Linux ALSA sound notes

Updated: 2008-01-23
Created: 2004-08-14
Licensing and disclaimer of warranty

This document is an incomplete draft.

Section menu

Top
Structure of ALSA and terminology
Configuring applications for ALSA
Common ALSA tasks
OSS emulation
Sharing a device
Issues with popular chipsets
Potential sound quality issue list
Troubleshooting
References

These notes are about *configuring* the *Linux ALSA* sound subsystem components (in particular the ALSA drivers and the ALSA library), and not about *installing* the various bits of ALSA.

These notes have some limitations, mostly due to my limited experiences, and my reluctance to include hearsay (but I do sometimes):

- The are very incomplete, both as a whole and each part. There are missing sections, links, paragraphs.
- The text has not been proofread or edited or checked.
- There is little if any mention of *MIDI*, sequencer or *IEC958*/*SPDIF* or digital or *AC3* etc. configuration and issues, as I don't have any of these. The focus is on stereo and 4 speaker surround, and AC3 etc. software conversion to these, if any.

Unless you are a *UNIX*/Linux developer don't even try to compile and install the various ALSA components yourself, unless you feel really lucky: just use the precompiled packages provided by your distribution for the kernel, kernel drivers, libraries and applications. You have been warned.

Structure of ALSA and terminology

Takashi Iwai has a nice diagram that illustrates the structure of ALSA.

- The hardware card level

- The kernel module level
- The `/proc/asound` kernel interface
- The `/dev/snd` device interface
- The ALSA library level

The hardware level

Sound cards come in a wide variety of types, with rather different internal organization.
All however are based on a *'chipset'*, and ALSA drivers are designed for chipsets, or even for families of similar chipsets, not specific cards.
However most chipsets are customizable, and when used in a card some of their capabilities may be omitted or somewhat modified.
Cards usually have at least some of these logical components:

- An interface to the bus (usually *PCI*) that uses an *'IRQ'*, some IO *'ports'* and possibly a memory range.
- Input ADC devices, that take analog sound signals and converts them to sound data.
- Output DAC devices, that take sound data and convert them to analog sound signals.
- Controls, which allow setting options on how the card operates.
- A mixer, which controls each devices for example as to volume, and routes among them.
- A MIDI device, which allows control of an external MIDI synthetizer.
- A sequencer, which is a builtin MIDI synthetizer.
- An output amplifier, which raises the strength of analog sound signals output by the card. The output amplifier as a rule is not very powerful, and is often not very good.
- An input amplifier, which raises the strength of microphone sound signals. The input amplifier, usually called `Mic boost` in the mixer, is often of terrible quality.

The kernel module level

Some important design principles of the ALSA *'kernel'* modules:

- The most important important aspect of the ALSA kernel modules is that they are designed to offer an interface that corresponds to that of the hardware, and no more. This was chosen so that the kernel modules be kept simple.
- The kernel modules are designed to provide both an operational interface via entries in the `/dev/` tree, and a status and configuration interface via entries in the `/proc/asound/` tree.
- The operational interface in `/dev/` contains three main types of devices, *'PCM'* devices for recording or playing digitized sound samples, *'CTL'* devices that allow manipulating the internal mixer and routing of the card, and *'MIDI'* devices to control the MIDI port of the card if any. There may be also *'sequencer'* devices if the card has a builtin sound synthesizer, and *'timer'* devices to help in using the sequencer.
- The kernel modules have been designed so that they offer a similar interface if they are for similar cards. In particular the names of controls have been deliberately kept similar, so that the master volume control, for example, is always called `Master Volume`.

A card may have multiple PCM devices, but the main one is always PCM device 0, and as a rule it is for analog multichannel sound. Additional PCM devices may exist for digital (IEC958, SPDIF) sound (usually device 2), or for additional channels (usually device 1), for example for the side speakers in a 7.1 card.

PCM devices come in two varieties: *'output'* and *'input'*. MIDI devices are output only, and so are sequencer devices. Whether a device is input or output is from the point of view of the *process* using the ALSA driver. An output device is one to which the process can write/send sound samples, and an input one is one from which it can read/receive sound samples.
Then the sound card has input or output *'sockets'*, and the *'mixer'*, which is controlled via the *'CTL'* device, routes sound samples from/to the PCM (or MIDI, sequencer) devices and the sockets, or among the devices.
Sound samples are written from a process to an output device to the mixer, and then usually routed to an output socket, and are read by a process from an input device usually associated with an input socket, but sometimes associated with the mixer.
Mixers therefore contain input and output routes; but they also contain copy (loopback) routes, which duplicate sound samples from a device to another; for example some copy routes take the sound samples written to an output device (or socket) and copy them to an input device, for example allowing to record the sound output of a process; some copy routes take the sound samples read from an input device (or socket) and copy them to an output device, for example allowing monitoring of what is being recorded via a microphone.
Usually input devices have associated *'capture'* controls in the mixer, and output devices have *'playback'* controls. Copy routes are implicit in the presence of both capture and playback controls for the same device; if an output device has both capture and playback mixer controls, there is an output-to-input copy route for it, and viceversa.

Names and functions of controls

When queried, the control interface of an ALSA card can return a list of controls for it. The command `amixer scontrols` prints a simplified list, and `amixer controls` prints the full list, and `amixer contents` prints the full lists with the current and possible values of each control.
There are three types of controls:

- *'Playback'* controls, which are associated with an output device and can be muted or unmuted, usually also have a volume level as well as the unmute *'switch'*, and are associated with output or copy (input-to-output) routes.
- *'Capture'* controls, which are associated with an input device, can be set to capture or not to capture, usually also have a volume level as well as the capture *'switch'*, and that are associated with input or copy (output-to-input) routes.
- Feature controls, that don't correspond to any devices, but drive instead some features of the card or mixer. Most of these are just a *'switch'* to enable or disable the feature. Some also have a level, to select different values of the feature.

Cards differ greatly in the availability of devices and features, and of device or feature controls. For example some have a distinct copy route for every input or output device, some have only one (often called `Capture`) that can be used only for one selected device; some can record from multiple input devices at the same time, some only from one selected input device. Some have volume controls for the copy routes independent of the volume of the subdevice being looped back, some don't.
For example a microphone socket may correspond to an input device with a capture control, and a copy route (of the input-to-output type) with a playback control. The capture control will have a capture switch and a volume level to enable recording and set the volume of the recording; the playback control will have an unmute switch to enable monitoring recording, and a volume level to set the monitoring volume.
As an example of a feature, some cards have a 3D spatializer; this can be represented by three controls, one with just a switch to enable or disable the spatializer, and two level controls, where the level of one control is the 3D depth, and the level of the other control is the degree of spatialization.

The most typical example of a feature control however is the `Master Volume` control; this allows controlling the internal amplifier feature of the card.

Channel assignments

Output devices don't necessarily allow playing aribtrary types of sound samples; most are limited in both the range of frequencies and formats of the sound samples, as well as in the number of channels in those samples.

When multichannel sound samples are written to an ALSA output device, every ALSA driver uses the same logical (in the sample) to physical (as to sockets) channel assignments.

This sound samples must contain these logical channel assignments for the samples to be correctly positioned by the drivers for stereo and 4.0 surround playback:

- 0: front left
- 1: front right
- 2: rear left
- 3: rear right

With 4.1 surround, channel 4 is the LFE, also called subwoofer, channel.

With 5.0 surround, channel 4 is the center channel.

With 5.1 surround, channel 4 is center, and channel 5 is the LFE/subwoofer.

With 7.1 surround, channel 6 is side left, and 7 is side right.

The assignment of sound channels to device channels might be reshaped, if appropriate, by using the `multi` plugin in `copy` mode, or as appropriate for lower or higher numbers of channels.

