risc PC the header for the audio connector was not fitted: If you have one of these machines, you can have a header fitted under the Acorn on-site warranty provided with the Risc PC.

1 Connect the 6-pin connector on the Acorn 16-bit Audio Card to the 6-pin header on the Risc PC board, with the component side of the card facing the front of the computer:

   The audio header on the Risc PC board can have either four or five pins fitted: The audio connector from the card is a 5-pin connector:

2 Remove any links that are currently fitted to the audio header:

3 Fit the 5-pin connector with the brown wire fitted to the first pin of the audio header (the one closest to the rear of the computer):

4 Refit any expansion cards you may have removed, and reassemble the computer as shown in the *Welcome Guide:*

Amplification

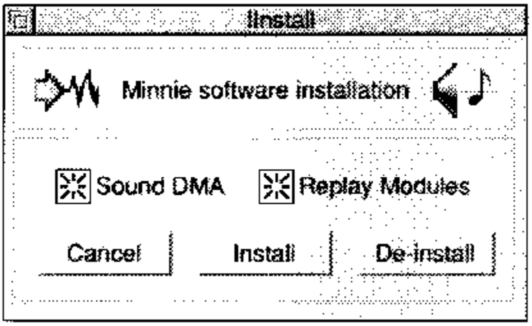
The output from the card will automatically play through the computer' s own speaker: But to hear the output to the best quality you will need to connect either headphones or an amplifier to the headphone socket on the back of the computer:

The software

The disc contains two applications: !Install and !Player: It also contains examples of 16-bit digital samples in the directory AudioDemo:

!Install

!Install provides a dialogue box with options to install or de-install the new sound DMA and the new I6-bit Replay modules: Run this



application first and choose **Install** for both SoundDMA and Replay modules: (Install copies the former into `!Boot . Choices . Boot . PreDesk . SoundDMA` and updates !ARMovie with the latter: It also configures the CMOS RAM to indicate that the card has been installed:)

If for any reason you need to remove the card, run !Install and choose **De-Install:** This will reset the CMOS RAM and return ! ARMovie to its previous state.

!Player

The !Player application provides a playback facility for a wide range of digital sample types and sample rates. You will need to copy the application from the floppy disc to your hard disc: You are recommended to place it in the Apps directory on your hard disc: To do this:

1 Open up the Apps directory :
2 Drag the !Player application to the Apps directory.

The AudioDemos directory contains examples of 16-bit audio data in a range of different styles. You are recommended to copy this directory into the Sound directory on your hard disc.

!ARPlayer

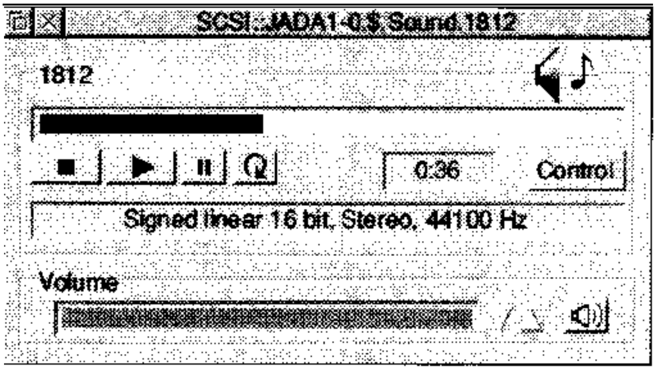
The disc includes a later version of ARPlayer (1:30), modified for 16-bit sound: Replace your current version (assuming it is earlier than 1:30) with this new one:

Playing Digital Audio

Run the !Player application from your hard disc:

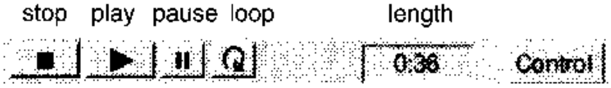
Double-click on the !Player icon

Open up the AudioDemo directory and drag a file onto the !Player icon: The main player window opens up:



Signed linear 16 bit, Stereo, 44100 Hz

Playing a sample is controlled by the following buttons:



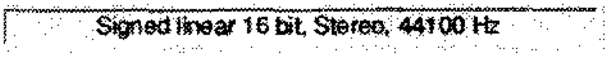
Volume

While the sample is playing the volume can be controlled either by clicking directly on the volume bar or using the arrows to the right of the bar to increase or decrease the volume. The sound can be muted by clicking on the mute icon.

