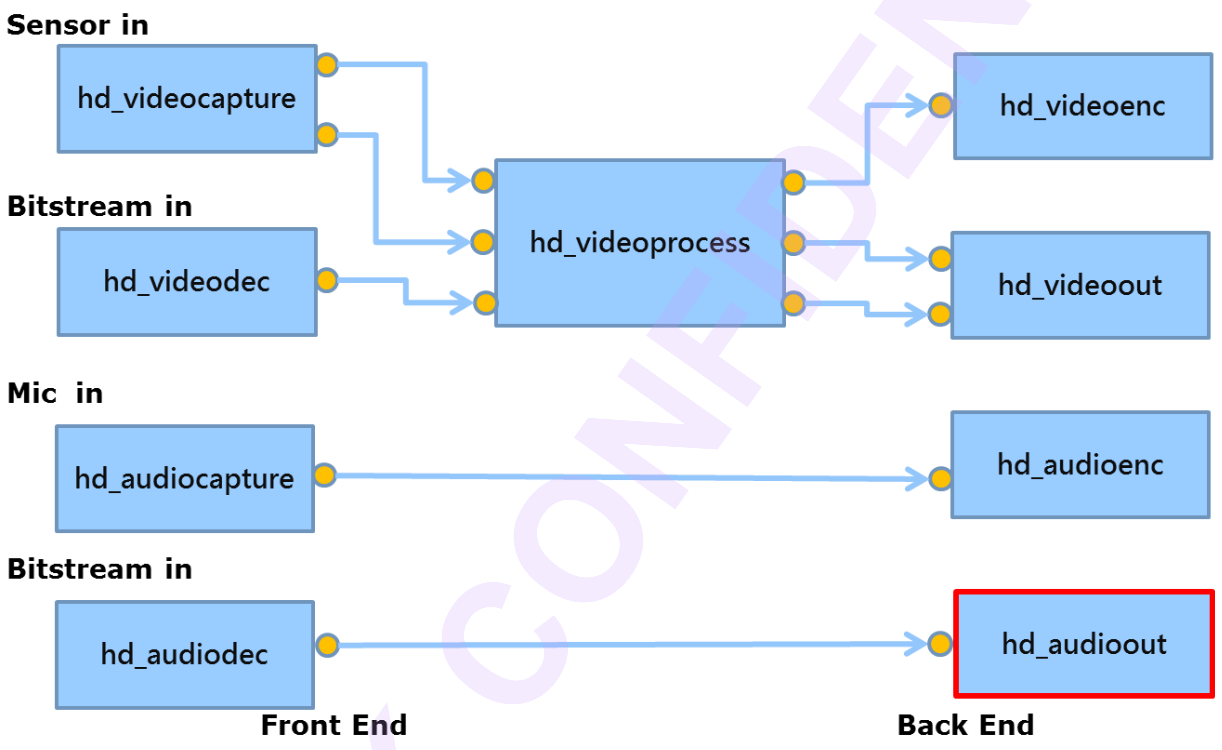
# Novatek HDAL Design Specification - hd_audioout

