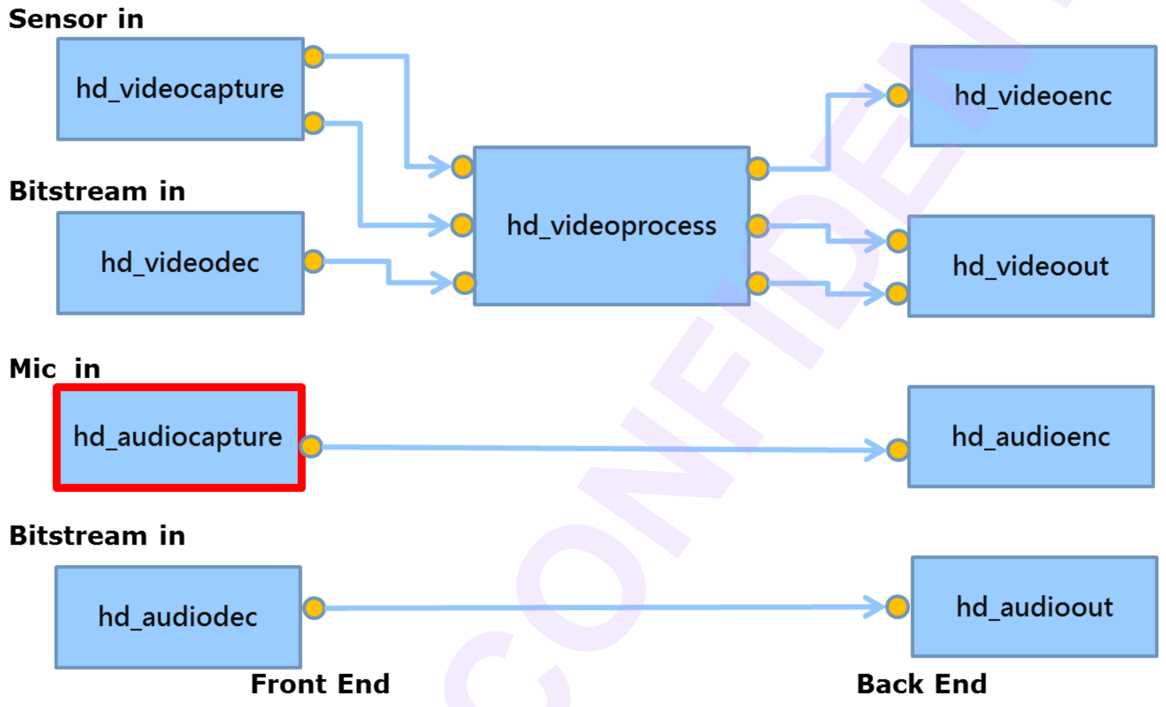
# *Novatek HDAL Design Specification - hd_audiocapture*

# *Table of Content*

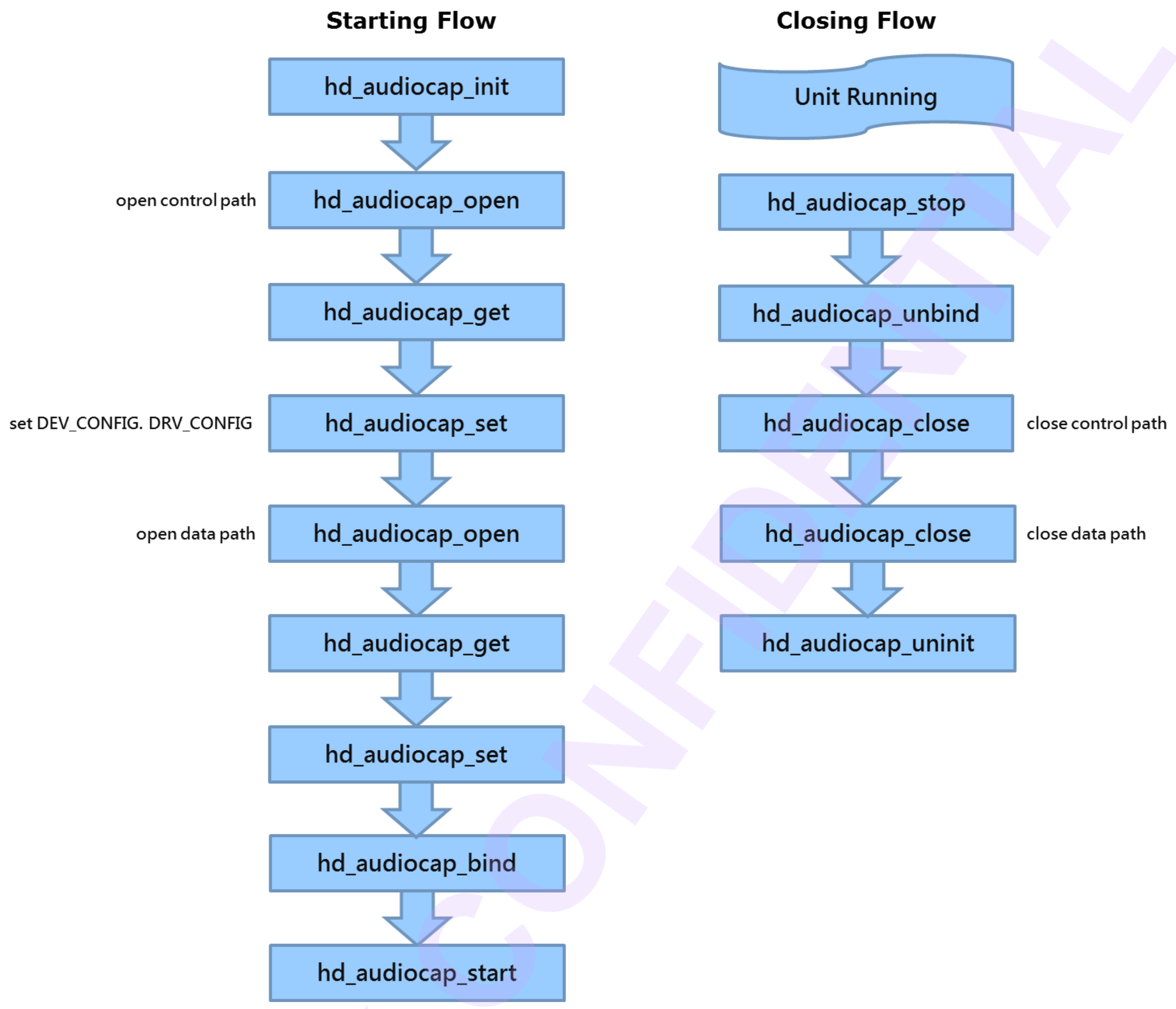
# 1 Introduction

The major purpose of hd_audiocapture is to get raw data from audio engine. This document will talk about the red block in the following diagram. The device driver is not the main point in this document.