Sample types

Digital samples can take a variety of forms: The !Player application recognises a number of the audio file types, including Acorn Replay, AudioWorks and WAV: It will play the first track in a Replay file.

To run a file, drop it into !Player (note that double-clicking on a sound file will not load it into !Player:). It will try to set the parameters in the player using information from the file:

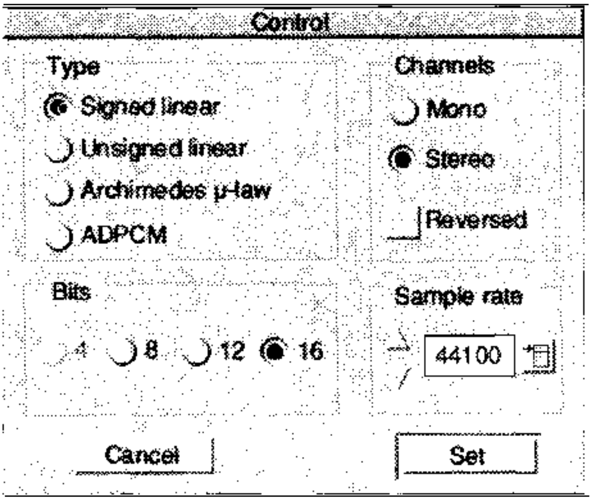


Signed linear 16 bit, Stereo, 44100 Hz

current sample setting

If !Player does not recognise the file format that you have provided, it will ask you if you would like the file loaded as raw data: Choose **OK** and the file will be loaded without setting the parameters. In such cases you will need to set them yourself, using your knowledge of how the file was created, or just by trial and error:

You can also change the parameters of known files to effect the way that they are played: In particular you might want to change the sample rate of playback: Click on the control button. A dialogue box to set the parameters is displayed:



Control

Type: [✓ Signed linear] Unsigned linear Archimedes µ-law ADPCM

Channels: Mono [✓ Stereo]

Reversed

Bits: 4 8 12 [✓ 16]

Sample rate: 44100

Cancel Set

Type

The module supports four types of digital storage:

- Signed linear
- Unsigned linear
- Archimedes Haw, the logarithmic scale used by the Archimedes sound system:
- ADPCM, a compressed format which turns I6-bit data into 4 bits: Used by Acorn Replay:

Bits

ADPCM can only be 4 bits; Archimedes µ-law can only be 8 bits: The linear types can be 8, 12 or 16 bits:

Channels

The channels used can be **Mono** or **Stereo:** If playing in stereo you can swap the left and right channels by choosing **Reversed:**

Sample Rate

The menu provides a list of the standard sample rates available: You can, however, choose any rate you wish by using the arrow icons or by writing directly into the icon: 1f the sample type is ADPCM, the only rates that are allowed are those provided in the menu:

Set and Cancel

Choose **Set** to enable the selected parameters' or **Cancel** to close the Control dialogue box without changes:
Note: Changing parameters causes play to start from the beginning:

Sample playback quality

The card is designed for 16-bit output: Because of this the best quality sounds will be those that have been sampled in 16 bits. The sample rate will also effect the quality, as will any filtering used when the sound was sampled. CDs store their data as

- signed linear

- 16 bits
- stereo
- 44100Hz (44.1kHz).

