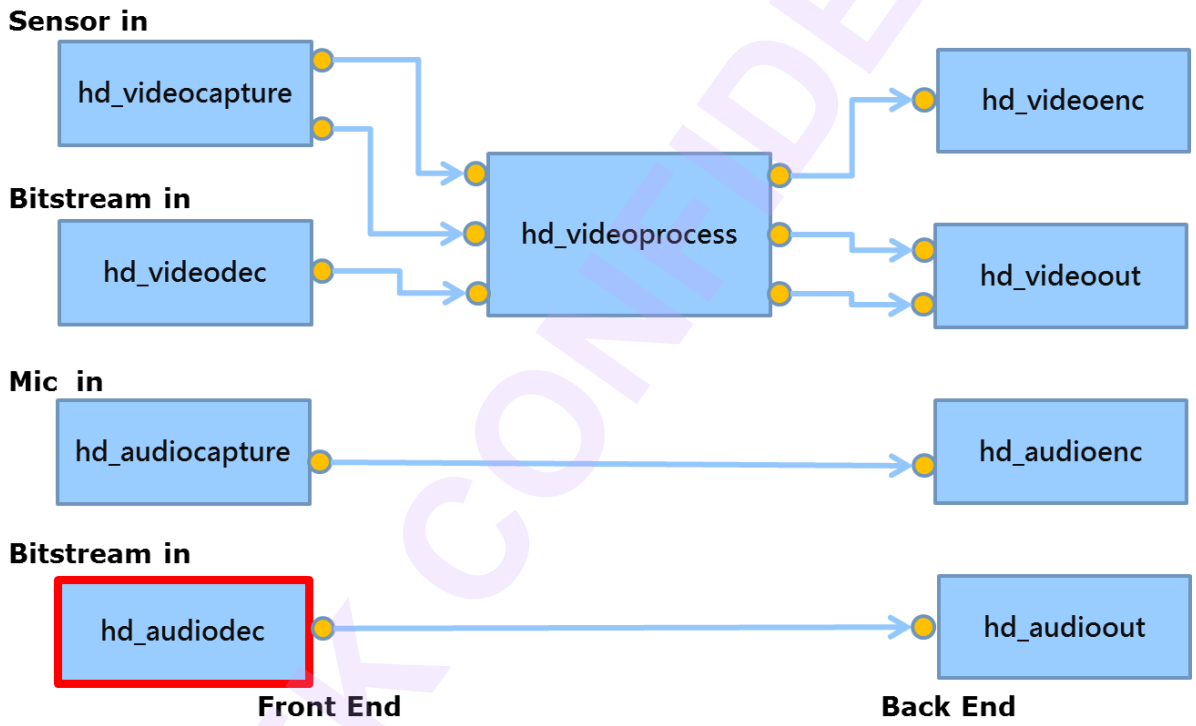
Novatek HDAL Design Specification - hd_audiodec

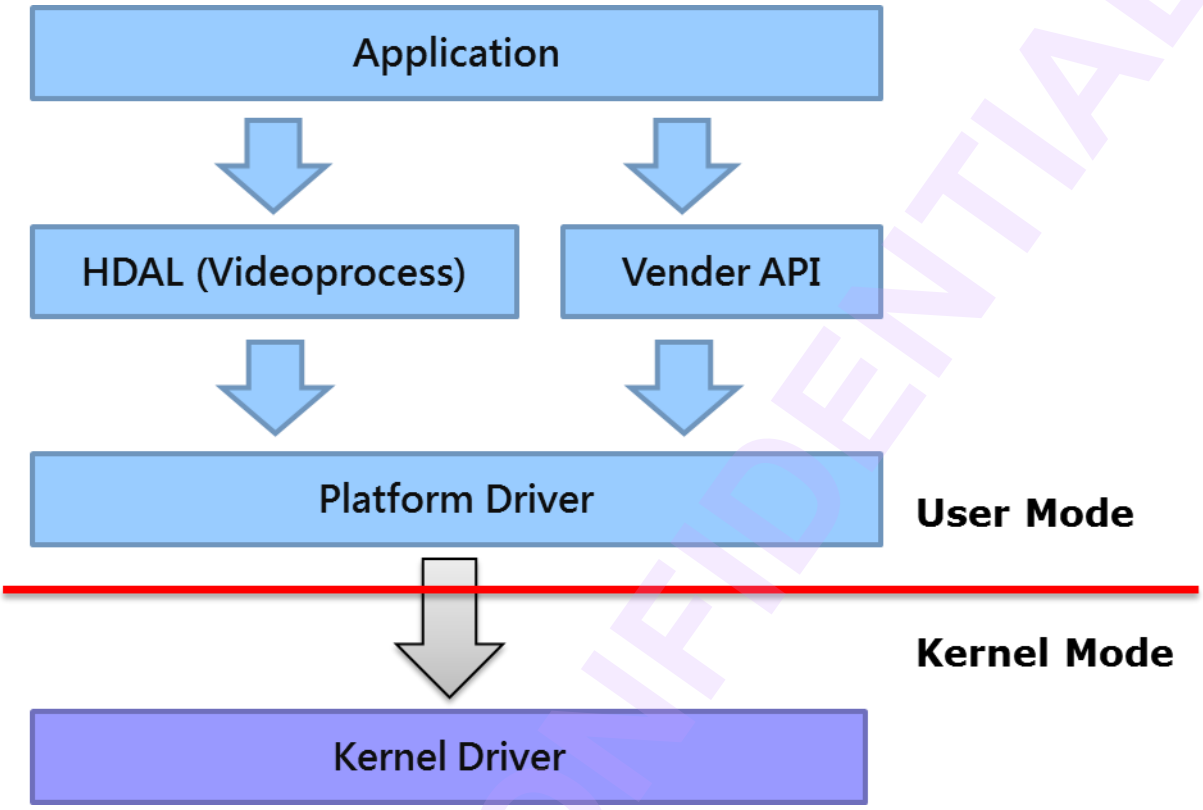
Table of Content

1 Introduction

The major purpose of hd_audiodec is to get bitstream data from upper module, and controls the audio decoder to decode the bitstream data then return the decoded raw data which can be used for sound playback. This document will talk about the red block in the following diagram. The device driver is not the main point in this document.