The `/proc/asound` kernel interface

For now just this (slightly edited) snippet from Takashi Iwai:

```
/proc/asound/cards (RO)
 the list of registered cards

/proc/asound/version (RO)
 the version and date the driver was built

/proc/asound/devices (RO)
 the list of registered ALSA devices (major=116)

/proc/asound/hwdep (RO)
 the list of hwdep (hardware dependent) controls

/proc/asound/meminfo (RO)
 memory usage information
        this proc file appears only when you build the alsa drivers
        with memory debug (or full) option so the file shows the
        currently allocated memories on kernel space.

/proc/asound/pcm (RO)
 the list of allocated pcm streams
```

```
/proc/asound/dev/
 the directory containing device files.  device files are created
 dynamically; in the case without devfs, this directory is usually
 linked to /dev/snd/

/proc/asound/oss/
 the directory containing info about oss emulation
```

```
/proc/asound/seq/
 the directory containing info about sequencer

/proc/asound/cardX/ (X = 0-7)
 the card-specific directory
```

```
/proc/asound/oss/
 info about OSS emulation
 the contents of the files under this directory are changed dynamically.
 when no oss emulation modules (snd-pcm-oss, snd-mixer-oss) are loaded,
 no pcm nor mixer devices will be listed.

/proc/asound/oss/devices (RO)
 the list of devices already registered

/proc/asound/oss/sndstat (RO)
 /dev/sndstat compatible list
```

```
/proc/asound/cardX/
  card-specific proc files

id (RO)
 the id string of the card

ac97#? (RO)
 AC97 codec information

ac97#?regs (RO)
 (printable) register dump

midi? (RO)
 the current of input/output on the
 rawmidi device

pcm?p
 the directory of the given pcm playback stream
pcm?c
 the directory of the given pcm capture stream

pcm stream information

pcm??/info (RO)
 the pcm stream general info (card, device, name, etc.)

pcm??/oss (RO)
 oss emulation info (shown only when the pcm is opened
 as an oss device).

pcm??/sub?
 the substream information directory

pcm??/sub?/info (RO)
 the pcm substream general info (card, device, name, etc.)

pcm??/sub?/status (RO)
 the current of the given pcm substream
 (status, position, delay, tick time, etc.)

pcm??/sub?/hw_params (RO)
 hw_params set-up on the substream
 (buffer size, format, etc.)

pcm??/sub?/sw_params (RO)
 sw_params set-up on the substream
 (threshold, etc.)

pcm??/sub?/prealloc (RW)
 the number of pre-allocated buffer size in kb.
 you can specify the buffer size by writing to this proc file:

 # echo 128 > /proc/asound/card0/pcm0p/sub0/prealloc
```

```
to allocate 128kbyte for playback, substream #0, stream #0
on the card #0.
```

The `/dev/snd` device interface

For now just this snippet from Takashi Iwai:

```
device files under /proc/asound/dev/ (or /dev/snd/)
generally the file is named as aaaCxDy,
where aaa is the service name,
x the card number (0-7),
y the device number (0-).

controlC? control devices (i.e. mixer, etc.)
hwC?D?  hwdep devices
midiC?D? rawmidi devices
pcmC?D?p pcm playback devices
pcmC?D?c pcm capture devices
seq sequencer device
timer timer device
```

The ALSA library level

- Internal data flow
- `/usr/share/alsa/alsa.conf`, `/usr/share/alsa/`, `/etc/asound.conf` and `~/.asoundrc`.
- Plugins
- Device names
- Environment variables
- ALSA Lisp sems to be a full Lisp implementation that is related to ALSA, and seems to be related to `asound.conf`.

`asound.conf`

Basic syntax, macros, external native code functions, runtime execution hooks.
This document comes with an extensive, generic, commented sample `asound.conf`.

Plugins

Plugins documentation.

Device Names

PCM device names.
Some aliases are conventional, in that their use is part of convention:

- `dsp0` is used by XMMS and the `aoss` script, unless changed by using the `ALSA_OSS_PCM_DEVICE` environment variable.

- `default` is used if no specific device name is used by an application other than XMMS and `aoss`.

From `/usr/share/alsa/alsa.conf` it looks like there are the following classes of device names:

- `defaults`:
- `pcm`:
- `ctl`:
- `rawmidi`:
- `seq`:
- `hwdep`:
- `timer_query`:
- `timer`:

Defaults

NAME	Default value	Notes
defaults.ctl.card	0	
defaults.pcm.card	0	
defaults.pcm.device	0	
defaults.pcm.subdevice	-1	
defaults.pcm.front.card	defaults.pcm.card	
defaults.pcm.front.device	defaults.pcm.device	
defaults.pcm.rear.card	defaults.pcm.card	
defaults.pcm.rear.device	defaults.pcm.device	
defaults.pcm.center_lfe.card	defaults.pcm.card	
defaults.pcm.center_lfe.device	defaults.pcm.device	
defaults.pcm.surround40.card	defaults.pcm.card	
defaults.pcm.surround40.device	defaults.pcm.device	
defaults.pcm.surround41.card	defaults.pcm.card	
defaults.pcm.surround41.device	defaults.pcm.device	
defaults.pcm.surround50.card	defaults.pcm.card	
defaults.pcm.surround50.device	defaults.pcm.device	
defaults.pcm.surround51.card	defaults.pcm.card	
defaults.pcm.surround51.device	defaults.pcm.device	
defaults.pcm.iec958.card	defaults.pcm.card	
defaults.pcm.iec958.device	defaults.pcm.device	
defaults.rawmidi.card	0	
defaults.rawmidi.device	0	
defaults.rawmidi.subdevice	-1	
defaults.hwdep.card	0	
defaults.hwdep.device	0	
defaults.timer.class	2	
defaults.timer.sclass	0	
defaults.timer.card	0	
defaults.timer.device	0	
defaults.timer.subdevice	0	

and the following default device names:

- These PCM names are shorthands for the similarly named plugin on the current card:
  - `pcm.hw`:
  - `pcm.plughw`:
  - `pcm.plug`:
  - `pcm.dmix`:
  - `pcm.dsnoop`:
  - `pcm.shm`:
  - `pcm.tee`:
  - `pcm.file`:
  - `pcm.null`:

  and `pcm.default` is the default PCm device and it is mostly equivalent to `pcm.plug`, so as to provide by default rate and format conversion. It contains references to the `ALSA_CARD`, `ALSA_PCM_CARD` and `ALSA_PCM_DEVICE` environment variables.

  These PCM names are shorthands for common device names:
  - `pcm.cards` (`cards.pcm`)
  - `pcm.iec958` and `pcm.spdif` (`cards.pcm.iec958`) define the default IEC958/SPDIF output device.
  - `pcm.front` (`cards.pcm.front`) `pcm.rear` (`cards.pcm.rear`) `pcm.center_lfe` (`cards.pcm.center_lfe`) name the front, read and center (or subwoofer) channel groups.
  - `pcm.surround40` (`cards.pcm.surround40`), `pcm.surround41` (`cards.pcm.surround41`), `pcm.surround50` (`cards.pcm.surround50`), `pcm.surround51` (`cards.pcm.surround51`) define the 4.0/4.1/5.0/5.1 channels output devices.
-
  - `ctl.hw`:
  - `ctl.shm`:
  - `ctl.default`:
-
  - `rawmidi.hw`:
  - `rawmidi.default`:
  - `rawmidi.virtual`:
-
  - `seq.default`:
  - `seq.hw`:
-
  - `hwdep.hw`:
  - `hwdep.default`:
-
  - `timer_query.hw`:
  - `timer_query.default`:
-
  - `timer.hw`:
  - `timer.default`:

Environment variables

From the config files:

- ALSA_CARD
- ALSA_CTL_CARD
- ALSA_PCM_CARD
- ALSA_PCM_DEVICE
- ALSA_HWDEP_CARD
- ALSA_HWDEP_DEVICE
- ALSA_IEC958_CARD
- ALSA_IEC958_DEVICE
- ALSA_RAWMIDI_CARD
- ALSA_RAWMIDI_DEVICE
- ALSA_FRONT_CARD
- ALSA_FRONT_DEVICE
- ALSA_CENTER_LFE_CARD
- ALSA_CENTER_LFE_DEVICE
- ALSA_REAR_CARD
- ALSA_REAR_DEVICE
- ALSA_SURROUND40_CARD
- ALSA_SURROUND40_DEVICE
- ALSA_SURROUND41_CARD
- ALSA_SURROUND41_DEVICE
- ALSA_SURROUND50_CARD
- ALSA_SURROUND50_DEVICE
- ALSA_SURROUND51_CARD
- ALSA_SURROUND51_DEVICE

From libasound.so:

- ALSA_CONFIG_PATH
- ALSA_ORDINARY_MIXER

From aoss:

- ALSA_OSS_PCM_DEVICE
- ALSA_OSS_DEBUG

Configuring applications for ALSA

General principles:

- If the application supports several sound systems, usually choose ALSA instead of *OSS*, or even a sound *'daemon'* like *aRts* or *ESoundD*: even if ALSA has an OSS compatibility mode it is more limited than the native ALSA API, and the ALSA library supports software sharing and rate and format conversion, thus making sound servers redundant inasmuch they are used for those purposes.
- If an application only supports OSS output, and there are problems, check the OSS emulation notes.
- if your applications supports OSS or a sound daemon, but not yet the ALSA 0.9/1 API, choose the sound daemon over OSS, as the sound daemon can be configured to use ALSA.

- Of the sound dæmons, such as aRts and ESoundD or *JACK* usually aRts and EsoundD are used only to provide services already provided by the ALSA library, so they should perhaps be disabled. But JACK is often used with a lot of cute sound processing plugins, so if you use them, keep using JACK.
- Using IEC958/SPDIF output, or direct DVD AC3/... output, is not trivial on many cards. Look carefully for card specific instructions on how to enable this.

There are some other notes here and here.

XMMS

With XMMS, download the ALSA output *'plugin'*, then use `CTRL-P` to get the preferences panel, and in `Audio I/O Plugins` select the `Output Plugin` called `ALSA` *1.x.y* `output plugin [libALSA.so]`.
In order to select a particular device click on `Configure` once the ALSA plugin has been selected, and either select a particular audio device, or type a name, in the `Audio device:` field. Remember to select the corresponding `Mixer card` number too.

libao

`libao` is a library used by many sound applications like for example *mpg321*. The default sound system for it can be configured by creating or editing the file `/etc/libao.conf` and ensuring it contains the line `default_driver=alsa09`.

OpenAL

OpenAL is a library used by many sound applications, in particular games. There are many, mostly buggy, versions of it floating around, some of which just don't support ALSA. Recent ones support ALSA, and there is a version that supports correctly positional/3D sound (this version may have been merged into newer versions of the official OpenAL sources, but I am not sure).
Its defaults are configured by the file `/etc/openalrc`, which is not awesomely documented, but there is something in the Info file for OpenAL (`info openal`) in the chapter *«The openal configuration file»*, and here is a suitable example:

```
(define devices                 '(alsa sdl arts esd native null))

(define alsa-device             "dsp0")
(define speaker-num             2)
;(define sampling-rate          22050)
```

SDL

The SDL library is a simple portability framework often used in games, and part of the framework is an audio portability layer. This is configured using environment variables. The relevant ones are `SDL_AUDIODRIVER` which should be set to `alsa`, and `AUDIODEV` or `SDL_PATH_DSP` which should be set to the name of the ALSA audio device, usually `default` or `dsp0`.

mpg321 or the proprietary *mpg123*

These have two parameters to select the sound system and device, `-o alsa09` select the ALSA 0.9.*x*/1.*x* sound system API, and `-a` *dev* selects the device. As to devices try `-a plughw:0,0` or `-a dsp0` for example. Since both use `libao` for sound output you can configure the default sound system as in the `libao` notes.

MPlayer

Configuring MPlayer depends on whether you use the command line version or one of the many GUI interfaces like `gmplayer`.
With the command line version, use option `-ao alsa:device=`*device* to select an ALSA device, usually as `-ao alsa:device=dsp0`.
To enable AC3 software processing there are some partly erroneous instructions in the MPlayer documentation that state that surround sound can only be obtained with OSS. Probably it is better support with ALSA, selecting a multiple channel device, as in `-channels 4 -ao alsa:device=surround40` or `-channels 6 -ao alsa:device=surround51` if supported by your card.

With some special options it is possible on some sound card to do AC3 pass-through with ALSA.

XINE

Command line `-A alsa` to select the ALSA output plugins, or equivalently set `audio.driver:alsa` in `~/.xine/config` and append `:device=`*dmixo* to select a specific output device. Also check a number of settings beginning with `audio.device.alsa_` in `~/.xine/config`.

VLC

The command line option `--aout alsa` to select ALSA output, and `--alsadev` *surround51* to select a specific device.

Mozilla, Firefox

Both Mozilla and Firefox only support OSS, so they must be used with OSS ALSA emulation (or perhaps the `aoss` wrapper). Similarly for most Mozilla and Firefox plugins.

Java

TODO

Flash

TODO

Icecast

From their «*Available input modules*» page, an example (the details on that page):

```
<input>
  <module>alsa</module>
  <param name="rate">44100</param>
  <param name="channels">2</param>
  <param name="device">plughw:0,0</param>
  <param name="periods">2</param>
  <param name="buffer-time">500</param>
  <param name="metadata">1</param>
  <param name="metadatafilename">/home/ices/metadata</param>
</input>
```

aRts

In theory, since ALSA supports sharing via the library even with cards that don't have hardware sharing or resampling or format conversion, a sound server like aRts is not needed with ALSA. But it may still be useful with older applications that have aRts, OSS or ALSA 0.5 output code, but haven't been yet updated to use the ALSA 0.9 or ALSA 1 API.
The recent versions of the aRts daemon default to the ALSA API, or they can be made to use it explicitly by using the `-a alsa` option to `artsd`, and use the `-D` to select a specific output device, if you are starting `artsd` on its own.
Usually however aRts is started within and by KDE, in which case you can configure it by opening `Control Center`, `Sound & Multimedia` and `Sound System`. Having done this if you wan to run aRts you click on `Enable the sound system` in the `General` tab. To configure aRts click on the `Hardware tab`. Under `Select the audio device` choose `Advanced Linux Sound Archictecture`. Normally you can enable `Full duplex` as ALSA supports it, and most cards allow it.
It is often useful, especially with recent chipsets that internally only support 48000Hz data, to click on `Use custom sampling rate` and select `48000Hz`, probably even if the ALSA library is configured to support resampling itself.
To select a specific output device, click on `Override device location` and put the ALSA device name (for example `dsp0` or `hw:0,0`) in the corresponding field. With cards that do not support hardware mixing make sure to specify the virtual PCM device name that uses the `asym`, `dsnoop` and `dmix` plugins. Usually this should be called something like `dsp0` for the same of programs that use that name by default, so try that.

EsoundD

In theory, since ALSA supports sharing via the library even with cards that don't have hardware sharing or resampling or format conversion, a sound server like EsoundD is not needed with ALSA. However like for aRts some applications depends on EsoundD or OSS,

and EsoundD is the lesser evil.
The oldest versions of EsounD do not support ALSA at all, old versions only support ALSA 0.5, and only fairly recent versions support ALSA 0.9/1.*x*.
EsoundD versions 0.2.27 and earlier needed a patch to support ALSA, but the patch was incorporated in versions 0.2.28 and later. However, usually EsoundD is packaged in two different and mutually exclusive versions, one of which supports OSS, the other ALSA, or some variation thereof. For example with Debian the `libesd0` package supports OSS and `libesd0-alsa` supports ALSA.