Therefore you will get CD-quality sounds with samples in this format. The quality of other sample types, bits and rates can vary considerably: A number of existing files, particularly Acorn Replay movies, may have been sampled at low sample rates and with 8 bits: This means that they may have poor quality audio data. When playing these files back through the old sound system the poor quality may not have been discernible: However, when played back through the Acorn 16-bit Audio Card using 16-bit output, any impurities in the sound may be accentuated. The !Player module uses oversampling techniques to try and ensure that files of all types will be played at the best quality possible:

Oversampling

If the sample playback rate is less 25kHz then the !Player module will play the data at two or more times the original sample rate. Each sample is repeated enough times to give the exact sample rate required but with increased quality: The process also uses a technique called fractional interpolation to smooth out the jumps in the repeated data, thus removing any noise introduced by the oversampling.

The Programmer's Interface

There are two levels of interface provided for the output of I6-bit Audio: The first is a high-level interface to the player module and the second is a low-level access to the SoundDMA handler:

The SamplePlayer module will attempt to extract the parameters from the loaded sound sample: If an unknown file type is loaded, it will use the existing parameters.

The Player Module *Commands

***AudioPlay**

*AudioPlay <Rate>

Play data in memory at optional new sample rate:

The data played must first have been loaded as a file using *AudioLoadFile or passed as memory pointers using *AudioSamplePointer:

***AudioStop**

Stop play.

***AudioPause**

*AudioPause <boolean>

Pause play:

<boolean>	on	pause play
	off	continue play

***AudioVolume**

*AudioVolume <value>

Set the current volume to <value>: <value> can be between 0 (no sound) and 128 (full volume): This can be done during playback:

***AudioLoadFIle**

*AudioLoadFile <filename>

Load a data file "filename" ready for playback: This can be raw data or a recognised filetype.

***AudioData**

*AudioData <Memptr> <Length>

Pass a pointer to sample data: Values expressed in decimal notation: The data must be in RMA.

***AudioType**

*AudioType <Bits> <Type> <Channels>

Set the type of data to be played: The data must be in RMA.

<Bits> 4, 8, 12 or 16; <Type> Signed, Unsigned, ArcLog or ADPCM; <Channels> Mono or Stereo:

***AudioLoop**

*AudioLoop <boolean>

Set the loop flag to on or off:

<boolean>	On	loop data
	Off	do not loop data

If the loop flag is on then the data will be repeated continuously:

***AudIoReversed**

*AudioReversed <boolean>

Set the stereo data reversed flag on or off.

<boolean>	On	data is reversed
	Off	data is not reversed

This is used to reverse the stereo position of the data:

***AudioMem**

*AudioMem <value>

Set the maximum amount of RMA to use when loading a file: If there is not sufficient space, play the data direct from disc.

***AudioPlayFile**

*AudioPlayFile <Filename> <Control> <Set>

Play a file direct from disc — typically an Acorn Replay file: The way that the file is played is determined by the <Control> parameter:

<Control>	KEY, MOUSE or POLL
KEY	Play the file until a key is pressed: The value in <Set> is optional. If <Set> is not specified, the playback will continue until any key is pressed. If <Set> is specified, playback will continue until the key with the ASCII value of <Set> is pressed.
MOUSE	Play the file until a mouse button is pressed: The value in <Set> is optional: If <Set> is not specified, playback will continue until any mouse button is pressed: 1f <Set> is specified, playback will continue until the mouse button with the value in <Set> is pressed: 4 = Select, 2 = Menu, 1 = Adjust
POLL	Wait for a *AudioPlayPoll to continue: Thisenables the playback routine to return control to the calling program, to enable a WimpPoll to take place, for example: A *AudioPlayPoll must be executed regularly to prevent the playback from pausing: <Set> is not required.

***AudioPlayPoll**

*AudioPlayPoll

Used to control PlayFile.

See AudioPlayFile above.