# *Table of Content*

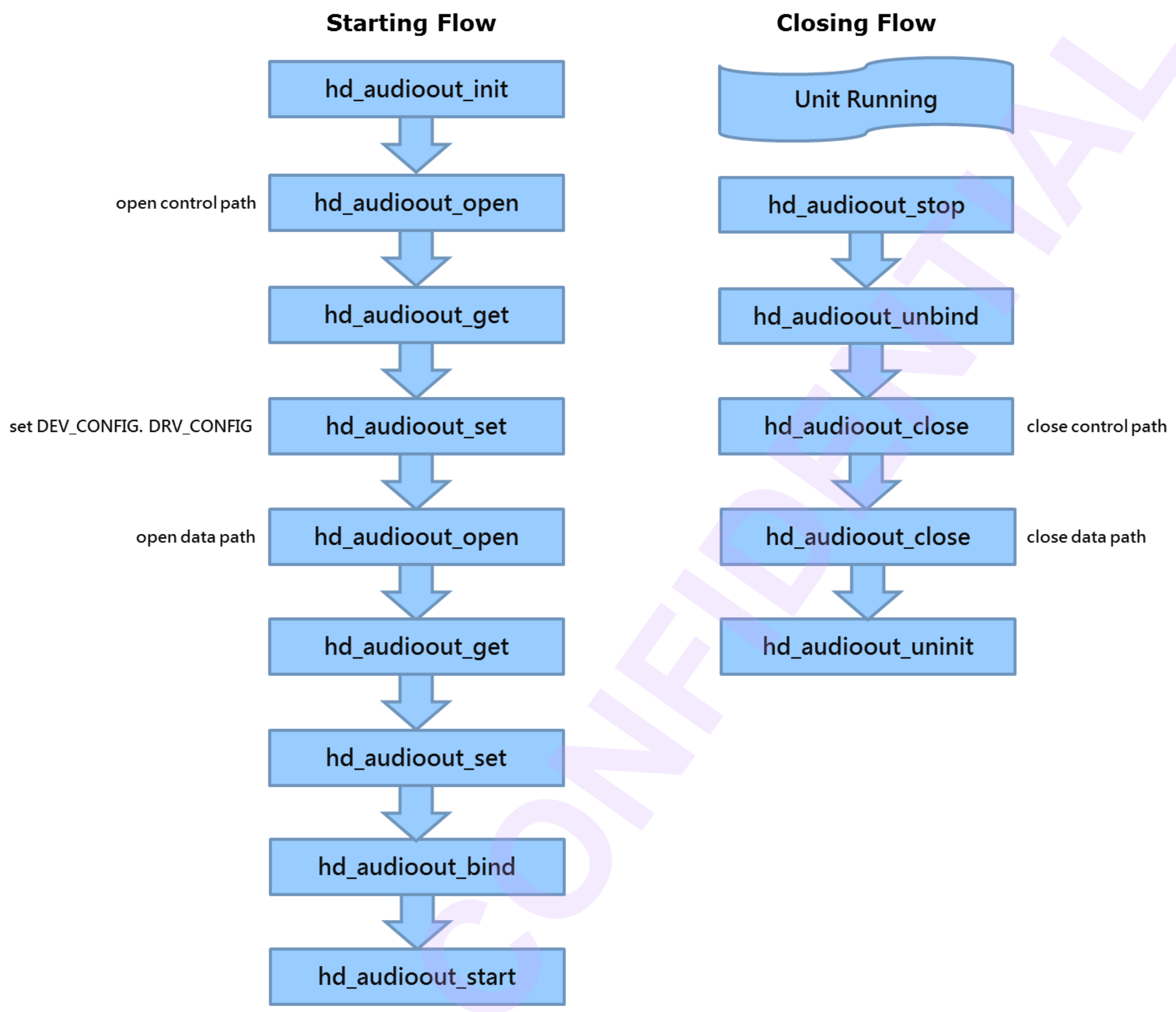
# 1 Introduction

The major purpose of hd_audioout is to send raw audio data to 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.

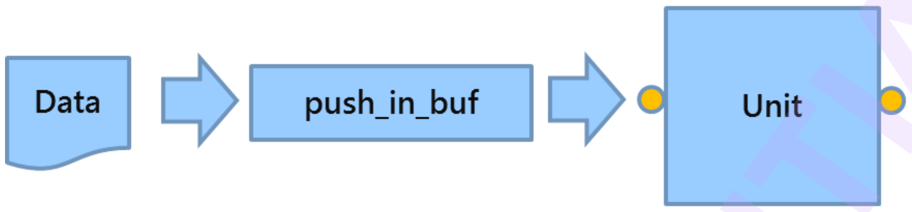


Now, below section in this chapter is mainly about what things to do in those functions above.