Timidity

Usually something like `timidity -oS -iA` is good, where `-oS` select as output an ALSA PCM, and with `-iA` Timidity becomes an ALSA soft sequencer. More context in a RoseGarden FAQ.

Doom 3

From version 12.86 Doom3 for Linux supports ALSA natively. A full discussion is provided in this section.

Quake 4

In theory Quake 4 uses the Doom 3 engine so its sound should be similar. However there is some kind of problem with its ALSA support, which manifests itself with error messages like:

```
snd_pcm_writei short write: 3763 out of 4096
```

and the only reliable workaround is to switch to the OSS driver API with

```
quake4 +set s_driver oss
```

This workaround should no longer be necessary starting with the 1.3 patch.

Common tasks with ALSA (060502)

Selecting a default card

When ALSA drivers get loaded they assume a particular order that influences which card become device number 0, 1 and so on. Device number 0 is the default one, and it is often desired to make a specific card to have device number 0.
It is not always easy to guarantee a particular order of loading of modules, because in modern Linux distribution there are several independent module loading programs that operate according to different principles.
The only reliable way to achieve this is to assign to *all* cards an explicit device number with the `index=` module parameter (as also described in this page).

Record sound

To record sound one must enable the relevant capture controls. Since different cards can have very different mixers, which ones are relevant depends a lot on the card. It is best to explore the

various channels and controls using `alsamixer` and pressing `F5` to see the full set of capture controls.

Despite the wide variation in mixers, there are some common types and names of controls:

- If input channels like `Mic` or `Line In` also have a playback control and a volume control that allows routing their input to the output, that means that their input can be routed to the output, which allows monitoring what is being recorded.
- If output channels like `PCM` or `Synth` also have a capture control that allows routing their output into the input, which enables recording what is being played.
- If there is a `Capture` channel usually that must be set in capture mode enabled for *any* recording to happen. If other channels, for example `Line In` can be set in capture mode that is to allow to control which input channels are mixed into the input channel.
- If there is a `Mix` channel that can be set in capture mode that must be set to allow recording of sounds being played. Usually if this channel exists there are no capture controls on individual output channels, and the complete mix of all output channels is recorded.

Probably each card needs a bit of experimenting to figure out the roles of the various channels and their controls. These can be rather unobvious.

Once the capture mode has been selected, one can use some kind of recording application, for example `arecord`, selecting the right recording device. This can be usually a direct specification like `hw:0,0` (for the analog recording device) or a virtual device name defined in an `asound.conf` file like `dsp0`; in the latter case it must be defined as a `dsnoop` plugin or an `asym` plugin.

With most cheap microphones out there one usually has to enable the (usually low quality) internal input amplifier by unmuting a suitable channel, usually called `Mic Boost`.

In most recent low cost chipset recording can only be done at a single fixed rate, usually 48000Hz, or at one of two fixed ones, usually 48000Hz and 44100Hz. If you want a different frequency you can resample after recording or use the `plug:` prefix on the name of the chosen recording device (or `plughw:` if it is a `hw:` device). For example:

```
arecord -D plughw:0,0 -f S16_LE -c 2 -r 22050 record.wav
```

Note that achieving good a good quality recording can be very difficult or impossible; for example some SoundBlaster Live! cards resample even at their native rate.

Record playing sound

This can be achieved in several ways:

- Physically loop back the sound output into the `Line In` socket.
- Enable playback loopback using the mixer. Most cards have the ability to loop back output channels into input channels. If this is the case the output channels will have a capture control. This is often not displayed by default in `alsamixer` and one has to press `F5` to see all the capture controls. Sometimes output-to-input routing is done globally with a capture control associated with a single channel, usually called `Mix`.
- Using the `file` plugin (possibly in conjuction with the `copy` plugin) to route the output of an ALSA application to a file.

OSS emulation (051101)

Takashi Iwai has written some important notes on OSS emulation.

In general OSS emulation relies on either the OSS compatibility devices, or the `aoss` wrapper that redirects sound to the ALSA devices instead.

This wrapper defaults to using the `dsp0` virtual PCM defined in the ALSA library configuration. This can be changed by setting the `ALSA_OSS_PCM_DEVICE` environment variable. The `ALSA_OSS_DEBUG` can be set to enable some debugging messages.

A newer and somewhat more flexibile and complex method is `oss2jack` which creates some virtual OSS devices that are fronts for the JACK. sound server.

Some common issues:

Even if ALSA is installed, some applications are still using the OSS interface

This can be checked with a command line like:

```
lsof | egrep '/dev/(sound|dsp|mixer)'
```

If there are applications listed that still use the OSS interface, please configure them to use the ALSA interface.

OSS applications don't share the sound card

By default the OSS compatibility devices respect POSIX rules and allow only exclusive opens unless the `O_NONBLOCK` option is given to `open`*(2)*. This can be worked around either by using the `aoss` program, `oss2jack`, the `non-block` configuration directive, or the `nonblock_open` options of the `snd-pcm-oss` modules.

OSS applications that only use `mmap` mode

Some OSS applications, notably Quake and other games, access the OSS compatibility devices in `mmap` mode, thus bypassing the ALSA library. Check the `mmap` related options in the `/etc/asound.conf` file syntax, and the `direct` and `disable` options to the OSS compatibility PCM device control device, for example `/proc/asound/card0/pcm0p/oss`.

OSS applications sound bad at higher sample rate or widths

This can be due to fragment size limits in the underlying chipset: several chipset have a limit of 4096 bytes, and any fragments larger than this get truncated. You can set explicitly the fragment size by writing the appropriate options to the OSS compatibility PCM device control device, for example `/proc/asound/card0/pcm0p/oss`.

The major applications that still only support OSS are Mozilla and Firefox and the Flash and Java browser plugins, and several older games.

But newer versions of the Flash plugin also support EsounD, which is far preferable in most situations as it allows for sharing.

Sharing a device

It is often desirable to be able to share a sound card among several processes running at the same time.

This requires the ability to mix the sound outputs of those processes into a single stream, that is multiplexing.

In order to achieve this with ALSA there are several different cases and techniques.

The cases depend on whether the sound card/chipset supports hardware mixing or not, and whether the processes access the sound card/chipset via the ALSA library, a sound server or OSS emulation.

In the beginning OSS often did not support sharing even if it was supported by the hardware. ALSA drivers, as a rule, will support sharing if the hardware supports it. The ALSA library supports sharing even if the hardware does not support it, but this requires some explicit configuration. For applications that use OSS, the `aoss` wrapper can make them use ALSA instead, which improves things.

Finally applications that use sound servers like EsounD, polypaudio, aRts, or JACK. most sound servers perform software mixing and support ALSA output.

The individual cases are:

- The card supports hardware mixing.
- The card does not support hardware mixing, but all processes accessing it run applications that use the ALSA library.
- The applications use a sound server to access the card.
- The applications use the OSS API to access the card.

The card supports hardware mixing

This is the best case. Most recent cards support hardware mixing, at least for output, and when they do they support it to up to a maximum number of streams that is so high that it is unlikely to be ever a problem.

If you can the simplest way to ensure sharing is to get a card that supports hardware mixing. Sound cards are cheap, often costing less than the time to implement workarounds.

The card does not support hardware mixing, but all processes accessing it run applications that use the ALSA library

In this case, it is fairly to create an ALSA library configuration file (see the `.asoundrc` documentation at OpenSrc.org and at ALSA-Project.org) that allows software mixing. This is achieved using the `dmix` (for output) and `dsnoop` (for input) plugins, and `asym` to tie them together. In ALSA library version 1.0.9rc2 and later versions this is already done in the standard configuration files.

