More Notes on HD-Audio Driver

Takashi Iwai <tiwai@suse.de>

General

HD-audio is the new standard on-board audio component on modern PCs after AC97. Although Linux has been supporting HD-audio since long time ago, there are often problems with new machines. A part of the problem is broken BIOS, and the rest is the driver implementation. This document explains the brief trouble-shooting and debugging methods for the HD-audio hardware.

The HD-audio component consists of two parts: the controller chip and the codec chips on the HD-audio bus. Linux provides a single driver for all controllers, snd-hda-intel. Although the driver name contains a word of a well-known hardware vendor, it's not specific to it but for all controller chips by other companies. Since the HD-audio controllers are supposed to be compatible, the single snd-hda-driver should work in most cases. But, not surprisingly, there are known bugs and issues specific to each controller type. The snd-hda-intel driver has a bunch of workarounds for these as described below.

A controller may have multiple codecs. Usually you have one audio codec and optionally one modem codec. In theory, there might be multiple audio codecs, e.g. for analog and digital outputs, and the driver might not work properly because of conflict of mixer elements. This should be fixed in future if such hardware really exists.

The snd-hda-intel driver has several different codec parsers depending on the codec. It has a generic parser as a fallback, but this functionality is fairly limited until now. Instead of the generic parser, usually the codec-specific parser (coded in patch_*.c) is used for the codec-specific implementations. The details about the codec-specific problems are explained in the later sections.

If you are interested in the deep debugging of HD-audio, read the HD-audio specification at first. The specification is found on Intel's web page, for example:

- https://www.intel.com/standards/hdaudio/

HD-Audio Controller

DMA-Position Problem

The most common problem of the controller is the inaccurate DMA pointer reporting. The DMA pointer for playback and capture can be read in two ways, either via a LPIB register or via a position-buffer map. As default the driver tries to read from the io-mapped position-buffer, and falls back to LPIB if the position-buffer appears dead. However, this detection isn't perfect on some devices. In such a case, you can change the default method via `position_fix` option.

`position_fix=1` means to use LPIB method explicitly. `position_fix=2` means to use the position-buffer. `position_fix=3` means to use a combination of both methods, needed for some VIA controllers. The capture stream position is corrected by comparing both LPIB and position-buffer values. `position_fix=4` is another combination available for all controllers, and uses LPIB for the playback and the position-buffer for the capture streams. `position_fix=5` is specific to Intel platforms, so far, for Skylake and onward. It applies the delay calculation for the precise position reporting. `position_fix=6` is to correct the position with the fixed FIFO size, mainly targeted for the recent AMD controllers. 0 is the default value for all other controllers, the automatic check and fallback to LPIB as described in the above. If you get a problem of repeated sounds, this option might help.

In addition to that, every controller is known to be broken regarding the wake-up timing. It wakes up a few samples before actually processing the data on the buffer. This caused a lot of problems, for example, with ALSA dmix or JACK. Since 2.6.27 kernel, the driver puts an artificial delay to the wake up timing. This delay is controlled via `bdl_pos_adj` option.

When `bdl_pos_adj` is a negative value (as default), it's assigned to an appropriate value depending on the controller chip. For Intel chips, it'd be 1 while it'd be 32 for others. Usually this works. Only in case it doesn't work and you get warning messages, you should change this parameter to other values.

Codec-Probing Problem

A less often but a more severe problem is the codec probing. When BIOS reports the available codec slots wrongly, the driver gets confused and tries to access the non-existing codec slot. This often results in the total screw-up, and destructs the further communication with the codec chips. The symptom appears usually as error messages like:

```
hda_intel: azx_get_response timeout, switching to polling mode:
      last cmd=0x12345678
hda_intel: azx_get_response timeout, switching to single_cmd mode:
      last cmd=0x12345678
```

The first line is a warning, and this is usually relatively harmless. It means that the codec response isn't notified via an IRQ. The driver uses explicit polling method to read the response. It gives very slight CPU overhead, but you'd unlikely notice it.

