The ALSA Driver API

The ALSA Driver API

Table of Contents

Chapter 1. Management of Cards and Devices

Card Management

snd_card_new

Name

`snd_card_new` — create and initialize a soundcard structure

Synopsis

```
struct snd_card * snd_card_new (int idx, const char * xid, struct module *
module, int extra_size);
```

Arguments

`idx`

card index (address) [0 ... (SNDRV_CARDS-1)]

`xid`

card identification (ASCII string)

`module`

top level module for locking

`extra_size`

allocate this extra size after the main soundcard structure

Description

Creates and initializes a soundcard structure.

Returns kmallocated snd_card structure. Creates the ALSA control interface (which is blocked until snd_card_register function is called).

snd_card_disconnect

Name

`snd_card_disconnect` — disconnect all APIs from the file-operations (user space)

Synopsis

```
int snd_card_disconnect (struct snd_card * card);
```

Arguments

card

soundcard structure

Description

Disconnects all APIs from the file-operations (user space).

Returns zero, otherwise a negative error code.

Note

The current implementation replaces all active file->f_op with special dummy file operations (they do nothing except release).

snd_card_register

Name

`snd_card_register` — register the soundcard

Synopsis

```
int snd_card_register (struct snd_card * card);
```

Arguments

card

soundcard structure

Description

This function registers all the devices assigned to the soundcard. Until calling this, the ALSA control interface is blocked from the external accesses. Thus, you should call this function at the end of the initialization of the card.

Returns zero otherwise a negative error code if the registrain failed.

snd_component_add

Name

`snd_component_add` — add a component string

Synopsis

```
int snd_component_add (struct snd_card * card, const char * component);
```

Arguments

card

soundcard structure

`component`

  the component id string

Description

This function adds the component id string to the supported list. The component can be referred from the alsa-lib.

Returns zero otherwise a negative error code.

snd_card_file_add

Name

`snd_card_file_add` — add the file to the file list of the card

Synopsis

```
int snd_card_file_add (struct snd_card * card, struct file * file);
```

Arguments

`card`

  soundcard structure

`file`

  file pointer

Description

This function adds the file to the file linked-list of the card. This linked-list is used to keep tracking the connection state, and to avoid the release of busy resources by hotplug.

Returns zero or a negative error code.

snd_card_file_remove

Name

`snd_card_file_remove` — remove the file from the file list

Synopsis

```
int snd_card_file_remove (struct snd_card * card, struct file * file);
```

Arguments

`card`

soundcard structure

`file`

file pointer

Description

This function removes the file formerly added to the card via `snd_card_file_add` function. If all files are removed and `snd_card_free_when_closed` was called beforehand, it processes the pending release of resources.

Returns zero or a negative error code.

snd_power_wait

Name

`snd_power_wait` — wait until the power-state is changed.

Synopsis

```
int snd_power_wait (struct snd_card * card, unsigned int power_state);
```

Arguments

card

soundcard structure

power_state

expected power state

Description

Waits until the power-state is changed.

Note

the power lock must be active before call.

Device Components

snd_device_new

Name

`snd_device_new` — create an ALSA device component

Synopsis

```
int snd_device_new (struct snd_card * card, snd_device_type_t type, void *
device_data, struct snd_device_ops * ops);
```

Arguments

`card`

the card instance

`type`

the device type, SNDRV_DEV_XXX

`device_data`

the data pointer of this device

`ops`

the operator table

Description

Creates a new device component for the given data pointer. The device will be assigned to the card and managed together by the card.

The data pointer plays a role as the identifier, too, so the pointer address must be unique and unchanged.

Returns zero if successful, or a negative error code on failure.

snd_device_free

Name

`snd_device_free` — release the device from the card

Synopsis

```
int snd_device_free (struct snd_card * card, void * device_data);
```

Arguments

`card`

the card instance

`device_data`

the data pointer to release

Description

Removes the device from the list on the card and invokes the callbacks, dev_disconnect and dev_free, corresponding to the state. Then release the device.

Returns zero if successful, or a negative error code on failure or if the device not found.

snd_device_register

Name

`snd_device_register` — register the device

Synopsis

```
int snd_device_register (struct snd_card * card, void * device_data);
```

Arguments

`card`

the card instance

`device_data`

the data pointer to register

Description

Registers the device which was already created via `snd_device_new`. Usually this is called from `snd_card_register`, but it can be called later if any new devices are created after invocation of `snd_card_register`.

Returns zero if successful, or a negative error code on failure or if the device not found.

KMOD and Device File Entries

snd_request_card

Name

`snd_request_card` — try to load the card module

Synopsis

```
void snd_request_card (int card);
```

Arguments

`card`

the card number

Description

Tries to load the module "snd-card-X" for the given card number via KMOD. Returns immediately if already loaded.

snd_lookup_minor_data

Name

`snd_lookup_minor_data` — get user data of a registered device

Synopsis

```
void * snd_lookup_minor_data (unsigned int minor, int type);
```

Arguments

`minor`

the minor number

`type`

device type (SNDRV_DEVICE_TYPE_XXX)

Description

Checks that a minor device with the specified type is registered, and returns its user data pointer.

snd_register_device_for_dev

Name

`snd_register_device_for_dev` — Register the ALSA device file for the card

Synopsis

```
int snd_register_device_for_dev (int type, struct snd_card * card, int dev,
const struct file_operations * f_ops, void * private_data, const char * name,
struct device * device);
```

Arguments

`type`

the device type, SNDRV_DEVICE_TYPE_XXX

`card`

the card instance

`dev`

the device index

`f_ops`

the file operations

`private_data`

user pointer for f_ops->`open`

`name`

the device file name

`device`

the &struct device to link this new device to

Description

Registers an ALSA device file for the given card. The operators have to be set in reg parameter.

Returns zero if successful, or a negative error code on failure.

snd_unregister_device

Name

`snd_unregister_device` — unregister the device on the given card

Synopsis

```
int snd_unregister_device (int type, struct snd_card * card, int dev);
```

Arguments

`type`

: the device type, SNDRV_DEVICE_TYPE_XXX

`card`

: the card instance

`dev`

: the device index

Description

Unregisters the device file already registered via `snd_register_device`.

Returns zero if sucecessful, or a negative error code on failure

Memory Management Helpers

copy_to_user_fromio

Name

`copy_to_user_fromio` — copy data from mmio-space to user-space

Synopsis

```
int copy_to_user_fromio (void __user * dst, const volatile void __iomem *
src, size_t count);
```

Arguments

`dst`

the destination pointer on user-space

`src`

the source pointer on mmio

`count`

the data size to copy in bytes

Description

Copies the data from mmio-space to user-space.