There is an example of using them below, and a much more extensive, ready-made, nearly universal 1 or 2 card configuration file that you can most often just drop-in and it will work here.

The applications use a sound server to access the card

Sound servers were mainly created to premix multiple streams for OSS, where even cards that supported hardware mixing did not support multiplexing.

If your system runs a sound servers like EsounD, for GNOME or aRts, for KDE, set the sound server to use ALSA as its output, and applications to use the sound server.

For KDE aRts problems see: here and try using `artsdsp` to make OSS applications use aRts instead. But it is probably preferable to make them use ALSA directly by using `aoss`.

The applications use the OSS API to access the card

Some applications cannot use ALSA or a sound server, but only the OSS API. In that case you can often make them use ALSA using the `aoss` wrapper. Also check these these notes and check carefully the description of non blocking options in these other notes.

Examples of ALSA lib configurations to use the software mixing plugins

Simple output only sharing example

```
# The top level shared pseudo device, with both PCM and CTL interfaces
# The device names "default", "dsp0", "mixer0" have conventional meanings.

# The top level shared pseudo device, with both PCM and CTL interfaces
# The ALSA default is "!default", but many programs like XMMS and aoss
# assume "dsp0" as default name for PCM and "mixer0" for CTL.

# Amazingly, XMMS has problems if one defines 'pcm.dsp0' to be
# 'plug' for 'pcm.asym0' and not directly as 'asym0'.

pcm.!default            { type                  plug;
                          slave.pcm             "dmix0"; }
ctl.!default            { type hw; card 0; }

pcm.dsp0                { type                  plug;
                          slave.pcm             "dmix0"; }
ctl.dsp0                { type hw; card 0; }
ctl.mixer0              { type hw; card 0; }

################################################################################

# Buffering (period time defaults to 125000 usecs).

# Size of period, expressed either in usec or byte units:
#   period_time USECS
#   period_size BYTES

# Size of buffers, expressed either in period, usec, or byte units:
#   periods     PERIODS
#   buffer_time USECS
#   buffer_size BYTES

# The ALSA docs have examples with 'period_time' set to 0,
# when 'period_size' and 'buffer_size' are used instead,
# but this can cause trouble in later releases of ALSA.

# For OSS compatibility, 'period_size' and 'buffer_size'
# should be powers of 2. Also, many cards cannot accept
# a 'period_size' much greater than 4096, so 4096 is safe.
# On my VIA 8233A, any value for 'period_time' greater than
# 85333 usecs (precisely!) causes hiccups in sound output.
# Why? At 48kHz, 85333 usec are are just over 4096 bytes/channel.

pcm.dmix0               { type                  dmix;
                          ipc_key               13759;
```

```
                        slave.pcm               "hw:0,0";
                        slave.channels          2;

                        slave.rate              48000;
                        slave.period_size       4096;
                        slave.buffer_size       16384;

                        slave.period_time       84000;
                        slave.buffer_time       340000;

                        # Map only the first two channels
                        bindings.0              0;
                        bindings.1              1; }
```

Sharing both input and output

To the output only example add:

```
# The top level shared pseudo device, with both PCM and CTL interfaces
# The ALSA default is "!default", but many programs like XMMS and aoss
# assume "dsp0" as default name for PCM and "mixer0" for CTL.

# Amazingly, XMMS has problems if one defines 'pcm.dsp0' to be
# 'plug' for 'pcm.asym0' and not directly as 'asym0'.

pcm.!default            { type                  asym;
                          capture.pcm           "dsnoop0";
                          playback.pcm          "dmix0"; }
ctl.!default            { type hw; card 0; }

pcm.dsp0                { type                  asym;
                          capture.pcm           "dsnoop0";
                          playback.pcm          "dmix0"; }
ctl.dsp0                { type hw; card 0; }
ctl.mixer0              { type hw; card 0; }

############################################################################

pcm.asym0               { type                  asym;
                          capture.pcm           "dsnoop0";
                          playback.pcm          "dmix0"; }

pcm.dsnoop0             { type                  dsnoop;
                          ipc_key               13758;
                          slave.pcm             "hw:0,0"; }
```

This defines a virtual ALSA PCM device called `asym0`. This device is capable of mixing several playback streams and sharing one capture stream amongst several applications. To get automatic samplerate conversion, etc, we defined the device `dmix0` which uses alsa's plug plugin.

Furthermore we defined a device called `!default`. This is equivalent to `dsp0`. The special name `!default` makes this device the default device for all well coded ALSA apps (sadly not too many are well coded).

And last we defined a device called `dsp0`. This device is used by the `aoss` script from the `alsa-oss` package.

First of all we test this basic setup with the standard ALSA aplay tool. You will need a `.wav` file for this test. If you have none, create one out of an MP3 with the following command:

```
mpg123 name.mp3 -w name.wav
```

With this `.wav` file we test the `dsp0` device now:

```
aplay -D dsp0 name.wav
```

This should playback the `.wav`. Even if you run this command in a second terminal at the same time, because the `dsp0` device does the mixing. Because we also defined the default alsa device

`!default` to use `asym0`, you should also be able to run the command without the `-D dsp0` parameter:

```
aplay name.wav
```

Not all apps honour the default device though. MPlayer for example is one of them.

To test this setup with MPlayer use:

```
mplayer -ao alsa1x:dsp0 name.avi
```

So, now is the time to test all your desired alsa apps to work with this setup.

- Some will need to be told explicitly to use `dsp0`.
- Others will happily use the default.

If some ALSA apps behave badly with `dsp0` (crackles, stutter), check the `dmix` plugin configuration page. It has quite a bit of troubleshooting. Tips: look at the samplerates of the slave, maybe play with the period_size parameter, etc.; in particular many cards have limits on the `period_size`, usually to 4096 bytes.

Some applications use `mmap`'ed audio data transfer. If your application complains about not being able to use mmap, then play around with the `mmap_emulation` setting in the `pcm.dsp0` definition.

Some applications and/or some cards will simply not work in `mmap` mode, so try disabling it.

Resources

- A version of the same information at the `ALSA.OpenSrc.org` site.
- Some similar notes with a simpler content mostly about ALSA lib.
- List of all currently supported ALSA cards/shipsets with those that have hardware sharing indicated.
- A new (200501) mixer program for ALSA, Mix2005.

Issues with popular chipsets

Unfortunately many sound chipsets are poorly designed or plain buggy, or are used in sound cards that are themselves poorly designed or buggy, and both can happen at the same time.

Even more unfortunately, some of these poorly designed or buggy chipsets or cards are very popular and end up in most systems.

Also note that chipset or card related issues do not necessarily happen in every system using them; they can depend on the motherboard chipsets or the sound hardware plugged into the card or even on the quality of ground of the power system of the area in which the card and the PC are used.

Usually the best way to solve any chipset or card related problems or limitation is to buy another one, as there are fairly decent and cheap soundcards around, and they usually cost much less than the time and effort required to investigate and work around chipset or card specific problems.

However be warned that some sound problems are caused by environmental and electrical issues, and they will happen with all chipsets and cards until the underlying causes are fixed.

As a rule it is important to read carefully the driver module documentation in the `ALSA-Configuration.txt` file in the Linux kernel sources and the ALSA driver sources, read the

related page on the ALSA Project/ and ALSA Wiki pages for the specific chipset or card.
This generic `asound.conf` will probably work with most recent chipsets on motherboards and on low end sound cards and fix a number of annoying problems if you use the `dsp0` device. Study it carefully.

All these said, here are some notes on some popular chipsets and cards:

Intel 8x0 compatible

There are some notes elsewhere too (1, 2).