The second line is, however, a fatal error. If this happens, usually it means that something is really wrong. Most likely you are accessing a non-existing codec slot.

Thus, if the second error message appears, try to narrow the probed codec slots via `probe_mask` option. It's a bitmask, and each bit corresponds to the codec slot. For example, to probe only the first slot, pass `probe_mask=1`. For the first and the third slots, pass `probe_mask=5` (where 5 = 1 | 4), and so on.

Since 2.6.29 kernel, the driver has a more robust probing method, so this error might happen rarely, though.

On a machine with a broken BIOS, sometimes you need to force the driver to probe the codec slots the hardware doesn't report for use. In such a case, turn the bit 8 (0x100) of `probe_mask` option on. Then the rest 8 bits are passed as the codec slots to probe unconditionally. For example, `probe_mask=0x103` will force to probe the codec slots 0 and 1 no matter what the hardware reports.

Interrupt Handling

HD-audio driver uses MSI as default (if available) since 2.6.33 kernel as MSI works better on some machines, and in general, it's better for performance. However, Nvidia controllers showed bad regressions with MSI (especially in a combination with AMD chipset), thus we disabled MSI for them.

There seem also still other devices that don't work with MSI. If you see a regression wrt the sound quality (stuttering, etc) or a lock-up in the recent kernel, try to pass `enable_msi=0` option to disable MSI. If it works, you can add the known bad device to the blacklist defined in hda_intel.c. In such a case, please report and give the patch back to the upstream developer.

HD-Audio Codec

Model Option

The most common problem regarding the HD-audio driver is the unsupported codec features or the mismatched device configuration. Most of codec-specific code has several preset models, either to override the BIOS setup or to provide more comprehensive features.

The driver checks PCI SSID and looks through the static configuration table until any matching entry is found. If you have a new machine, you may see a message like below:

```
hda_codec: ALC880: BIOS auto-probing.
```

Meanwhile, in the earlier versions, you would see a message like:

```
hda_codec: Unknown model for ALC880, trying auto-probe from BIOS...
```

Even if you see such a message, DON'T PANIC. Take a deep breath and keep your towel. First of all, it's an informational message, no warning, no error. This means that the PCI SSID of your device isn't listed in the known preset model (white-)list. But, this doesn't mean that the driver is broken. Many codec-drivers provide the automatic configuration mechanism based on the BIOS setup.

The HD-audio codec has usually "pin" widgets, and BIOS sets the default configuration of each pin, which indicates the location, the connection type, the jack color, etc. The HD-audio driver can guess the right connection judging from these default configuration values. However -- some codec-support codes, such as patch_analog.c, don't support the automatic probing (yet as of 2.6.28). And, BIOS is often, yes, pretty often broken. It sets up wrong values and screws up the driver.

The preset model (or recently called as "fix-up") is provided basically to overcome such a situation. When the matching preset model is found in the white-list, the driver assumes the static configuration of that preset with the correct pin setup, etc. Thus, if you have a newer machine with a slightly different PCI SSID (or codec SSID) from the existing one, you may have a good chance to re-use the same model. You can pass the `model` option to specify the preset model instead of PCI (and codec-) SSID look-up.

What `model` option values are available depends on the codec chip. Check your codec chip from the codec proc file (see "Codec Proc-File" section below). It will show the vendor/product name of your codec chip. Then, see Documentation/sound/hd-audio/models.rst file, the section of HD-audio driver. You can find a list of codecs and `model` options belonging to each codec. For example, for Realtek ALC262 codec chip, pass `model=ultra` for devices that are compatible with Samsung Q1 Ultra.

Thus, the first thing you can do for any brand-new, unsupported and non-working HD-audio hardware is to check HD-audio codec and several different `model` option values. If you have any luck, some of them might suit with your device well.