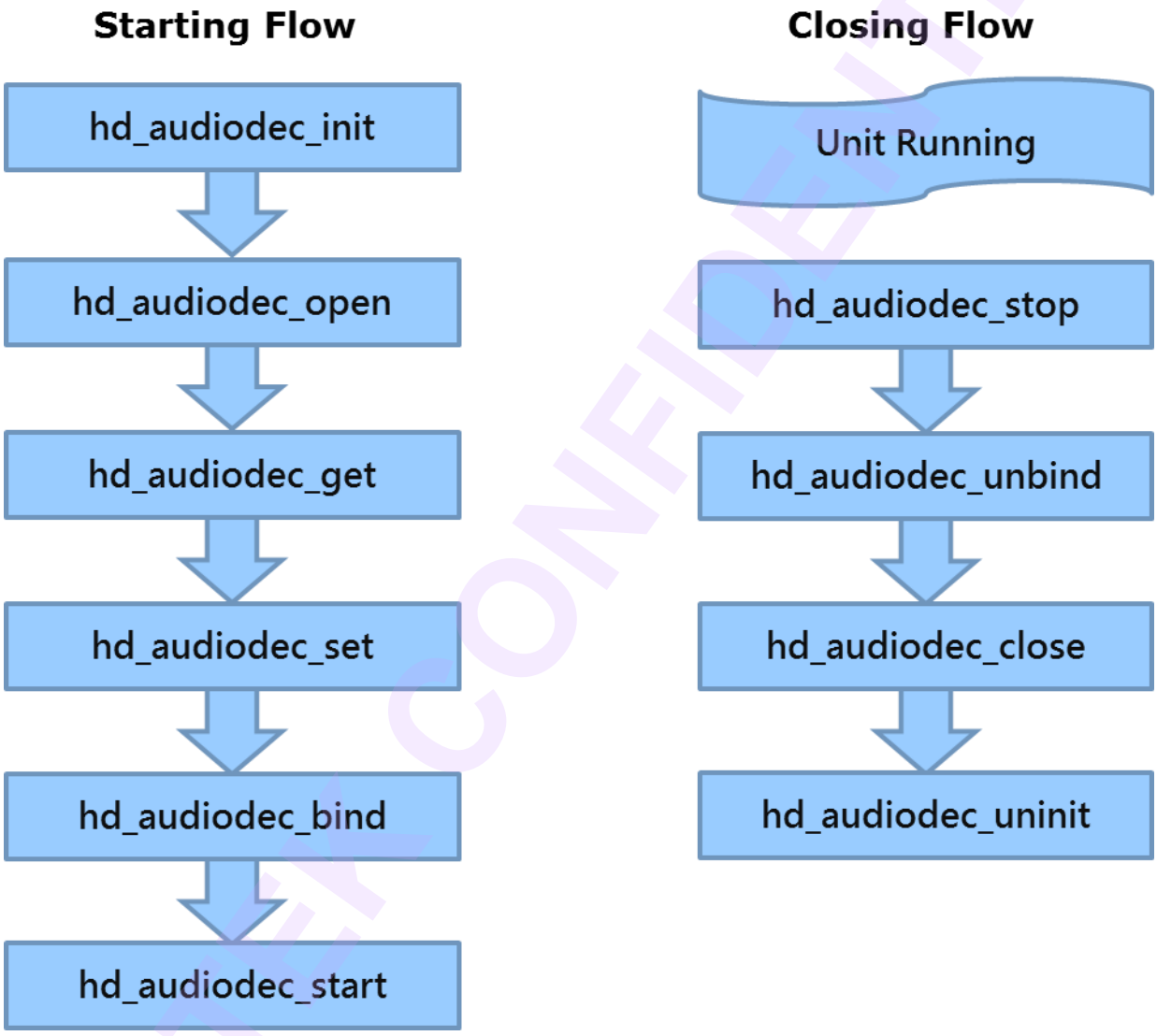


Module diagram is shown as below:



1.1 Basic Flow

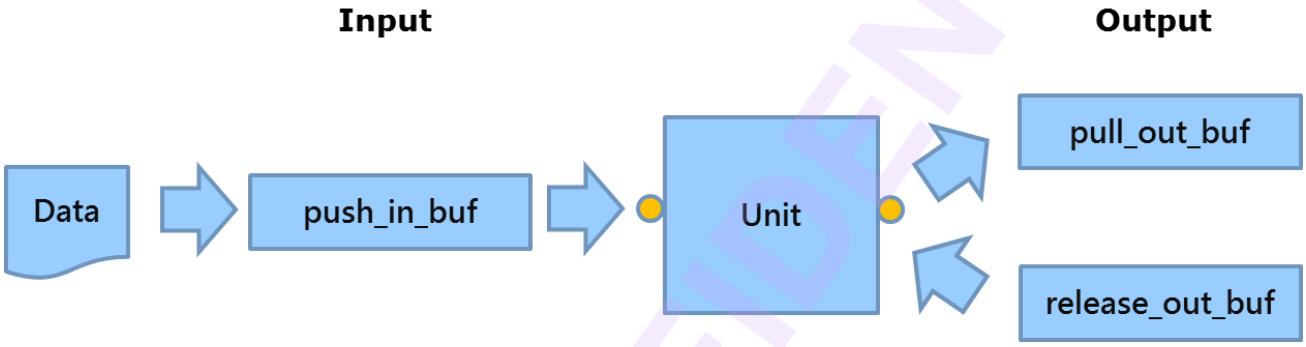
The call sequence is needed to be done correctly for the unit. The standard starting flows of most modules are init, open, get, set and start. The standard closing flows of most modules are stop, unbind, close and uninit. The basic flow is shown as below.



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 bitstream from audio capture; or decode one bitstream to frame by using audio decoder. There are two types of functions for the input port and output port. The sequence for input port is new, push and release; the sequence for output port is pull and release. The flow is shown as below.



2 Function and data structure definition

2.1 General function

2.1.1 hd_audiodec_init

[Description]
Initialize the unit

[Syntax]
HD_RESULT hd_audiodec_init(VOID);

[Parameter]

Value	Description
VOID	Not available

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.2 hd_audiodec_open

[Description]
Open the unit

[Syntax]
HD_RESULT hd_audiodec_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_audiodec_get

[Description]
Get parameters from unit by path id

[Syntax]
HD_RESULT hd_audiodec_get(HD_PATH_ID path_id, HD_AUDIODEC_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_audiodec_set

[Description]
Set parameters to unit by path id

[Syntax]
HD_RESULT hd_audiodec_set(HD_PATH_ID path_id, HD_AUDIODEC_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_audiodec_bind

[Description]
Bind this unit with destination unit

[Syntax]
HD_RESULT hd_audiodec_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_audiodec_start

[Description]
Start the unit

[Syntax]
HD_RESULT hd_audiodec_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_audiodec_stop

[Description]
Stop the unit

[Syntax]
HD_RESULT hd_audiodec_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_audiodec_unbind

[Description]
Unbind the unit

[Syntax]
HD_RESULT hd_audiodec_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_audiodec_close

[Description]
Close the unit

[Syntax]
HD_RESULT hd_audiodec_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_audiodec_uninit

[Description]
Uninitialize the unit

[Syntax]
HD_RESULT hd_audiodec_uninit(VOID);

[Parameter]

Value	Description
VOID	Not available

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.11 hd_audiodec_push_in_buf

[Description]
Push the audio frame buffer to unit

[Syntax]
HD_RESULT hd_audiodec_push_in_buf(HD_PATH_ID path_id, HD_AUDIO_BS* p_in_audio_bs, HD_AUDIO_FRAME* p_user_out_audio_frame, INT32 wait_ms);

[Parameter]

Value	Description
path_id	the path id
p_in_audio_bs	pointer of the input audio bitstream
p_user_out_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.

2.1.12 hd_audiodec_pull_out_buf

[Description]

Pull the audio bitstream buffer from unit

[Syntax]

HD_RESULT hd_audiodec_pull_out_buf(HD_PATH_ID path_id,
HD_AUDIO_FRAEM* 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.

2.1.13 hd_audiodec_release_out_buf

[Description]
Release the audio frame buffer which is get from unit

[Syntax]
HD_RESULT hd_audiodec_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 frame

[Return Value]

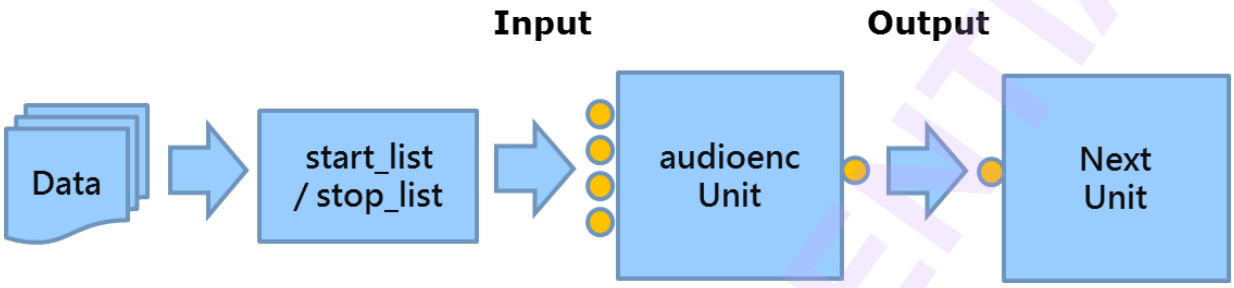
Value	Description
HD_OK	Success
HD_ERR_NG	Failure

[Difference]

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

2.2 Multi List Operation

Multi list operation is used to send mulit bitstream simultaneously, it is very efficidecy in the multi channels case. The flow is shown as below:



2.2.1 hd_audiodec_start_list

[Description]

Execute multi-start in one command

[Syntax]

HD_RESULT hd_audiodec_start_list(HD_PATH_ID *path_id, UINT num);

[Parameter]

Value	Description
path_id	the path id
num	path number

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

[Difference]

Chip	Description
IPC	Not supported.
NVR	All functions are supported.