## 1.1 Basic Flow

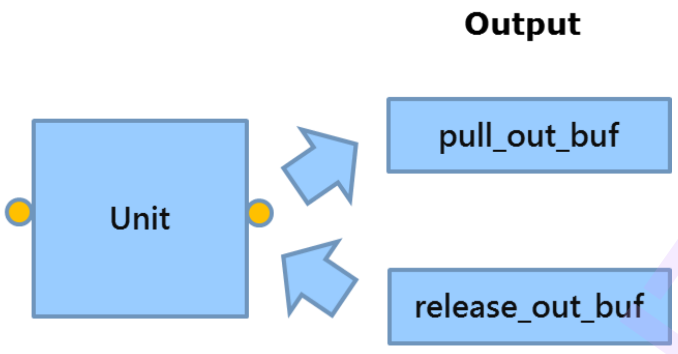
The call sequence is needed to be done correctly for the unit. The standard starting flows of most modules are init, open, get, set, bind and start. The standard closing flows of most modules are stop, unbind, close and uninit. The basic flow is shown as below.



Now, below section in this chapter is mainly about what things to do in those functions above.

## 1.2 Single Trigger Operation

Single trigger operation is used to trigger the unit to do one job, such as to grab one PCM frame from audio capture. There are two types of functions for output port. The sequence for output port is pull and release. The flow is shown as below.


Output
Unit
pull_out_buf
release_out_buf

# 2 Function and data structure definition

## 2.1 General function

### 2.1.1 hd_audiocap_init

[Description]
Initialize the unit

[Syntax]
HD_RESULT hd_audiocap_init(VOID);

[Parameter]

| Value | Description |
|---|---|
| VOID | Not available |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |

### 2.1.2 hd_audiocap_open

[Description]
Open the unit

[Syntax]
HD_RESULT hd_audiocap_open(HD_IN_ID in_id, HD_OUT_ID out_id, HD_PATH_ID* p_path_id)

[Parameter]

| Value | Description |
|---|---|

| | |
|---|---|
| in_id | id of input port. |
| out_id | id of output port. |
| p_path_id | pointer of the path id |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |

### 2.1.3 hd_audiocap_get

[Description]

Get parameters from unit by path id

[Syntax]

HD_RESULT hd_audiocap_get(HD_PATH_ID path_id, HD_AUDIOCAP_PARAM_ID id, VOID* p_param)

[Parameter]

| Value | Description |
|---|---|
| path_id | the path id |
| id | id of parameters |
| p_param | pointer of parameters |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |
| HD_ERR_NOT_SUPPORT | Not support this parameter |

### 2.1.4 hd_audiocap_set

[Description]

Set parameters to unit by path id

[Syntax]

HD_RESULT hd_audiocap_set(HD_PATH_ID path_id, HD_AUDIOCAP_PARAM_ID id, VOID* p_param)

[Parameter]

| Value | Description |
|---|---|
| path_id | the path id |
| id | id of parameters |
| p_param | pointer of parameters |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |
| HD_ERR_NOT_SUPPORT | Not support this parameter |

### 2.1.5 hd_audiocap_bind

[Description]

Bind this unit with destination unit

[Syntax]

HD_RESULT hd_audiocap_bind(HD_OUT_ID out_id, HD_IN_ID dest_in_id)

[Parameter]

| Value | Description |
|---|---|
| out_id | id of output port. |
| dest_in_id | id of input port. |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |

### 2.1.6 hd_audiocap_start

[Description]
Start the unit

[Syntax]
HD_RESULT hd_audiocap_start(HD_PATH_ID path_id)

[Parameter]

| Value | Description |
|---|---|
| path_id | pointer of the path id |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |

### 2.1.7 hd_audiocap_stop

[Description]
Stop the unit

[Syntax]
HD_RESULT hd_audiocap_stop(HD_PATH_ID path_id)

[Parameter]

| Value | Description |
|---|---|
| path_id | pointer of the path id |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |

### 2.1.8 hd_audiocap_unbind

[Description]
Unbind the unit

[Syntax]
HD_RESULT hd_audiocap_open(HD_IN_ID in_id, HD_OUT_ID out_id, HD_PATH_ID* p_path_id)

[Parameter]

| Value | Description |
|---|---|
| in_id | id of input port. |
| out_id | id of output port. |
| p_path_id | pointer of the path id |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |

### 2.1.9 hd_audiocap_close

[Description]
Close the unit

[Syntax]
HD_RESULT hd_audiocap_close(HD_PATH_ID path_id)

[Parameter]

| Value | Description |
|---|---|
| path_id | pointer of the path id |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |

### 2.1.10 hd_audiocap_uninit

[Description]
Uninitialize the unit

[Syntax]
HD_RESULT hd_audiocap_uninit(VOID);

[Parameter]

| Value | Description |
|---|---|
| VOID | Not available |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |

### 2.1.11 hd_audiocap_start_list

[Description]
Do start for a list of paths

[Syntax]
HD_RESULT hd_audiocap_start_list(HD_PATH_ID *path_id, UINT num);

[Parameter]

| Value | Description |
|---|---|
| path_id | An array of paths |
| num | The number of paths |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |

[Difference]

| Chip | Description |
|---|---|
| IPC | All functions are NOT supported. |
| NVR | All functions are supported. |

### 2.1.12 hd_audiocap_stop_list

[Description]

Do stop for a list of paths

[Syntax]

HD_RESULT hd_audiocap_stop_list(HD_PATH_ID *path_id, UINT num);

[Parameter]

| Value | Description |
|---|---|
| path_id | An array of paths |
| num | The number of paths |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |

[Difference]

| Chip | Description |
|---|---|
| IPC | All functions are NOT supported. |
| NVR | All functions are supported. |

### 2.1.13 hd_audiocap_pull_out_buf

[Description]

Pull the audio frame buffer from unit

[Syntax]

HD_RESULT hd_audiocap_pull_out_buf(HD_PATH_ID path_id, HD_AUDIO_FRAME *p_audio_frame, INT32 wait_ms)

[Parameter]

| Value | Description |
|---|---|
| path_id | the path id |
| p_audio_frame | pointer of the output audio frame |
| wait_ms | timeout value in ms |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |

[Difference]

| Chip | Description |
|---|---|
| IPC | All functions are supported. |
| NVR | All functions are NOT supported. |

### 2.1.14 hd_audiocap_release_out_buf

[Description]

Release the audio frame buffer which is get from unit

[Syntax]

HD_RESULT hd_audiocap_release_out_buf(HD_PATH_ID path_id, HD_AUDIO_FRAME *p_audio_frame)

[Parameter]

| Value | Description |
|---|---|
| path_id | the path id |
| p_audio_frame | pointer of the output audio p_audio_frame |

[Return Value]

| Value | Description |
|---|---|
| HD_OK | Success |
| HD_ERR_NG | Failure |

[Difference]

| Chip | Description |
|---|---|
| IPC | All functions are supported. |
| NVR | All functions are NOT supported. |

## 2.2 Data structure definition

The function hd_audiocap_get and hd_audiocap_set provides the following parameter IDs:

- HD_AUDIOCAP_PARAM_DEVCOUNT
  - ☐ NVR/IPC. support get with ctrl path
  - ☐ using HD_DEVCOUNT struct (device id max count)
- HD_AUDIOCAP_PARAM_SYSCAPS
  - ☐ NVR/IPC. support get with ctrl path
  - ☐ using HD_AUDIOCAP_SYSCAPS
- HD_AUDIOCAP_PARAM_SYSINFO
  - ☐ NVR/IPC. support get with ctrl path
  - ☐ using HD_AUDIOCAP_SYSINFO
- HD_AUDIOCAP_PARAM_DEV_CONFIG
  - ☐ NVR/IPC. support get/set with ctrl path
  - ☐ using HD_AUDIOCAP_DEV_CONFIG struct
- HD_AUDIOCAP_PARAM_DRV_CONFIG
  - ☐ NVR/IPC. support get/set with ctrl path
  - ☐ using HD_AUDIOCAP_DRV_CONFIG struct
- HD_AUDIOCAP_PARAM_IN
  - ☐ NVR/IPC. support get/set with i/o path
  - ☐ using HD_AUDIOCAP_IN struct
- HD_AUDIOCAP_PARAM_OUT
  - ☐ NVR/IPC. support get/set with i/o path
  - ☐ using HD_AUDIOCAP_OUT struct
- HD_AUDIOCAP_PARAM_OUT_AEC

- ☐ IPC only. support get/set with i/o path
- ☐ using HD_AUDIOCAP_AEC struct

- HD_AUDIOCAP_PARAM_OUT _ANR
  - ☐ IPC only. support get/set with i/o path
  - ☐ using HD_AUDIOCAP_ANR struct
- HD_AUDIOCAP_PARAM_VOLUME
  - ☐ NVR/IPC. support get/set with ctrl path
  - ☐ using HD_AUDIOCAP_VOLUME struct
- HD_AUDIOCAP_PARAM_BUFINFO
  - ☐ IPC only. support get with ctrl path
  - ☐ using HD_AUDIOCAP_BUFINFO struct

### 2.2.1 HD_ AUDIOCAP_SYSCAPS

[Description]
System capability

[Parameter]

| Value | Description |
|---|---|
| dev_id | device id |
| chip_id | chip id of this device |
| max_in_count | max count of input of this device |
| max_out_count | max count of output of this device |
| dev_caps | capability of device, combine caps of HD_DEVICE_CAPS and HD_AUDIOCAP_DEVCAPS |
| in_caps | capability of input, cap of HD_AUDIO_CAPS |
| out_caps | capability of output, cap of HD_AUDIO_CAPS |
| support_in_sr | sample rate capability of input, cap of HD_AUDIOCAP_SRCAPS |
| support_out_sr | sample rate capability of output, cap of HD_AUDIOCAP_SRCAPS |

### 2.2.2 HD_AUDIOCAP_DEV_CONFIG

[Description]

Device configuration.

IPC: support close-time change.

[Parameter]

| Value | Description |
|---|---|
| in_max.sample_rate | NVR/IPC. maximum input sample rate<br>Please refer to HD_AUDIO_SR enum.<br>Default value: HD_AUDIO_SR_48000<br>IPC: support close-time change. |
| in_max.sample_bit | NVR/IPC. maximum input sample bit<br>Please refer to HD_AUDIO_BIT_WIDTH enum.<br>Default value: HD_AUDIO_BIT_WIDTH_16<br>IPC: support close-time change. |
| in_max.mode | NVR/IPC. maximum input sound mode<br>Please refer to HD_AUDIO_SOUND_MODE enum.<br>Default value: HD_AUDIO_SOUND_MODE_STEREO<br>IPC: support close-time change. |
| in_max.frame_sample | NVR/IPC. maximum sample count of each frame<br>Value range: 1024, 2048, 3072, 4096<br>Default value: 1024<br>IPC: support close-time change. |
| frame_num_max | NVR/IPC. maximum frame number in buffer<br>Value range: [4, 50]<br>Default value: 10<br>IPC: support close-time change. |
| out_max.sample_rate | NVR/IPC. maximum output sample rate.<br>Please refer to HD_AUDIO_SR enum<br>Value is 0: not support resampling<br>Default value: 0<br>IPC: support close-time change. |
| aec_max.enabled | NVR/IPC. AEC enable |