## 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.

**Input**



# 2 Function and data structure definition

## 2.1 General function

### 2.1.1 hd_audioout_init

[Description]
Initialize the unit

[Syntax]
HD_RESULT hd_audioout_init(VOID);

[Parameter]

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

[Return Value]

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

### 2.1.2 hd_audioout_open

[Description]
Open the unit

[Syntax]
HD_RESULT hd_audioout_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_audioout_get

[Description]

Get parameters from unit by path id

[Syntax]

HD_RESULT hd_audioout_get(HD_PATH_ID path_id, HD_AUDIOOUT_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_audioout_set

[Description]

Set parameters to unit by path id

[Syntax]

HD_RESULT hd_audioout_set(HD_PATH_ID path_id, HD_AUDIOOUT_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_audioout_start

[Description]

Start the unit

[Syntax]

HD_RESULT hd_audioout_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 |

### 2.1.6 hd_audioout_stop

[Description]
Stop the unit

[Syntax]
HD_RESULT hd_audioout_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 |

### 2.1.7 hd_audioout_close

[Description]
Close the unit

[Syntax]
HD_RESULT hd_audioout_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 |

### 2.1.8 hd_audioout_uninit

[Description]
Uninitialize the unit

[Syntax]
HD_RESULT hd_audioout_uninit(VOID);

[Parameter]

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

[Return Value]

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

### 2.1.9 hd_audioout_start_list

[Description]
Do start for a list of paths

[Syntax]
HD_RESULT hd_audioout_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. |

### 2.1.10 hd_audioout_stop_list

[Description]
Do stop for a list of paths

[Syntax]
HD_RESULT hd_audioout_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. |

### 2.1.11 hd_audioout_push_in_buf

[Description]
Push the audio frame buffer to audioout module

[Syntax]
HD_RESULT hd_audioout_push_in_buf(HD_PATH_ID path_id, HD_AUDIO_FRAME*

p_in_audio_frame, INT32 wait_ms)

[Parameter]

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

[Return Value]

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

## 2.2 Data structure definition

The function hd_audioout_get and hd_audioout_set provides the following parameter IDs:

- HD_AUDIOOUT_PARAM_DEVCOUNT
  - NVR/IPC. support get with ctrl path
  - using HD_DEVCOUNT struct (device id max count)
- HD_AUDIOOUT_PARAM_SYSCAPS
  - NVR/IPC. support get with ctrl path
  - using HD_AUDIOOUT_SYSCAPS
- HD_AUDIOOUT_PARAM_SYSINFO
  - NVR/IPC. support get with ctrl path
  - using HD_AUDIOOUT_SYSINFO
- HD_AUDIOOUT_PARAM_DEV_CONFIG
  - NVR/IPC. support get/set with ctrl path
  - using HD_AUDIOOUT_DEV_CONFIG struct
- HD_AUDIOOUT_PARAM_DRV_CONFIG
  - NVR/IPC. support get/set with ctrl path
  - using HD_AUDIOOUT_DRV_CONFIG struct
- HD_AUDIOOUT_PARAM_OUT

- ☐ NVR/IPC. support get/set with i/o path
- ☐ using HD_AUDIOOUT_OUT struct

- HD_AUDIOOUT_PARAM_IN
  - ☐ NVR/IPC. support get/set with i/o path
  - ☐ using HD_AUDIOOUT_IN struct
- HD_AUDIOOUT_PARAM_VOLUME
  - ☐ NVR/IPC. support get/set with ctrl path
  - ☐ using HD_AUDIOOUT_VOLUME struct
- HD_AUDIOOUT_PARAM_CLEAR_BUF
  - ☐ NVR only. support set with i/o path
  - ☐ no parameter

### 2.2.1 HD_ AUDIOOUT_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_AUDIOOUT_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_AUDIOOUT_SRCAPS |
| support_out_sr | sample rate capability of output, cap of HD_AUDIOOUT_SRCAPS |

### 2.2.2 HD_AUDIOOUT_DEV_CONFIG

[Description]

Device configuration

[Parameter]

| Value | Description |
|---|---|
| out_max.sample_rate | maximum output sample rate<br>Please refer to HD_AUDIO_SR enum.<br>Default value: HD_AUDIO_SR_48000<br>IPC: support close-time change. |
| out_max.sample_bit | maximum output 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. |
| out_max.mode | maximum output 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. |
| frame_sample_max | 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 | maximum frame number in buffer<br>Value range: [4, 50]<br>Default value: 10<br>IPC: support close-time change. |
| in_max.sample_rate | maximum input 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. |

### 2.2.3 HD_AUDIOOUT_SYSINFO

[Description]
System information

[Parameter]