- There are *many* variants of the Intel 8x0 sound chips, and the `snd-intel8x0` driver is being constantly updated as new variants appear. Many such variants differ in the way their mixers are set up or other details.
- In particular the ICH5 and ICH6 versions of the Intel sound chips is supported only by ALSA versions 1.0.5 or later. In particular, the Linux 2.6 kernels up to version 2.6.8.1 inclusive don't support them because they only have ALSA versions up to 1.0.4; note also that there are very many variants of the ICH6 chipset (check `include/linux/pci_ids.h` in recent kernel sources), and up to 1.0.8 only the one with device id `8086:266e` is recognized. More may be added with time, so check the latest 2.6 kernels and ALSA driver packages for 2.4.
- High Definition Audio versions of the chipset, ICH6, ICH7 and ICH8 (with their Realtek codecs) are supported by the `snd-hda-intel` driver, which is only available from ALSA 1.0.9, or with kernels later than 2.6.11; an earlier version of this driver, from ALSA 1.0.8, was called `snd-azx`.
- Since there are a very large number of variants of the ICH chipsets, it is very important to use a driver version that has specific support for that chipset, down to the specific type of motherboard or laptop they are embedded in. For laptops in particular sometimes the routing of the output audio to a specific device (headphone socket or builtin speakers usually) is laptop dependent and may not be part of chipset function. It also also usually important to use the right argument to the `model=` module parameter.
- As an example on the ICH8 variant with an AL268 codec on 2008 vintage Toshiba laptops is supported starting from ALSA 1.0.15, even if 1.0.14 recognizes them but does not quite handle them correctly. Usually before one of the innumerable ICH variants is supported officially by ALSA some patches appear, so it is worthwhile to look for them or download development versions of the ALSA drivers. For example for the Toshiba ICH8 ALC268 combination there are several (1, 2, 3, 4).
- These chipsets usually do not support hardware sharing, so make sure you have software sharing configured.
- Also, carefully read about the parameters to the `snd-intel8x0` driver module, in particular the `ac97_clock`, `ac97_quirk` and `dxs_support` ones. They are described in the `ALSA-Configuration.txt` file in the Linux kernel sources and the ALSA driver sources. For many variants of the ICH it is *essential* to specify the appropriate value for the `model=` driver parameter.
- It is usually best to set `ac97_clock=48000`, and to set the output plugin's sampling rate accordingly in the `/etc/asound.conf`.

VIA VT82*xx*

There are some notes elsewhere too (1, 2).
This chipset is somewhat similar to the Intel 8x0 one, and most of the information above about the latter applies here, in particular the use of the `ac97_clock` parameter, and reading the chipset specific notes in the `ALSA-Configuration.txt` file in the Linux kernel sources and the ALSA driver sources.
This chipset is usually a frontend to some other sound chip, like an ALC650 or CMI9761. The mixer can be slightly peculiar especially as to the recording. There are some input channels for specific inputs and then a `Capture` channel. The volume controls associated with the specific channels control the copy, not the input, volume; the input volume is controlled by the volume of the `Capture` channel.

The capture buttons of the input input selects from which one recording happens. Only one input input channel can be in capture mode, and always one; as it is the capture button of the `Capture` channel control whether recording is done at all, and is a toggle, not a selector.

SoundBlaster *non*-Live! variants

There are some notes elsewhere too (1, 2, 3).
The SoundBlaster PCI, 128, 1024 etc. are usually based on the Ensoniq ES137*x* chipsets, which are all fairly similar and supported by the `snd-ens1371` driver module. They have neither software sharing nor, usually, a hardware synthesizer.

SoundBlaster Audigy LS and Live! 24 bit

There are some notes elsewhere too (1, 2).
The first bad news is that apparently some Audigy variants, like some Live! variants, don't have a DSP chip, and are thus fundamentally different from the other Audigy cards. Beware!
The driver has *"recently"* changed name to `snd-ca0106` and also supports the SB Live! 24 bit.

SoundBlaster Audigy and Audigy 2 variants

There are some notes elsewhere too (1, 2). 3).
Both the Audigy and Audigy 2 variants (except for the Audigy LS) still use the `snd-emu10k1` driver module, and most of the description related to the SB Live! range applies to them too.

SoundBlaster Live! variants

There are some notes elsewhere too (1, 2 and especially 3).

- Almost all SoundBlaster Live! variants use the EMU10K1 DSP chipset, and are supported by the `snd-emu10k1` driver module, but some newer cards share the name but not the chipset.
- The SB Live! 24 bit is not an SB Live! card in the sense of having an EMU10K1 chipset, and is supported by the `snd-ca0106` driver.
- The SB Live! Dell OEM is not really an SB Live! card, as it does not have an EMU10K1 chipset, and is supported by the `snd-emu10k1x` driver.
- The SB Live! mixer is vast and mysteryous, some notes on the function of its many controls can be found in the STAC chipset documentation and in the apposite ALSA Wiki page.
- Apparently the SB Live! cards operate internally at 48kHz, like many others. Unfortunately this means that they resample to 48kHz on recording, and apparently they resample to 48kHz even inputs that are *already* at 48kHz, introducing artifacts.
- If you are using the IEC958/SPDIF bracket (which may also be purchased separately from HoonTech), read carefully the documentation for the `extin` and `extout` parameters to the `snd-emu10k1` driver module. Also, you can do AC3 pass-through with a little mixer firmware update as described here.

CMI973*x*

There are some notes elsewhere too (1, 2), This chipset is increasingly popular on various motherboards, and has some significant limitations, for example the digital output can only be at 48000Hz.
More importantly, the chip seems not to have the ability to change sound volume, and one needs a software volume control, like the newly introduced `softvol` plugin for `/etc/asound.conf`.

NVIDIA MCP51

On laptops sometimes this chipset has somewhat different routing for the external headphones. This page points to a patch to fix the the routing.

Now for completeness some popular chipsets or cards that I know of and that usually work nicely and without much trouble, but may have some small issues:

CMI8738

There are some notes elsewhere too (1, 2),
CMI8738 is a pretty decent chipset supported by the `snd-cmipci` driver, which works well and easy to configure, has a fairly uncomplicated set of mixer controls, and is available as very cheap PCI cards and on some motherboards as the built in sound chipset.

- The major limitation of the CMI8738 does not support hardware sharing, but you can just setup software sharing.
- There is a Trust card with this chipset which is cheap and has a IEC958/SPDIF bracket with both optical and coaxial input and output. It also has a MIDI/gameport socket. The gameport has a know electrical problem and basically does not work. Other than that the card is pretty decent.
- IEC958/SPDIF output can be simple or somewhat complicated to enable and configure. Check the references above.
- The output stage runs at 48kHz and playing 44.1kHz sound involves resampling and further resampling (44.1kHz to 48kHz to 44.1kHz) if the (digital) output is to 44.1kHz again. In practice this card should be considered to be 48kHz only, even if nominally can resample to 44.1kHz itself.
- As listed by `aplay -l` device 0 is the first DAC, 1 is the second DAC, and 2 is the IEC958 output. The second DAC does the rear two channels if the mixer control `Four Channel Mode` is disabled, but it has 4/6 channels and drives both output socket if enabled; this means that 4/6 channel mode can be used with `dmix` if device 1 is specified and `Four Channel Mode` is enabled.

Also, someone has done a nice utility to control its detailed features but perhaps it works only under OSS.

ICE17*xx*/Envy24*xx*

There are some notes elsewhere too (1, 2, 3, 4, 5).
These chipsets, now owned by VIA, are well supported by ALSA, and there is even a special and high level mixer program for Envy based cards. There are several 5.1 (ICE1712/Envy24) and 7.1 (ICE1724/Envy24HT) cards using this chipset, for example from Terratec.

CS46*xx* (Crystal Sound Fusion)

There are some notes elsewhere too (1, 2).
The Crystal SoundFusion chipsets are fairly well supported and featured, and the CS4624 has 5.1 channels, and the CS4630 has 7.1 channels.