2.2.2 hd_audiodec_stop_list

[Description]
Execute multi-stop in one command

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

[Parameter]

Value	Description
path_id	the path id
num	path number

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

[Difference]

Chip	Description
IPC	Not supported.
NVR	All functions are supported.

2.2.3 hd_audiodec_send_list

[Description]
Send multi-channel bitstreams for audiodec module

[Syntax]
HD_RESULT hd_audiodec_send_list(HD_AUDIODEC_SEND_LIST *p_audiodec_bs, UINT32 num, INT32 wait_ms);

[Parameter]

Value	Description
p_audiodec_bs	An array of bitstream structure filled by

	user for multi channels
num	The number of channels to send bitstream
wait_ms	The timeout value in millisecond while sending

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

[Difference]

Chip	Description
IPC	Not supported.
NVR	All functions are supported.

2.3 Data structure definition

The audiodec provides the following parameter IDs:

- HD_AUDIODEC_PARAM_DEVCOUNT
  - NVR/IPC. support get with ctrl path
  - using HD_DEVCOUNT struct (device id max count)
- HD_AUDIODEC_PARAM_SYSCAPS
  - NVR/IPC. support get with ctrl path
  - using HD_AUDIODEC_SYSCAPS
- HD_AUDIODEC_PARAM_PATH_CONFIG
  - IPC only. support get/set with i/o path
  - using HD_AUDIODEC_PATH_CONFIG
- HD_AUDIODEC_PARAM_IN
  - NVR/IPC. support get/set with i/o path
  - using HD_AUDIODEC_IN struct (input bitstream parameter)

2.3.1 HD_AUDIODEC_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_AUDIODEC_DEVCAPS
in_caps	capability of input, combine caps of HD_AUDIO_CAPS and

	HD_AUDIODEC_INCAPS

2.3.2 HD_AUDIODEC_PATH_CONFIG

[Description]
Path configuration

[Parameter]

Value	Description
max_mem	maximum memory information see HD_AUDIODEC_MAXMEM

2.3.3 HD_AUDIODEC_MAXMEM

[Description]
Maximum memory information

[Parameter]

Value	Description
sample_rate	sample rate
sample_bit	sample bit
mode	sound mode

[Difference]

Chip	Description
IPC	Supported
NVR	Not supported

2.3.4 HD_AUDIODEC_IN

[Description]
Input bitstream parameter

[Parameter]

Value	Description

sample_rate	sample rate
sample_bit	sample bit
mode	sound mode

2.3.5 HD_AUDIODEC_USER_BS

[Description]

Audio bitstream data and relative information

[Parameter]

Value	Description
sign	signature = MAKEFOURCC('A','S','T','M')
p_next	pointer to next meta
acodec_format	encoded format of audio frame
timestamp	encoded timestamp
p_user_buf	Bitstream buffer pointer
user_buf_size	size of encoded data

[Difference]

Chip	Description
IPC	Not supported.
NVR	Supported.

2.3.6 HD_AUDIODEC_SEND_LIST

[Description]

The audio bitstream item including path information

Use this type to form an array in hd_audiodec_send_list() to send the audio bitstreams for all paths.

[Parameter]

Value	Description
path_id	path id
user_bs	audio decode user bitstream
retval	The return value. If less than 0: send bistream

	fail

[Difference]

Chip	Description
IPC	Not supported.
NVR	Supported.

3 Trouble shooting

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

3.1 Debug menu for IPC

In application, call hd_debug_run_menu() to open the debug menu.

```
==============================
 HDAL
------------------------------
 01 : AUDIOCAPTURE
 02 : AUDIOOUT
 03 : AUDIOENC
 04 : AUDIODEC
 05 : VIDEOCAPTURE
 06 : VIDEOOUT
 07 : VIDEOPROCESS
 08 : VIDEOENC
 09 : VIDEODEC
 10 : OSG
 11 : COMMON
 12 : UTIL
 13 : DEBUG
------------------------------
 254 : Quit
 255 : Return
------------------------------
```

Enter “4” to open AUDIODEC debug menu

```
==============================
 AUDIODEC
------------------------------
```

```
01 : dump status
------------------------------
254 : Quit
255 : Return
------------------------------
```

Note: The items in the menu may vary for IPC or NVR/DVR.

3.1.1 dump status

Enter “1” to show the status of audiodec

```
Run: 01 : dump status
HDAL_VERSION: 00010001:00010001


-------------------------AUDIODEC 0  PATH & BIND -------------------------------
in    out   state   bind_src           bind_dest
0     0     START   (null)              AUDIOOUT_0_IN_0
-------------------------AUDIODEC 0  PATH CONFIG -------------------------------
in    out   sr    ch    bit   codec
0     0     48000 2     16    AAC
-------------------------AUDIODEC 0  IN FRAME ----------------------------------
in    sr    ch    bit   codec
0     48000 2     16    AAC
------------------------ AUDIODEC 0  IN WORK STATUS -------------------------
in   PUSH drop wrn  err  PROC drop wrn  err  REL
0    48   0    0    0    48   0    0    0    48
------------------------ AUDIODEC 0  OUT WORK STATUS ------------------------
out  NEW  drop wrn  err  PROC drop wrn  err  PUSH drop wrn  err
0    48   0    0    0    48   0    0    0    48   0    0    0
------------------------ AUDIODEC 0  USER WORK STATUS -----------------------
out  PULL  drop wrn  err  REL
0    0     0    0    0    0
```

As above, the debug menu shows the path & bind information, path_config , input frame / output bitstream information, more detail can see the table as below.

