Novatek HDAL Design Specification - hd_videodec

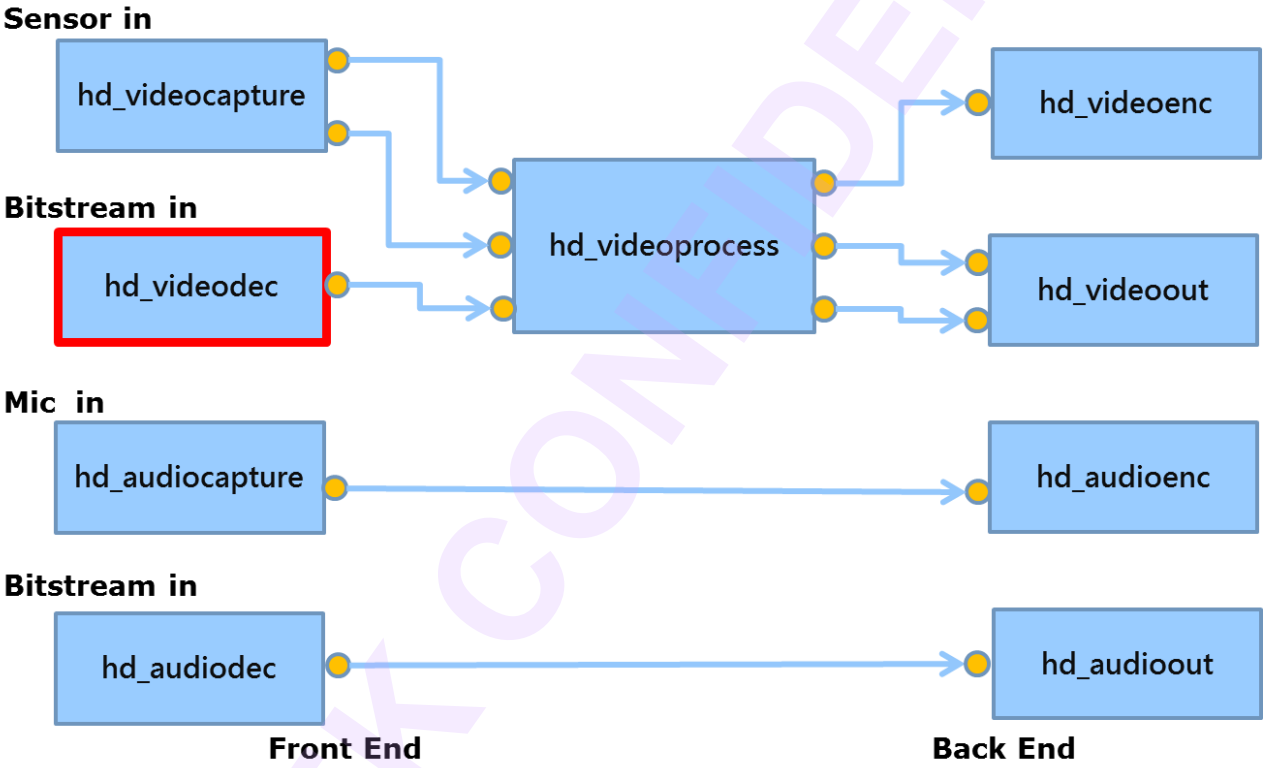
Table of Content

Novatek HDAL Design Specification - hd_videodec 1
1 Introduction 4
1.1 Block Diagram 6
1.1.1 IPC Block Diagram (videodec) 6
1.1.2 NVR Block Diagram (videodec) 7
1.2 Basic Flow 8
1.3 Single Trigger Operation 9
1.4 Multi List Operation 10
2 Functions Definition 11
2.1 hd_videodec_init 11
2.2 hd_videodec_open 11
2.3 hd_videodec_get 12
2.4 hd_videodec_set 13
2.5 hd_videodec_bind 13
2.6 hd_videodec_start 14
2.7 hd_videodec_stop 14
2.8 hd_videodec_unbind 15
2.9 hd_videodec_close 16
2.10 hd_videodec_uninit 16
2.11 hd_videodec_push_in_buf 17
2.12 hd_videodec_pull_out_buf 17
2.13 hd_videodec_release_out_buf 18
2.14 hd_videodec_start_list 19
2.15 hd_videodec_stop_list 19
3 Parameter IDs and Data Structures 21
3.1 Parameter IDs 21
3.2 Data structure definition 22
3.2.1 HD_VIDEODEC_SYSCAPS 22
3.2.2 HD_VIDEODEC_SYSINFO 22
3.2.3 HD_VIDEODEC_PATH_CONFIG 23
3.2.4 HD_VIDEODEC_MAXMEM 23
4 Trouble shooting 24
4.1 Debug Menu of IPC 24
4.1.1 dump status 25

