Notes on Universal Interface for Intel High Definition Audio Codec

Takashi Iwai <tiwai@suse.de>

[Still a draft version]

General

The snd-hda-codec module supports the generic access function for the High Definition (HD) audio codecs. It's designed to be independent from the controller code like ac97 codec module. The real accessors from/to the controller must be implemented in the lowlevel driver.

The structure of this module is similar with ac97_codec module.
Each codec chip belongs to a bus class which communicates with the controller.

Initialization of Bus Instance

The card driver has to create struct hda_bus at first. The template struct should be filled and passed to the constructor:

```
struct hda_bus_template {
        void *private_data;
        struct pci_dev *pci;
        const char *modelname;
        struct hda_bus_ops ops;
};
```

The card driver can set and use the private_data field to retrieve its own data in callback functions. The pci field is used when the patch needs to check the PCI subsystem IDs, so on. For non-PCI system, it doesn't have to be set, of course.
The modelname field specifies the board's specific configuration. The string is passed to the codec parser, and it depends on the parser how the string is used.
These fields, private_data, pci and modelname are all optional.

The ops field contains the callback functions as the following:

```
struct hda_bus_ops {
        int (*command)(struct hda_codec *codec, hda_nid_t nid, int direct,
                       unsigned int verb, unsigned int parm);
        unsigned int (*get_response)(struct hda_codec *codec);
        void (*private_free)(struct hda_bus *);
#ifdef CONFIG_SND_HDA_POWER_SAVE
        void (*pm_notify)(struct hda_codec *codec);
#endif
};
```

The command callback is called when the codec module needs to send a
VERB to the controller. It's always a single command.
The get_response callback is called when the codec requires the answer
for the last command. These two callbacks are mandatory and have to
be given.
The third, private_free callback, is optional. It's called in the
destructor to release any necessary data in the lowlevel driver.

The pm_notify callback is available only with
CONFIG_SND_HDA_POWER_SAVE kconfig. It's called when the codec needs
to power up or may power down. The controller should check the all
belonging codecs on the bus whether they are actually powered off
(check codec->power_on), and optionally the driver may power down the
controller side, too.

The bus instance is created via snd_hda_bus_new(). You need to pass
the card instance, the template, and the pointer to store the
resultant bus instance.

```
int snd_hda_bus_new(struct snd_card *card, const struct hda_bus_template *temp,
                    struct hda_bus **busp);
```

It returns zero if successful. A negative return value means any
error during creation.

Creation of Codec Instance
==========================

Each codec chip on the board is then created on the BUS instance.
To create a codec instance, call snd_hda_codec_new().

```
int snd_hda_codec_new(struct hda_bus *bus, unsigned int codec_addr,
                      struct hda_codec **codecp);
```

The first argument is the BUS instance, the second argument is the
address of the codec, and the last one is the pointer to store the
resultant codec instance (can be NULL if not needed).

The codec is stored in a linked list of bus instance. You can follow
the codec list like:

```
        struct hda_codec *codec;
        list_for_each_entry(codec, &bus->codec_list, list) {
                ...
        }
```

The codec isn't initialized at this stage properly. The
initialization sequence is called when the controls are built later.

Codec Access
============

To access codec, use snd_hda_codec_read() and snd_hda_codec_write().
snd_hda_param_read() is for reading parameters.

For writing a sequence of verbs, use snd_hda_sequence_write().

There are variants of cached read/write, snd_hda_codec_write_cache(), snd_hda_sequence_write_cache(). These are used for recording the register states for the power-management resume. When no PM is needed, these are equivalent with non-cached version.

To retrieve the number of sub nodes connected to the given node, use snd_hda_get_sub_nodes(). The connection list can be obtained via snd_hda_get_connections() call.

When an unsolicited event happens, pass the event via snd_hda_queue_unsol_event() so that the codec routines will process it later.

(Mixer) Controls
================

To create mixer controls of all codecs, call snd_hda_build_controls(). It then builds the mixers and does initialization stuff on each codec.

PCM Stuff
=========

snd_hda_build_pcms() gives the necessary information to create PCM streams. When it's called, each codec belonging to the bus stores codec->num_pcms and codec->pcm_info fields. The num_pcms indicates the number of elements in pcm_info array. The card driver is supposed to traverse the codec linked list, read the pcm information in pcm_info array, and build pcm instances according to them.

The pcm_info array contains the following record:

```
/* PCM information for each substream */
struct hda_pcm_stream {
        unsigned int substreams;        /* number of substreams, 0 = not exist
*/
        unsigned int channels_min;      /* min. number of channels */
        unsigned int channels_max;      /* max. number of channels */
        hda_nid_t nid;  /* default NID to query rates/formats/bps, or set up */
        u32 rates;      /* supported rates */
        u64 formats;    /* supported formats (SNDRV_PCM_FMTBIT_) */
        unsigned int maxbps;    /* supported max. bit per sample */
        struct hda_pcm_ops ops;
};

/* for PCM creation */
struct hda_pcm {
        char *name;
        struct hda_pcm_stream stream[2];
};
```

The name can be passed to snd_pcm_new(). The stream field contains

the information for playback (SNDRV_PCM_STREAM_PLAYBACK = 0) and capture (SNDRV_PCM_STREAM_CAPTURE = 1) directions. The card driver should pass substreams to snd_pcm_new() for the number of substreams to create.

