NAME

sound, **pcm**, **snd** - FreeBSD PCM audio device infrastructure

SYNOPSIS

To compile this driver into the kernel, place the following line in your kernel configuration file:

```
device sound
```

DESCRIPTION

The **sound** driver is the main component of the FreeBSD sound system. It works in conjunction with a bridge device driver on supported devices and provides PCM audio record and playback once it attaches. Each bridge device driver supports a specific set of audio chipsets and needs to be enabled together with the **sound** driver. PCI and ISA PnP audio devices identify themselves so users are usually not required to add anything to */boot/device.hints*.

Some of the main features of the **sound** driver are: multichannel audio, per-application volume control, dynamic mixing through virtual sound channels, true full duplex operation, bit perfect audio, rate conversion and low latency modes.

The **sound** driver is enabled by default, along with several bridge device drivers. Those not enabled by default can be loaded during runtime with kldload(8) or during boot via loader.conf(5). The following bridge device drivers are available:

- snd_ai2s(4) (enabled by default on powerpc)
- snd_als4000(4)
- snd_atiixp(4)
- snd_cmi(4) (enabled by default on amd64, i386)
- snd_cs4281(4)
- snd_csa(4) (enabled by default on amd64, i386)
- snd_davbus(4) (enabled by default on powerpc)
- snd_emu10k1(4)
- snd_emu10kx(4) (enabled by default on amd64, i386)
- snd_envy24(4)
- snd_envy24ht(4)
- snd_es137x(4) (enabled by default on amd64, i386)
- snd_fm801(4)
- snd_hda(4) (enabled by default on amd64, i386)
- snd_hdspe(4)
- snd_ich(4) (enabled by default on amd64, i386)
- snd_maestro3(4)

- snd_neomagic(4)
- snd_solo(4)
- snd_spicds(4)
- snd_uaudio(4) (enabled by default on amd64, i386, powerpc)
- snd_via8233(4) (enabled by default on amd64, i386)
- snd_via82c686(4)
- snd_vibes(4)

Refer to the manual page for each bridge device driver for driver specific settings and information.

Legacy Hardware

For old legacy ISA cards, the driver looks for MSS cards at addresses 0x530 and 0x604. These values can be overridden in */boot/device.hints*. Non-PnP sound cards require the following lines in device.hints(5):

```
hint.pcm.0.at="isa"
hint.pcm.0.irq="5"
hint.pcm.0.drq="1"
hint.pcm.0.flags="0x0"
```

Apart from the usual parameters, the flags field is used to specify the secondary DMA channel (generally used for capture in full duplex cards). Flags are set to 0 for cards not using a secondary DMA channel, or to 0x10 + C to specify channel C.

Boot Variables

In general, the module *snd_foo* corresponds to **device snd_foo** and can be loaded by the boot loader(8) via loader.conf(5) or from the command line using the kldload(8) utility. Options which can be specified in */boot/loader.conf* include:

snd_driver_load ("NO") If set to "YES", this option loads all available drivers.

snd_hda_load ("NO") If set to "YES", only the Intel High Definition Audio bridge device driver and dependent modules will be loaded.

snd_foo_load ("NO") If set to "YES", load driver for card/chipset foo.

To define default values for the different mixer channels, set the channel to the preferred value using hints, e.g.: *hint.pcm.0.line*="0". This will mute the input channel per default.

Multichannel Audio

Multichannel audio, popularly referred to as "surround sound" is supported and enabled by default. The FreeBSD multichannel matrix processor supports up to 18 interleaved channels, but the limit is currently set to 8 channels (as commonly used for 7.1 surround sound). The internal matrix mapping can handle reduction, expansion or re-routing of channels. This provides a base interface for related multichannel **ioctl**() support. Multichannel audio works both with and without VCHANs.

Most bridge device drivers are still missing multichannel matrixing support, but in most cases this should be trivial to implement. Use the *dev.pcm.%d.[play|rec].vchanformat* sysctl(8) to adjust the number of channels used. The current multichannel interleaved structure and arrangement was implemented by inspecting various popular UNIX applications. There were no single standard, so much care has been taken to try to satisfy each possible scenario, despite the fact that each application has its own conflicting standard.

EQ

The Parametric Software Equalizer (EQ) enables the use of "tone" controls (bass and treble). Commonly used for ear-candy or frequency compensation due to the vast difference in hardware quality. EQ is disabled by default, but can be enabled with the *hint.pcm.%d.eq* tunable.

VCHANs

Each device can optionally support more playback and recording channels than physical hardware provides by using "virtual channels" or VCHANs. VCHAN options can be configured via the sysctl(8) interface but can only be manipulated while the device is inactive.