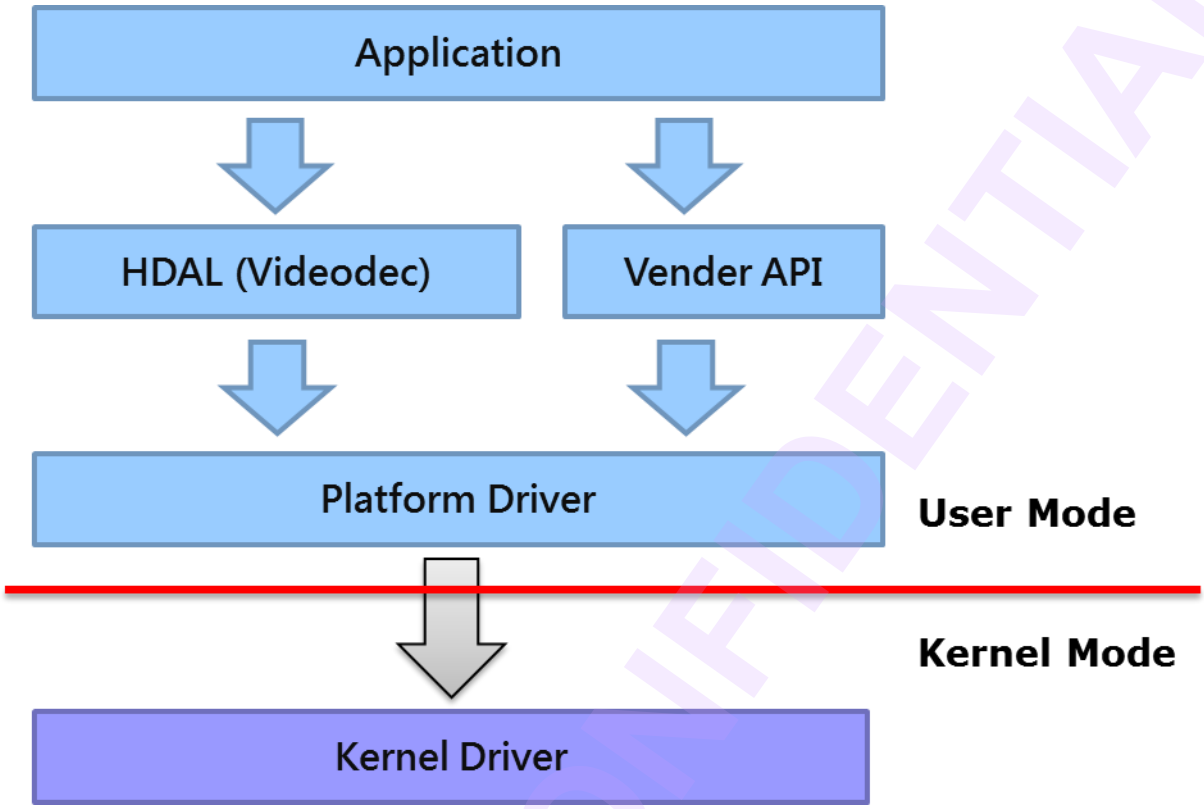
4.2 proc command for IPC 26
4.2.1 dump status 26
4.2.2 debug command 27
4.2.3 trace command 28
4.2.4 probe command 29
4.2.5 perf command 30
4.2.6 save command 30
4.3 Proc Command of NVR 31
4.3.1 dump setting 31
4.4 Debug Menu for NVR 32
4.4.1 dump info 32
5 Sample Codes 33
5.1 Video_playback (IPC) 33
5.2 video_decode_only (IPC) 36
5.3 user_videodec (NVR) 38
6 Q & A 41

1 Introduction

The major purpose of hd_videodec is to get bitstream data from upper unit, and controls the video decoder to decode the bitstream data then return the YUV frame data which can be used for displaying. This document will talk about the red block in the following diagram. The device driver is not the main point in this document.