There are a few special model option values:

- when 'nofixup' is passed, the device-specific fixups in the codec parser are skipped.
- when `generic` is passed, the codec-specific parser is skipped and only the generic parser is used.

A new style for the model option that was introduced since 5.15 kernel is to pass the PCI or codec SSID in the form of `model=XXXX:YYYY` where XXXX and YYYY are the sub-vendor and sub-device IDs in hex numbers, respectively. This is a kind of aliasing to another device; when this form is given, the driver will refer to that SSID as a reference to the quirk table. It'd be useful especially when the target quirk isn't listed in the model table. For example, passing model=103c:8862 will apply the quirk for HP ProBook 445 G8 (which isn't found in the model table as of writing) as long as the device is handled equivalently by the same driver.

Speaker and Headphone Output

One of the most frequent (and obvious) bugs with HD-audio is the silent output from either or both of a built-in speaker and a headphone jack. In general, you should try a headphone output at first. A speaker output often requires more additional controls like

the external amplifier bits. Thus a headphone output has a slightly better chance.

Before making a bug report, double-check whether the mixer is set up correctly. The recent version of snd-hda-intel driver provides mostly "Master" volume control as well as "Front" volume (where Front indicates the front-channels). In addition, there can be individual "Headphone" and "Speaker" controls.

Ditto for the speaker output. There can be "External Amplifier" switch on some codecs. Turn on this if present.

Another related problem is the automatic mute of speaker output by headphone plugging. This feature is implemented in most cases, but not on every preset model or codec-support code.

In anyway, try a different model option if you have such a problem. Some other models may match better and give you more matching functionality. If none of the available models works, send a bug report. See the bug report section for details.

If you are masochistic enough to debug the driver problem, note the following:

- The speaker (and the headphone, too) output often requires the external amplifier. This can be set usually via EAPD verb or a certain GPIO. If the codec pin supports EAPD, you have a better chance via SET_EAPD_BTL verb (0x70c). On others, GPIO pin (mostly it's either GPIO0 or GPIO1) may turn on/off EAPD.
- Some Realtek codecs require special vendor-specific coefficients to turn on the amplifier. See patch_realtek.c.
- IDT codecs may have extra power-enable/disable controls on each analog pin. See patch_sigmatel.c.
- Very rare but some devices don't accept the pin-detection verb until triggered. Issuing GET_PIN_SENSE verb (0xf09) may result in the codec-communication stall. Some examples are found in patch_realtek.c.

Capture Problems

The capture problems are often because of missing setups of mixers. Thus, before submitting a bug report, make sure that you set up the mixer correctly. For example, both "Capture Volume" and "Capture Switch" have to be set properly in addition to the right "Capture Source" or "Input Source" selection. Some devices have "Mic Boost" volume or switch.

When the PCM device is opened via "default" PCM (without pulse-audio plugin), you'll likely have "Digital Capture Volume" control as well. This is provided for the extra gain/attenuation of the signal in software, especially for the inputs without the hardware volume control such as digital microphones. Unless really needed, this should be set to exactly 50%, corresponding to 0dB -- neither extra gain nor attenuation. When you use "hw" PCM, i.e., a raw access PCM, this control will have no influence, though.

It's known that some codecs / devices have fairly bad analog circuits, and the recorded sound contains a certain DC-offset. This is no bug of the driver.

Most of modern laptops have no analog CD-input connection. Thus, the recording from CD input won't work in many cases although the driver provides it as the capture source. Use CDDA instead.

The automatic switching of the built-in and external mic per plugging is implemented on some codec models but not on every model. Partly because of my laziness but mostly lack of testers. Feel free to submit the improvement patch to the author.

Direct Debugging

If no model option gives you a better result, and you are a tough guy to fight against evil, try debugging via hitting the raw HD-audio codec verbs to the device. Some tools are available: hda-emu and hda-analyzer. The detailed description is found in the sections below. You'd need to enable hwdep for using these tools. See "Kernel Configuration" section.