Returns zero if successful, or non-zero on failure.

copy_from_user_toio

Name

`copy_from_user_toio` — copy data from user-space to mmio-space

Synopsis

```
int copy_from_user_toio (volatile void __iomem * dst, const void __user *
src, size_t count);
```

Arguments

`dst`

the destination pointer on mmio-space

`src`

the source pointer on user-space

`count`

the data size to copy in bytes

Description

Copies the data from user-space to mmio-space.

Returns zero if successful, or non-zero on failure.

snd_malloc_pages

Name

`snd_malloc_pages` — allocate pages with the given size

Synopsis

```
void * snd_malloc_pages (size_t size, gfp_t gfp_flags);
```

Arguments

`size`

the size to allocate in bytes

`gfp_flags`

the allocation conditions, GFP_XXX

Description

Allocates the physically contiguous pages with the given size.

Returns the pointer of the buffer, or NULL if no enoguh memory.

snd_free_pages

Name

`snd_free_pages` — release the pages

Synopsis

```
void snd_free_pages (void * ptr, size_t size);
```

Arguments

ptr

the buffer pointer to release

size

the allocated buffer size

Description

Releases the buffer allocated via `snd_malloc_pages`.

snd_dma_alloc_pages

Name

`snd_dma_alloc_pages` — allocate the buffer area according to the given type

Synopsis

```
int snd_dma_alloc_pages (int type, struct device * device, size_t size,
struct snd_dma_buffer * dmab);
```

Arguments

type

the DMA buffer type

device

the device pointer

size

the buffer size to allocate

dmab

buffer allocation record to store the allocated data

Description

Calls the memory-allocator function for the corresponding buffer type.

Returns zero if the buffer with the given size is allocated successfuly, other a negative value at error.

snd_dma_alloc_pages_fallback

Name

`snd_dma_alloc_pages_fallback` — allocate the buffer area according to the given type with fallback

Synopsis

```
int snd_dma_alloc_pages_fallback (int type, struct device * device, size_t
size, struct snd_dma_buffer * dmab);
```

Arguments

`type`

  the DMA buffer type

`device`

  the device pointer

`size`

  the buffer size to allocate

`dmab`

  buffer allocation record to store the allocated data

Description

Calls the memory-allocator function for the corresponding buffer type. When no space is left, this function reduces the size and tries to allocate again. The size actually allocated is stored in res_size argument.

Returns zero if the buffer with the given size is allocated successfuly, other a negative value at error.

snd_dma_free_pages

Name

`snd_dma_free_pages` — release the allocated buffer

Synopsis

```
void snd_dma_free_pages (struct snd_dma_buffer * dmab);
```

Arguments

`dmab`

: the buffer allocation record to release

Description

Releases the allocated buffer via `snd_dma_alloc_pages`.

snd_dma_get_reserved_buf

Name

`snd_dma_get_reserved_buf` — get the reserved buffer for the given device

Synopsis

```
size_t snd_dma_get_reserved_buf (struct snd_dma_buffer * dmab, unsigned int
id);
```

Arguments

`dmab`

: the buffer allocation record to store

`id`

: the buffer id

Description

Looks for the reserved-buffer list and re-uses if the same buffer is found in the list. When the buffer is found, it's removed from the free list.

Returns the size of buffer if the buffer is found, or zero if not found.

snd_dma_reserve_buf

Name

`snd_dma_reserve_buf` — reserve the buffer

Synopsis

```
int snd_dma_reserve_buf (struct snd_dma_buffer * dmab, unsigned int id);
```

Arguments

dmab

the buffer to reserve

id

the buffer id

Description

Reserves the given buffer as a reserved buffer.

Returns zero if successful, or a negative code at error.

Chapter 2. PCM API

PCM Core

snd_pcm_new_stream

Name

`snd_pcm_new_stream` — create a new PCM stream

Synopsis

```
int snd_pcm_new_stream (struct snd_pcm * pcm, int stream, int
substream_count);
```

Arguments

pcm

the pcm instance

stream

the stream direction, SNDRV_PCM_STREAM_XXX

substream_count

the number of substreams

Description

Creates a new stream for the pcm. The corresponding stream on the pcm must have been empty before calling this, i.e. zero must be given to the argument of `snd_pcm_new`.

Returns zero if successful, or a negative error code on failure.

snd_pcm_new

Name

`snd_pcm_new` — create a new PCM instance

Synopsis

```
int snd_pcm_new (struct snd_card * card, char * id, int device, int
playback_count, int capture_count, struct snd_pcm ** rpcm);
```

Arguments

card

the card instance

id

the id string

device

the device index (zero based)

playback_count

the number of substreams for playback

capture_count

the number of substreams for capture

rpcm

the pointer to store the new pcm instance

Description

Creates a new PCM instance.

The pcm operators have to be set afterwards to the new instance via `snd_pcm_set_ops`.

Returns zero if successful, or a negative error code on failure.

snd_pcm_set_ops

Name

`snd_pcm_set_ops` — set the PCM operators

Synopsis

```
void snd_pcm_set_ops (struct snd_pcm * pcm, int direction, struct snd_pcm_ops
* ops);
```

Arguments

`pcm`

the pcm instance

`direction`

stream direction, SNDRV_PCM_STREAM_XXX

`ops`

the operator table

Description

Sets the given PCM operators to the pcm instance.

snd_pcm_set_sync

Name

`snd_pcm_set_sync` — set the PCM sync id

Synopsis

```
void snd_pcm_set_sync (struct snd_pcm_substream * substream);
```

Arguments

`substream`

: the pcm substream

Description

Sets the PCM sync identifier for the card.

snd_interval_refine

Name

`snd_interval_refine` — refine the interval value of configurator

Synopsis

```
int snd_interval_refine (struct snd_interval * i, const struct snd_interval *
v);
```

Arguments

`i`

: the interval value to refine

`v`

: the interval value to refer to

Description

Refines the interval value with the reference value. The interval is changed to the range satisfying both intervals. The interval status (min, max, integer, etc.) are evaluated.

Returns non-zero if the value is changed, zero if not changed.

snd_interval_ratnum

Name

`snd_interval_ratnum` — refine the interval value

Synopsis

```
int snd_interval_ratnum (struct snd_interval * i, unsigned int rats_count,
struct snd_ratnum * rats, unsigned int * nump, unsigned int * denp);
```

Arguments

i

interval to refine

rats_count

number of ratnum_t

rats

ratnum_t array

nump

pointer to store the resultant numerator

denp

pointer to store the resultant denominator

Description

Returns non-zero if the value is changed, zero if not changed.

snd_interval_list

Name

`snd_interval_list` — refine the interval value from the list

Synopsis

```
int snd_interval_list (struct snd_interval * i, unsigned int count, unsigned
int * list, unsigned int mask);
```

Arguments

`i`

the interval value to refine

`count`

the number of elements in the list

`list`

the value list

`mask`

the bit-mask to evaluate

Description

Refines the interval value from the list. When mask is non-zero, only the elements corresponding to bit 1 are evaluated.