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

### 2.2.4 HD_AUDIOOUT_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_LEFT<br>IPC: support stop-time change. |
| output | NVR/IPC. audio output path<br>Please refer to HD_AUDIOOUT_OUTPUT enum.<br>Default value:<br>HD_AUDIOOUT_OUTPUT_SPK<br>IPC: support stop-time change. |
| ssp_config | NVR only. audio ssp config |

### 2.2.5 HD_AUDIOOUT_SSP_CONFIG

[Description]
SSP configuration

[Parameter]

| Value | Description |
|---|---|
| enable | NVR only. playout enable for each ssp interface |
| resample_ratio | NVR only. resample ratio of for each ssp interface |
| playback_chmap | NVR only. channel map of playbach ch |

### 2.2.6 HD_AUDIOOUT_OUT

[Description]
Output parameter

[Parameter]

| Value | Description |
|---|---|
| sample_rate | sample rate<br>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. |

### 2.2.7 HD_AUDIOOUT_IN

[Description]

Input parameter

[Parameter]

| Value | Description |
|---|---|
| sample_rate | input 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_AUDIOOUT_VOLUME

[Description]
Output volume

[Parameter]

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

### 2.2.9 HD_AUDIOOUT_PARAM_CLEAR_BUF

[Description]
Clear audio data in the queue.

[Parameter]
No parameters.

# 3 Debug command

The audioout 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/aout/info
```

The result will show the audioout information by five parts.

1. **PATH & BIND**: bind status of hd_audioout.
2. **DEV CONFIG**: device configuration, referring to HD_AUDIOOUT_DEV_CONFIG.
3. **DRV CONFIG**: driver configuration, referring to HD_AUDIOOUT_DRV_CONFIG.
4. **Volume**: volume configuration, referring to HD_AUDIOOUT_VOLUME.
5. **OUT FRAME**: output configuration for resampling, referring to HD_AUDIOOUT_OUT.
6. **IN FRAME**: input configuration, referring to HD_AUDIOOUT_IN.

[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 |
| out.sr | maximum sample rate |
| out.ch | maximum channel count (sound mode) |

| | |
|---|---|
| out.bit | maximum bit width |
| frm_sample | maximum frame sample |
| frm_num | maximum frame number |

[DRV CONFIG]

| Status | Description |
|---|---|
| mono | audio mono channel |
| output | audio output path |

[VOLUME]

| Status | Description |
|---|---|
| vol | output volume |

[OUT FRAME]

| Value | Description |
|---|---|
| out | output id of path |
| sr | current output sample rate |
| ch | current output channel count (sound mode) |
| bit | current output bit width |

[IN FRAME]

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

Example:

```
------------------------- AUDIOOUT 0  PATH & BIND -----------------------------

in    out   state bind_src            bind_dest

0     0    START AUDIOCAP_0_OUT_0      (null)

------------------------- AUDIOOUT 0  DEV CONFIG ------------------------------

max   out.sr     out.ch    out.bit   frm_sample  frm_num

0     48000     2        16       1024       10

------------------------- AUDIOOUT 0  DRV CONFIG ------------------------------

mono  output
```

```
0    0

------------------------- AUDIOOUT 0  VOLUME ----------------------------------

vol

50

------------------------- AUDIOOUT 0  OUT FRAME -------------------------------

out   sr    ch   bit

0    48000  2    16

------------------------- AUDIOOUT 0  IN FRAME --------------------------------

in    sr

0    0

------------------------- AUDIOOUT 0  IN WORK STATUS --------------------------

in    PUSH  drop  wrn   err   PROC  drop  wrn   err   REL

0     47    0     0     0     47    0     0     0     47
```

### 3.1.2 debug command

```
[debug port]

echo debug [dev] [i/o] [mask] > /proc/hdal/aout/cmd

where [dev] = d0 , [i/o] = i0, i1, i2, ..., o0, o1, o2, ... , [mask] = show info mask


[ Sample ]

echo debug d0 o0 mfff > /proc/hdal/aout/cmd
```

this debug command can show more debug log on console