Other Issues

Kernel Configuration

In general, I recommend you to enable the sound debug option, `CONFIG_SND_DEBUG=y`, no matter whether you are debugging or not. This enables snd_printd() macro and others, and you'll get additional kernel messages at probing.

In addition, you can enable `CONFIG_SND_DEBUG_VERBOSE=y`. But this will give you far more messages. Thus turn this on only when you are sure to want it.

Don't forget to turn on the appropriate `CONFIG_SND_HDA_CODEC_*` options. Note that each of them corresponds to the codec chip, not the controller chip. Thus, even if lspci shows the Nvidia controller, you may need to choose the option for other vendors. If you are unsure, just select all yes.

`CONFIG_SND_HDA_HWDEP` is a useful option for debugging the driver. When this is enabled, the driver creates hardware-dependent devices (one per each codec), and you have a raw access to the device via these device files. For example, `hwC0D2` will be created for the codec slot #2 of the first card (#0). For debug-tools such as hda-verb and hda-analyzer, the hwdep device has to be enabled. Thus, it'd be better to turn this on always.

`CONFIG_SND_HDA_RECONFIG` is a new option, and this depends on the hwdep option above. When enabled, you'll have some sysfs files under the corresponding hwdep directory. See "HD-audio reconfiguration" section below.

`CONFIG_SND_HDA_POWER_SAVE` option enables the power-saving feature. See "Power-saving" section below.

Codec Proc-File

The codec proc-file is a treasure-chest for debugging HD-audio. It shows most of useful information of each codec widget.

The proc file is located in /proc/asound/card*/codec#*, one file per each codec slot. You can know the codec vendor, product id and names, the type of each widget, capabilities and so on. This file, however, doesn't show the jack sensing state, so far. This is because the jack-sensing might be depending on the trigger state.

This file will be picked up by the debug tools, and also it can be fed to the emulator as the primary codec information. See the debug tools section below.

This proc file can be also used to check whether the generic parser is used. When the generic parser is used, the vendor/product ID name will appear as "Realtek ID 0262", instead of "Realtek ALC262".

HD-Audio Reconfiguration

This is an experimental feature to allow you re-configure the HD-audio codec dynamically without reloading the driver. The following sysfs files are available under each codec-hwdep device directory (e.g. /sys/class/sound/hwC0D0):

vendor_id
: Shows the 32bit codec vendor-id hex number. You can change the vendor-id value by writing to this file.

subsystem_id
: Shows the 32bit codec subsystem-id hex number. You can change the subsystem-id value by writing to this file.

revision_id
: Shows the 32bit codec revision-id hex number. You can change the revision-id value by writing to this file.

afg
: Shows the AFG ID. This is read-only.

mfg
: Shows the MFG ID. This is read-only.

name
: Shows the codec name string. Can be changed by writing to this file.

modelname
: Shows the currently set `model` option. Can be changed by writing to this file.

init_verbs
: The extra verbs to execute at initialization. You can add a verb by writing to this file. Pass three numbers: nid, verb and parameter (separated with a space).

hints
: Shows / stores hint strings for codec parsers for any use. Its format is `key = value`. For example, passing `jack_detect = no` will disable the jack detection of the machine completely.

init_pin_configs
: Shows the initial pin default config values set by BIOS.

driver_pin_configs
: Shows the pin default values set by the codec parser explicitly. This doesn't show all pin values but only the changed values by the parser. That is, if the parser doesn't change the pin default config values by itself, this will contain nothing.

user_pin_configs
: Shows the pin default config values to override the BIOS setup. Writing this (with two numbers, NID and value) appends the new value. The given will be used instead of the initial BIOS value at the next reconfiguration time. Note that this config will override even the driver pin configs, too.

reconfig
: Triggers the codec re-configuration. When any value is written to this file, the driver re-initialize and parses the codec tree again. All the changes done by the sysfs entries above are taken into account.

clear
: Resets the codec, removes the mixer elements and PCM stuff of the specified codec, and clear all init verbs and hints.