SWI Calls

Audio Set (SWI &452C0)

To set up the values for the current data:

On entry

R0 = sample rate

0	read the current rate
>0	set a new rate

R1 = sample type

0	read current type
1	Signed linear
2	Unsigned linear
3	Archimedes log (µ-law)
4	ADPCM

R2 = number of channels

Bits 0 to 1

0	read the current number of channels
1	mono
2	stereo

Bits 2 to 5 - unused

Bit 6	Stereo reversed flag
Bit 7	Unused

R3 = bits per sample

0	read the current number of bits
4	set to 4 bits; only for ADPCM
8	8 bits (1 byte)
12	12 bits (stereo only 3 bytes)
16	16 bits (2 bytes)

On exit

R0 = current sample rate
R1 = current sample type
R2 = current channels
R3 = current bits

Audio_Play (SWI &452C2)

Play the data currently in memory.

The data must first have been loaded using Audio_LoadFile or AudioSamplePointer and the parameters set using Audio_Set

On entry

R0 = rate

0	return current rate in R0
> 0	set new rate for playback

R1 = from

0	start from the beginning
> 0	start from byte R1

R2 = to

0	continue to the end
> 0	continue until R2 bytes into the data

On exit

R0 = current rate

Audio_Stop (SWI &452C3)

Stop playback immediately:

Audio_Pause (SWI &452C4)

Pause playback:

On entry

R0 = Boolean for pause on or off

0	pause off
1	pause on
-1	read pause state

On exit

R0 preserved, or pause state if R0 = -1 on entry.

Audio_Volume (SWI &452C5)

Set the volume level for playback:

This has immediate effect and can take place during playback:

On entry

R0 = volume level

-1	read current volume
0-128	set volume level

On exit

R0 = previous volume level

Audio Mute (SWI &452C6)

Mute sound:

On entry

R0 = Boolean for mute on or off

-1	read current value
0	mute off
1	mute on

On exit

R0 = current value

Audio_Loop (SWI &452C7)

Loop playback:

On entry

R0 = Boolean for loop on or off

-1	read current value
0	loop off
1	loop on

On exit

R0 = current value

Audio_SamplePointer (SWI &452C8)

Set data pointers to data information given.

On entry

R0 = data pointer

0	read current data pointer
> 0	set data pointer to R0

R1 = Data length

0	read current data length
> 0	set data length to RI

On exit

R0 = previous data pointer

R1 = previous length

Note: The data must be in RMA.

Audio_LoadFile (SWI &452C9)

Load a file into memory for later playback:

If the module recognises the file type, it will set the sample type parameters accordingly:

On entry

R0 = filename pointer to null-terminated filename

On exit

R0 = sample rate

R1 = sample type

R2 = channels

R3 = bits

R4 = file length in bytes

Audio_MaxMem (SWI &452CA)

Set the maximum RAM to use for Load File, otherwise play from file.

On entry

R0 = Memory value

0	use whatever is available in RMA
>0	set value to R0
1	read current value

On exit

R0 = current memory value

Audio_PlayFile (SWI &452CB)

Play a file from disc:

This is typically an Acorn Replay file: The way the file is played is determined by the value in RI; see *AudioPlayFile for further details.

On entry

R0 = Pointer to null-terminated filename

R1 = Control

1	play until a key is pressed
2	play until a mouse button is pressed
3	continue play on receipt of an Audio_Poll

R2 = Set, used for additional mouse & key information

0	if key then any key, if mouse then any button
> 0	if mouse then specify mouse button - 1 2 4 if key then ASCII value of key

R3 = from

0	start from the beginning
> 0	pointer to byte from which to play from

R4 = to

0	continue to the end
> 0	pointer to end byte

Audio_Poll (SWI &452CC)

Used to keep track of playback and continue file playback if POLL is selected in Audio_PlayFile.