| | |
|---|---|
| | Value is 0: not support AEC<br>Value is 1: support AEC<br>Default value: 0<br>IPC: support close-time change. |
| aec_max.leak_estimate_enabled | NVR/IPC. AEC leak estimate enable<br>Value is 0: disable aec leak estimate<br>Value is 1: enable aec leak estimate<br>Default value: 0<br>IPC: support close-time change. |
| aec_max.leak_estimate_value | NVR/IPC. AEC initial condition of the leak estimate<br>Value range: [25, 99]<br>Default value: 0<br>IPC: support close-time change. |
| aec_max.noise_cancel_level | NVR/IPC. AEC noise cancel level<br>Value range: [-40, -3]<br>Default value: 0<br>IPC: support close-time change. |
| aec_max.echo_cancel_level | NVR/IPC. AEC echo cancel level<br>Value range: [-60, -30]<br>Default value: 0<br>IPC: support close-time change. |
| aec_max.filter_length | NVR/IPC. AEC internal filter length<br>Default value: 0<br>IPC: support close-time change. |
| aec_max.frame_size | NVR/IPC. AEC internal frame size<br>Default value: 0<br>IPC: support close-time change. |
| aec_max.notch_radius | NVR/IPC. AEC notch filter radius.<br>Value range: [0, 1000]<br>Default value: 0<br>IPC: support close-time change. |
| aec_max.lb_channel | NVR/IPC. AEC audio output loopback channel.<br>Please refer to HD_AUDIOCAP_LB_CH.<br>Default value: 0<br>IPC: support stop-time change. |

| | |
|---|---|
| anr_max.enabled | NVR/IPC. ANR enable<br>Value is 0: not support ANR<br>Value is 1: support ANR<br>Default value: 0<br>IPC: support close-time change. |
| anr_max.suppress_level | NVR/IPC. ANR suppression level of noise<br>Value range: [3, 35]<br>Default value: 0<br>IPC: support close-time change. |
| anr_max.hpf_cut_off_freq | NVR/IPC. ANR cut-off frequency of HPF pre-filtering<br>Default value: 0<br>IPC: support close-time change. |
| anr_max.bias_sensitive | NVR/IPC. ANR bias sensitive<br>Value range: [1, 9]<br>Default value: 0<br>IPC: support close-time change. |
| data_pool | NVR only. pool memory information |

### 2.2.3 HD_AUDIOCAP_SYSINFO

[Description]

System information

[Parameter]

| Value | Description |
|---|---|
| dev_id | device id |
| cur_in_sample_rate | current input sample rate |
| cur_sample_bit | current sample bit width |
| cur_mode | current sound mode |
| cur_out_sample_rate | current output sample rate |

### 2.2.4 HD_AUDIOCAP_DRV_CONFIG

[Description]

Driver configuration

[Parameter]

| Value | Description |
|---|---|
| mono | IPC only. audio mono channel<br>Please refer to HD_AUDIO_MONO enum.<br>Default value: HD_AUDIO_MONO_RIGHT<br>IPC: support stop-time change. |
| ssp_config | NVR only. audio ssp config |

### 2.2.5 HD_AUDIOCAP_SSP_CONFIG

[Description]

SSP configuration

[Parameter]

| Value | Description |
|---|---|
| ssp_num | NVR only. the hw path for this ssp |
| enable | NVR only. enable for each ssp interface |
| ssp_chan | NVR only. the channel count for this ssp |
| sample_size | NVR only. audio sample size |
| sample_rate | NVR only. audio sample rate |
| ssp_clock | NVR only. ssp clock for each ssp interface |
| bit_clock | NVR only. bit clock for each ssp interface |
| ssp_master | NVR only. select mode for this ssp |
| live_sound_ch | NVR only. channel source of live sound |

### 2.2.6 HD_AUDIOCAP_IN

[Description]

Input parameter

[Parameter]

| Value | Description |
|---|---|
| sample_rate | sample rate |

| | |
|---|---|
| | Please refer to HD_AUDIO_SR enum.<br>Default value: HD_AUDIO_SR_48000<br>IPC: support stop-time change. |
| sample_bit | sample bit<br>Please refer to HD_AUDIO_BIT_WIDTH enum.<br>Default value: HD_AUDIO_BIT_WIDTH_16<br>IPC: support stop-time change. |
| mode | sound mode<br>Please refer to HD_AUDIO_SOUND_MODE enum.<br>Default value: HD_AUDIO_SOUND_MODE_STEREO<br>IPC: support stop-time change. |
| frame_sample | sample count of each frame<br>Value range:<br>Default value: 1024<br>IPC: support stop-time change. |

### 2.2.7 HD_AUDIOCAP_OUT

[Description]
Output parameter

[Parameter]

| Value | Description |
|---|---|
| sample_rate | output sample rate (for resampling)<br>Please refer to HD_AUDIO_SR enum.<br>Value is 0: disable resampling<br>Default value: 0<br>IPC: support stop-time change. |

### 2.2.8 HD_AUDIOCAP_VOLUME

[Description]

Input volume.

[Parameter]

| Value | Description |
|---|---|
| volume | input volume<br>Value range: [0,100]<br>Default value: 100. |

### 2.2.9 HD_AUDIOCAP_POOL

[Description]
Pool memory information

[Parameter]

| Value | Description |
|---|---|
| ddr_id | DDR ID |
| frame_sample_size | the buffer size of audio frame |
| counts | count of buffer |
| max_counts | max counts of buffer |
| min_counts | min counts of buffer |
| mode | pool mode |

### 2.2.10 HD_AUDIOCAP_AEC

[Description]
AEC configuration

[Parameter]

| Value | Description |
|---|---|
| enabled | AEC enable<br>Value is 0: disable AEC<br>Value is 1: enable AEC<br>Default value: 0<br>IPC: support stop-time change. |
| leak_estimate_enabled | leak estimate enable |

| | |
|---|---|
| | Value is 0: disable aec leak estimate<br>Value is 1: enable aec leak estimate<br>Default value: 0<br>IPC: support stop-time change. |
| leak_estimate_value | initial condition of the leak estimate<br>Value range: [25, 99]<br>Default value: 0<br>IPC: support stop-time change. |
| noise_cancel_level | noise cancel level<br>Value range: [-40, -3]<br>Default value: 0<br>IPC: support stop-time change. |
| echo_cancel_level | echo cancel level<br>Value range: [-60, -30]<br>Default value: 0<br>IPC: support stop-time change. |
| filter_length | internal filter length<br>Default value: 0<br>IPC: support stop-time change. |
| frame_size | internal frame size<br>Default value: 0<br>IPC: support stop-time change. |
| notch_radius | notch filter radius<br>Value range: [0, 1000]<br>Default value: 0<br>IPC: support stop-time change. |
| lb_channel | audio output loopback channel<br>Please refer to HD_AUDIOCAP_LB_CH.<br>Default value: 0<br>IPC: support stop-time change. |