```
root@NVTEVM:~$ hd_audio_output_only
[   67.920948] hd_reset - begin
[   67.925729] hd_reset - end
HDAL_VERSION: 00010001:00010001
[   67.933998]  "audout".out[0]: set aud-max-frame(10) aud-max-bitpersec(16) aud-max-sndmode(2)
aud-max-samplerate(48000,0)
[   67.945770]  "audout".out[0]: set param(00018008)=1024
[   67.951781]  "audout".out[0]: set param(00018005)=10
[   67.957629]  "audout".out[0]: set aud-max-frame(10) aud-max-bitpersec(16) aud-max-sndmode(2)
aud-max-samplerate(0,0)
[   67.969013]  "audout".out[0]: set param(00018003)=0
[   67.974759]  "audout".out[0]: set param(00018002)=0
[   67.980511]
[   67.980511]  "audout".out[0]: open begin, state=0
[   67.987645]  "audout".out[0]: cmd OPEN
[   67.992422]  "audout".out[0]: open end, state=1
[   67.997843]  "audout".out[0]: set aud-bitpersec(16) aud-sndmode(2) samplecnt(0)
[   68.006070]  "audout".out[0]: set aud-samplerate(48000,2)
[   68.012342]  "audout".ctrl: set param(00018001)=100
[   68.018098]  "audout".out[0]: set aud-samplerate(0,100)
[   68.024323]
[   68.024323]  "audout".out[0]: start begin, state=1
[   68.031558]  "audout".out[0]: cmd RDYSYNC
play file: [/mnt[   68.036440]  "audout".out[0]: cmd START
/sd/audio_bs_16_[   68.042922]  "audout".out[0]: start end, state=2
2_48000_pcm.dat]
len file: [/mnt/sd/audio_bs_16_2_48000_pcm.len]
Enter q to exit
alloc bs_buf: start(0x75300000) curr(0x75300000) end(0x75400000) size(0x100000)
q
[   69.773905]
[   69.773905]  "audout".out[0]: stop begin, state=2
[   69.781056]  "audout".out[0]: cmd STOP
[   69.785865]  "audout".out[0]: stop end, state=1
[   69.791295]
[   69.791295]  "audout".out[0]: close begin, state=1
[   69.798533]  "audout".out[0]: cmd CLOSE
[   69.803322]  "audout".out[0]: close end, state=0
```

### 3.1.3 trace command

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

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

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

### 3.1.4 probe command

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

[ Sample ]
echo probe d0 i0 mffff > /proc/hdal/aout/cmd
```

this probe command could print per-data status

```
[  204.278920] "audout".in[0] - REL - rel -- h=00000001 (result=0) OK
[  204.300209] "audout".in[0] - REL - rel -- h=00000001 (result=0) OK
[  204.321542] "audout".in[0] - REL - rel -- h=00000001 (result=0) OK
[  204.342873] "audout".in[0] - REL - rel -- h=00000001 (result=0) OK
[  204.364207] "audout".in[0] - REL - rel -- h=00000001 (result=0) OK
[  204.385541] "audout".in[0] - REL - rel -- h=00000001 (result=0) OK
```

### 3.1.5 perf command

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

[ Sample ]
echo perf d0 i0 > /proc/hdal/aout/cmd
```

### 3.1.6 save command

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

[ Sample ]
echo save d0 i0 > /proc/hdal/aout/cmd
```

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

## 3.2 Debug menu for IPC

The currently supported audioout module debug menu is as below.

```
==============================
 AUDIOOUT
------------------------------
 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/aout/info | dump audioout 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
------------------------- AUDIOOUT 0  PATH & BIND -----------------------------
in      out     state   bind_src              bind_dest
0       0       OPEN    -                     -
------------------------- AUDIOOUT 0  IN -----------------------------
out     rate    bit     samples
0       8000
```

## 3.4 Debug menu for NVR

Calling hd_debug_run_menu() from app will pop out debug_menu.