[PATH & BIND]

Status	Description	Value
in	input id of path	0 ~ [max_in_count]
out	output id of path	0 ~ [max_out_count]
state	state of path	OFF/OPEN/START (default OFF)
bind_src	current binding source of input	bind: [module]_[device_id]_OUT_[output_id] not-bind: (null)
bind_dest	current binding source of output	bind: [module]_[device_id]_IN_[input_id] not-bind: (null)

[PATH CONFIG]

Value	Description	Value
in	input id of path	0 ~ [max_in_count]
out	output id of path	0 ~ [max_out_count]
sr	maximum sample rate	enum: user assign sample rate see HD_AUDIO_SR default 0 (n/a)
ch	maximum channel count (sound mode)	enum: user assign channel count see HD_AUDIO_SOUND_MODE default 0 (n/a)
bit	maximum bit width	enum: user assign bit width see HD_AUDIO_BIT_WIDTH default 0 (n/a)

[IN BS]

Value	Description	Value
in	input id of path	0 ~ [max_in_count]
sr	current input sample rate	enum: user assign sample rate see HD_AUDIO_SR default 0 (n/a)
ch	current input channel count (sound mode)	enum: user assign channel count, see HD_AUDIO_SOUND_MODE default 0 (n/a)
bit	current input bit width	enum: user assign bit width

		see HD_AUDIO_BIT_WIDTH default 0 (n/a)

3.2 proc command for IPC

User can obtained debugging information from the proc file system of Linux.

3.2.1 dump status

```
[dump info]
cat /proc/hdal/adec/info
```

the result is exactly the same as 3.1.1 Dump status

```
root@NVTEVM:~$ cat /proc/hdal/adec/info
Run: 01 : dump status
HDAL_VERSION: 00010001:00010001


------------------------AUDIODEC 0  PATH & BIND -------------------------------
in    out    state   bind_src            bind_dest
0     0      START   (null)              AUDIOOUT_0_IN_0
------------------------AUDIODEC 0  PATH CONFIG -------------------------------
in    out    sr    ch    bit    codec
0     0      48000 2     16     AAC
------------------------AUDIODEC 0  IN FRAME ----------------------------------
in    sr     ch    bit    codec
0     48000  2     16     AAC
------------------------ AUDIODEC 0  IN WORK STATUS -------------------------
in   PUSH  drop  wrn   err   PROC  drop  wrn   err   REL
0    48    0     0     0     48    0     0     0     48
------------------------ AUDIODEC 0  OUT WORK STATUS ------------------------
out   NEW   drop  wrn   err   PROC  drop  wrn   err   PUSH  drop  wrn   err
0     48    0     0     0     48    0     0     0     48    0     0     0
------------------------ AUDIODEC 0  USER WORK STATUS -----------------------
```

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

3.2.2 debug command

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

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

this debug command can show more debug log on console

```
root@NVTEVM:~$ hd_audio_playback
[ 1117.361571] hd_reset - begin
[ 1117.366560] hd_reset - end
HDAL_VERSION: 00010001:00010001
[ 1117.370797]
[ 1117.370797]  "auddec".out[0]: open begin, state=0
[ 1117.378518]  "auddec".out[0]: cmd OPEN

[ 1117.384604]  "auddec".out[0]: open end, state=1
[ 1117.390208]  "auddec".out[0]: set param(08000a04)=2
[ 1117.395987]  "auddec".out[0]: set param(08000a00)=2
[ 1117.401741]  "auddec".out[0]: set param(08000a00)=2
[ 1117.407497]  "auddec".out[0]: set param(08000a01)=48000
[ 1117.413591]  "auddec".out[0]: set param(08000a02)=2
[ 1117.419337]  "auddec".out[0]: set param(08000a03)=16
[ 1117.425278]
[ 1117.425278]  "auddec".out[0]: bind begin, ("audout".in[0])
[ 1117.433197]  "auddec".out[0]: cmd CONNECT
[ 1117.438082]  "auddec".out[0]: cmd RDYSYNC
[ 1117.442960]  "auddec".out[0]: bind end
[ 1117.447583]
[ 1117.447583]  "auddec".out[0]: start begin, state=1
[ 1117.454808]  "auddec".out[0]: cmd RDYSYNC
```

```
[ 1117.459684]  "auddec".out[0]: cmd START
[ 1117.464707]  "auddec".out[0]: start end, state=2
[ 1117.470207]  "auddec".out[0]: get param(08000a07)=311508992
[ 1117.476648]  "auddec".out[0]: get param(08000a08)=192000
Enter q to exit, Enter d to debug
play file: [/mnt/sd/audio_bs_16_2_48000_aac.dat]
len file: [/mnt/sd/audio_bs_16_2_48000_aac.len]
alloc bs_buf: start(0x75300000) curr(0x75300000) end(0x75400000) size(0x100000)


q
[ 1123.751397]
[ 1123.751397]  "auddec".out[0]: stop begin, state=2
[ 1123.758584]  "auddec".out[0]: cmd STOP
[ 1123.763220]  "auddec".out[0]: stop end, state=1
[ 1123.768812]
[ 1123.768812]  "auddec".out[0]: unbind begin, ("audout".in[0])
[ 1123.776909]  "auddec".out[0]: cmd DISCONNECT
[ 1123.782046]  "auddec".out[0]: unbind end
[ 1123.786851]  "auddec".out[0]: set param(08000a04)=0
[ 1123.792613]
[ 1123.792613]  "auddec".out[0]: close begin, state=1
[ 1123.799843]  "auddec".out[0]: cmd CLOSE
[ 1123.804597]  "auddec".out[0]: close end, state=0
```