For example, when you want to change the pin default configuration value of the pin widget 0x14 to 0x9993013f, and let the driver re-configure based on that state, run like below:

```
# echo 0x14 0x9993013f > /sys/class/sound/hwC0D0/user_pin_configs
# echo 1 > /sys/class/sound/hwC0D0/reconfig
```

Hint Strings

The codec parser have several switches and adjustment knobs for matching better with the actual codec or device behavior. Many of them can be adjusted dynamically via "hints" strings as mentioned in the section above. For example, by passing `jack_detect = no` string via sysfs or a patch file, you can disable the jack detection, thus the codec parser will skip the features like auto-mute or mic auto-switch. As a boolean value, either `yes`, `no`, `true`, `false`, `1` or `0` can be passed.

The generic parser supports the following hints:

jack_detect (bool)
: specify whether the jack detection is available at all on this machine; default true

inv_jack_detect (bool)
: indicates that the jack detection logic is inverted

trigger_sense (bool)
: indicates that the jack detection needs the explicit call of AC_VERB_SET_PIN_SENSE verb

inv_eapd (bool)
: indicates that the EAPD is implemented in the inverted logic

pcm_format_first (bool)
: sets the PCM format before the stream tag and channel ID

sticky_stream (bool)
: keep the PCM format, stream tag and ID as long as possible; default true

spdif_status_reset (bool)
: reset the SPDIF status bits at each time the SPDIF stream is set up

pin_amp_workaround (bool)
: the output pin may have multiple amp values

single_adc_amp (bool)
: ADCs can have only single input amps

auto_mute (bool)
: enable/disable the headphone auto-mute feature; default true

auto_mic (bool)
: enable/disable the mic auto-switch feature; default true

line_in_auto_switch (bool)
: enable/disable the line-in auto-switch feature; default false

need_dac_fix (bool)
: limits the DACs depending on the channel count

primary_hp (bool)
: probe headphone jacks as the primary outputs; default true

multi_io (bool)
: try probing multi-I/O config (e.g. shared line-in/surround, mic/clfe jacks)

multi_cap_vol (bool)
: provide multiple capture volumes

inv_dmic_split (bool)
: provide split internal mic volume/switch for phase-inverted digital mics

indep_hp (bool)
: provide the independent headphone PCM stream and the corresponding mixer control, if available

add_stereo_mix_input (bool)
: add the stereo mix (analog-loopback mix) to the input mux if available

add_jack_modes (bool)
: add "xxx Jack Mode" enum controls to each I/O jack for allowing to change the headphone amp and mic bias VREF capabilities

power_save_node (bool)
: advanced power management for each widget, controlling the power sate (D0/D3) of each widget node depending on the actual pin and stream states

power_down_unused (bool)
: power down the unused widgets, a subset of power_save_node, and will be dropped in future

add_hp_mic (bool)
: add the headphone to capture source if possible

hp_mic_detect (bool)
: enable/disable the hp/mic shared input for a single built-in mic case; default true

vmaster (bool)
: enable/disable the virtual Master control; default true

mixer_nid (int)
: specifies the widget NID of the analog-loopback mixer

Early Patching

When `CONFIG_SND_HDA_PATCH_LOADER=y` is set, you can pass a "patch" as a firmware file for modifying the HD-audio setup before initializing the codec. This can work basically like the reconfiguration via sysfs in the above, but it does it before the first codec configuration.