The currently supported audioout module debug menu is as below.

```
==============================
 AUDIOOUT
------------------------------
 01 : dump status
------------------------------
 254 : Quit
 255 : Return
------------------------------
1
Run: 01 : dump status
------------------------- AUDIOOUT 0  PATH & BIND -----------------------------
in      out     state   bind_src              bind_dest
0       0       OPEN    -                     -
------------------------- AUDIOOUT 0  IN -----------------------------
out     rate
0       8000
```

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_output_only (IPC)

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

```
/* Set out configuration */
ret = hd_audioout_open(0, HD_AUDIOOUT_0_CTRL, &audio_out_ctrl);
audio_dev_cfg.out_max.sample_rate = HD_AUDIO_SR_48000;
audio_dev_cfg.out_max.sample_bit = HD_AUDIO_BIT_WIDTH_16;
audio_dev_cfg.out_max.mode = HD_AUDIO_SOUND_MODE_STEREO;
audio_dev_cfg.frame_sample_max = 1024;
audio_dev_cfg.frame_num_max = 10;
ret = hd_audioout_set(audio_out_ctrl, HD_AUDIOOUT_PARAM_DEV_CONFIG, &audio_dev_cfg);
if (ret != HD_OK) { return ret; }
audio_drv_cfg.mono = HD_AUDIO_MONO_LEFT;
audio_drv_cfg.output = HD_AUDIOOUT_OUTPUT_SPK;
ret = hd_audioout_set(audio_out_ctrl, HD_AUDIOOUT_PARAM_DRV_CONFIG, &audio_drv_cfg);
```

```
if (ret != HD_OK) { return ret; }

/* Set out parameter */
ret = hd_audioout_open(HD_AUDIOOUT_0_IN_0, HD_AUDIOOUT_0_OUT_0, &audio_out_path);
audio_out_param.sample_rate = HD_AUDIO_SR_48000;
audio_out_param.sample_bit = HD_AUDIO_BIT_WIDTH_16;
audio_out_param.mode = HD_AUDIO_SOUND_MODE_STEREO;
ret = hd_audioout_set(audio_out_path, HD_AUDIOOUT_PARAM_OUT, &audio_out_param);
if (ret != HD_OK) { return ret; }
audio_out_vol.volume = 100;
ret = hd_audioout_set(audio_out_ctrl, HD_AUDIOOUT_PARAM_VOLUME, &audio_out_vol);
if (ret != HD_OK) { return ret; }

/* Push in buffer */
blk = hd_common_mem_get_block(HD_COMMON_MEM_USER_POOL_BEGIN, blk_size, ddr_id);
pa = hd_common_mem_blk2pa(blk);
va = (UINT32)hd_common_mem_mmap(HD_COMMON_MEM_MEM_TYPE_CACHE, pa, blk_size);
fread((void *)va, 1, bs_size, bs_fd);

bs_in_buf.sign = MAKEFOURCC('A','F','R','M');
bs_in_buf.phy_addr[0] = pa;
bs_in_buf.size = bs_size;
bs_in_buf.ddr_id = ddr_id;
bs_in_buf.timestamp = hd_gettime_us();
bs_in_buf.bit_width = HD_AUDIO_BIT_WIDTH_16;
bs_in_buf.sound_mode = HD_AUDIO_SOUND_MODE_STEREO;
bs_in_buf.sample_rate = HD_AUDIO_SR_48000;
ret = hd_audioout_push_in_buf(audio_out_path, &bs_in_buf, 100);
if (ret != HD_OK) {
     return ret;
}

/* Release in buffer */
hd_common_mem_munmap((void *)va, blk_size)
hd_common_mem_release_block(blk);
```

## 4.2 user_audioout (NVR)

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

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <pthread.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "hdal.h"

#define PATTERN_NAME      "audio_8khz_16bit.pcm"
#define FRAME_SAMPLES     1024
#define LOOP_COUNT        50
#define BUFFER_TIME_MS    500

typedef struct {
     //for real use
     HD_AUDIO_FRAME audio_frame;

     //memory info
     UINT64 pool;
     INT size;
     HD_COMMON_MEM_DDR_ID ddr_id;

     //address
     HD_COMMON_MEM_VB_BLK blk;
     VOID *va;

} app_buffer_t;

typedef struct _USER_AUDIOOUT {
     INT mem_fd;
     INT au_frame_ms;
     FILE *pattern_file;
     HD_PATH_ID aout_path_id;
} USER_AUDIOOUT;


#define TIME_DIFF(new_val, old_val)    ((int)(new_val) - (int)(old_val))
unsigned int get_current_time(void)
{
     struct  timeval now_timeval;
     UINT ts_in_ms;
     gettimeofday(&now_timeval, NULL);
     ts_in_ms = 1000 * ((unsigned long long)now_timeval.tv_sec) +
                          ((unsigned long long)now_timeval.tv_usec / 1000);
```