3.2.3 trace command

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


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

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

3.2.4 probe command

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

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

this probe command could print per-data status

```
[   94.719069] "auddec".out[0] - NEW - new -- h=00000001 size=00000000 addr=00000001 OK
[   94.730238] "auddec".out[0] - PUSH - rel -- h=00000004 (result=0) OK
[   94.737604] "auddec".out[0] - PUSH - rel -- h=00000002 (result=0) OK
[   94.739243] "auddec".out[0] - NEW - new -- h=00000003 size=00000000 addr=00000003 OK
[   94.753667] "auddec".out[0] - PUSH - rel -- h=00000001 (result=0) OK
[   94.761028] "auddec".out[0] - PUSH - rel -- h=00000003 (result=0) OK
[   94.769488] "auddec".out[0] - NEW - new -- h=00000001 size=00000000 addr=00000001 OK
[   94.778300] "auddec".out[0] - PUSH - rel -- h=00000001 (result=0) OK
[   94.789666] "auddec".out[0] - NEW - new -- h=00000001 size=00000000 addr=00000001 OK
[   94.798479] "auddec".out[0] - PUSH - rel -- h=00000001 (result=0) OK
[   94.809853] "auddec".out[0] - NEW - new -- h=00000001 size=00000000 addr=00000001 OK
[   94.818652] "auddec".out[0] - PUSH - rel -- h=00000001 (result=0) OK
[   94.830012] "auddec".out[0] - NEW - new -- h=00000001 size=00000000 addr=00000001 OK
[   94.838797] "auddec".out[0] - PUSH - rel -- h=00000001 (result=0) OK
[   94.850181] "auddec".out[0] - NEW - new -- h=00000001 size=00000000 addr=00000001 OK
[   94.859056] "auddec".out[0] - PUSH - rel -- h=00000001 (result=0) OK
[   94.870372] "auddec".out[0] - NEW - new -- h=00000001 size=00000000 addr=00000001 OK
[   94.879169] "auddec".out[0] - PUSH - rel -- h=00000001 (result=0) OK
```

3.2.5 perf command

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

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

this perf command could print data count per second

```
[  122.560530] "auddec".in[0] Perf! -- (AudioBs) 20 KByte/sec
[  123.565613] "auddec".in[0] Perf! -- (AudioBs) 20 KByte/sec
[  124.570037] "auddec".in[0] Perf! -- (AudioBs) 20 KByte/sec
```

3.2.6 save command

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

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

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

```
[  175.313165] save i/o begin: "auddec".in[0] count=1
[  178.149785] "auddec".in[0] Save -- h=00000000 t=000000000bf51ec0 (ASTM: 94002000 433)
_2_48000_aac.dat]
[  178.166955] "auddec".in[0] Save -- //mnt//sd//isf_  auddec_in[0]_c0.bsa ok

[  178.175872] save port end
```

3.3 proc command for NVR

3.3.1 Dump info

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

The result will show the audiodec information.

```
root@NVTEVM:/$ cat /proc/videograph/hdal_setting
------------------------ AUDIODEC -4096 IN -----------------------------
out     rate    bit     samples
0       8000    16      MONO
```

3.4 Debug menu for NVR

Calling hd_debug_run_menu() from app will pop out debug_menu.

The currently supported audiodec module debug menu is as below.

```
============================
 AUDIODEC
----------------------------
 01 : dump status
----------------------------
 254 : Quit
 255 : Return
----------------------------
1
Run: 01 : dump status
------------------------ AUDIODEC -4096 IN ---------------------------
out     rate    bit     samples
0       8000    16      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_playback (IPC)

This sample code shows that audiodec binding audioout, user push bitstream data to audiodec, which can decode bitstream to PCM and push to audioout to play the sound by speaker.

```
/* Set dec path configuration */
audio_path_cfg.max_mem.codec_type = dec_type;
audio_path_cfg.max_mem.sample_rate = HD_AUDIO_SR_48000;
audio_path_cfg.max_mem.sample_bit = HD_AUDIO_BIT_WIDTH_16;
audio_path_cfg.max_mem.mode = HD_AUDIO_SOUND_MODE_STEREO;
ret = hd_audiodec_set(audio_dec_path, HD_AUDIODEC_PARAM_PATH_CONFIG, &audio_path_cfg);
if (ret != HD_OK) { return ret; }

/* Set dec parameter */
audio_in_param.codec_type = dec_type;
audio_in_param.sample_rate = HD_AUDIO_SR_48000;
audio_in_param.sample_bit = HD_AUDIO_BIT_WIDTH_16;
audio_in_param.mode = HD_AUDIO_SOUND_MODE_STEREO;
ret = hd_audiodec_set(audio_dec_path, HD_AUDIODEC_PARAM_IN, &audio_in_param);
if (ret != HD_OK) { return ret; }

/* 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 */
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; }

/* Bind audio_playback module */
hd_audiodec_bind(HD_AUDIODEC_0_OUT_0, HD_AUDIOOUT_0_IN_0);

/* Push in buffer */
blk = hd_common_mem_get_block(HD_COMMON_MEM_COMMON_POOL, blk_size, ddr_id);
pa = hd_common_mem_blk2pa(blk);
va= hd_common_mem_mmap(HD_COMMON_MEM_MEM_TYPE_CACHE, pa, blk_size);
fread((void *)va, 1, length, bs_fd);
audio_bs.phy_addr[0] = pa;
audio_bs.size = length;
ret = hd_audiodec_push_in_buf(p_stream0->dec_path, &audio_bs, NULL, 0);
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 audio_decode_only (IPC)