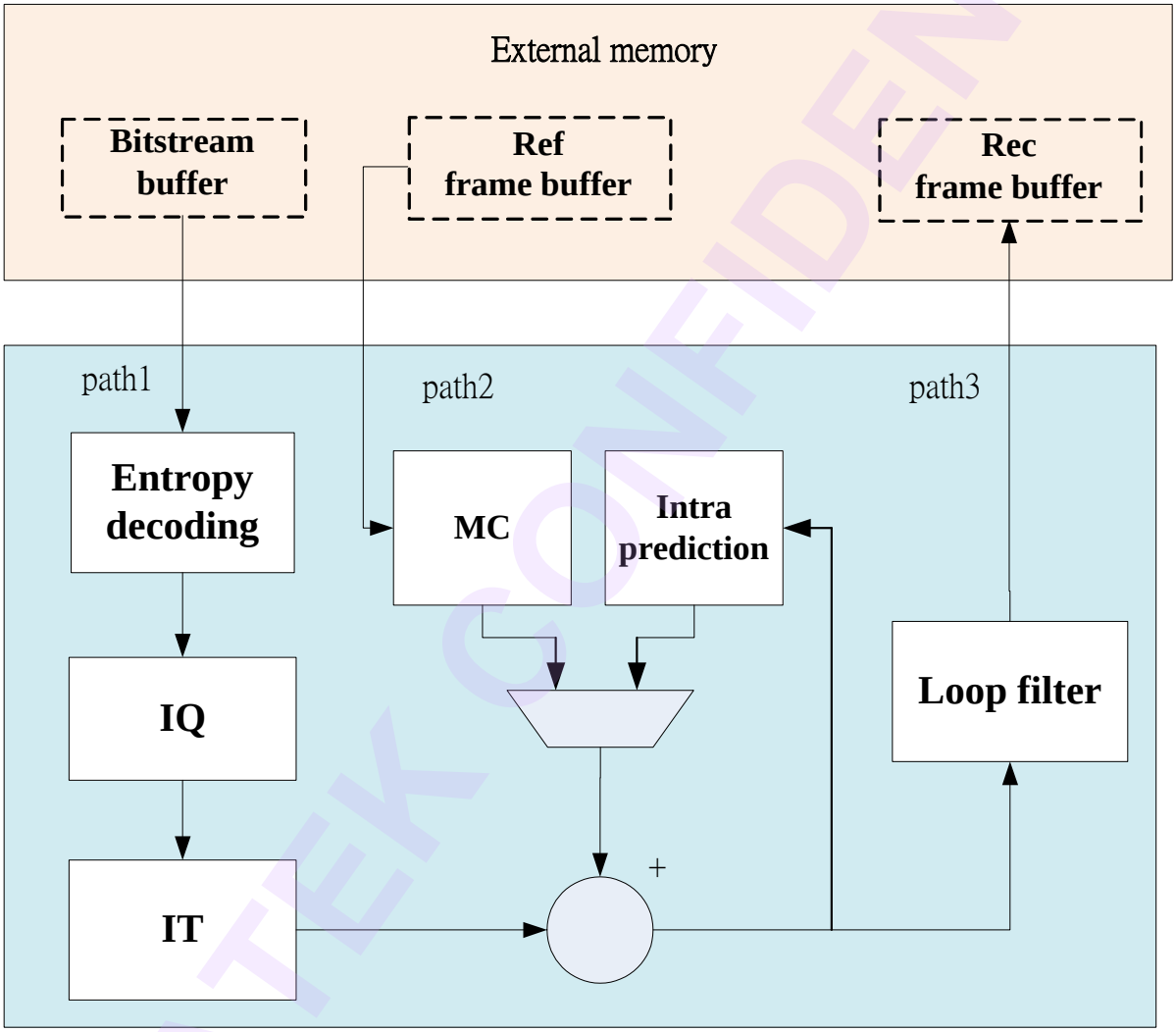
Module diagram is shown as below:



1.1 Block Diagram

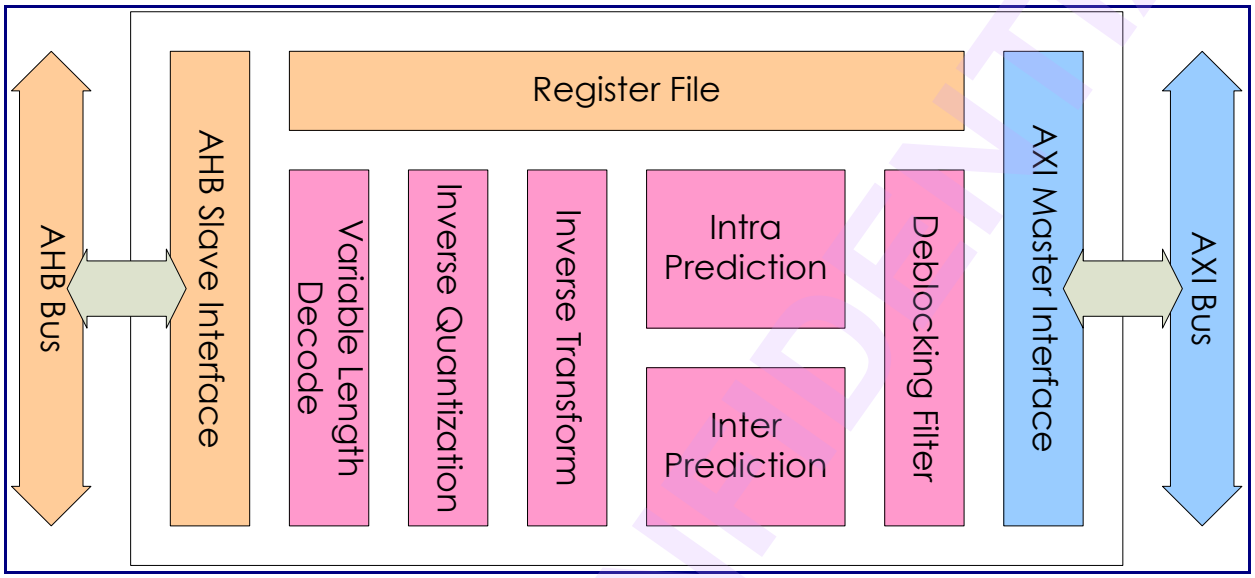
1.1.1 IPC Block Diagram (videodec)

The block diagram of H264/H265 codec is shown as below:

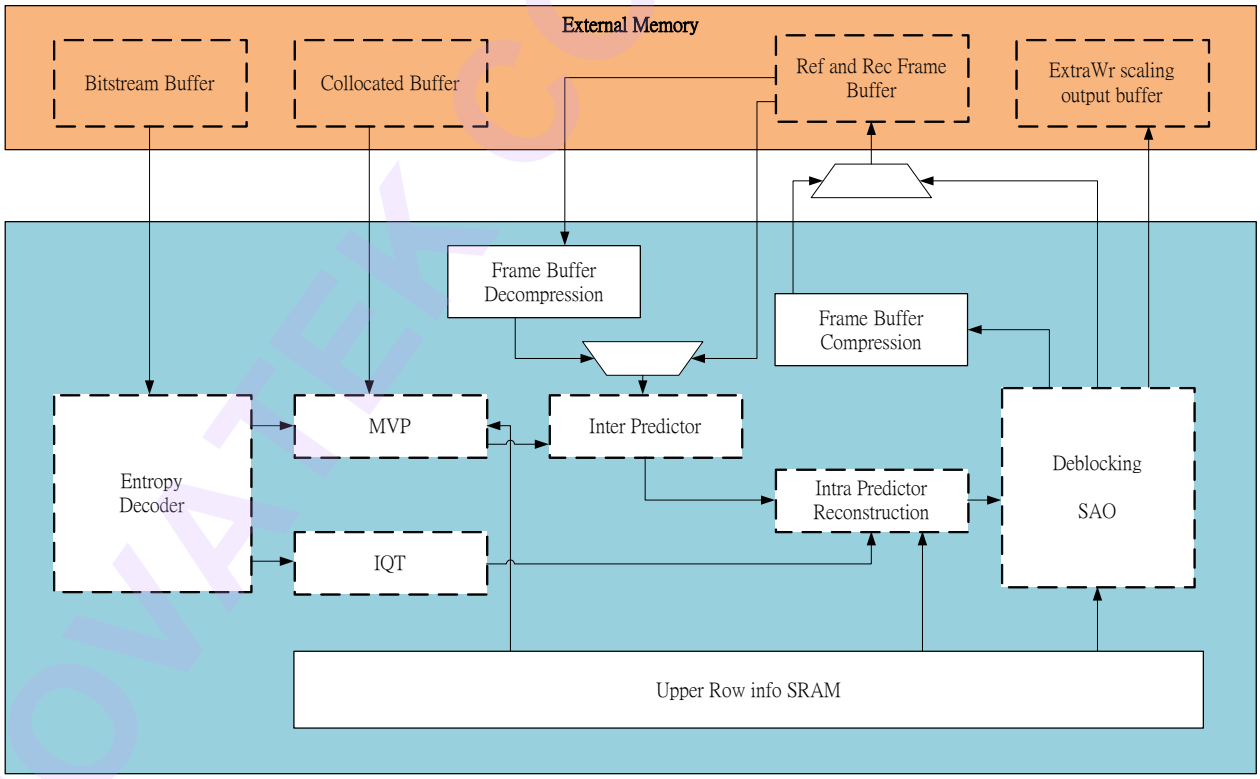


1.1.2 NVR Block Diagram (videodec)

The block diagram of H264 codec is shown as below:

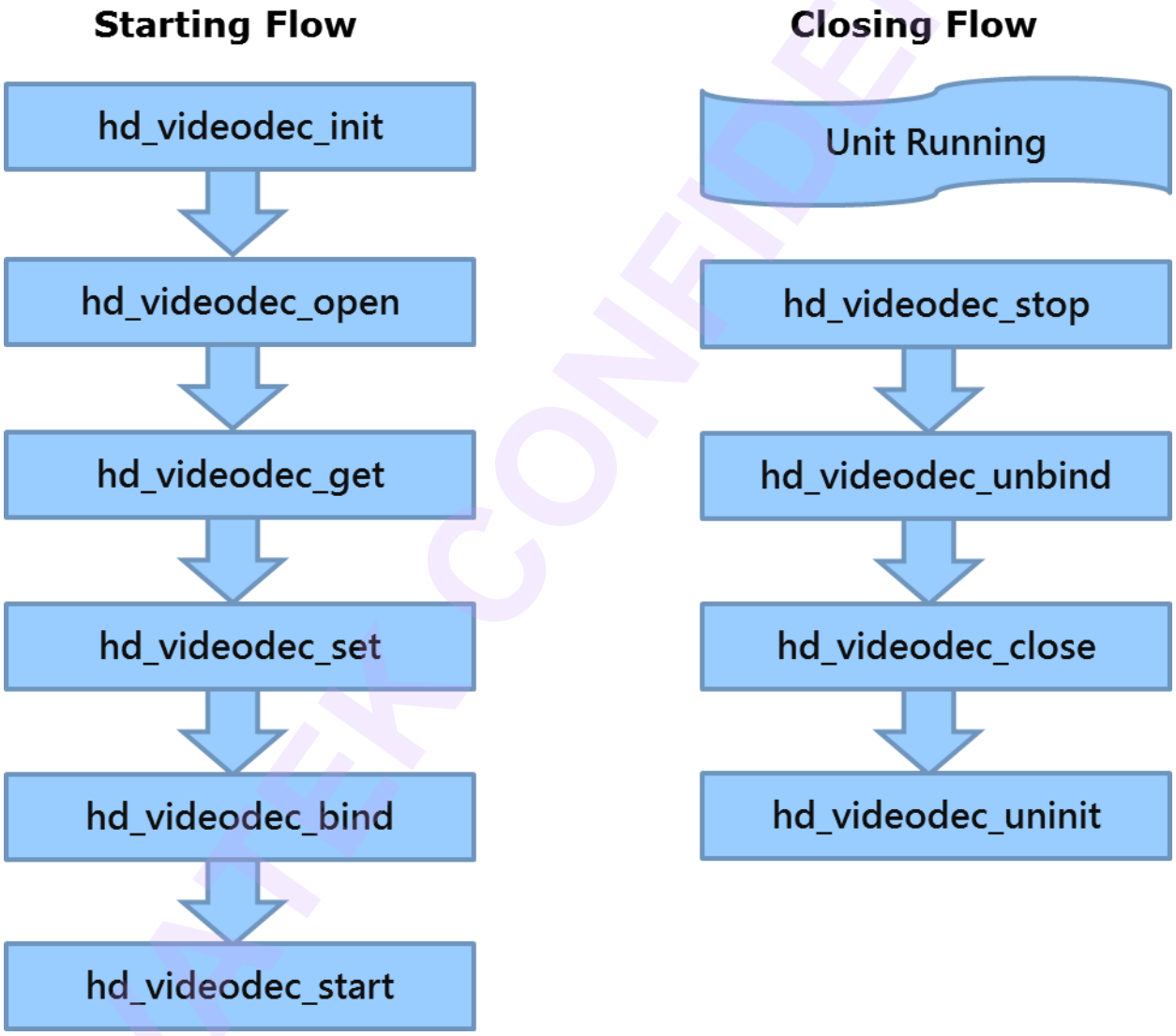


The block diagram of H265 codec is shown as below:



1.2 Basic Flow

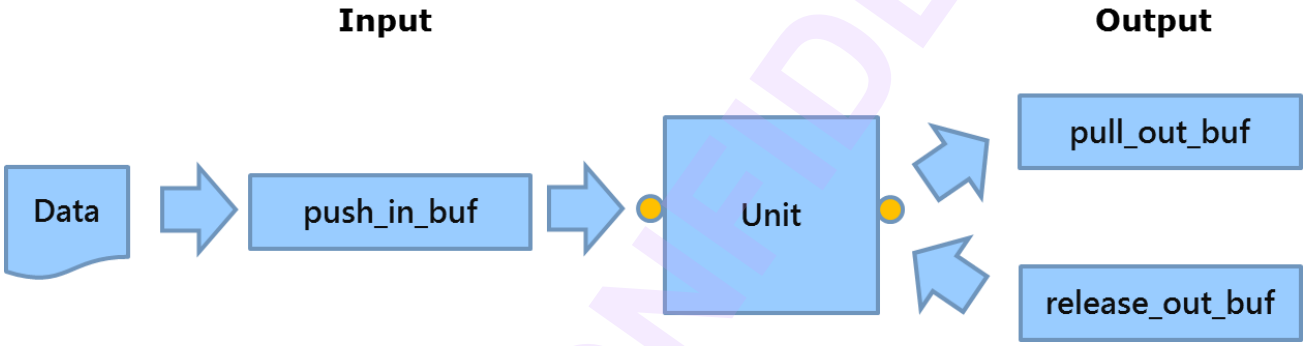
The call sequence is needed to be done correctly for the unit. The standard starting flows of most modules are init, open, get, set and start. The standard closing flows of most modules are stop, unbind, close and uninit. The basic flow is shown as below.



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

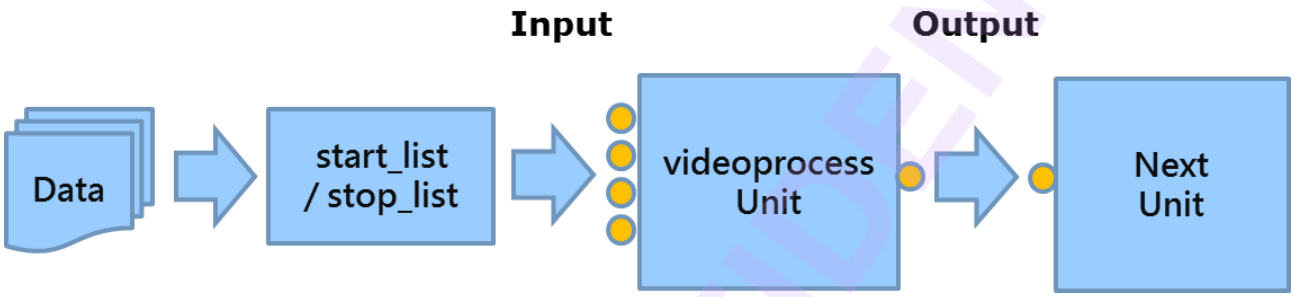
1.3 Single Trigger Operation

Single trigger operation is used to trigger the unit to do one job, such as to grab one YUV frame from video capture; or encode one frame to bitstream by using video encoder. 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.