On entry

On exit

R0 = current position in data

R 1 = stop flag

0	still playing
1	stopped

16-bit stereo sound support for RISC OS

The Sound DMA

The Risc PC uses the VIDC20 video controller to provide both video and sound output facilities. As standard, the VIDC20's internal sound DACs are used to implement 8-channel stereo sound, using the 8-bit Haw format to give wide dynamic range; this is fully backwards-compatible with the sound system provided on earlier Acorn 32-bit RISC computers: VIDC20 also supports a digital sound output interface, allowing for full 16-bit linear (CD-style) stereo sound output, when used in conjunction with the Acorn 16-bit Audio Card. Only one of these sound output mechanisms may be used at one time:

To support 16-bit sound output, a revised version of the RISC OS SoundDMA system software module has been produced: This module continues to support the original Haw format, both as normal on an unexpanded Risc PC, and also by software emulation, with output through the Acorn 16-bit Audio Card, when the Acorn 16-bit Audio Card has been installed: Selection of the relevant modes of operation is by means of new bits in the computer's CMOS RAM.

This following sections describe the facilities provided by the new SoundDMA module (version 1:41 or later), to support sound output through the Acorn 16-bit Audio Card.

Software emulation of µ-law sound

The SoundDMA module now supports the output of 8-bit Haw format sound data in either of two different modes:

- If the CMOS RAM configuration indicates that the computer does not have 16-bit sound output hardware, then the standard VIDC20 internal µ-law DACs will be used, as before:
- However if 16-bit sound output hardware is fitted, and this is reflected in the CMOS RAM configuration, then all sound output (originated in both µ-law and 16-bit linear formats) will be performed through the 16-bit sound path and not through the VIDC20 µ-law DACs. This is because the 16-bit sound

upgrade disconnects the analogue sound output coming from VIDC20, and routes the sound from the 16-bit DAC outputs through the headphone socket and the internal speaker; this maximises the sound quality achieved, but requires that software emulation be performed to support µ-law sound, for compatibility:

µ-law format emulation is done transparently (albeit at a small additional cost in processor time over direct hardware µ-law output), and in general no difference from the standard sound system will be apparent: However one point in particular — sample rate support — needs to be considered: Most applications using µ-law format sound produce it at the system default sampling rate of 20.83kHz (48 microsecond sample period), and indeed many applications can only use this rate — all the standard voice generators assume it. However, a few applications may choose to use other sample rates: The software emulation does provide for a wide range of the sample rates available with the standard sound system, but not all of them. In those cases where the exact sample period (in microseconds), requested by means of the Sound_Configure SWI, is not available, a nearby sample period value will be used instead. If exact sample rate is important, an application which uses the Sound_Configure SWI to set a particular rate can perform a second call of the SWI, with all input registers set to 0, and check that the current period value, returned in R2, is as required: It should be noted that this method is only applicable for programs which produce µ-law sound using the standard sound system interfaces: Programs producing 16-bit linear sound should use the new SWI calls documented below for all functions, including setting the sample rate:

New sound system SWI calls

Three new SWIs are implemented to support 16-bit sound: They are:

- Sound_Mode
- Sound_LinearHandler
- Sound_SampleRate.

Sound_Mode (SWI &40144)

Examine/control sound system configuration (16-bit sound):

On entry

R0 = function code

R1 = parameter as required for function

On exit

Registers set as determined by function code

Use

New sound applications, particularly those capable of 16-bit sound output, should always use this SWI first, with function code 0, in order to identify the current configuration of the internal sound system: If an older version of the SoundDMA module (prior to 1:41) is currently installed, it will appear as a system implementing only the original μ-law sound output facility: Function code I allows the automatic oversampling facility to be controlled: The new CMOS RAM bits defined below are read only when the SoundDMA module is initialised; hence any adjustment of hardware configuration cannot be performed without also re-initialising the SoundDMA module: However automatic oversampling may be selected dynamically; on initialisation it defaults to the value defined in CMOS RAM:

Sound_Mode 0 (SWI &40144)

Read current sound system configuration:

On entry

R0 = 0 (function code)

On exit

R0 = 0: sound system supports original (Haw) sound only;
SWIs Sound_LinearHandler and Sound_SampleRate, and all other functions of Sound_Mode, are not supported

R0 = 1: sound system is 16-bit capable (and supports µ-law sound output by emulation): the SWI calls Sound_LinearHandler and Sound_SampleRate, and full Sound_Mode functionality are all available