Returns non-zero if the value is changed, zero if not changed.

snd_pcm_hw_rule_add

Name

`snd_pcm_hw_rule_add` — add the hw-constraint rule

Synopsis

```
int snd_pcm_hw_rule_add (struct snd_pcm_runtime * runtime, unsigned int cond,
int var, snd_pcm_hw_rule_func_t func, void * private, int dep, ... ...);
```

Arguments

`runtime`

the pcm runtime instance

`cond`

condition bits

`var`

the variable to evaluate

`func`

the evaluation function

`private`

the private data pointer passed to function

`dep`

the dependent variables

`...`

variable arguments

Description

Returns zero if successful, or a negative error code on failure.

snd_pcm_hw_constraint_integer

Name

`snd_pcm_hw_constraint_integer` —

Synopsis

```
int snd_pcm_hw_constraint_integer (struct snd_pcm_runtime * runtime,
snd_pcm_hw_param_t var);
```

Arguments

runtime

PCM runtime instance

var

hw_params variable to apply the integer constraint

Description

Apply the constraint of integer to an interval parameter.

snd_pcm_hw_constraint_minmax

Name

`snd_pcm_hw_constraint_minmax` —

Synopsis

```
int snd_pcm_hw_constraint_minmax (struct snd_pcm_runtime * runtime,
snd_pcm_hw_param_t var, unsigned int min, unsigned int max);
```

Arguments

runtime

PCM runtime instance

var

hw_params variable to apply the range

min

the minimal value

max

the maximal value

Description

Apply the min/max range constraint to an interval parameter.

snd_pcm_hw_constraint_list

Name

`snd_pcm_hw_constraint_list` —

Synopsis

```
int snd_pcm_hw_constraint_list (struct snd_pcm_runtime * runtime, unsigned
int cond, snd_pcm_hw_param_t var, struct snd_pcm_hw_constraint_list * l);
```

Arguments

runtime

PCM runtime instance

cond

condition bits

var

hw_params variable to apply the list constraint

l

list

Description

Apply the list of constraints to an interval parameter.

snd_pcm_hw_constraint_ratnums

Name

`snd_pcm_hw_constraint_ratnums` —

Synopsis

```
int snd_pcm_hw_constraint_ratnums (struct snd_pcm_runtime * runtime, unsigned
int cond, snd_pcm_hw_param_t var, struct snd_pcm_hw_constraint_ratnums * r);
```

Arguments

runtime

PCM runtime instance

cond

condition bits

var

hw_params variable to apply the ratnums constraint

r

struct snd_ratnums constriants

snd_pcm_hw_constraint_ratdens

Name

snd_pcm_hw_constraint_ratdens —

Synopsis

```
int snd_pcm_hw_constraint_ratdens (struct snd_pcm_runtime * runtime, unsigned
int cond, snd_pcm_hw_param_t var, struct snd_pcm_hw_constraint_ratdens * r);
```

Arguments

runtime

PCM runtime instance

cond

condition bits

var

hw_params variable to apply the ratdens constraint

r

struct snd_ratdens constriants

snd_pcm_hw_constraint_msbits

Name

`snd_pcm_hw_constraint_msbits` —

Synopsis

```
int snd_pcm_hw_constraint_msbits (struct snd_pcm_runtime * runtime, unsigned
int cond, unsigned int width, unsigned int msbits);
```

Arguments

runtime

PCM runtime instance

cond

condition bits

width

sample bits width

msbits

msbits width

snd_pcm_hw_constraint_step

Name

`snd_pcm_hw_constraint_step` —

Synopsis

`int` **`snd_pcm_hw_constraint_step`** `(struct snd_pcm_runtime *` *`runtime`*`, unsigned int` *`cond`*`, snd_pcm_hw_param_t` *`var`*`, unsigned long` *`step`*`);`

Arguments

`runtime`

PCM runtime instance

`cond`

condition bits

`var`

hw_params variable to apply the step constraint

`step`

step size

snd_pcm_hw_constraint_pow2

Name

`snd_pcm_hw_constraint_pow2` —

Synopsis

`int` **`snd_pcm_hw_constraint_pow2`** `(struct snd_pcm_runtime *` *`runtime`*`, unsigned int` *`cond`*`, snd_pcm_hw_param_t` *`var`*`);`

Arguments

`runtime`

PCM runtime instance

`cond`

condition bits

`var`

hw_params variable to apply the power-of-2 constraint

snd_pcm_hw_param_value

Name

`snd_pcm_hw_param_value` —

Synopsis

```
int snd_pcm_hw_param_value (const struct snd_pcm_hw_params * params,
snd_pcm_hw_param_t var, int * dir);
```

Arguments

`params`

the hw_params instance

`var`

parameter to retrieve

`dir`

pointer to the direction (-1,0,1) or NULL

Description

Return the value for field PAR if it’s fixed in configuration space defined by PARAMS. Return -EINVAL otherwise

snd_pcm_hw_param_first

Name

`snd_pcm_hw_param_first` —

Synopsis

```
int snd_pcm_hw_param_first (struct snd_pcm_substream * pcm, struct
snd_pcm_hw_params * params, snd_pcm_hw_param_t var, int * dir);
```

Arguments

pcm

PCM instance

params

the hw_params instance

var

parameter to retrieve

dir

pointer to the direction (-1,0,1) or NULL

Description

Inside configuration space defined by PARAMS remove from PAR all values > minimum. Reduce configuration space accordingly. Return the minimum.

snd_pcm_hw_param_last

Name

`snd_pcm_hw_param_last` —

Synopsis

```
int snd_pcm_hw_param_last (struct snd_pcm_substream * pcm, struct
snd_pcm_hw_params * params, snd_pcm_hw_param_t var, int * dir);
```

Arguments

`pcm`

PCM instance

`params`

the hw_params instance

`var`

parameter to retrieve

`dir`

pointer to the direction (-1,0,1) or NULL

Description

Inside configuration space defined by PARAMS remove from PAR all values < maximum. Reduce configuration space accordingly. Return the maximum.

snd_pcm_lib_ioctl

Name

`snd_pcm_lib_ioctl` — a generic PCM ioctl callback

Synopsis

```
int snd_pcm_lib_ioctl (struct snd_pcm_substream * substream, unsigned int
cmd, void * arg);
```

Arguments

substream

the pcm substream instance

cmd

ioctl command

arg

ioctl argument

Description

Processes the generic ioctl commands for PCM. Can be passed as the ioctl callback for PCM ops.