### 2.2.11 HD_AUDIOCAP_ANR

[Description]

ANR configuration

[Parameter]

| Value | Description |
|---|---|
| enabled | ANR enable<br>Value is 0: disable ANR<br>Value is 1: enable ANR<br>Default value: 0<br>IPC: support stop-time change. |
| suppress_level | suppression level of noise<br>Value range: [3, 35]<br>Default value: 0<br>IPC: support stop-time change. |
| hpf_cut_off_freq | cut-off frequency of HPF pre-filtering<br>Default value: 0<br>IPC: support stop-time change. |
| bias_sensitive | bias sensitive<br>Value range: [1, 9]<br>Default value: 0<br>IPC: support stop-time change. |

### 2.2.12 HD_AUDIOCAP_PARAM_CLEAR_BUF

[Description]

Clear audio data in the queue.

[Parameter]

No parameters.

# 3 Debug command

The audiocapture module supports two kinds of debug mechanism for user. User can use proc command or debug menu to debug.

## 3.1 proc command for IPC

### 3.1.1 Dump info

```
[dump info]

cat /proc/hdal/acap/info
```

The result will show the audiocapture information by five parts.

1. **PATH & BIND**: bind status of hd_audiocapture.
2. **DEV CONFIG**: device configuration, referring to HD_AUDIOCAP_DEV_CONFIG.
3. **DRV CONFIG**: driver configuration, referring to HD_AUDIOCAP_DRV_CONFIG.
4. **Volume**: volume configuration, referring to HD_AUDIOCAP_VOLUME.
5. **IN FRAME**: input configuration, referring to HD_AUDIOCAP_IN.
6. **OUT FRAME**: output configuration for resampling, referring to HD_AUDIOCAP_OUT.
7. **AEC**: AEC configuration, referring to HD_AUDIOCAP_AEC.
8. **ANR**: ANR configuration, referring to HD_AUDIOCAP_ANR.

[PATH & BIND]

| Status | Description |
|---|---|
| in | input id of path |
| out | output id of path |
| state | state of path |
| bind_src | current binding source of input |
| bind_dest | current binding source of output |

[DEV CONFIG]

| Status | Description |
|---|---|
| max | device ID |

| | |
|---|---|
| in.sr | maximum sample rate |
| in.ch | maximum channel count (sound mode) |
| in.bit | maximum bit width |
| in.frm_sample | maximum frame sample |
| frm_num | maximum frame number |
| aec.en | AEC enable |
| aec.leak_est_en | AEC leak estimate enable |
| aec.leak_est | AEC initial condition of the leak estimate |
| aec.noise_lvl | AEC noise cancel level |
| aec.echo_lvl | AEC echo cancel level |
| aec.filter_len | AEC internal filter length |
| aec.frm_size | AEC internal frame size |
| aec.notch_radius | AEC notch filter radius |
| aec.lb_ch | AEC audio output loopback channel |
| anr.en | ANR enable |
| anr.suppress_level | ANR maximum suppression level of noise |
| anr.hpf_freq | ANR cut-off frequency of HPF pre-filtering |
| anr.bias_sensitive | ANR bias sensitive |

[DRV CONFIG]

| Status | Description |
|---|---|
| mono | audio mono channel |

[VOLUME]

| Status | Description |
|---|---|
| vol | input volume |

[IN FRAME]

| Value | Description |
|---|---|
| in | input id of path |
| sr | current input sample rate |
| ch | current input channel count (sound mode) |
| bit | current input bit width |
| frm_sample | current frame sample |

[OUT FRAME]

| Value | Description |
|---|---|
| out | output id of path |
| sr | current output sample rate (for resampling) |

[AEC CONFIG]

| Status | Description |
|---|---|
| out | output id of path |
| aec.en | AEC enable |
| aec.leak_est_en | AEC leak estimate enable |
| aec.leak_est | AEC initial condition of the leak estimate |
| aec.noise_lvl | AEC noise cancel level |
| aec.echo_lvl | AEC echo cancel level |
| aec.filter_len | AEC internal filter length |
| aec.frm_size | AEC internal frame size |
| aec.notch_radius | AEC notch filter radius |
| aec.lb_ch | AEC audio output loopback channel |

[ANR CONFIG]

| Status | Description |
|---|---|
| out | output id of path |
| anr.en | ANR enable |
| anr.suppress_level | ANR maximum suppression level of noise |
| anr.hpf_freq | ANR cut-off frequency of HPF pre-filtering |
| anr.bias_sensitive | ANR bias sensitive |

Example:

```
------------------------- AUDIOCAP 0  PATH & BIND -----------------------------

in    out   state bind_src            bind_dest

0     0     START (null)              AUDIOOUT_0_IN_0

------------------------- AUDIOCAP 0  DEV CONFIG ------------------------------

max

0

in.sr        in.ch              in.bit        in.frm_sample      frm_num

48000        2                  16            1024               10
```

```
aec.en       aec.leak_est_en     aec.leak_est   aec.noise_lvl

0            0                   0              0

aec.echo_lvl  aec.filter_len     aec.frm_size  aec.notch_radius  aec.lb_ch

0             0                  0             0                 0

anr.en       anr.suppress_level  anr.hpf_freq  anr.bias_sensitive

0            0                   0             0

------------------------- AUDIOCAP 0  DRV CONFIG ------------------------------

mono

1

------------------------- AUDIOCAP 0  VOLUME ----------------------------------

vol

100

------------------------- AUDIOCAP 0  IN FRAME --------------------------------

in    sr     ch    bit   frm_sample

0     48000  2     16    1024

------------------------- AUDIOCAP 0  OUT FRAME -------------------------------

out   sr

0     0

------------------------- AUDIOCAP 0  AEC CONFIG ------------------------------

out

0

aec.en       aec.leak_est_en     aec.leak_est  aec.noise_lvl

0            0                   0             0

aec.echo_lvl  aec.filter_len     aec.frm_size  aec.notch_radius  aec.lb_ch

0             0                  0             0                 0

------------------------- AUDIOCAP 0  ANR CONFIG ------------------------------

out  anr.en  anr.suppress_level  anr.hpf_freq  anr.bias_sensitive

0    0       0                   0             0

------------------------- AUDIOCAP 0  OUT WORK STATUS -------------------------

out   NEW   drop  wrn   err   PROC  drop  wrn   err   PUSH  drop  wrn   err

0     47    0     0     0     47    0     0     0     47    0     0     0

------------------------- AUDIOCAP 0  USER WORK STATUS ------------------------
```

```
out   PULL  drop  wrn   err   REL
0     0     0     0     0     0
```