1.4 Multi List Operation

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



2 Functions Definition

2.1 hd_videodec_init

[Description]
Initialize the unit

[Syntax]
HD_RESULT hd_videodec_init(VOID);

[Parameter]

Value	Description
VOID	Not available

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.2 hd_videodec_open

[Description]
Open the unit

[Syntax]
HD_RESULT hd_videodec_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

2.3 hd_videodec_get

[Description]
Get parameters from unit by path id

[Syntax]
HD_RESULT hd_videodec_get(HD_PATH_ID path_id, HD_VIDEODEC_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

2.4 hd_videodec_set

[Description]
Set parameters to unit by path id

[Syntax]
HD_RESULT hd_videodec_set(HD_PATH_ID path_id, HD_VIDEODEC_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

2.5 hd_videodec_bind

[Description]
Bind this unit with destination unit

[Syntax]
HD_RESULT hd_videodec_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

2.6 hd_videodec_start

[Description]
Start the unit

[Syntax]
HD_RESULT hd_videodec_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

2.7 hd_videodec_stop

[Description]
Stop the unit

[Syntax]
HD_RESULT hd_videodec_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

2.8 hd_videodec_unbind

[Description]
Unbind the unit

[Syntax]
HD_RESULT hd_videodec_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

2.9 hd_videodec_close

[Description]
Close the unit

[Syntax]
HD_RESULT hd_videodec_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

2.10 hd_videodec_uninit

[Description]
Uninitialize the unit

[Syntax]
HD_RESULT hd_videodec_uninit(VOID);

[Parameter]

Value	Description
VOID	Not available

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.11 hd_videodec_push_in_buf

[Description]
Push the video buffer to unit

[Syntax]
HD_RESULT hd_videodec_push_in_buf(HD_PATH_ID path_id, HD_VIDEO_FRAME* p_in_video_frame, HD_VIDEO_FRAME* p_user_out_video_frame, INT32 wait_ms);

[Parameter]

Value	Description
path_id	the path id
p_in_video_frame	pointer of the input video buffer
p_user_out_video_frame	pointer of the output video buffer
wait_ms	timeout value in ms

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.12 hd_videodec_pull_out_buf

[Description]
Pull the video buffer from unit

[Syntax]
HD_RESULT hd_videodec_pull_out_buf(HD_PATH_ID path_id, HD_VIDEO_FRAME* p_video_frame, INT32 wait_ms);

[Parameter]

Value	Description
path_id	the path id
p_video_frame	pointer of the output video buffer
wait_ms	timeout value in ms

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.13 hd_videodec_release_out_buf

[Description]

Release the video frame buffer which is get from unit

[Syntax]

HD_RESULT hd_videodec_release_out_buf(HD_PATH_ID path_id, HD_VIDEO_FRAME* p_video_frame)

[Parameter]

Value	Description
path_id	the path id
p_video_frame	pointer of the output video buffer

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.14 hd_videodec_start_list

[Description]
Start to send multi bitstream data to the unit

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

[Parameter]

Value	Description
path_id	the path id
num	number of bitstream data

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

[Difference]

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

2.15 hd_videodec_stop_list

[Description]
Stop to send multi bitstream data to the unit

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

[Parameter]

Value	Description
path_id	the path id
num	number of bitstream data

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

[Difference]

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

3 Parameter IDs and Data Structures

3.1 Parameter IDs

The videodec provides the following parameter IDs:

- **HD_VIDEODEC_PARAM_DEVCOUNT**
  - NVR/IPC. support get with ctrl path
  - using HD_DEVCOUNT struct (device id max count)
- **HD_VIDEODEC_PARAM_SYSCAPS**
  - NVR/IPC. support get with ctrl path
  - using HD_VIDEODEC_SYSCAPS
- **HD_VIDEODEC_PARAM_PATH_CONFIG**
  - NVR/IPC only. support get/set with i/o path
  - using HD_VIDEODEC_PATH_CONFIG