A patch file is a plain text file which looks like below:

```
[codec]
0x12345678 0xabcd1234 2

[model]
auto

[pincfg]
0x12 0x411111f0
```

```
[verb]
0x20 0x500 0x03
0x20 0x400 0xff

[hint]
jack_detect = no
```

The file needs to have a line `[codec]`. The next line should contain three numbers indicating the codec vendor-id (0x12345678 in the example), the codec subsystem-id (0xabcd1234) and the address (2) of the codec. The rest patch entries are applied to this specified codec until another codec entry is given. Passing 0 or a negative number to the first or the second value will make the check of the corresponding field be skipped. It'll be useful for really broken devices that don't initialize SSID properly.

The `[model]` line allows to change the model name of the each codec. In the example above, it will be changed to model=auto. Note that this overrides the module option.

After the `[pincfg]` line, the contents are parsed as the initial default pin-configurations just like `user_pin_configs` sysfs above. The values can be shown in user_pin_configs sysfs file, too.

Similarly, the lines after `[verb]` are parsed as `init_verbs` sysfs entries, and the lines after `[hint]` are parsed as `hints` sysfs entries, respectively.

Another example to override the codec vendor id from 0x12345678 to 0xdeadbeef is like below:

```
[codec]
0x12345678 0xabcd1234 2

[vendor_id]
0xdeadbeef
```

In the similar way, you can override the codec subsystem_id via `[subsystem_id]`, the revision id via `[revision_id]` line. Also, the codec chip name can be rewritten via `[chip_name]` line.

```
[codec]
0x12345678 0xabcd1234 2

[subsystem_id]
0xffff1111

[revision_id]
0x10

[chip_name]
My-own NEWS-0002
```

The hd-audio driver reads the file via request_firmware(). Thus, a patch file has to be located on the appropriate firmware path, typically, /lib/firmware. For example, when you pass the option `patch=hda-init.fw`, the file /lib/firmware/hda-init.fw must be present.

The patch module option is specific to each card instance, and you need to give one file name for each instance, separated by commas. For example, if you have two cards, one for an on-board analog and one for an HDMI video board, you may pass patch option like below:

```
options snd-hda-intel patch=on-board-patch,hdmi-patch
```

Power-Saving

The power-saving is a kind of auto-suspend of the device. When the device is inactive for a certain time, the device is automatically turned off to save the power. The time to go down is specified via `power_save` module option, and this option can be changed dynamically via sysfs.

The power-saving won't work when the analog loopback is enabled on some codecs. Make sure that you mute all unneeded signal routes when you want the power-saving.

The power-saving feature might cause audible click noises at each power-down/up depending on the device. Some of them might be solvable, but some are hard, I'm afraid. Some distros such as openSUSE enables the power-saving feature automatically when the power cable is unplugged. Thus, if you hear noises, suspect first the power-saving. See /sys/module/snd_hda_intel/parameters/power_save to check the current value. If it's non-zero, the feature is turned on.

The recent kernel supports the runtime PM for the HD-audio controller chip, too. It means that the HD-audio controller is also powered up / down dynamically. The feature is enabled only for certain controller chips like Intel LynxPoint. You can enable/disable this feature forcibly by setting `power_save_controller` option, which is also available at /sys/module/snd_hda_intel/parameters directory.

Tracepoints

The hd-audio driver gives a few basic tracepoints. `hda:hda_send_cmd` traces each CORB write while `hda:hda_get_response` traces the response from RIRB (only when read from the codec driver). `hda:hda_bus_reset` traces the bus-reset due to fatal

error, etc, `hda:hda_unsol_event` traces the unsolicited events, and `hda:hda_power_down` and `hda:hda_power_up` trace the power down/up via power-saving behavior.