Yamaha YMF72*xx* and later chipsets

There are some notes elsewhere too (1, 2).
These are supported by the `snd-ymfpci` driver module and are also pretty nice and simple.

Another page has a list of supported sound cards, usually oriented towards audiophiles.

Potential sound quality issue checklist

Sound hardware depends on particularly tight realtime constraints, and this can give rise to a number of issues.
Sound hardware is often also designed and manufactured without much care, especially in the case on motherboard chipsets.
There can be in particular many sound quality issues related to latency, and these are discussed in a separate document on Linux sound latency issues.

The sound quality HOWTO also has a lot of advice on quality issues.

These are issues that are generally possible, and not necessarily related to the specific chipset or sound card used:

BIOS: disable `PCI Delay Transaction`
For better performance and latency this BIOS setting should be enabled, but there are enough cards that don't support it well that if there are problems one should try to disable it.

BIOS: change IRQ sharing
Ideally your sound chipset should be on a dedicated IRQ, but this is often not possible on a system with many chipsets. Unfortunately some chipsets do not share IRQs well, and it is difficult to know in advance. So experiment with rearranging IRQ assignments in the BIOS, and/or swapping PCI cards around, as IRQs are assigned to slots, not cards as such.
To ensure your explicit IRQ assignments it is probably necessary to disable in the BISO the `PnP OS` setting (usually a good idea regardless) and to use the kernel boot parameter `pci=biosirq`.

System: tweaking the PCI latency timers
If your sound is metallic or crackly or slightly distorted perhaps you need to set the PCI latency times of some PCI card to less aggressive values, for example with the command:

```
setpci -v -s '*:*' latency_timer=20
```

(but reset the latency timer to 0 for the host bridge, usually with id `00:00.0`). There are more details on this in the notes on sound latency. Another possible workaround is adding `Option "PciRetry" "true"` to the X server configuration.

System: ensure proper MTRR
The MTRR settings for your system can matter greatly to the amount of bus traffic, and this can impact the performance and quality of your sound. Check the plausibility of the contents of `/proc/mtrr`; main memory should be `write-back`, and the frame buffer of video cards (at some very high address) should be `write-combinining`. There are some documents and articles on the WWW that discuss MTRR settings.

System: ensure the ATA/IDE drives are on DMA
If the ATA/IDE drives (usually hard discs) are not using DMA then IO may take a lot of CPU time and this can impact very strongly sound quality. Use `hdparm -v` to check if DMA is enabled.

ALSA: `fragment_size` <= 4096
Many cards, in particular of the AC97 sort, have low limits on the size of sound samples they can accept, usually only a few kilobytes. This means that the fragment size for that card should be set to be low. This can be done inside the program writing sound data to the card, but it can also be done by defining a suitable plugin in `/etc/asound.conf` with the desired fragment size.

ALSA: ensure volume is less than 70%
Many cards have somewhat poor internal mixers and amplifiers, and sound will be distorted at high volume levels. Some cards have problems with too high levels of the master volume, some with too high volumes of the individual channel volumes, some with both. Some will have problems with volume setting higher than 50%, some with setting higher than 65-70%.

ALSA: check the number of channels in `/etc/asound.conf`
If you specify the wrong number of channels in a virtual device definition the resulting sound can be quite crackly. Some cards for example don't have a 2 channel mode, they just work in 5.1 channel mode.

ALSA: ensure all non necessary inputs are muted
Sometimes the controls for input devices are left unmuted; in theory this should affect the sound, especially if nothing is muted into them. But unmuted recording controls can collect electrical noise which gets mixed into the output. Makes sure that all non necessary input channels are muted; merely setting the capture volume to zero is not enough.

ALSA: use external amplifier and line out

To avoid sound distortion problems it is often a good idea to use an external amplifier, instead of relying on the internal amplifier on the card. Several ALSA drivers allow disabling the internal amplifier by unmuting the control called `External Amplifier`. If this is not possible or does not work, one can plug the external amplifier into the line-out socket of the card, if any.

ALSA: use external preamp for microphone

There are many, many issues related to mismatches in impedance and other electrical characteristics between microphones and cards. Sometimes recording is better if one uses the `Mic Boost` control for the microphone channel, but usually quality will not be optimal. If quality is desired, an external microphone amplifier is probably# a lot better.

Troubleshooting (060502)

An important first step: compile the relevant information using `aadebug`.

Also have a look at the FAQ and the drivers documentation pages (1) for troubleshooting your specific soundcard.

The ALSA tips page might have useful stuff too, and this PDF of a presentation by an ALSA developer contains a number of interesting and important troubleshooting steps and details.

Check that the ALSA drivers are compiled as modules

Make sure you have installed ALSA as modules, and not compiled into the kernel.
ALSA fails for all sorts of reasons when compiled into the kernel (this may no longer be true for kernels after v2.6.5).
Anything that mentions sound in the kernel config, even if it is not directly to do with ALSA, set it's option to `M` if you can.
If you compile you own kernel: when you configure the kernel, make sure you see `M` (for module) and not `*` (compiled into the kernel).

Check the ALSA driver version

Look at `/proc/asound/version` and that this says something like:

```
Advanced Linux Sound Architecture Driver Version 1.0.4.
```

and that the version is 1.0.4 or above.
If it is not, download and compile newer drivers for kernel v2.4, or update your v2.6 kernel to version v2.6.5 or later.
There are several ALSA installation pages on the web, which are usually distribution dependent. . Check the Wiki or documentation pages for your distribution for details.

Check the ALSA library version

How to check your ALSA library version is distribution dependent. Usually, you can use the package (e.g. RPM or DPKG) or dependency manager (e.g. APT, aptitude, synaptic, yum, up2date, YAST2) used by your distribution to check the version of installed packages.
You can also try something like

```
grep VERSION_STR /usr/include/alsa/version.h
```

and check that the version is at least 1.0.4, and usually it is best if it matches the driver version.

Check the sound drivers for your card are active

First check that the ALSA drivers are installed and have recognized your card.
Make sure that `/proc/asound/cards` lists your card, as card number zero. If not, make

sure that the appropriate driver module is loaded.
To figure out which modules you need, use the

```
lspci | egrep audio
```

command. This usually will list the name and type of your sound chipset. The main ALSA website then contains a list of chipset and corresponding drivers.
As a wild guess, for most recent low cost AC97 based motherboards and laptops, try the `snd-intel8x0` driver; for the other common sound card type, the Creative SB Live! range, use `snd-emu10k1`.
To make sure your card is recognized and the right driver is selected you can also try the `alsaconf` command, and use it to configure your sound card.
It can also be that the ALSA driver has not been loaded but an OSS driver has been loaded. Check the contents of `/dev/sndstat` and if it exists and you see your card listed there, and there is no line talking about ALSA emulation, it is being driven by an OSS driver. Unload it and load the ALSA driver.
If you have multiple sound cards, check this item to discover how to configure ALSA to renumber them if the cards are not in the order you would like them to be.

Check that you have the right sound devices

Run `alsamixer` as `root`. If you get

```
alsamixer: function snd_ctl_open failed for default: No such device
```

the right device special files in `/dev/` might be missing. Run `ls /dev/snd/` and make sure there are several entries there.
If the directory does not exist, or it is empty, use the ALSA `snddevices` script or similar to create it and the device special files in it.

Check that non-`root` users can access the sound device special files

Run `alsamixer` both as `root` and as a non-`root` ordinary user.
If the latter fails, the permissions on the device special files don't allow access by ordinary users.
There are several options to fix this, depending on security requirements, your distribution and how your PC is set up:

- Allow all users read and write access to all sound devices, for example with

  ```
  chmod -R a+rwX /dev/snd/.
  ```

  This might be slightly insecure.
- Make sure that the sound device files belong to a specific group, they have group read/write permissions, and add all the users that should have access to the sound cards to that group. In many distributions this is the `audio` or `sound` group.
- If only one user needs access to the sound card, change the ownership of the sound device files to that user.