Returns zero if successful, or a negative error code on failure.

snd_pcm_period_elapsed

Name

`snd_pcm_period_elapsed` — update the pcm status for the next period

Synopsis

```
void snd_pcm_period_elapsed (struct snd_pcm_substream * substream);
```

Arguments

`substream`

  the pcm substream instance

Description

This function is called from the interrupt handler when the PCM has processed the period size. It will update the current pointer, wake up sleepers, etc.

Even if more than one periods have elapsed since the last call, you have to call this only once.

snd_pcm_stop

Name

`snd_pcm_stop` —

Synopsis

```
int snd_pcm_stop (struct snd_pcm_substream * substream, int state);
```

Arguments

`substream`

  the PCM substream instance

`state`

  PCM state after stopping the stream

Description

Try to stop all running streams in the substream group. The state of each stream is changed to the given value after that unconditionally.

snd_pcm_suspend

Name

`snd_pcm_suspend` —

Synopsis

```
int snd_pcm_suspend (struct snd_pcm_substream * substream);
```

Arguments

`substream`

the PCM substream

Description

Trigger SUSPEND to all linked streams. After this call, all streams are changed to SUSPENDED state.

snd_pcm_suspend_all

Name

`snd_pcm_suspend_all` —

Synopsis

```
int snd_pcm_suspend_all (struct snd_pcm * pcm);
```

Arguments

`pcm`

 the PCM instance

Description

Trigger SUSPEND to all substreams in the given pcm. After this call, all streams are changed to SUSPENDED state.

PCM Format Helpers

snd_pcm_format_signed

Name

`snd_pcm_format_signed` — Check the PCM format is signed linear

Synopsis

```
int snd_pcm_format_signed (snd_pcm_format_t format);
```

Arguments

`format`

 the format to check

Description

Returns 1 if the given PCM format is signed linear, 0 if unsigned linear, and a negative error code for non-linear formats.

snd_pcm_format_unsigned

Name

`snd_pcm_format_unsigned` — Check the PCM format is unsigned linear

Synopsis

```
int snd_pcm_format_unsigned (snd_pcm_format_t format);
```

Arguments

`format`

the format to check

Description

Returns 1 if the given PCM format is unsigned linear, 0 if signed linear, and a negative error code for non-linear formats.

snd_pcm_format_linear

Name

`snd_pcm_format_linear` — Check the PCM format is linear

Synopsis

```
int snd_pcm_format_linear (snd_pcm_format_t format);
```

Arguments

`format`

the format to check

Description

Returns 1 if the given PCM format is linear, 0 if not.

snd_pcm_format_little_endian

Name

`snd_pcm_format_little_endian` — Check the PCM format is little-endian

Synopsis

```
int snd_pcm_format_little_endian (snd_pcm_format_t format);
```

Arguments

`format`

the format to check

Description

Returns 1 if the given PCM format is little-endian, 0 if big-endian, or a negative error code if endian not specified.

snd_pcm_format_big_endian

Name

`snd_pcm_format_big_endian` — Check the PCM format is big-endian

Synopsis

```
int snd_pcm_format_big_endian (snd_pcm_format_t format);
```

Arguments

`format`

the format to check

Description

Returns 1 if the given PCM format is big-endian, 0 if little-endian, or a negative error code if endian not specified.

snd_pcm_format_width

Name

`snd_pcm_format_width` — return the bit-width of the format

Synopsis

```
int snd_pcm_format_width (snd_pcm_format_t format);
```

Arguments

`format`

the format to check

Description

Returns the bit-width of the format, or a negative error code if unknown format.

snd_pcm_format_physical_width

Name

`snd_pcm_format_physical_width` — return the physical bit-width of the format

Synopsis

```
int snd_pcm_format_physical_width (snd_pcm_format_t format);
```

Arguments

`format`

the format to check

Description

Returns the physical bit-width of the format, or a negative error code if unknown format.

snd_pcm_format_size

Name

`snd_pcm_format_size` — return the byte size of samples on the given format

Synopsis

```
ssize_t snd_pcm_format_size (snd_pcm_format_t format, size_t samples);
```

Arguments

`format`

the format to check

`samples`

-- undescribed --

Description

Returns the byte size of the given samples for the format, or a negative error code if unknown format.

snd_pcm_format_silence_64

Name

`snd_pcm_format_silence_64` — return the silent data in 8 bytes array

Synopsis

```
const unsigned char * snd_pcm_format_silence_64 (snd_pcm_format_t format);
```

Arguments

`format`

the format to check

Description

Returns the format pattern to fill or NULL if error.

snd_pcm_format_set_silence

Name

`snd_pcm_format_set_silence` — set the silence data on the buffer

Synopsis

```
int snd_pcm_format_set_silence (snd_pcm_format_t format, void * data,
unsigned int samples);
```

Arguments

`format`

the PCM format

`data`

the buffer pointer

`samples`

the number of samples to set silence

Description

Sets the silence data on the buffer for the given samples.

Returns zero if successful, or a negative error code on failure.

snd_pcm_limit_hw_rates

Name

`snd_pcm_limit_hw_rates` — determine rate_min/rate_max fields

Synopsis

```
int snd_pcm_limit_hw_rates (struct snd_pcm_runtime * runtime);
```

Arguments

`runtime`

the runtime instance

Description

Determines the rate_min and rate_max fields from the rates bits of the given runtime->hw.

Returns zero if successful.

snd_pcm_rate_to_rate_bit

Name

`snd_pcm_rate_to_rate_bit` — converts sample rate to SNDRV_PCM_RATE_xxx bit

Synopsis

```
unsigned int snd_pcm_rate_to_rate_bit (unsigned int rate);
```

Arguments

rate

the sample rate to convert

Description

Returns the SNDRV_PCM_RATE_xxx flag that corresponds to the given rate, or SNDRV_PCM_RATE_KNOT for an unknown rate.

PCM Memory Management

snd_pcm_lib_preallocate_free_for_all

Name

`snd_pcm_lib_preallocate_free_for_all` — release all pre-allocated buffers on the pcm

Synopsis

```
int snd_pcm_lib_preallocate_free_for_all (struct snd_pcm * pcm);
```

Arguments

pcm

the pcm instance

Description

Releases all the pre-allocated buffers on the given pcm.

Returns zero if successful, or a negative error code on failure.

snd_pcm_lib_preallocate_pages

Name

`snd_pcm_lib_preallocate_pages` — pre-allocation for the given DMA type

Synopsis

```
int snd_pcm_lib_preallocate_pages (struct snd_pcm_substream * substream, int
type, struct device * data, size_t size, size_t max);
```

Arguments

`substream`

the pcm substream instance

`type`

DMA type (SNDRV_DMA_TYPE_*)

`data`

DMA type dependant data

`size`

the requested pre-allocation size in bytes

`max`

the max. allowed pre-allocation size

Description

Do pre-allocation for the given DMA buffer type.