VPC

FreeBSD supports independent and individual volume controls for each active application, without touching the master **sound** volume. This is sometimes referred to as Volume Per Channel (VPC). The VPC feature is enabled by default.

Loader Tunables

The following loader tunables are used to set driver configuration at the loader(8) prompt before booting the kernel, or they can be stored in */boot/loader.conf* in order to automatically set them before booting the kernel. It is also possible to use kenv(1) to change these tunables before loading the **sound** driver. The following tunables can not be changed during runtime using sysctl(8).

hint.pcm.%d.eq

Set to 1 or 0 to explicitly enable (1) or disable (0) the equalizer. Requires a driver reload if changed. Enabling this will make bass and treble controls appear in mixer applications. This tunable is undefined by default. Equalizing is disabled by default.

hint.pcm.%d.vpc

Set to 1 or 0 to explicitly enable (1) or disable (0) the VPC feature. This tunable is undefined by default. VPC is however enabled by default.

Runtime Configuration

There are a number of sysctl(8) variables available which can be modified during runtime. These values can also be stored in */etc/sysctl.conf* in order to automatically set them during the boot process. *hw.snd.** are global settings and *dev.pcm.** are device specific.

hw.snd.compat_linux_mmap

Linux mmap(2) compatibility. The following values are supported (default is 0):

-1 Force disabling/denying PROT_EXEC mmap(2) requests.

0 Auto detect proc/ABI type, allow mmap(2) for Linux applications, and deny for everything else.

1 Always allow PROT_EXEC page mappings.

hw.snd.default_auto

Automatically assign the default sound unit. The following values are supported (default is 1):

0 Do not assign the default sound unit automatically.

1 Use the best available sound device based on playing and recording capabilities of the device.

2 Use the most recently attached device.

hw.snd.default_unit

Default sound card for systems with multiple sound cards. When using devfs(5), the default device for */dev/dsp*. Equivalent to a symlink from */dev/dsp* to */dev/dsp${hw.snd.default_unit}*.

hw.snd.feeder_eq_exact_rate

Only certain rates are allowed for precise processing. The default behavior is however to allow sloppy processing for all rates, even the unsupported ones. Enable to toggle this requirement and only allow processing for supported rates.

hw.snd.feeder_rate_max

Maximum allowable sample rate.

hw.snd.feeder_rate_min

Minimum allowable sample rate.

hw.snd.feeder_rate_polyphase_max

Adjust to set the maximum number of allowed polyphase entries during the process of building resampling filters. Disabling polyphase resampling has the benefit of reducing memory usage, at the expense of slower and lower quality conversion. Only applicable when the SINC interpolator is used. Default value is 183040. Set to 0 to disable polyphase resampling.

hw.snd.feeder_rate_quality

Sample rate converter quality. Default value is 1, linear interpolation. Available options include:

0 Zero Order Hold, ZOH. Very fast, but with poor quality.

1 Linear interpolation. Fast, quality is subject to personal preference. Technically the quality is poor however, due to the lack of anti-aliasing filtering.

2 Bandlimited SINC interpolator. Implements polyphase banking to boost the conversion speed, at the cost of memory usage, with multiple high quality polynomial interpolators to improve the conversion accuracy. 100% fixed point, 64bit accumulator with 32bit coefficients and high precision sample buffering. Quality values are 100dB stopband, 8 taps and 85% bandwidth.

3 Continuation of the bandlimited SINC interpolator, with 100dB stopband, 36 taps and 90% bandwidth as quality values.

4 Continuation of the bandlimited SINC interprolator, with 100dB stopband, 164 taps and 97% bandwidth as quality values.

hw.snd.feeder_rate_round

Sample rate rounding threshold, to avoid large prime division at the cost of accuracy. All requested sample rates will be rounded to the nearest threshold value. Possible values range between 0 (disabled) and 500. Default is 25.

hw.snd.latency

Configure the buffering latency. Only affects applications that do not explicitly request blocksize / fragments. This tunable provides finer granularity than the *hw.snd.latency_profile* tunable. Possible values range between 0 (lowest latency) and 10 (highest latency).

hw.snd.latency_profile

Define sets of buffering latency conversion tables for the *hw.snd.latency* tunable. A value of 0 will use a low and aggressive latency profile which can result in possible underruns if the application cannot keep up with a rapid irq rate, especially during high workload. The default value is 1, which is considered a moderate/safe latency profile.