```
	return ts_in_ms;
}

int read_input(FILE *pfile, void *buffer, int read_size)
{
	INT ret;
read:
	ret = fread(buffer, 1, read_size, pfile);
	if (read_size != ret) {
		fseek(pfile, 0, SEEK_SET);
		goto read;
	}
	return ret;
}

INT allocate_buffer(UINT64 pool, int size, HD_COMMON_MEM_DDR_ID ddr_id, app_buffer_t *p_app_buf)
{
	INT ret = 0;
	HD_COMMON_MEM_VB_BLK blk;
	UINT32 pa;
	VOID *va;

	p_app_buf->pool = pool;
	p_app_buf->size = size;
	p_app_buf->ddr_id = ddr_id;

	blk = hd_common_mem_get_block(pool, size, ddr_id);
	if (HD_COMMON_MEM_VB_INVALID_BLK == blk) {
		printf("hd_common_mem_get_block fail\r\n");
		ret =  -1;
		goto exit;
	}
	pa = hd_common_mem_blk2pa(blk);
	if (pa == 0) {
		printf("hd_common_mem_blk2pa fail, blk = %#lx\r\n", blk);
		hd_common_mem_release_block(blk);
		ret =  -1;
		goto exit;
	}
	va = hd_common_mem_mmap(HD_COMMON_MEM_MEM_TYPE_NONCACHE, pa , size);
	p_app_buf->blk = blk;
	p_app_buf->va = va;

	p_app_buf->audio_frame.phy_addr[0] = pa;
exit:
	return ret;
}

INT free_buffer(app_buffer_t *p_app_buf)
{
	INT ret = 0;
	HD_RESULT hd_ret;

	hd_ret = hd_common_mem_munmap(p_app_buf->va, p_app_buf->size);
	if (hd_ret != HD_OK) {
		printf("hd_common_mem_munmap fail\r\n");
		ret =  -1;
```

```
            goto exit;
      }
      hd_ret = hd_common_mem_release_block((HD_COMMON_MEM_VB_BLK)p_app_buf->blk);
      if (hd_ret != HD_OK) {
            printf("hd_common_mem_munmap fail\r\n");
            ret =  -1;
            goto exit;
      }
exit:
      return ret;
}

HD_RESULT prepare_buffers(USER_AUDIOOUT *p_usr_audioout, app_buffer_t *p_audio_buffer)
{
      HD_RESULT ret;
      HD_AUDIO_FRAME *p_audio_frame = &p_audio_buffer->audio_frame;

      p_audio_frame->ddr_id = 0;
      p_audio_frame->size = FRAME_SAMPLES * 2;

      ret = allocate_buffer(HD_COMMON_MEM_COMMON_POOL, p_audio_frame->size,
                                    p_audio_frame->ddr_id, p_audio_buffer);
      if (ret != HD_OK) {
            printf("allocate_buffer fail\n");
            goto exit;
      }

      read_input(p_usr_audioout->pattern_file, p_audio_buffer->va, p_audio_frame->size);
      p_audio_frame->sample_rate = HD_AUDIO_SR_8000;
      p_audio_frame->bit_width = HD_AUDIO_BIT_WIDTH_16;
      p_audio_frame->sound_mode = HD_AUDIO_SOUND_MODE_MONO;

      // au_frame_ms: the time(in ms) of each audio frame
      p_usr_audioout->au_frame_ms = FRAME_SAMPLES * 1000 / p_audio_frame->sample_rate;
exit:
      return ret;
}

HD_RESULT return_buffers(app_buffer_t *p_audio_buffer)
{
      HD_RESULT ret;

      ret = free_buffer(p_audio_buffer);
      if (ret != HD_OK) {
            printf("free_buffer fail\n");
      }
      return ret;
}

int main(int argc, char *argv[])
{
      INT count = 0, elapse_time, au_buf_time;
      app_buffer_t audio_buffer;
      UINT start_time, data_time;
      USER_AUDIOOUT usr_audioout;
      HD_RESULT ret = HD_OK;
```