When substream->dma_buf_id is set, the function tries to look for the reserved buffer, and the buffer is not freed but reserved at destruction time. The dma_buf_id must be unique for all systems (in the same DMA buffer type) e.g. using `snd_dma_pci_buf_id`.

Returns zero if successful, or a negative error code on failure.

snd_pcm_lib_preallocate_pages_for_all

Name

`snd_pcm_lib_preallocate_pages_for_all` — pre-allocation for continous memory type (all substreams)

Synopsis

```
int snd_pcm_lib_preallocate_pages_for_all (struct snd_pcm * pcm, int type,
void * data, size_t size, size_t max);
```

Arguments

`pcm`

the pcm instance

`type`

DMA type (SNDRV_DMA_TYPE_*)

`data`

DMA type dependant data

`size`

the requested pre-allocation size in bytes

`max`

the max. allowed pre-allocation size

Description

Do pre-allocation to all substreams of the given pcm for the specified DMA type.

Returns zero if successful, or a negative error code on failure.

snd_pcm_sgbuf_ops_page

Name

`snd_pcm_sgbuf_ops_page` — get the page struct at the given offset

Synopsis

```
struct page * snd_pcm_sgbuf_ops_page (struct snd_pcm_substream * substream,
unsigned long offset);
```

Arguments

`substream`

  the pcm substream instance

`offset`

  the buffer offset

Description

Returns the page struct at the given buffer offset. Used as the page callback of PCM ops.

snd_pcm_lib_malloc_pages

Name

`snd_pcm_lib_malloc_pages` — allocate the DMA buffer

Synopsis

```
int snd_pcm_lib_malloc_pages (struct snd_pcm_substream * substream, size_t
size);
```

Arguments

`substream`

the substream to allocate the DMA buffer to

`size`

the requested buffer size in bytes

Description

Allocates the DMA buffer on the BUS type given earlier to `snd_pcm_lib_preallocate_xxx_pages`.

Returns 1 if the buffer is changed, 0 if not changed, or a negative code on failure.

snd_pcm_lib_free_pages

Name

`snd_pcm_lib_free_pages` — release the allocated DMA buffer.

Synopsis

```
int snd_pcm_lib_free_pages (struct snd_pcm_substream * substream);
```

Arguments

`substream`

the substream to release the DMA buffer

Description

Releases the DMA buffer allocated via `snd_pcm_lib_malloc_pages`.

Returns zero if successful, or a negative error code on failure.

Chapter 3. Control/Mixer API

General Control Interface

snd_ctl_new1

Name

`snd_ctl_new1` — create a control instance from the template

Synopsis

```
struct snd_kcontrol * snd_ctl_new1 (const struct snd_kcontrol_new * ncontrol,
void * private_data);
```

Arguments

ncontrol

the initialization record

private_data

the private data to set

Description

Allocates a new struct snd_kcontrol instance and initialize from the given template. When the access field of ncontrol is 0, it's assumed as READWRITE access. When the count field is 0, it's assumes as one.

Returns the pointer of the newly generated instance, or NULL on failure.

snd_ctl_free_one

Name

`snd_ctl_free_one` — release the control instance

Synopsis

```
void snd_ctl_free_one (struct snd_kcontrol * kcontrol);
```

Arguments

`kcontrol`

the control instance

Description

Releases the control instance created via `snd_ctl_new` or `snd_ctl_new1`. Don't call this after the control was added to the card.

snd_ctl_add

Name

`snd_ctl_add` — add the control instance to the card

Synopsis

```
int snd_ctl_add (struct snd_card * card, struct snd_kcontrol * kcontrol);
```

Arguments

`card`

the card instance

`kcontrol`

the control instance to add

Description

Adds the control instance created via `snd_ctl_new` or `snd_ctl_new1` to the given card. Assigns also an unique numid used for fast search.

Returns zero if successful, or a negative error code on failure.

It frees automatically the control which cannot be added.

snd_ctl_remove

Name

`snd_ctl_remove` — remove the control from the card and release it

Synopsis

```
int snd_ctl_remove (struct snd_card * card, struct snd_kcontrol * kcontrol);
```

Arguments

`card`

the card instance

`kcontrol`

the control instance to remove

Description

Removes the control from the card and then releases the instance. You don't need to call `snd_ctl_free_one`. You must be in the write lock - down_write(&card->controls_rwsem).

Returns 0 if successful, or a negative error code on failure.

snd_ctl_remove_id

Name

`snd_ctl_remove_id` — remove the control of the given id and release it

Synopsis

```
int snd_ctl_remove_id (struct snd_card * card, struct snd_ctl_elem_id * id);
```

Arguments

`card`

   the card instance

`id`

   the control id to remove

Description

Finds the control instance with the given id, removes it from the card list and releases it.

Returns 0 if successful, or a negative error code on failure.

snd_ctl_rename_id

Name

`snd_ctl_rename_id` — replace the id of a control on the card

Synopsis

```
int snd_ctl_rename_id (struct snd_card * card, struct snd_ctl_elem_id *
src_id, struct snd_ctl_elem_id * dst_id);
```

Arguments

card

the card instance

src_id

the old id

dst_id

the new id

Description

Finds the control with the old id from the card, and replaces the id with the new one.

Returns zero if successful, or a negative error code on failure.

snd_ctl_find_numid

Name

`snd_ctl_find_numid` — find the control instance with the given number-id

Synopsis

```
struct snd_kcontrol * snd_ctl_find_numid (struct snd_card * card, unsigned
int numid);
```

Arguments

card

the card instance

numid

the number-id to search

Description

Finds the control instance with the given number-id from the card.

Returns the pointer of the instance if found, or NULL if not.

The caller must down card->controls_rwsem before calling this function (if the race condition can happen).

snd_ctl_find_id

Name

`snd_ctl_find_id` — find the control instance with the given id

Synopsis

```
struct snd_kcontrol * snd_ctl_find_id (struct snd_card * card, struct
snd_ctl_elem_id * id);
```

Arguments

card

the card instance

id

the id to search

Description

Finds the control instance with the given id from the card.

Returns the pointer of the instance if found, or NULL if not.

The caller must down card->controls_rwsem before calling this function (if the race condition can happen).

AC97 Codec API

snd_ac97_write

Name

`snd_ac97_write` — write a value on the given register

Synopsis

```
void snd_ac97_write (struct snd_ac97 * ac97, unsigned short reg, unsigned
short value);
```

Arguments

ac97

the ac97 instance

reg

the register to change

value

the value to set

Description

Writes a value on the given register. This will invoke the write callback directly after the register check. This function doesn't change the register cache unlike #`snd_ca97_write_cache`, so use this only when you don't want to reflect the change to the suspend/resume state.

snd_ac97_read

Name

`snd_ac97_read` — read a value from the given register

Synopsis

```
unsigned short snd_ac97_read (struct snd_ac97 * ac97, unsigned short reg);
```

Arguments

ac97

the ac97 instance

reg

the register to read

Description

Reads a value from the given register. This will invoke the read callback directly after the register check.