hw.snd.maxautovchans

Global VCHAN setting that only affects devices with at least one playback or recording channel available. The sound system will dynamically create up to this many VCHANs. Set to "0" if no VCHANs are desired. Maximum value is 256.

hw.snd.report_soft_formats

Controls the internal format conversion if it is available transparently to the application software. When disabled or not available, the application will only be able to select formats the device natively supports.

hw.snd.report_soft_matrix

Enable seamless channel matrixing even if the hardware does not support it. Makes it possible to play multichannel streams even with a simple stereo sound card.

hw.snd.verbose

Level of verbosity for the */dev/sndstat* device. Higher values include more output and the highest level, four, should be used when reporting problems. Other options include:

0 Installed devices and their allocated bus resources.

1 The number of playback, record, virtual channels, and flags per device.

2 Channel information per device including the channel’s current format, speed, and pseudo device statistics such as buffer overruns and buffer underruns.

3 File names and versions of the currently loaded sound modules.

4 Various messages intended for debugging.

hw.snd.vpc_0db

Default value for **sound** volume. Increase to give more room for attenuation control. Decrease for more amplification, with the possible cost of sound clipping.

hw.snd.vpc_autoreset

When a channel is closed the channel volume will be reset to 0db. This means that any changes

to the volume will be lost. Enabling this will preserve the volume, at the cost of possible confusion when applications tries to re-open the same device.

hw.snd.vpc_mixer_bypass
The recommended way to use the VPC feature is to teach applications to use the correct **ioctl**(): SNDCTL_DSP_GETPLAYVOL, SNDCTL_DSP_SETPLAYVOL, SNDCTL_DSP_SETRECVOL, SNDCTL_DSP_SETRECVOL. This is however not always possible. Enable this to allow applications to use their own existing mixer logic to control their own channel volume.

hw.snd.vpc_reset
Enable to restore all channel volumes back to the default value of 0db.

dev.pcm.%d.bitperfect
Enable or disable bitperfect mode. When enabled, channels will skip all dsp processing, such as channel matrixing, rate converting and equalizing. The pure **sound** stream will be fed directly to the hardware. If VCHANs are enabled, the bitperfect mode will use the VCHAN format/rate as the definitive format/rate target. The recommended way to use bitperfect mode is to disable VCHANs and enable this sysctl. Default is disabled.

dev.pcm.%d.[play|rec].vchans
The current number of VCHANs allocated per device. This can be set to preallocate a certain number of VCHANs. Setting this value to "0" will disable VCHANs for this device.

dev.pcm.%d.[play|rec].vchanformat
Format for VCHAN mixing. All playback paths will be converted to this format before the mixing process begins. By default only 2 channels are enabled. Available options include:

s16le:1.0
Mono.

s16le:2.0
Stereo, 2 channels (left, right).

s16le:2.1
3 channels (left, right, LFE).

s16le:3.0
3 channels (left, right, rear center).

s16le:4.0
Quadraphonic, 4 channels (front/rear left and right).

s16le:4.1
5 channels (4.0 + LFE).

s16le:5.0
5 channels (4.0 + center).

s16le:5.1
6 channels (4.0 + center + LFE).

s16le:6.0
6 channels (4.0 + front/rear center).

s16le:6.1
7 channels (6.0 + LFE).

s16le:7.1
8 channels (4.0 + center + LFE + left and right side).

dev.pcm.%d.[play|rec].vchanmode
VCHAN format/rate selection. Available options include:

fixed
Channel mixing is done using fixed format/rate. Advanced operations such as digital passthrough will not work. Can be considered as a "legacy" mode. This is the default mode for hardware channels which lack support for digital formats.

passthrough
Channel mixing is done using fixed format/rate, but advanced operations such as digital passthrough also work. All channels will produce sound as usual until a digital format playback is requested. When this happens all other channels will be muted and the latest incoming digital format will be allowed to pass through undisturbed. Multiple concurrent digital streams are supported, but the latest stream will take precedence and mute all other streams.

adaptive
Works like the "passthrough" mode, but is a bit smarter, especially for multiple **sound** channels with different format/rate. When a new channel is about to start, the entire list of

virtual channels will be scanned, and the channel with the best format/rate (usually the highest/biggest) will be selected. This ensures that mixing quality depends on the best channel. The downside is that the hardware DMA mode needs to be restarted, which may cause annoying pops or clicks.

dev.pcm.%d.[play|rec].vchanrate
Sample rate speed for VCHAN mixing. All playback paths will be converted to this sample rate before the mixing process begins.

dev.pcm.%d.polling
Experimental polling mode support where the driver operates by querying the device state on each tick using a callout(9) mechanism. Disabled by default and currently only available for a few device drivers.