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

```
/* Set dec path configuration */
audio_path_cfg.max_mem.codec_type = dec_type;
audio_path_cfg.max_mem.sample_rate = HD_AUDIO_SR_48000;
audio_path_cfg.max_mem.sample_bit = HD_AUDIO_BIT_WIDTH_16;
audio_path_cfg.max_mem.mode = HD_AUDIO_SOUND_MODE_STEREO;
ret = hd_audiodec_set(audio_dec_path, HD_AUDIODEC_PARAM_PATH_CONFIG, &audio_path_cfg);
if (ret != HD_OK) { return ret; }

/* Set dec parameter */
audio_in_param.codec_type = dec_type;
audio_in_param.sample_rate = HD_AUDIO_SR_48000;
audio_in_param.sample_bit = HD_AUDIO_BIT_WIDTH_16;
audio_in_param.mode = HD_AUDIO_SOUND_MODE_STEREO;
ret = hd_audiodec_set(audio_dec_path, HD_AUDIODEC_PARAM_IN, &audio_in_param);
if (ret != HD_OK) { return ret; }

/* Push in buffer */
blk = hd_common_mem_get_block(HD_COMMON_MEM_COMMON_POOL, blk_size, ddr_id);
pa = hd_common_mem_blk2pa(blk);
va= hd_common_mem_mmap(HD_COMMON_MEM_MEM_TYPE_CACHE, pa, blk_size);
fread((void *)va, 1, length, bs_fd);
audio_bs.phy_addr[0] = pa;
audio_bs.size = length;
ret = hd_audiodec_push_in_buf(p_stream0->dec_path, &audio_bs, NULL, 0);
if (ret != HD_OK) { return ret; }

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

/* Pull out buffer */
```

```
ret = hd_audiodec_pull_out_buf(p_stream0->dec_path, &data_pull, -1);
if (ret == HD_OK) {
     hd_audiodec_get(p_stream0->dec_path, HD_AUDIODEC_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);
     #define PHY2VIRT_MAIN(pa) (vir_addr_main + (pa - phy_buf_main.buf_info.phy_addr))
     va = PHY2VIRT_MAIN(data_pull.phy_addr);
     size = data_pull.size;
     sprintf(filename, "dump_frm_main.dat");
     save_output(filename, va, size);
}

/* Release out buffer */
hd_common_mem_munmap(vir_addr_main, phy_buf_main.buf_info.buf_size);
hd_audiodec_release_out_buf(p_stream0->dec_path, &data_pull);
```

4.3 audio_playback(NVR)

audiodec doesn't support push/pull operation. User must bind it with audioout to play the audio data out. The following demonstrates how to send PCM data and play it 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"
#include "hd_debug.h"

#define MAX_BITSTREAM_NUM   1
#define BITSTREAM_LEN      12800
#define PATTERN_NAME       "audio_8khz_16bit.pcm"
#define PATTERN_LEN_NAME    "audio_8khz_16bit.pcmlen"
#define BUFFER_TIME_MS     500


typedef struct _AUDIO_PLAYBACK {
     HD_PATH_ID auddec_path_id;
```

```
    HD_PATH_ID audout_path_id;
    UINT32 pb_exit;
    int out_dev;        //0:Front-End DAC  1:ADDA301  2:HDMI
} AUDIO_PLAYBACK;


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

HD_RESULT open_module(AUDIO_PLAYBACK *p_pb_info)
{
    HD_RESULT ret;
    if((ret = hd_audiodec_open(HD_AUDIODEC_IN(0, 0), HD_AUDIODEC_OUT(0, 0),
&p_pb_info->auddec_path_id)) != HD_OK)
      return ret;
    if((ret = hd_audioout_open(HD_AUDIOOUT_IN(p_pb_info->out_dev, 0),
HD_AUDIOOUT_OUT(p_pb_info->out_dev, 0), &p_pb_info->audout_path_id)) != HD_OK)
      return ret;
    return HD_OK;
}

HD_RESULT close_module(AUDIO_PLAYBACK *p_pb_info)
{
    HD_RESULT ret;
    if((ret = hd_audiodec_close(p_pb_info->auddec_path_id)) != HD_OK)
      return ret;
    if((ret = hd_audioout_close(p_pb_info->audout_path_id)) != HD_OK)
      return ret;
    return HD_OK;
}

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


#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;
}


static void *playback_thread(void *arg)
{
   AUDIO_PLAYBACK *p_pb_info = (AUDIO_PLAYBACK *) arg;
    INT ret, length, elapse_time, au_buf_time;
    CHAR filename[50];
    FILE *bs_fd, *len_fd;
    CHAR *bitstream_data;
    HD_AUDIODEC_SEND_LIST send_list[1];
    UINT start_time, data_time;

    //open files for reading
    sprintf(filename, "%s", PATTERN_NAME);
    bs_fd = fopen(filename, "rb");
    if (bs_fd == NULL) {
        printf("[ERROR] Open %s failed!!\n", filename);
        exit(1);
    }
    printf("Play file: [%s]\n", filename);
    sprintf(filename, "%s", PATTERN_LEN_NAME);
    len_fd = fopen(filename, "rb");
    if (len_fd == NULL) {
        printf("[ERROR] Open %s failed!!\n", filename);
        exit(1);
    }