Enabling all tracepoints can be done like

```
# echo 1 > /sys/kernel/debug/tracing/events/hda/enable
```

then after some commands, you can traces from /sys/kernel/debug/tracing/trace file. For example, when you want to trace what codec command is sent, enable the tracepoint like:

```
# cat /sys/kernel/debug/tracing/trace
# tracer: nop
#
#       TASK-PID    CPU#    TIMESTAMP  FUNCTION
#          | |       |          |         |
       <...>-7807  [002] 105147.774889: hda_send_cmd: [0:0] val=e3a019
       <...>-7807  [002] 105147.774893: hda_send_cmd: [0:0] val=e39019
       <...>-7807  [002] 105147.999542: hda_send_cmd: [0:0] val=e3a01a
       <...>-7807  [002] 105147.999543: hda_send_cmd: [0:0] val=e3901a
       <...>-26764 [001] 349222.837143: hda_send_cmd: [0:0] val=e3a019
       <...>-26764 [001] 349222.837148: hda_send_cmd: [0:0] val=e39019
       <...>-26764 [001] 349223.058539: hda_send_cmd: [0:0] val=e3a01a
       <...>-26764 [001] 349223.058541: hda_send_cmd: [0:0] val=e3901a
```

Here `[0:0]` indicates the card number and the codec address, and `val` shows the value sent to the codec, respectively. The value is a packed value, and you can decode it via hda-decode-verb program included in hda-emu package below. For example, the value e3a019 is to set the left output-amp value to 25.

```
% hda-decode-verb 0xe3a019
raw value = 0x00e3a019
cid = 0, nid = 0x0e, verb = 0x3a0, parm = 0x19
raw value: verb = 0x3a0, parm = 0x19
verbname = set_amp_gain_mute
amp raw val = 0xa019
output, left, idx=0, mute=0, val=25
```

Development Tree

The latest development codes for HD-audio are found on sound git tree:

- git://git.kernel.org/pub/scm/linux/kernel/git/tiwai/sound.git

The master branch or for-next branches can be used as the main development branches in general while the development for the current and next kernels are found in for-linus and for-next branches, respectively.

Sending a Bug Report

If any model or module options don't work for your device, it's time to send a bug report to the developers. Give the following in your bug report:

- Hardware vendor, product and model names
- Kernel version (and ALSA-driver version if you built externally)
- `alsa-info.sh` output; run with `--no-upload` option. See the section below about alsa-info

If it's a regression, at best, send alsa-info outputs of both working and non-working kernels. This is really helpful because we can compare the codec registers directly.

Send a bug report either the following:

kernel-bugzilla
: https://bugzilla.kernel.org/

alsa-devel ML
: alsa-devel@alsa-project.org

Debug Tools

This section describes some tools available for debugging HD-audio problems.

alsa-info

The script `alsa-info.sh` is a very useful tool to gather the audio device information. It's included in alsa-utils package. The latest version can be found on git repository:

- git://git.alsa-project.org/alsa-utils.git

The script can be fetched directly from the following URL, too:

- https://www.alsa-project.org/alsa-info.sh

Run this script as root, and it will gather the important information such as the module lists, module parameters, proc file contents including the codec proc files, mixer outputs and the control elements. As default, it will store the information onto a web server on alsa-project.org. But, if you send a bug report, it'd be better to run with `--no-upload` option, and attach the generated file.

There are some other useful options. See `--help` option output for details.

When a probe error occurs or when the driver obviously assigns a mismatched model, it'd be helpful to load the driver with `probe_only=1` option (at best after the cold reboot) and run alsa-info at this state. With this option, the driver won't configure the mixer and PCM but just tries to probe the codec slot. After probing, the proc file is available, so you can get the raw codec information before modified by the driver. Of course, the driver isn't usable with `probe_only=1`. But you can continue the configuration via hwdep sysfs file if hda-reconfig option is enabled. Using `probe_only` mask 2 skips the reset of HDA codecs (use `probe_only=3` as module option). The hwdep interface can be used to determine the BIOS codec initialization.

hda-verb

hda-verb is a tiny program that allows you to access the HD-audio codec directly. You can execute a raw HD-audio codec verb with this. This program accesses the hwdep device, thus you need to enable the kernel config `CONFIG_SND_HDA_HWDEP=y` beforehand.