Recording Channels

On devices that have more than one recording source (ie: mic and line), there is a corresponding */dev/dsp%d.r%d* device. The mixer(8) utility can be used to start and stop recording from an specific device.

Statistics

Channel statistics are only kept while the device is open. So with situations involving overruns and underruns, consider the output while the errant application is open and running.

IOCTL Support

The driver supports most of the OSS **ioctl**() functions, and most applications work unmodified. A few differences exist, while memory mapped playback is supported natively and in Linux emulation, memory mapped recording is not due to VM system design. As a consequence, some applications may need to be recompiled with a slightly modified audio module. See <*sys/soundcard.h*> for a complete list of the supported **ioctl**() functions.

FILES

The **sound** drivers may create the following device nodes:

/dev/audio%d.%d Sparc-compatible audio device.
/dev/dsp%d.%d Digitized voice device.
/dev/dspW%d.%d
Like */dev/dsp*, but 16 bits per sample.
/dev/dsp%d.p%d Playback channel.
/dev/dsp%d.r%d Record channel.
/dev/dsp%d.vp%d

Virtual playback channel.
/dev/dsp%d.vr%d Virtual recording channel.
/dev/sndstat Current **sound** status, including all channels and drivers.

The first number in the device node represents the unit number of the **sound** device. All **sound** devices are listed in */dev/sndstat*. Additional messages are sometimes recorded when the device is probed and attached, these messages can be viewed with the dmesg(8) utility.

The above device nodes are only created on demand through the dynamic devfs(5) clone handler. Users are strongly discouraged to access them directly. For specific sound card access, please instead use */dev/dsp* or */dev/dsp%d*.

EXAMPLES

Use the sound metadriver to load all **sound** bridge device drivers at once (for example if it is unclear which the correct driver to use is):

```
kldload snd_driver
```

Load a specific bridge device driver, in this case the Intel High Definition Audio driver:

```
kldload snd_hda
```

Check the status of all detected **sound** devices:

```
cat /dev/sndstat
```

Change the default sound device, in this case to the second device. This is handy if there are multiple **sound** devices available:

```
sysctl hw.snd.default_unit=1
```

DIAGNOSTICS

pcm%d:play:%d:dsp%d.p%d: play interrupt timeout, channel dead The hardware does not generate interrupts to serve incoming (play) or outgoing (record) data.

unsupported subdevice XX A device node is not created properly.

SEE ALSO

snd_ai2s(4), snd_als4000(4), snd_atiixp(4), snd_cmi(4), snd_cs4281(4), snd_csa(4), snd_davbus(4), snd_emu10k1(4), snd_emu10kx(4), snd_envy24(4), snd_envy24ht(4), snd_es137x(4), snd_fm801(4),

snd_hda(4), snd_hdspe(4), snd_ich(4), snd_maestro3(4), snd_neomagic(4), snd_solo(4), snd_spicds(4), snd_t4dwave(4), snd_uaudio(4), snd_via8233(4), snd_via82c686(4), snd_vibes(4), devfs(5), device.hints(5), loader.conf(5), dmesg(8), kldload(8), mixer(8), sysctl(8)

Cookbook formulae for audio EQ biquad filter coefficients (Audio-EQ-Cookbook.txt), by Robert Bristow-Johnson, https://www.musicdsp.org/en/latest/Filters/197-rbj-audio-eq-cookbook.html.

Julius O'Smith's Digital Audio Resampling, http://ccrma.stanford.edu/~jos/resample/.

Polynomial Interpolators for High-Quality Resampling of Oversampled Audio, by Olli Niemitalo, http://yehar.com/blog/wp-content/uploads/2009/08/deip.pdf.

The OSS API, http://www.opensound.com/pguide/oss.pdf.

HISTORY

The **sound** device driver first appeared in FreeBSD 2.2.6 as **pcm**, written by Luigi Rizzo. It was later rewritten in FreeBSD 4.0 by Cameron Grant. The API evolved from the VOXWARE standard which later became OSS standard.

AUTHORS

Luigi Rizzo <*luigi@iet.unipi.it*> initially wrote the **pcm** device driver and this manual page. Cameron Grant <*gandalf@vilnya.demon.co.uk*> later revised the device driver for FreeBSD 4.0. Seigo Tanimura <*tanimura@r.dl.itc.u-tokyo.ac.jp*> revised this manual page. It was then rewritten for FreeBSD 5.2.

BUGS

Some features of your sound card (e.g., global volume control) might not be supported on all devices.