    //prepare buffer for sending data
    bitstream_data = (char *)malloc(BITSTREAM_LEN);
    if (!bitstream_data) {
        printf("Error allocation\n");
        exit(1);
    }

    start_time = get_current_time();
    data_time = 0;
    while (1) {
        if (p_pb_info->pb_exit == 1) {
            break;
        }
retry:
        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;
        }

        //read audio data
        if (fscanf(len_fd, "%d\n", &length) == EOF) {
            fseek(bs_fd, 0, SEEK_SET);
            fseek(len_fd, 0, SEEK_SET);
            fscanf(len_fd, "%d\n", &length);
        }
        if (length == 0) {
```

```
                continue;
            }
            if (length > BITSTREAM_LEN) {
                printf("Invalid length, len(%d)\n", length);
                exit(1);
            }
            fread(bitstream_data, 1, length, bs_fd);

            //fill the structure and send it to audiodec
            memset(send_list, 0, sizeof(send_list));  //clear all mutli bs
            send_list[0].path_id = p_pb_info->auddec_path_id;
            send_list[0].user_bs.p_user_buf = bitstream_data;
            send_list[0].user_bs.user_buf_size = length;
            if ((ret = hd_audiodec_send_list(send_list, 1, 500)) < 0) {
                printf("<send bitstream fail(%d)!>\n", ret);
            }
        }

        //close files and free buffer
        fclose(bs_fd);
        fclose(len_fd);
        free(bitstream_data);
        return 0;
}

HD_RESULT set_param(AUDIO_PLAYBACK *p_pb_info)
{
        HD_AUDIODEC_IN audiodec_param;
        HD_AUDIOOUT_OUT audioout_param;
        HD_RESULT ret;

        //set audiodec parameters
        ret = hd_audiodec_get(p_pb_info->auddec_path_id, HD_AUDIODEC_PARAM_IN,
&audiodec_param);
        if (ret != HD_OK) {
            printf("hd_audiodec_get(HD_AUDIODEC_PARAM_IN) fail\n");
            goto exit;
        }
        audiodec_param.codec_type = HD_AUDIO_CODEC_PCM;
        audiodec_param.sample_rate = HD_AUDIO_SR_8000;
        audiodec_param.sample_bit = HD_AUDIO_BIT_WIDTH_16;
        audiodec_param.mode = HD_AUDIO_SOUND_MODE_MONO;
        ret = hd_audiodec_set(p_pb_info->auddec_path_id, HD_AUDIODEC_PARAM_IN,
&audiodec_param);
        if (ret != HD_OK) {
            printf("hd_audiodec_set(HD_AUDIODEC_PARAM_IN) fail\n");
            goto exit;
        }

        //set audioout parameters
        ret = hd_audioout_get(p_pb_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_pb_info->audout_path_id, HD_AUDIOOUT_PARAM_OUT,
&audioout_param);
    if (ret != HD_OK) {
        printf("hd_audiodec_set(HD_AUDIOOUT_PARAM_OUT) fail\n");
        goto exit;
    }
exit:
    return ret;
}

int main(int argc, char** argv)
{
    HD_RESULT ret;
    INT key, value;
    pthread_t thread_id;
    AUDIO_PLAYBACK pb_info = {0};

    if (argc == 2) {
    value = atoi(argv[1]);
        if (value >= 0 && value <= 2) {
            printf(" Output to devid(%d)\n", value);
        } else {
            printf("Wrong argument, out_dev range(0~2)\n");
            exit(0);
        }
    } else {
    value = 0;
    }

    //set default out-dev port
    pb_info.out_dev = value;

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

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

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

    //setup runtime parameters
    ret = set_param(&pb_info);
```

```
if(ret != HD_OK) {
    printf("set param fail\n");
    goto exit;
}

//bind audio playback: audiodec -> audioout
ret = hd_audiodec_bind(HD_AUDIODEC_OUT(0, 0), HD_AUDIOOUT_IN(pb_info.out_dev, 0));
if(ret != HD_OK) {
    printf("bind fail\n");
    goto exit;
}

  //start to run
ret = hd_audiodec_start(pb_info.auddec_path_id);
if(ret != HD_OK) {
    printf("start audiodec fail\n");
    goto exit;
}
ret = hd_audioout_start(pb_info.audout_path_id);
if(ret != HD_OK) {
    printf("start audioout fail\n");
    goto exit;
}

//create a thread to put audio data
  ret = pthread_create(&thread_id, NULL, playback_thread, (void *)&pb_info);
  if (ret < 0) {
       printf("create playback thread failed");
       return -1;
  }

  //main waiting loop
  printf("Enter q to exit\n");
  while (1) {
       key = getchar();
       if (key == 'q') {
            pb_info.pb_exit = 1;
            break;
       } else if (key == 'd') {
            hd_debug_run_menu();
       }
  }

  //wait thread finish
  pthread_join(thread_id, NULL);

  //stop modules and unbind the connection
  ret = hd_audiodec_stop(pb_info.auddec_path_id);
  if(ret != HD_OK) {
       printf("stop audiodec fail\n");
  }
  ret = hd_audioout_stop(pb_info.audout_path_id);
  if(ret != HD_OK) {
       printf("stop audioout fail\n");
  }
  ret = hd_audiodec_unbind(HD_AUDIODEC_OUT(0, 0));
  if(ret != HD_OK) {
```

```
            printf("unbind fail\n");
        }
exit:
        //close and uninit modules
        ret = close_module(&pb_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");
         goto exit;
      }
        return 0
}
```