### 3.1.2 debug command

```
[debug port]
echo debug [dev] [i/o] [mask] > /proc/hdal/acap/cmd
where [dev] = d0 , [i/o] = i0, i1, i2, ..., o0, o1, o2, ... , [mask] = show info mask

[ Sample ]
echo debug d0 o0 mfff > /proc/hdal/acap/cmd
```

this debug command can show more debug log on console

```
root@NVTEVM:~$ hd_audio_capture_only
[ 2243.828960] hd_reset - begin
[ 2243.833958] hd_reset - end
HDAL_VERSION: 00[ 2243.838071]  "audcap".out[0]: set aud-max-frame(10) aud-max-bitpersec(16)
aud-max-sndmode(2) aud-max-samplerate(48000,-2141852768)
[ 2243.851277]  "audcap".ctrl: set param(00019016)=1024

[ 2243.858471]  "audcap".out[0]: set param(00019005)=10
[ 2243.864460]  "audcap".out[0]: set param(00019018)=0
[ 2243.870302]  "audcap".out[0]: set param(00019011)=0
[ 2243.876057]  "audcap".out[0]: set param(00019012)=0
[ 2243.881829]  "audcap".out[0]: set aud-max-frame(10) aud-max-bitpersec(16) aud-max-sndmode(2)
aud-max-samplerate(0,164)
[ 2243.893387]  "audcap".out[0]: set param(00019003)=1
[ 2243.899139]
[ 2243.899139]  "audcap".out[0]: open begin, state=0
[ 2243.906272]  "audcap".out[0]: cmd OPEN
[ 2243.911160]  "audcap".out[0]: open end, state=1
[ 2243.916587]  "audcap".ctrl: get param(00019014)=310398976
[ 2243.922857]  "audcap".ctrl: get param(00019015)=45928
[ 2243.928779]  "audcap".out[0]: set aud-bitpersec(16) aud-sndmode(2) samplecnt(0)
[ 2243.936955]  "audcap".out[0]: set aud-samplerate(48000,2)
[ 2243.943222]  "audcap".ctrl: set param(00019010)=1024
[ 2243.949063]  "audcap".out[0]: set aud-samplerate(0,1024)
[ 2243.955380]
[ 2243.955380]  "audcap".out[0]: start begin, state=1
[ 2243.962607]  "audcap".out[0]: cmd RDYSYNC
[ 2243.967486]  "audcap".out[0]: cmd START
[ 2244.173095]  "audcap".out[0]: start end, state=2
Enter q to exit, Enter d to debug

dump main bitstream to file (/mnt/sd/audio_bs_16_2_48000_pcm.dat) ....

if you want to stop, enter "q" to exit !!

q
[ 2247.181497]
[ 2247.181497]  "audcap".out[0]: stop begin, state=2
[ 2247.188685]  "audcap".out[0]: cmd STOP
[ 2247.193495]  "audcap".out[0]: stop end, state=1
[ 2247.198928]
[ 2247.198928]  "audcap".out[0]: close begin, state=1
[ 2247.206151]  "audcap".out[0]: cmd CLOSE
[ 2247.210976]  "audcap".out[0]: close end, state=0
```

### 3.1.3 trace command

```
[trace port]
echo trace [dev] [i/o] [mask] > /proc/hdal/acap/cmd
where [dev] = d0 , [i/o] = i0, i1, i2, ..., o0, o1, o2, ... , [mask] = show info mask

[ Sample ]
echo trace d0 o0 mfff > /proc/hdal/acap/cmd
```

this trace command could enable module internal debug message to know what's going on for the AUDIOCAPTURE module.

### 3.1.4 probe command

```
[probe port]
```

```
echo probe [dev] [i/o] [mask] > /proc/hdal/acap/cmd
where [dev] = d0 , [i/o] = i0, i1, i2, ..., o0, o1, o2, ... , [mask] = show info mask

[ Sample ]
echo probe d0 o0 mffff > /proc/hdal/acap/cmd
```

this probe command could print per-data status

```
[ 2522.848423] "audcap".out[0] - NEW - new -- h=00000001 size=00000000 addr=00000001 OK
[ 2522.857270] "audcap".out[0] - PUSH - rel -- h=00000009 (result=0) OK
[ 2522.864702] "audcap".out[0] - PUSH - rel -- h=00000001 (result=0) OK
[ 2522.868444] "audcap".out[0] - NEW - new -- h=00000002 size=00000000 addr=00000002 OK
[ 2522.880814] "audcap".out[0] - PUSH - rel -- h=00000002 (result=0) OK
[ 2522.889769] "audcap".out[0] - NEW - new -- h=00000001 size=00000000 addr=00000001 OK
[ 2522.898557] "audcap".out[0] - PUSH - rel -- h=00000001 (result=0) OK
[ 2522.911102] "audcap".out[0] - NEW - new -- h=00000001 size=00000000 addr=00000001 OK
[ 2522.919878] "audcap".out[0] - PUSH - rel -- h=00000001 (result=0) OK
```

### 3.1.5 perf command

```
[perf port]
echo perf [dev] [i/o] > /proc/hdal/acap/cmd

[ Sample ]
echo perf d0 o0 > /proc/hdal/acap/cmd
```

this perf command could print data count per second

```
[   86.934367] "audcap".out[0] Perf! -- (Audio) 0 KSample/sec
[   87.935854] "audcap".out[0] Perf! -- (Audio) 0 KSample/sec
[   88.938517] "audcap".out[0] Perf! -- (Audio) 0 KSample/sec
```

### 3.1.6 save command

```
[save port]
echo save [dev] [i/o] [count] > /proc/hdal/acap/cmd
where [count] means how many i/o datas to save

[ Sample ]
```

```
echo save d0 o0 > /proc/hdal/acap /cmd
```

this save command could save i/o data to SDCard for debug purpose.

```
[ 2623.112009] save i/o begin: "audcap".out[0] count=1

[ 2623.131650] "audcap".out[0] Save -- h=00000001 t=000000009dbbc1ac (ARAW: 16.2.48000 9400c368
00000000 4096)

[ 2623.142721] "audcap".out[0] Save -- //mnt//sd//isf_  audcap_out[0]_16_2_48000_c0.aud ok


[ 2623.151593] save port end
```

## 3.2 Debug menu for IPC

The currently supported audiocapture module debug menu is as below.

```
==============================
 AUDIOCAP
------------------------------
 01 : dump info
------------------------------
```

User can choose the number to dump the status what you want. The dump result is just like the example shows on 3.1.1.