Check that the sound card mixer channels are unmuted

If your card's driver is installed and its name appears in `/proc/asound/cards`, and you still hear no sound, the most likely cause is that you haven't unmuted the right mixer channels and set their volume to nonzero. Note that ALSA sort of misnomes the channels of the mixers of many cards.
Use `alsamixer` to play around with the settings of the most obvious sounding channels. There are often descriptions of what the channels are in the page about your specific sound cards.
Usually, make sure that the at least the `Master` and `PCM` (and `Wave` or `Headphone` if present) channels are unmuted and have non zero volume.
For laptop users, try toggling the `External Amplifier` switch.

The main channels are unmuted and non zero volume, but no sound

Many cards can do both analog and digital (labeled SPDIF/IEC958) output, but some cannot do both at the same time.
If you hear no sound, it may be because the card is in analog mode and you have digital speakers or viceversa.

To determine which mixer channels controls the switch between analog and digital for your card look at the page for your card.
Often this is called `Analog/Digital Output Jack`. If present, mute/unmute it to switch between digital and analog sound output.

Check whether your application uses OSS instead of ALSA

If you still hear no sound, your application may be set up to use OSS. Check with your applications preferences to see if this is the case. The best fix is to set you application up to use ALSA instead of OSS. If your application does not support ALSA, there are three possible solutions:

- Make it use a sound server that uses ALSA for output.
- Load the OSS compatibility modules for ALSA.
- Use the `aoss` wrapper to run the application.

There are some other pages on ALSA emulation of OSS (1, 2).

Still no sound with your favourite application

Make sure that it is not a problem with the way the sound application you are using is configured. Try

```
dd if=/dev/urandom bs=8000 count=1 | aplay -D DEVNAME
```

to see if you can hear something. If you dont know which *DEVNAME* you should specify, use `plughw:0,0` to test sound card 0 in analog mode, and `plughw:1,0` for sound card 1.

Check whether your sound card or application only work with or without `mmap`

Some sound cards and/or some applications will not work in `mmap` mode, so try disabling it. Viceversa if it is disabled try to enable it.

Sound can be heard but with an echo

This happens particularly with applications where there is both input ad output, for example voice chat or voice IP ones, and you are using some chipset like the Intel ICH ones. If this happens it probably happens only with a few chipsets: the cause is that those chipsets allow more than one capture source channel to be specified, usually two, and the two are different.
This can only be checked looking at non simplified controls with `amixer`, for example with:

```
amixer cget name='Capture Source'
```

which might return:

```
  ; type=ENUMERATED,access=rw---,values=2,items=8
  ; Item #0 'Mic'
  ; Item #1 'CD'
  ; Item #2 'Video'
  ; Item #3 'Aux'
  ; Item #4 'Line'
  ; Item #5 'Mix'
  ; Item #6 'Mix Mono'
  ; Item #7 'Phone'
  : values=0,5
```

This indicates that there are 8 possible capture channels, and `values=0,5` indicates that both `Mic` and `Mix` are enabled at the same time. This often is the result of a bug in `alsamixer`.
If this is not intentional, it is possible to ensure only one channel is enabled for capture by setting both values to the same number, for example for `Mic` in this case with:

```
amixer cset name='Capture Source' 0,0
```

Sound can be heard but it is distorted

- Some cards, notably SB Live! ones, suffer from distortion if the volume on some subchannels is higher than 66%. Some suffer distortion if the `Master` volume is higher than 66%, for some others the threashold is 50%. Reduce all volume setting. If the distortion goes away, experiment until you determine which are the highest volume settings that don't trigger distortion.

- Some cards, especially motherboard chipsets, have small limits on the fragment size (specified for example in the dmix plugin configuration. Many cards cannot handle fragment sizes greater than 4096 bytes, and if it is bigger the fragment seems to be truncated and the sound becomes choppy.
- Some cards, especially recent ones, can only handle a fixed set of frequencies. Some can only handle a single frequency, usually 48000Hz. try to use the `plug:` plugin prefix (as in, for example: `plug:default` when playing. If your card can only play a single fixed frequency you must ensure that the driver is told that (by the use of driver-specific option parameters), and the ALSA library is setup up to output sample only at that frequency. Usually this will involve using the `plug` plugin in `/etc/asound.conf`.
- Choppy sound can also be due to IRQ issues, in particular with recent systems that have an ACPI BIOS and a Linux kernel with ACPI support. Try adding `acpi=off` to the linux kernel boot arguments

Also see the potential sound quality issue list section in this document.

You can hear stereo sound from the front speakers, but not from the other speakers

The default ALSA output virtual device is 2 channel only. In order to hear sound on more than two speakers you need to use the name of a virtual device that supports more than two channels.

These virtual device names begin with `surround`; the most commonly used are `surround40` and `surround51`. To get a full list of those defined for your card run:

```
aplay -L 2>&1 | egrep surround
```

For some cards, in particular USB based cards, ALSA does not (yet?) define any `surround` style virtual device names, in part because these cards usually *only* work in surround mode, accepting for example only 5.1 channel samples.

Note that if you play 2 channel sound to a `surround`*NN* device, you will still hear only the front speakers. This is as it should be.

If you want to artificially *"spread"* the 2 channel sound onto more channels you need to create a special virtual device in the `/etc/asound.conf` file to do that using the `route` plugin. There are several examples of this in this sample `asound.conf` file (additional notes: 1, 2).

You can hear sound coming from the front and rear speakers, but not from the other speakers you have

This may be because your sound has fewer channels than you have speakers, for example because you are playing 4 channel sound onto a 5.1 channel speaker system. In this case there is no problem, things are working as they should.

But it can be because your card uses the same sockets for some input and output channels. Often the `Line in` socket is also used for the `LFE` output channel and the `Mic` socket for the `Center` output channel, or viceversa. If this is the case you must use the appropriate toggles in the mixer controls to ensure that those sockets are used for output and not input.

In order to test that all speakers work you can use multiple channel sound files or use the `speaker-test` utility, which has been included in the `alsa-utils` package for a while now.

Things work except when you use a device for sw sharing, one that relies on the `dmix` plugin

This can be due to the kernel lacking the SystemV compatible shared memory implementation. Make sure the kernel has it.

References (091115)

- ALSA project home page.
- ALSA unofficial Wiki.
- Some ALSA notes similar to these with a more tutorial attitude.
- nice diagram that illustrates the structure of ALSA.

Colophon

This document is the result of many hours spent:

- Trying to figure out ALSA for my own system, using my VT8233A, SB Live!, CMI8738, and CS4261 chipset and cards that I have used at various times. *"Thanks"* to Creative in particular for half-assedly designing my *SB Live!*, despite their immense corporate resources, and thus leading me to investigate the many pointless absurdities of this range of cards.
- Helping often hapless, lazy, and irritating people in the #ALSA channel on Freenode, *"thanks"* to them for showing me what are the most common issues and misunderstandings with ALSA.

Thanks to the ALSA authors for writing the software, which is after all a huge improvement on OSS, and *"thanks"* to them also for under-documenting and over-complicating it gleefully probably to protect their own jobs and their employer from competition (really?), and thus giving me the motivation to spend a lot of time, that maybe I could have spent more usefully, trying to collate obscure hints here and there, reverse engineering the sources, and experimenting, to gather the information condensed in these notes.

Thanks to `<peer>` for the pointer to the chipset docs of the SB Live!.

Thanks to `<hermanro>` for the pointer to the RoseGarden FAQ on soft sequencers.

Thanks to the fellow sufferers and inmates of the ALSA asylum, those who have scratched on the walls of their cells their horror stories, painful discoveries and hints for those that would come after them.

---

notes@notes.for.sabi.co.uk