Returns the read value.

Description

Reads a value from the given register. This will invoke the read callback directly after the register check.

Returns the read value.

snd_ac97_write_cache

Name

`snd_ac97_write_cache` — write a value on the given register and update the cache

Synopsis

```
void snd_ac97_write_cache (struct snd_ac97 * ac97, unsigned short reg,
unsigned short value);
```

Arguments

ac97

the ac97 instance

reg

the register to change

value

the value to set

Description

Writes a value on the given register and updates the register cache. The cached values are used for the cached-read and the suspend/resume.

snd_ac97_update

Name

`snd_ac97_update` — update the value on the given register

Synopsis

```
int snd_ac97_update (struct snd_ac97 * ac97, unsigned short reg, unsigned
short value);
```

Arguments

`ac97`

the ac97 instance

`reg`

the register to change

`value`

the value to set

Description

Compares the value with the register cache and updates the value only when the value is changed.

Returns 1 if the value is changed, 0 if no change, or a negative code on failure.

snd_ac97_update_bits

Name

`snd_ac97_update_bits` — update the bits on the given register

Synopsis

```
int snd_ac97_update_bits (struct snd_ac97 * ac97, unsigned short reg,
unsigned short mask, unsigned short value);
```

Arguments

ac97

the ac97 instance

reg

the register to change

mask

the bit-mask to change

value

the value to set

Description

Updates the masked-bits on the given register only when the value is changed.

Returns 1 if the bits are changed, 0 if no change, or a negative code on failure.

snd_ac97_get_short_name

Name

`snd_ac97_get_short_name` — retrieve codec name

Synopsis

```
const char * snd_ac97_get_short_name (struct snd_ac97 * ac97);
```

Arguments

ac97

the codec instance

Description

Returns the short identifying name of the codec.

snd_ac97_bus

Name

`snd_ac97_bus` — create an AC97 bus component

Synopsis

```
int snd_ac97_bus (struct snd_card * card, int num, struct snd_ac97_bus_ops *
ops, void * private_data, struct snd_ac97_bus ** rbus);
```

Arguments

card

the card instance

num

the bus number

ops

the bus callbacks table

private_data

private data pointer for the new instance *rbus*

the pointer to store the new AC97 bus instance.

Description

Creates an AC97 bus component. An struct snd_ac97_bus instance is newly allocated and initialized.

The ops table must include valid callbacks (at least read and write). The other callbacks, wait and reset, are not mandatory.

The clock is set to 48000. If another clock is needed, set (*rbus)->clock manually.

The AC97 bus instance is registered as a low-level device, so you don't have to release it manually.

Returns zero if successful, or a negative error code on failure.

snd_ac97_mixer

Name

`snd_ac97_mixer` — create an Codec97 component

Synopsis

```
int snd_ac97_mixer (struct snd_ac97_bus * bus, struct snd_ac97_template *
template, struct snd_ac97 ** rac97);
```

Arguments

bus

the AC97 bus which codec is attached to

template

the template of ac97, including index, callbacks and the private data.

rac97

the pointer to store the new ac97 instance.

Description

Creates an Codec97 component. An struct snd_ac97 instance is newly allocated and initialized from the template. The codec is then initialized by the standard procedure.

The template must include the codec number (num) and address (addr), and the private data (private_data).

The ac97 instance is registered as a low-level device, so you don't have to release it manually.

Returns zero if successful, or a negative error code on failure.

snd_ac97_update_power

Name

`snd_ac97_update_power` — update the powerdown register

Synopsis

```
int snd_ac97_update_power (struct snd_ac97 * ac97, int reg, int powerup);
```

Arguments

ac97

the codec instance

reg

the rate register, e.g. AC97_PCM_FRONT_DAC_RATE

powerup

non-zero when power up the part

Description

Update the AC97 powerdown register bits of the given part.

snd_ac97_suspend

Name

`snd_ac97_suspend` — General suspend function for AC97 codec

Synopsis

```
void snd_ac97_suspend (struct snd_ac97 * ac97);
```

Arguments

`ac97`

the ac97 instance

Description

Suspends the codec, power down the chip.

snd_ac97_resume

Name

`snd_ac97_resume` — General resume function for AC97 codec

Synopsis

```
void snd_ac97_resume (struct snd_ac97 * ac97);
```

Arguments

ac97

the ac97 instance

Description

Do the standard resume procedure, power up and restoring the old register values.

snd_ac97_tune_hardware

Name

`snd_ac97_tune_hardware` — tune up the hardware

Synopsis

```
int snd_ac97_tune_hardware (struct snd_ac97 * ac97, struct ac97_quirk *
quirk, const char * override);
```

Arguments

ac97

the ac97 instance

quirk

quirk list

override

explicit quirk value (overrides the list if non-NULL)

Description

Do some workaround for each pci device, such as renaming of the headphone (true line-out) control as “Master”. The quirk-list must be terminated with a zero-filled entry.

Returns zero if successful, or a negative error code on failure.

snd_ac97_set_rate

Name

`snd_ac97_set_rate` — change the rate of the given input/output.

Synopsis

```
int snd_ac97_set_rate (struct snd_ac97 * ac97, int reg, unsigned int rate);
```

Arguments

ac97

  the ac97 instance

reg

  the register to change

rate

  the sample rate to set

Description

Changes the rate of the given input/output on the codec. If the codec doesn't support VAR, the rate must be 48000 (except for SPDIF).

The valid registers are AC97_PMC_MIC_ADC_RATE, AC97_PCM_FRONT_DAC_RATE, AC97_PCM_LR_ADC_RATE. AC97_PCM_SURR_DAC_RATE and AC97_PCM_LFE_DAC_RATE are accepted if the codec supports them. AC97_SPDIF is accepted as a pseudo register to modify the SPDIF status bits.