The channels_min, channels_max, rates and formats should be copied to runtime->hw record. They and maxbps fields are used also to compute the format value for the HDA codec and controller. Call snd_hda_calc_stream_format() to get the format value.

The ops field contains the following callback functions:

```
struct hda_pcm_ops {
        int (*open)(struct hda_pcm_stream *info, struct hda_codec *codec,
                    struct snd_pcm_substream *substream);
        int (*close)(struct hda_pcm_stream *info, struct hda_codec *codec,
                     struct snd_pcm_substream *substream);
        int (*prepare)(struct hda_pcm_stream *info, struct hda_codec *codec,
                       unsigned int stream_tag, unsigned int format,
                       struct snd_pcm_substream *substream);
        int (*cleanup)(struct hda_pcm_stream *info, struct hda_codec *codec,
                       struct snd_pcm_substream *substream);
};
```

All are non-NULL, so you can call them safely without NULL check.

The open callback should be called in PCM open after runtime->hw is set up. It may override some setting and constraints additionally. Similarly, the close callback should be called in the PCM close.

The prepare callback should be called in PCM prepare. This will set up the codec chip properly for the operation. The cleanup should be called in hw_free to clean up the configuration.

The caller should check the return value, at least for open and prepare callbacks. When a negative value is returned, some error occurred.

Proc Files

Each codec dumps the widget node information in /proc/asound/card*/codec#* file. This information would be really helpful for debugging. Please provide its contents together with the bug report.

Power Management

It's simple:
Call snd_hda_suspend() in the PM suspend callback.
Call snd_hda_resume() in the PM resume callback.

Codec Preset (Patch)
====================

To set up and handle the codec functionality fully, each codec may
have a codec preset (patch). It's defined in struct hda_codec_preset:

```
	struct hda_codec_preset {
		unsigned int id;
		unsigned int mask;
		unsigned int subs;
		unsigned int subs_mask;
		unsigned int rev;
		const char *name;
		int (*patch)(struct hda_codec *codec);
	};
```

When the codec id and codec subsystem id match with the given id and
subs fields bitwise (with bitmask mask and subs_mask), the callback
patch is called. The patch callback should initialize the codec and
set the codec->patch_ops field. This is defined as below:

```
	struct hda_codec_ops {
		int (*build_controls)(struct hda_codec *codec);
		int (*build_pcms)(struct hda_codec *codec);
		int (*init)(struct hda_codec *codec);
		void (*free)(struct hda_codec *codec);
		void (*unsol_event)(struct hda_codec *codec, unsigned int res);
	#ifdef CONFIG_PM
		int (*suspend)(struct hda_codec *codec, pm_message_t state);
		int (*resume)(struct hda_codec *codec);
	#endif
	#ifdef CONFIG_SND_HDA_POWER_SAVE
		int (*check_power_status)(struct hda_codec *codec,
					  hda_nid_t nid);
	#endif
	};
```

The build_controls callback is called from snd_hda_build_controls().
Similarly, the build_pcms callback is called from
snd_hda_build_pcms(). The init callback is called after
build_controls to initialize the hardware.
The free callback is called as a destructor.

The unsol_event callback is called when an unsolicited event is
received.

The suspend and resume callbacks are for power management.
They can be NULL if no special sequence is required. When the resume
callback is NULL, the driver calls the init callback and resumes the
registers from the cache. If other handling is needed, you'd need to
write your own resume callback. There, the amp values can be resumed
via

```
	void snd_hda_codec_resume_amp(struct hda_codec *codec);
```

and the other codec registers via

```
	void snd_hda_codec_resume_cache(struct hda_codec *codec);
```

The check_power_status callback is called when the amp value of the given widget NID is changed. The codec code can turn on/off the power appropriately from this information.

Each entry can be NULL if not necessary to be called.

Generic Parser

When the device doesn't match with any given presets, the widgets are parsed via th generic parser (hda_generic.c). Its support is limited: no multi-channel support, for example.

Digital I/O

Call snd_hda_create_spdif_out_ctls() from the patch to create controls related with SPDIF out.

Helper Functions

snd_hda_get_codec_name() stores the codec name on the given string.

snd_hda_check_board_config() can be used to obtain the configuration information matching with the device. Define the model string table and the table with struct snd_pci_quirk entries (zero-terminated), and pass it to the function. The function checks the modelname given as a module parameter, and PCI subsystem IDs. If the matching entry is found, it returns the config field value.

snd_hda_add_new_ctls() can be used to create and add control entries. Pass the zero-terminated array of struct snd_kcontrol_new

Macros HDA_CODEC_VOLUME(), HDA_CODEC_MUTE() and their variables can be used for the entry of struct snd_kcontrol_new.

The input MUX helper callbacks for such a control are provided, too: snd_hda_input_mux_info() and snd_hda_input_mux_put(). See patch_realtek.c for example.