```
	//prepare resource for audio playback
	memset(&usr_audioout, 0, sizeof(usr_audioout));
	usr_audioout.mem_fd = -1;

	if ((usr_audioout.pattern_file = fopen(PATTERN_NAME, "rb")) == NULL) {
		printf("[ERROR] Open File %s failed!!\n", PATTERN_NAME);
		exit(1);
	}
	printf("Play file: %s\n", PATTERN_NAME);

	if ((usr_audioout.mem_fd = open("/dev/mem", O_RDWR | O_SYNC)) < 0) {
		printf("open /dev/mem failed.\n");
		goto exit;
	}

	//init hdal and open audio modules
	ret = hd_common_init(1);
	if (ret != HD_OK) {
		printf("hd_common_init fail\n");
		goto exit;
	}
	ret = hd_audioout_init();
	if (ret != HD_OK) {
		printf("hd_audioout_init fail\n");
		goto exit;
	}
	ret = hd_audioout_open(HD_AUDIOOUT_0_IN_0, HD_AUDIOOUT_0_OUT_0,
&usr_audioout.aout_path_id);
	if (ret != HD_OK) {
		printf("hd_audioout_open fail\n");
		goto exit;
	}

	//main thread for sending adio data
	start_time = get_current_time();
	data_time = 0;
	while (count++ < LOOP_COUNT) {
retry:
		//control the depth and audio buffer
		elapse_time = TIME_DIFF(get_current_time(), start_time);
		au_buf_time = data_time - elapse_time;
		if (au_buf_time > BUFFER_TIME_MS) {
			usleep(10000);
			goto retry;
		}

		//prepare buffer used for sending audio data
		ret = prepare_buffers(&usr_audioout, &audio_buffer);
		if (ret != HD_OK) {
			printf("prepare_buffers fail\n");
			goto exit;
		}
		data_time += usr_audioout.au_frame_ms;

		//put the data to hdal
		ret = hd_audioout_push_in_buf(usr_audioout.aout_path_id, &audio_buffer.audio_frame,
500);
```

```
            if (ret != HD_OK) {
                printf("hd_audioout_push_in_buf fail\n");
                goto exit;
            }
            printf("Put audio frame len(%lu) - %d\n", audio_buffer.audio_frame.size, count);

            //release the buffer
            ret = return_buffers(&audio_buffer);
            if (ret != HD_OK) {
                printf("prepare_buffers fail\n");
                goto exit;
            }
        }

exit:
        //close modules, release resources, and exit hdal
        ret = hd_audioout_close(usr_audioout.aout_path_id);
        if (ret != HD_OK) {
            printf("hd_audioout_close fail\n");
            return -1;
        }
        ret = hd_audioout_uninit();
        if (ret != HD_OK) {
            printf("hd_audioout_uninit fail\n");
            return -1;
        }
        ret = hd_common_uninit();
        if (ret != HD_OK) {
            printf("hd_common_uninit fail\n");
            return -1;
        }
        if (usr_audioout.mem_fd != -1) {
            close(usr_audioout.mem_fd);
        }
        fclose(usr_audioout.pattern_file);

        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 | PGA gain (dB) |
|---|---|
| 0 | Mute |
| 1-2 | -30.0 |
| 3-5 | -28.8 |
| 6-8 | -27.6 |
| 9-11 | -26.5 |
| 12-14 | -25.3 |
| 15-17 | -24.1 |
| 18-20 | -23.0 |
| 21-23 | -21.8 |
| 24-26 | -20.7 |
| 27-29 | -19.5 |
| 30-32 | -18.3 |
| 33-35 | -17.2 |
| 36-38 | -16.0 |
| 39-41 | -14.9 |
| 42-44 | -13.7 |
| 45-47 | -12.5 |
| 48-50 | -11.4 |
| 51-53 | -10.2 |
| 54-56 | -9.0 |
| 57-59 | -7.9 |
| 60-62 | -6.7 |
| 63-65 | -5.6 |

| | |
|---|---|
| 66-68 | -4.4 |
| 69-71 | -3.2 |
| 72-74 | -2.1 |
| 75-77 | -0.9 |
| 89-80 | +0.1 |
| 81-83 | +1.3 |
| 84-86 | +2.5 |
| 87-89 | +3.6 |
| 90-92 | +4.8 |
| 93-100 | +6.0 |