RI is affected differently in different cases: For versions of the SoundDMA module before 1:41, RI will be unaltered (and R0 will be 0): Otherwise, if R0 on return is 0, RI will also be set to 0; if R0 is 1 on exit, RI indicates the current sound system configuration, as below:

bits 3 - 0 sound format/clock control, as read from CMOS RAM at initialisation of SoundDMA module:

0	never returned
1	16-bit output; sample rates 44:1 kHz, 22:05 kHz, 11:025 kHz, and selected original µ-law system compatible sample rates are available
2	16-bit output; selected original µ-law system compatible sample rates only
3:.15	reserved values, not yet defined

bit 4 oversampling control:

0 - automatic linear 2x oversampling is disabled
1 - automatic linear 2x oversampling is enabled
bits 31 - 5 reserved for expansion, undefined

Interrupts

Interrupt status is undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Not defined

Use

This SWI call should always be used before attempting to access any of the other functionality introduced in support of 16-bit sound output: It allows an application to determine whether 16-bit sound output is supported by the configured sound output hardware:

If 16-bit hardware has been configured, indication is given of any external sound clock hardware facilities present (a separate oscillator is required to support CD-rate (44:1kHz) or other sample rates) and the current state of the automatic oversampling flag is also returned in R2, for convenience. A subset of the original sound system's sample rates will be available — see the description of Sound_SampleRate for more information:

If I6-bit sound output is not configured in CMOS RAM, R0 will be 0 on return, and only the original sound system SWIs, in particular Sound_Configure, should be used to determine and control sound output parameters such as sampling rate; the sound system will behave in a fully-backwards-compatible manner:

Sound_Mode 1 (SWI &40144)

Enable or disable automatic oversampling:

On entry

R0 = 1 (function code)

RI = new state:

0	disable linear 2x oversampling
1	enable linear 2x oversampling

other values - do not use

On exit

R0 preserved

R1 = old state of automatic oversampling

0	was previously disabled
1	was previously enabled

Interrupts

Interrupt status is undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Not defined

Use

This SWI allows automatic oversampling to be used, when 16-bit sound output hardware is fitted, to improve the output sound quality at lower sample rates: If automatic oversampling is enabled, then at all sample rates up to and including 25kHz, the output data stream will be oversampled by a factor of two, by simple linear interpolation, before it reaches the digital-to-analogue converters: This reduces the audible level of the undesirable high frequency image (sometimes inaccurately called the 'alias') of the sound output signal which is sent to the loudspeaker or headphones: The image signal appears in the analogue audio output as a high-pitched noise correlated with the main signal; it is a by-product of the digital-to-analogue conversion process, and is generally audible only with lower sample rates: As a side effect of the oversampling process, higher frequencies within the sound output signal are slightly reduced in amplitude; however in most cases this slight loss of 'treble' is outweighed in subjective terms by the benefit of reduced image noise level: Note: Oversampling when active consumes a small amount of processor time on each sound system interrupt, at worst approximately 3% of a 30 MHz ARM610 processor with a selected sample rate of 25 kHz:

Sound_LinearHandler (SWI &40145)

Declare the 16-bit linear stereo sound handler:

On entry

R0 = function code:

0	return current handler
1	install new handler

R1 = new handler code address (if R0 = 1), or 0 to remove the handler (value irrelevant if R0 = 0)

R2 = new handler parameter (if R0 = 1) (value irrelevant if R0 = 0)

On exit

R0 preserved

R1 = current/previous handler code address, or 0 if no handler is was installed

R2 = current/previous handler parameter, or -1 if no handler is/was installed

Interrupts

Interrupt status is undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Not defined

Use

The 16-bit sound handler is specified as two 32-bit quantities, a code address, to which control will pass when more data is required, and a parameter which will be passed to the handler in R0 as it is called: Typically the parameter will be a pointer to a data area containing any information the handler may need to perform its task. If no handler is installed, the code address is specified as 0:

This SWI must not be used unless 16-bit sound output is configured, as determined by a preceding call of Sound_Mode 0:

The handler is passed the address of a buffer to fill with 16-bit linear stereo sound data: The data in the buffer may already have been produced by conversion from µ-law multi-channel data generated in compatibility mode via the original sound system interfaces: In this case (as indicated by the flags in R3 on the call), the handler may either overwrite the data or else mix its own data with it, as selected by a user-configurable option supported by the handler:

The handler will be called under the following conditions:

On entry

R0 = parameter value as specified on installation

R1 = base of quadword-aligned buffer, to fill with 16-bit stereo data, stored as pairs of signed (2's complement) 16-bit values; each word has bits 31-16 left channel data, bits 15-0 right channel data:

R2 = end of buffer (address of first word beyond buffer)

R3 = flags:

bits 2 - 0 initial buffer content indicator:

0	data in buffer is invalid and must be overwritten
1	data has been converted from N-channel µ-law sound system, and is not all 0: It may be either overwritten, or mixed with new data produced by the handler, at the handler's option: Handlers should in general be user-configurable in respect of this behaviour
2	data in buffer is all 0: if handler would generate

	silent output, it may simply return: 3..
7	reserved values

bits 31 - 8 not yet defined — ignore

R4 = sample frequency at which data will be played, in units of 1/1024 Hz (e:g: for 20kHz, R4 would be 20000*1024 = 20480000).

On exit

R0-R10 may be corrupted

R11 ,R12, R13 must be preserved

Processor mode must be preserved

Interrupts

Interrupts may be enabled during execution of the handle:

Processor mode

Handler may be called in either IRO mode or SVC mode:

Notes

The size of the buffer passed to the handler is as determined by the Sound_Configure SWI, and is independent of the number of channels of Haw sound configured: RI as passed in to/out of Sound_Configure indicates how many sample-times the data in the buffer represents. Since with 16-bit linear stereo sound, the size of the data per sample-time is fixed at 32 bits, the size of the buffer is then just that number of words: This is different from the case with multi-channel Haw sound, where the buffer size passed to the channel handler is a function of how many channels are active: The linear sound handler is called to generate 16-bit stereo data before any oversampling is performed. The value passed in R4 will therefore be the same as would be returned by

Sound_SampleRate 1:

Sound_SampleRate (SWI &40146)

Determine/control sound sample rate

On entry

R0 = function code:

0	read number of sample rates, NSR
1	read current sample rate index and value
2	convert sample rate index to sample rate
3	set sample rate by index

R1 = parameter if required by function code

On exit

R0 preserved

R1, R2 = results as determined by function code

Interrupts (for all function codes)

Interrupt status is undefined
Fast interrupts are enabled

Processor mode (for all function codes)

Processor is in SVC mode

Re-entrancy (for all function codes)

Not defined

Use

See individual function code descriptions for further details: This SW1 must not be used unless 16-bit sound output is configured, as determined by a preceding call of Sound_Mode 0:

Sound_SampleRate 0 (SWI &40146)

Read number of available sample rates, NSR

On entry

R0 = 0 (function code)

On exit

R0 preserved

R1 = total number of available sample rates, NSR:

R2 preserved

Use

Available sample rates are indexed by numbers in the range I ..NSR, and increase with increasing index number: The number of available sample rates, and the assignment of index values to specific sample rates, is affected by the sound output hardware configuration:

Therefore this function code should in general be used before any manipulation of sample rates is performed; in particular, the SWIs Sound_SampleRate 2 and Sound_SampleRate 3 require that their respective parameters be in the range 1..NSR.

Notes

Future hardware or software developments might result in a change in the value of NSR for any given hardware configuration. For maximum portability and compatibility with possible future enhancements, sound applications should always fully determine the working configuration of the sound system before using it, and should not assume any particular value for NSR, or any particular binding of sample rates to index values (see Sound_SampleRate 2 and 3):

Sound_SampleRate 1 (SWI &40146)

Read current sample rate.