Returns zero if successful, or a negative error code on failure.

snd_ac97_pcm_assign

Name

`snd_ac97_pcm_assign` — assign AC97 slots to given PCM streams

Synopsis

```
int snd_ac97_pcm_assign (struct snd_ac97_bus * bus, unsigned short
pcms_count, const struct ac97_pcm * pcms);
```

Arguments

`bus`

the ac97 bus instance

`pcms_count`

count of PCMs to be assigned

`pcms`

PCMs to be assigned

Description

It assigns available AC97 slots for given PCMs. If none or only some slots are available, pcm->xxx.slots and pcm->xxx.rslots[] members are reduced and might be zero.

snd_ac97_pcm_open

Name

`snd_ac97_pcm_open` — opens the given AC97 pcm

Synopsis

```
int snd_ac97_pcm_open (struct ac97_pcm * pcm, unsigned int rate, enum
ac97_pcm_cfg cfg, unsigned short slots);
```

Arguments

pcm

the ac97 pcm instance

rate

rate in Hz, if codec does not support VRA, this value must be 48000Hz

cfg

output stream characteristics

slots

a subset of allocated slots (snd_ac97_pcm_assign) for this pcm

Description

It locks the specified slots and sets the given rate to AC97 registers.

snd_ac97_pcm_close

Name

`snd_ac97_pcm_close` — closes the given AC97 pcm

Synopsis

```
int snd_ac97_pcm_close (struct ac97_pcm * pcm);
```

Arguments

pcm

the ac97 pcm instance

Description

It frees the locked AC97 slots.

snd_ac97_pcm_double_rate_rules

Name

snd_ac97_pcm_double_rate_rules — set double rate constraints

Synopsis

```
int snd_ac97_pcm_double_rate_rules (struct snd_pcm_runtime * runtime);
```

Arguments

runtime

the runtime of the ac97 front playback pcm

Description

Installs the hardware constraint rules to prevent using double rates and more than two channels at the same time.

Chapter 4. MIDI API

Raw MIDI API

snd_rawmidi_receive

Name

`snd_rawmidi_receive` — receive the input data from the device

Synopsis

```
int snd_rawmidi_receive (struct snd_rawmidi_substream * substream, const
unsigned char * buffer, int count);
```

Arguments

`substream`

the rawmidi substream

`buffer`

the buffer pointer

`count`

the data size to read

Description

Reads the data from the internal buffer.

Returns the size of read data, or a negative error code on failure.

snd_rawmidi_transmit_empty

Name

`snd_rawmidi_transmit_empty` — check whether the output buffer is empty

Synopsis

```
int snd_rawmidi_transmit_empty (struct snd_rawmidi_substream * substream);
```

Arguments

substream

the rawmidi substream

Description

Returns 1 if the internal output buffer is empty, 0 if not.

snd_rawmidi_transmit_peek

Name

`snd_rawmidi_transmit_peek` — copy data from the internal buffer

Synopsis

```
int snd_rawmidi_transmit_peek (struct snd_rawmidi_substream * substream,
unsigned char * buffer, int count);
```

Arguments

`substream`

the rawmidi substream

`buffer`

the buffer pointer

`count`

data size to transfer

Description

Copies data from the internal output buffer to the given buffer.

Call this in the interrupt handler when the midi output is ready, and call `snd_rawmidi_transmit_ack` after the transmission is finished.

Returns the size of copied data, or a negative error code on failure.

snd_rawmidi_transmit_ack

Name

`snd_rawmidi_transmit_ack` — acknowledge the transmission

Synopsis

```
int snd_rawmidi_transmit_ack (struct snd_rawmidi_substream * substream, int
count);
```

Arguments

`substream`

the rawmidi substream

`count`

the tranferred count

Description

Advances the hardware pointer for the internal output buffer with the given size and updates the condition. Call after the transmission is finished.

Returns the advanced size if successful, or a negative error code on failure.

snd_rawmidi_transmit

Name

`snd_rawmidi_transmit` — copy from the buffer to the device

Synopsis

```
int snd_rawmidi_transmit (struct snd_rawmidi_substream * substream, unsigned
char * buffer, int count);
```

Arguments

`substream`

the rawmidi substream

`buffer`

the buffer pointer

`count`

the data size to transfer

Description

Copies data from the buffer to the device and advances the pointer.

Returns the copied size if successful, or a negative error code on failure.

snd_rawmidi_new

Name

`snd_rawmidi_new` — create a rawmidi instance

Synopsis

```
int snd_rawmidi_new (struct snd_card * card, char * id, int device, int
output_count, int input_count, struct snd_rawmidi ** rrawmidi);
```

Arguments

`card`

the card instance

`id`

the id string

`device`

the device index

`output_count`

the number of output streams

`input_count`

the number of input streams

`rrawmidi`

the pointer to store the new rawmidi instance

Description

Creates a new rawmidi instance. Use `snd_rawmidi_set_ops` to set the operators to the new instance.

Returns zero if successful, or a negative error code on failure.

snd_rawmidi_set_ops

Name

`snd_rawmidi_set_ops` — set the rawmidi operators

Synopsis

```
void snd_rawmidi_set_ops (struct snd_rawmidi * rmidi, int stream, struct
snd_rawmidi_ops * ops);
```

Arguments

`rmidi`

   the rawmidi instance

`stream`

   the stream direction, SNDRV_RAWMIDI_STREAM_XXX

`ops`

   the operator table

Description

Sets the rawmidi operators for the given stream direction.

MPU401-UART API

snd_mpu401_uart_interrupt

Name

`snd_mpu401_uart_interrupt` — generic MPU401-UART interrupt handler

Synopsis

```
irqreturn_t snd_mpu401_uart_interrupt (int irq, void * dev_id);
```

Arguments

irq

the irq number

dev_id

mpu401 instance

Description

Processes the interrupt for MPU401-UART i/o.

snd_mpu401_uart_interrupt_tx

Name

`snd_mpu401_uart_interrupt_tx` — generic MPU401-UART transmit irq handler

Synopsis

```
irqreturn_t snd_mpu401_uart_interrupt_tx (int irq, void * dev_id);
```

Arguments

irq

the irq number

dev_id

mpu401 instance

Description

Processes the interrupt for MPU401-UART output.

snd_mpu401_uart_new

Name

`snd_mpu401_uart_new` — create an MPU401-UART instance

Synopsis

```
int snd_mpu401_uart_new (struct snd_card * card, int device, unsigned short
hardware, unsigned long port, unsigned int info_flags, int irq, int
irq_flags, struct snd_rawmidi ** rrawmidi);
```

Arguments

card

the card instance

`device`

the device index, zero-based

`hardware`

the hardware type, MPU401_HW_XXXX

`port`

the base address of MPU401 port

`info_flags`

bitflags MPU401_INFO_XXX

`irq`

the irq number, -1 if no interrupt for mpu

`irq_flags`

the irq request flags (SA_XXX), 0 if irq was already reserved.

`rrawmidi`

the pointer to store the new rawmidi instance

Description

Creates a new MPU-401 instance.

Note that the rawmidi instance is returned on the rrawmidi argument, not the mpu401 instance itself. To access to the mpu401 instance, cast from rawmidi->private_data (with struct snd_mpu401 magic-cast).

Returns zero if successful, or a negative error code.