The proc command and debug menu mapping table is as below:

| Proc command | Debug menu |
|---|---|
| cat /proc/hdal/acap/info | dump audiocapture information |

## 3.3 proc command for NVR

### 3.3.1 Dump info

```
[dump info]
cat /proc/videograph/hdal_setting
```

The result will show the audiocapture information.

```
root@NVTEVM:/$ cat /proc/videograph/hdal_setting
```

```
------------------------- AUDIOCAP 0  PATH & BIND -----------------------------
in      out     state   bind_src              bind_dest
0       0       START   -                   AUDIOENC_0_IN_0
------------------------- AUDIOCAP 0  IN -----------------------------
out     rate    bit     samples
0       8000    16      320     MONO
```

## 3.4 Debug menu for NVR

Calling hd_debug_run_menu() from app will pop out debug_menu.
The currently supported audiocapture module debug menu is as below.

```
==============================
 AUDIOCAP
------------------------------
 01 : dump status
------------------------------
 254 : Quit
 255 : Return
------------------------------
1
Run: 01 : dump status
------------------------- AUDIOCAP 0  PATH & BIND -----------------------------
in      out     state   bind_src              bind_dest
0       0       START   -                   AUDIOENC_0_IN_0
------------------------- AUDIOCAP 0  IN -----------------------------
out     rate    bit     samples
0       8000    16      320     MONO
```

User can choose the number to dump the status what you want. The dump result is just like the example shown on 3.3.

# 4 Sample Codes

## 4.1 audio_capture_only (IPC)

This sample code demonstrates how to use the single trigger operation to get the PCM data.

```
/* Set cap configuration */
ret = hd_audiocap_open(0, HD_AUDIOCAP_0_CTRL, &audio_cap_ctrl); //open this for device control
audio_dev_cfg.in_max.sample_rate = HD_AUDIO_SR_48000;
audio_dev_cfg.in_max.sample_bit = HD_AUDIO_BIT_WIDTH_16;
audio_dev_cfg.in_max.mode = HD_AUDIO_SOUND_MODE_STEREO;
audio_dev_cfg.in_max.frame_sample = 1024;
audio_dev_cfg.frame_num_max = 10;
ret = hd_audiocap_set(audio_cap_ctrl, HD_AUDIOCAP_PARAM_DEV_CONFIG, &audio_dev_cfg);
if (ret != HD_OK) { return ret; }
audio_drv_cfg.mono = HD_AUDIO_MONO_RIGHT;
ret = hd_audiocap_set(audio_cap_ctrl, HD_AUDIOCAP_PARAM_DRV_CONFIG, &audio_drv_cfg);
if (ret != HD_OK) { return ret; }

/* Set cap parameter */
ret = hd_audiocap_open(HD_AUDIOCAP_0_IN_0, HD_AUDIOCAP_0_OUT_0, &audio_cap_path);
audio_cap_param.sample_rate = HD_AUDIO_SR_48000;
audio_cap_param.sample_bit = HD_AUDIO_BIT_WIDTH_16;
audio_cap_param.mode = HD_AUDIO_SOUND_MODE_STEREO;
audio_cap_param.frame_sample = 1024;
ret = hd_audiocap_set(audio_cap_path, HD_AUDIOCAP_PARAM_IN, &audio_cap_param);
if (ret != HD_OK) { return ret; }

/* Pull out buffer */
#define PHY2VIRT_MAIN(pa) (vir_addr_main + (pa - phy_buf_main.buf_info.phy_addr))

ret = hd_audiocap_pull_out_buf(audio_cap_path, &data_pull, 200);

if (ret == HD_OK) {
     UINT8 *ptr = 0;
     UINT32 size = 0;
     hd_audiocap_get(audio_cap_ctrl, HD_AUDIOCAP_PARAM_BUFINFO, &phy_buf_main);
     vir_addr_main = (UINT32)hd_common_mem_mmap(HD_COMMON_MEM_MEM_TYPE_CACHE,
                                              phy_buf_main.buf_info.phy_addr,
                                              phy_buf_main.buf_info.buf_size);
     ptr = (UINT8 *)PHY2VIRT_MAIN(data_pull.phy_addr[0]);
     size = data_pull.size;
}

/* Release out buffer */
ret = hd_audiocap_release_out_buf(audio_cap_path, &data_pull);
hd_common_mem_munmap((void *)vir_addr_main, phy_buf_main.buf_info.buf_size);
```

## 4.2 audio_livesound(NVR)

audiocapture doesn't support push/pull operation. User must bind it with audioenc or audioout to get audio data or livesound functions. The following demonstrates how to run livesound by using binding method.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <pthread.h>
```

```
#include "hdal.h"


typedef struct _AUDIO_LIVESOUND {
     UINT32 is_exit;
     HD_PATH_ID audcap_path_id;
     HD_PATH_ID audout_path_id;
} AUDIO_LIVESOUND;

HD_RESULT init_module(void)
{
    HD_RESULT ret;
      if((ret = hd_audiocap_init()) != HD_OK)
        return ret;
    if((ret = hd_audioout_init()) != HD_OK)
        return ret;
    return HD_OK;
}


HD_RESULT open_module(AUDIO_LIVESOUND *p_ls_info)
{
    HD_RESULT ret;
      if((ret = hd_audiocap_open(HD_AUDIOCAP_0_IN_0, HD_AUDIOCAP_0_OUT_0,
&p_ls_info->audcap_path_id)) != HD_OK)
        return ret;
      if((ret = hd_audioout_open(HD_AUDIOOUT_0_IN_0, HD_AUDIOOUT_0_OUT_0,
&p_ls_info->audout_path_id)) != HD_OK)
        return ret;
    return HD_OK;
}