The hda-verb program takes four arguments: the hwdep device file, the widget NID, the verb and the parameter. When you access to the codec on the slot 2 of the card 0, pass /dev/snd/hwC0D2 to the first argument, typically. (However, the real path name depends on the system.)

The second parameter is the widget number-id to access. The third parameter can be either a hex/digit number or a string corresponding to a verb. Similarly, the last parameter is the value to write, or can be a string for the parameter type.

```
% hda-verb /dev/snd/hwC0D0 0x12 0x701 2
nid = 0x12, verb = 0x701, param = 0x2
value = 0x0

% hda-verb /dev/snd/hwC0D0 0x0 PARAMETERS VENDOR_ID
nid = 0x0, verb = 0xf00, param = 0x0
value = 0x10ec0262

% hda-verb /dev/snd/hwC0D0 2 set_a 0xb080
nid = 0x2, verb = 0x300, param = 0xb080
value = 0x0
```

Although you can issue any verbs with this program, the driver state won't be always updated. For example, the volume values are usually cached in the driver, and thus changing the widget amp value directly via hda-verb won't change the mixer value.

The hda-verb program is included now in alsa-tools:

- git://git.alsa-project.org/alsa-tools.git

Also, the old stand-alone package is found in the ftp directory:

- ftp://ftp.suse.com/pub/people/tiwai/misc/

Also a git repository is available:

- git://git.kernel.org/pub/scm/linux/kernel/git/tiwai/hda-verb.git

See README file in the tarball for more details about hda-verb program.

hda-analyzer

hda-analyzer provides a graphical interface to access the raw HD-audio control, based on pyGTK2 binding. It's a more powerful version of hda-verb. The program gives you an easy-to-use GUI stuff for showing the widget information and adjusting the amp values, as well as the proc-compatible output.

The hda-analyzer:

- https://git.alsa-project.org/?p=alsa.git;a=tree;f=hda-analyzer

is a part of alsa.git repository in alsa-project.org:

- git://git.alsa-project.org/alsa.git

Codecgraph

Codecgraph is a utility program to generate a graph and visualizes the codec-node connection of a codec chip. It's especially useful when you analyze or debug a codec without a proper datasheet. The program parses the given codec proc file and converts to SVG via graphiz program.

The tarball and GIT trees are found in the web page at:

- http://helllabs.org/codecgraph/

hda-emu

hda-emu is an HD-audio emulator. The main purpose of this program is to debug an HD-audio codec without the real hardware. Thus, it doesn't emulate the behavior with the real audio I/O, but it just dumps the codec register changes and the ALSA-driver internal changes at probing and operating the HD-audio driver.

The program requires a codec proc-file to simulate. Get a proc file for the target codec beforehand, or pick up an example codec from the codec proc collections in the tarball. Then, run the program with the proc file, and the hda-emu program will start parsing the codec file and simulates the HD-audio driver:

```
% hda-emu codecs/stac9200-dell-d820-laptop
# Parsing..
hda_codec: Unknown model for STAC9200, using BIOS defaults
hda_codec: pin nid 08 bios pin config 40c003fa
....
```

The program gives you only a very dumb command-line interface. You can get a proc-file dump at the current state, get a list of control (mixer) elements, set/get the control element value, simulate the PCM operation, the jack plugging simulation, etc.

The program is found in the git repository below:

- git://git.kernel.org/pub/scm/linux/kernel/git/tiwai/hda-emu.git

See README file in the repository for more details about hda-emu program.

hda-jack-retask

hda-jack-retask is a user-friendly GUI program to manipulate the HD-audio pin control for jack retasking. If you have a problem about the jack assignment, try this program and check whether you can get useful results. Once when you figure out the proper pin assignment, it can be fixed either in the driver code statically or via passing a firmware patch file (see "Early Patching" section).

The program is included in alsa-tools now:

- git://git.alsa-project.org/alsa-tools.git