On entry

R0 = 1 (function code)

On exit

R0 = preserved

R1 = index of current sample rate

R2 = current sample rate value

Use

This call reads the current sample rate index; for convenience, the current sample rate value is also returned. The sample rate index is an integer in the range 1..NSR (as determined by Sound_SampleRate 0):

The sample rate value is measured in units of 1/1024 Hz: Hence, 20kHz (20,000 Hz), for example, would be reported as 20000*1024 = 20480000:

Sound_SampleRate 2 (SWI &40146)

Convert sample rate index to sample rate value:

On entry

R0 = 2 (function code)

R1 = sample rate index to be converted, in range 1.NSR

On exit

R0 preserved

R1 preserved

R2 = sample rate value corresponding to given index

Use

This call allows the set of available sample rates to be determined by repeated use of the SWI with varying values of R 1 : By setting RI successively to all values in the range 1..NSR, the full set of rates may be discovered; alternatively, if some particular sample rate is required, a "binary chop" algorithm may be used to reduce the cost of finding it (or the closest available rate, if an approximation is acceptable), since sample rates increase monotonically with increasing sample rate index: As usual, the returned sample rate value is measured in units of 1/1024 Hz:

Notes

The mapping of index numbers to sample rates is fixed for any given hardware & software configuration (i:e: with a particular set of versions of the sound system modules, and a particular sound system configuration setting in CMOS RAM) but is not guaranteed to remain constant over future hardware or software changes.

Sound_SampleRate 3 (SWI &40146)

Select sound sample rate

On entry

R0 = 3 (function code)

R1 = new sample rate index, in range 1 NSR

On exit

R0 preserved

R1 = previous sample rate index

R2 = previous sample rate value

Use

This SWI is provided to select a new sample rate by means of the index given in RI on call: This must be in the range 1..NSR. The previous values of sample rate index and associated value are returned in RI and R2 respectively:

Notes

Never assume a given index corresponds to a particular sample rate value; check first using Sound_SampleRate 2.

New CMOS RAM sound configuration options

Three CMOS RAM bits have been allocated to hold new sound hardware and software configuration options, particularly in support of the 16-bit output facilities: They are defined below:

CMOS RAM address

&84

Bit assignments

Bits 4-0 = other unrelated functions; must be preserved

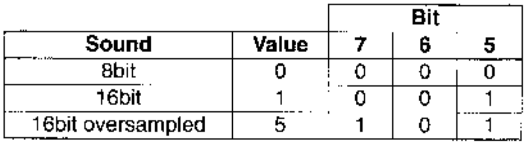
Bits 6-5 = 16-bit sound control, on VIDC20-based machines including Risc PC:

0	no 16-bit sound output; use standard R-law (default)
1	DAC clock is slave: 11:2896MHz external clock: selected standard VIDC20 rates or 44:1 kHz/{ 1,2,4} can be used
2	DAC clock is slave: no external clock: selected standard VIDC20 rates only
3	DAC clock is master: external clock must be used for 16-bit sound (suitable sound clock driver must be installed).

Bit 7 = automatic interpolation flag

0	do not automatically interpolate
1	automatically interpolate at sample rates <= 25 kHz to improve sound output quality.

The functions of bits 5 to 7 may be summarised as follows:



		Bit		
Sound	**Value**	**7**	**6**	**5**
8bit	0	0	0	0
16bit	1	0	0	1
16bit oversampled	5	1	0	1

Configuration can be set by:

```
*Configure SoundSystem 8bit I 16bit [oversampled] I <Value>
```

and read by:

```
*Status SoundSystem
```

The setting of bits 6-5 must be 0, on a machine without 16-bit sound hardware fitted, or 1 for the standard Acorn 16-bit sound upgrade: Bit 7 may be configured as required – refer to the description of oversampling under SWI Sound_Mode 1 (see page Sound_Mode 1 (SW1 &40144)29): Note that it is only effective when 16-bit sound output hardware is fitted; it is ignored otherwise:

Published by Acorn Computers Limited
Part number 0491,805
Issue 1, November 1994