HD_RESULT close_module(AUDIO_LIVESOUND *p_ls_info)
{
    HD_RESULT ret;
      if((ret = hd_audiocap_close(p_ls_info->audcap_path_id)) != HD_OK)
        return ret;
      if((ret = hd_audioout_close(p_ls_info->audout_path_id)) != HD_OK)
        return ret;
    return HD_OK;
}

HD_RESULT exit_module(void)
{
    HD_RESULT ret;
      if((ret = hd_audiocap_uninit()) != HD_OK)
        return ret;
    if((ret = hd_audioout_uninit()) != HD_OK)
        return ret;
    return HD_OK;
}

HD_RESULT set_param(AUDIO_LIVESOUND *p_livesound_info)
{
    HD_RESULT ret;
      HD_AUDIOCAP_IN audiocap_param;
      HD_AUDIOOUT_OUT audioout_param;
```

```
	//set audiocap parameters
	ret = hd_audiocap_get(p_livesound_info->audcap_path_id, HD_AUDIOCAP_PARAM_IN, 
&audiocap_param);
	if (ret != HD_OK) {
		printf("hd_audiocap_get(HD_AUDIOCAP_PARAM_IN) fail\n");
		goto exit;
	}
	audiocap_param.sample_rate = HD_AUDIO_SR_8000;
	audiocap_param.sample_bit = HD_AUDIO_BIT_WIDTH_16;
	audiocap_param.mode = HD_AUDIO_SOUND_MODE_MONO;
	audiocap_param.frame_sample = 320;  // for 25fps: 8000/25=320
	ret = hd_audiocap_set(p_livesound_info->audcap_path_id, HD_AUDIOCAP_PARAM_IN, 
&audiocap_param);
	if (ret != HD_OK) {
		printf("hd_audiocap_set(HD_AUDIOCAP_PARAM_IN) fail\n");
		goto exit;
	}

	//set audioout parameters
	ret = hd_audioout_get(p_livesound_info->audout_path_id, HD_AUDIOOUT_PARAM_OUT, 
&audioout_param);
	if (ret != HD_OK) {
		printf("hd_audioout_get(HD_AUDIOOUT_PARAM_OUT) fail\n");
		goto exit;
	}
	audioout_param.sample_rate = HD_AUDIO_SR_8000;
    audioout_param.sample_bit = HD_AUDIO_BIT_WIDTH_16;
    audioout_param.mode = HD_AUDIO_SOUND_MODE_MONO;
	ret = hd_audioout_set(p_livesound_info->audout_path_id, HD_AUDIOOUT_PARAM_OUT, 
&audioout_param);
	if (ret != HD_OK) {
		printf("hd_audioout_get(HD_AUDIOOUT_PARAM_OUT) fail\n");
		goto exit;
	}

exit:
	return ret;
}

int main(void)
{
    HD_RESULT ret;
    INT key;
    AUDIO_LIVESOUND livesound_info = {0};

    //init hdal
	ret = hd_common_init(1);
    if(ret != HD_OK) {
        printf("common init fail\n");
        goto exit;
    }

    // init audiocap and audioout modules
    ret = init_module();
    if(ret != HD_OK) {
        printf("init fail\n");
        goto exit;
    }

    //open audiocap and audioout modules
    ret = open_module(&livesound_info);
    if(ret != HD_OK) {
        printf("open fail\n");
        goto exit;
    }

    //setup runtime parameters
      ret = set_param(&livesound_info);
      if(ret != HD_OK) {
            printf("set param fail\n");
            goto exit;
      }

    //bind live sound: audiocap -> audioout
    ret = hd_audiocap_bind(HD_AUDIOCAP_0_OUT_0, HD_AUDIOOUT_0_IN_0);
    if(ret != HD_OK) {
        printf("bind fail\n");
        goto exit;
    }

      //start to run
    ret = hd_audiocap_start(livesound_info.audcap_path_id);
    if(ret != HD_OK) {
        printf("start audiocap fail\n");
        goto exit;
    }
    ret = hd_audioout_start(livesound_info.audout_path_id);
    if(ret != HD_OK) {
        printf("start audioout fail\n");
        goto exit;
    }

    //main waiting loop
      printf("Enter q to exit\n");
      while (1) {
            key = getchar();
            if (key == 'q') {
                  livesound_info.is_exit = 1;
                  break;
            }
      }

      //stop modules and unbind the connection
      ret = hd_audiocap_stop(livesound_info.audcap_path_id);
      if(ret != HD_OK) {
            printf("stop audiocap fail\n");
      }
      ret = hd_audioout_stop(livesound_info.audout_path_id);
      if(ret != HD_OK) {
            printf("stop audiocap fail\n");
      }
      ret = hd_audiocap_unbind(HD_AUDIOCAP_0_OUT_0);
      if(ret != HD_OK) {
            printf("unbind fail\n");
        }

exit:
        //close and uninit modules
        ret = close_module(&livesound_info);
        if(ret != HD_OK) {
            printf("close fail\n");
        }
        ret = exit_module();
        if(ret != HD_OK) {
            printf("exit fail\n");
        }
        ret = hd_common_uninit();
    if(ret != HD_OK) {
        printf("uninit fail\n");
    }
        return 0;
```

}}

# 5 Frequently asked questions

## 5.1 [NVR ONLY]

TBD

## 5.2 [IPCAM ONLY]

### 5.2.1 Sample rate

The sample rate of audiocapture and audioout must be the same when they start simultaneously.

### 5.2.2 Volume

Volume mapping table. When volume is larger than 100 (100~160), each step will increase the volume by 0.5 dB digital gain.

| Volume | Step | dB |
|---|---|---|
| 0 | 0 | mute |
| 1-11 | 1 | -27 |
| 12-23 | 2 | -24 |
| 24-35 | 3 | -21 |
| 36-47 | 4 | -18 |
| 48-59 | 5 | -15 |
| 60-71 | 6 | -12 |
| 72-83 | 7 | -9 |
| 84-100 | 8 | -6 |