Chapter 5. Proc Info API

Proc Info Interface

snd_iprintf

Name

`snd_iprintf` — printf on the procfs buffer

Synopsis

```
int snd_iprintf (struct snd_info_buffer * buffer, char * fmt, ... ...);
```

Arguments

`buffer`

the procfs buffer

`fmt`

the printf format

`...`

variable arguments

Description

Outputs the string on the procfs buffer just like `printf`.

Returns the size of output string.

snd_info_get_line

Name

`snd_info_get_line` — read one line from the procfs buffer

Synopsis

```
int snd_info_get_line (struct snd_info_buffer * buffer, char * line, int
len);
```

Arguments

buffer

the procfs buffer

line

the buffer to store

len

the max. buffer size - 1

Description

Reads one line from the buffer and stores the string.

Returns zero if successful, or 1 if error or EOF.

snd_info_get_str

Name

`snd_info_get_str` — parse a string token

Synopsis

```
char * snd_info_get_str (char * dest, char * src, int len);
```

Arguments

dest

  the buffer to store the string token

src

  the original string

len

  the max. length of token - 1

Description

Parses the original string and copy a token to the given string buffer.

Returns the updated pointer of the original string so that it can be used for the next call.

snd_info_create_module_entry

Name

`snd_info_create_module_entry` — create an info entry for the given module

Synopsis

```
struct snd_info_entry * snd_info_create_module_entry (struct module * module,
const char * name, struct snd_info_entry * parent);
```

Arguments

`module`

the module pointer

`name`

the file name

`parent`

the parent directory

Description

Creates a new info entry and assigns it to the given module.

Returns the pointer of the new instance, or NULL on failure.

snd_info_create_card_entry

Name

`snd_info_create_card_entry` — create an info entry for the given card

Synopsis

```
struct snd_info_entry * snd_info_create_card_entry (struct snd_card * card,
const char * name, struct snd_info_entry * parent);
```

Arguments

`card`

the card instance

`name`

the file name

`parent`

the parent directory

Description

Creates a new info entry and assigns it to the given card.

Returns the pointer of the new instance, or NULL on failure.

snd_card_proc_new

Name

`snd_card_proc_new` — create an info entry for the given card

Synopsis

```
int snd_card_proc_new (struct snd_card * card, const char * name, struct
snd_info_entry ** entryp);
```

Arguments

`card`

the card instance

`name`

the file name

`entryp`

the pointer to store the new info entry

Description

Creates a new info entry and assigns it to the given card. Unlike `snd_info_create_card_entry`, this function registers the info entry as an ALSA device component, so that it can be unregistered/released without explicit call. Also, you don’t have to register this entry via `snd_info_register`, since this will be registered by `snd_card_register` automatically.

The parent is assumed as card->proc_root.

For releasing this entry, use `snd_device_free` instead of `snd_info_free_entry`.

Returns zero if successful, or a negative error code on failure.

snd_info_free_entry

Name

`snd_info_free_entry` — release the info entry

Synopsis

```
void snd_info_free_entry (struct snd_info_entry * entry);
```

Arguments

`entry`

the info entry

Description

Releases the info entry. Don’t call this after registered.

snd_info_register

Name

`snd_info_register` — register the info entry

Synopsis

```
int snd_info_register (struct snd_info_entry * entry);
```

Arguments

entry

the info entry

Description

Registers the proc info entry.

Returns zero if successful, or a negative error code on failure.

Chapter 6. Miscellaneous Functions

Hardware-Dependent Devices API

snd_hwdep_new

Name

`snd_hwdep_new` — create a new hwdep instance

Synopsis

```
int snd_hwdep_new (struct snd_card * card, char * id, int device, struct
snd_hwdep ** rhwdep);
```

Arguments

card

the card instance

id

the id string

device

the device index (zero-based)

rhwdep

the pointer to store the new hwdep instance

Description

Creates a new hwdep instance with the given index on the card. The callbacks (hwdep->ops) must be set on the returned instance after this call manually by the caller.

Returns zero if successful, or a negative error code on failure.

ISA DMA Helpers

snd_dma_program

Name

`snd_dma_program` — program an ISA DMA transfer

Synopsis

```
void snd_dma_program (unsigned long dma, unsigned long addr, unsigned int
size, unsigned short mode);
```

Arguments

`dma`

the dma number

`addr`

the physical address of the buffer

`size`

the DMA transfer size

`mode`

the DMA transfer mode, DMA_MODE_XXX

Description

Programs an ISA DMA transfer for the given buffer.

snd_dma_disable

Name

`snd_dma_disable` — stop the ISA DMA transfer

Synopsis

```
void snd_dma_disable (unsigned long dma);
```

Arguments

`dma`

the dma number

Description

Stops the ISA DMA transfer.

snd_dma_pointer

Name

`snd_dma_pointer` — return the current pointer to DMA transfer buffer in bytes

Synopsis

```
unsigned int snd_dma_pointer (unsigned long dma, unsigned int size);
```

Arguments

dma

the dma number

size

the dma transfer size

Description

Returns the current pointer in DMA tranfer buffer in bytes

Other Helper Macros

snd_register_device

Name

`snd_register_device` — Register the ALSA device file for the card

Synopsis

```
int snd_register_device (int type, struct snd_card * card, int dev, const
struct file_operations * f_ops, void * private_data, const char * name);
```

Arguments

type

the device type, SNDRV_DEVICE_TYPE_XXX

card

the card instance

`dev`

the device index

`f_ops`

the file operations

`private_data`

user pointer for f_ops->`open`

`name`

the device file name

Description

Registers an ALSA device file for the given card. The operators have to be set in reg parameter.

This function uses the card's device pointer to link to the correct &struct device.

Returns zero if successful, or a negative error code on failure.

snd_printk

Name

`snd_printk` — printk wrapper

Synopsis

```
snd_printk ( fmt,  args...);
```

Arguments

`fmt`

format string

`args...`

Description

Works like `print` but prints the file and the line of the caller when configured with CONFIG_SND_VERBOSE_PRINTK.

snd_printd

Name

`snd_printd` — debug printk

Synopsis

```
snd_printd ( fmt,  args...);
```

Arguments

`fmt`

format string

`args...`

Description

Works like `snd_printk` for debugging purposes. Ignored when CONFIG_SND_DEBUG is not set.

snd_assert

Name

`snd_assert` — run-time assertion macro

Synopsis

```
snd_assert ( expr,  args...);
```

Arguments

`expr`

expression

`args...`

Description

This macro checks the expression in run-time and invokes the commands given in the rest arguments if the assertion is failed. When CONFIG_SND_DEBUG is not set, the expression is executed but not checked.

snd_printdd

Name

`snd_printdd` — debug printk

Synopsis

```
snd_printdd ( format,  args...);
```

Arguments

`format`

format string

`args...`

Description

Works like `snd_printk` for debugging purposes. Ignored when CONFIG_SND_DEBUG_DETECT is not set.