3.2 Data structure definition

3.2.1 HD_VIDEODEC_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_VIDEODEC_DEVCAPS
in_caps	capability of input, combine caps of HD_VIDEO_CAPS and HD_VIDEODEC_INCAPS
out_caps	capability of output, combine caps of HD_VIDEO_CAPS and HD_VIDEODEC_OUTCAPS
max_dim	max dimension of videodecoder
max_bitrate	max bitrate of videodecoder

3.2.2 HD_VIDEODEC_SYSINFO

[Description]
System information

[Parameter]

Value	Description

dev_id	device id
cur_in_fps	current input fps
cur_out_fps	current output fps

3.2.3 HD_VIDEODEC_PATH_CONFIG

[Description]

Device configuration

[Parameter]

Value	Description
max_mem	NVR/IPC. maximum memory information see HD_VIDEODEC_MAXMEM
data_pool	NVR only. pool memory information

3.2.4 HD_VIDEODEC_MAXMEM

[Description]

Maximum memory information

[Parameter]

Value	Description
dim	video image dimension
frame_rate	video frame rate

4 Trouble shooting

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

4.1 Debug Menu of 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 “9” to open VIDEODEC debug menu

```
==============================
 VIDEODEC
------------------------------
 01 : dump status
```

```
-----------------------------
 254 : Quit
 255 : Return
-----------------------------
```

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

4.1.1 dump status

Enter “1” to show the status of videodec

```
Run: 01 : dump status

HDAL_VERSION: 00010001:00010001

------------------------- VIDEODEC 0  PATH & BIND -----------------------------

in    out   state bind_src            bind_dest

0     0     START (null)              VIDEOPROC_0_IN_0

------------------------- VIDEODEC 0  PATH CONFIG -----------------------------

in    out   max_w max_h  codec

0     0     640   480    JPEG

------------------------- VIDEODEC 0  IN FRAME --------------------------------

in    codec

0     JPEG
```

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	bind: [module]_[device_id]_IN_[input_id]

	source of output	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]
max_w	maximum dimension width	16 ~ 65532: user assign width default 0 (n/a)
max_h	maximum dimension height	16 ~ 65532: user assign height default 0 (n/a)

[IN FRAME]

Value	Description	Value
in	input id	0 ~ [max_in_count]
codec	current input video codec type	enum: user assign codec type see HD_VIDEO_CODEC default 0 (n/a)

4.2 proc command for IPC

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

4.2.1 dump status

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

the result is exactly the same as 4.1.1 Dump status

```
Run: 01 : dump status

HDAL_VERSION: 00010001:00010001

------------------------ VIDEODEC 0  PATH & BIND -----------------------------

in    out   state bind_src            bind_dest

0     0     START (null)              VIDEOPROC_0_IN_0
```

```
------------------------- VIDEODEC 0  PATH CONFIG -----------------------------
in    out   max_w max_h  codec
0     0     640   480    JPEG
------------------------- VIDEODEC 0  IN FRAME --------------------------------
in    codec
0     JPEG
```

4.2.2 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 mffff > /proc/hdal/vdec/cmd
```

this debug command can show more debug log on console

```
root@NVTEVM:/mnt/sd$ hd_video_playback 2
[ 4183.595988] hd_reset - begin
[ 4183.601853] hd_reset - end
HDAL_VERSION: 00010001:00010002
[ 4183.657738]
[ 4183.657738] hd: "vdodec".out[0]: open begin, state=0
[ 4183.665326] hd: "vdodec".out[0]: cmd OPEN
[ 4183.678796] hd: "vdodec".out[0]: open end, state=1
[ 4183.685213] hd: "vdodec".out[0]: set param(08000a05)=1
[ 4183.708678] hd: "vdodec".out[0]: set param(08000a00)=1

[ 4183.716255] hd: "vdodec".out[0]: set param(08000a00)=1
[ 4183.722596]
[ 4183.722596] hd: "vdodec".out[0]: bind begin, ("vdoprc0".in[0])

[ 4183.732387] hd: "vdodec".out[0]: cmd CONNECT
```

```
[ 4183.737854] hd: "vdodec".out[0]: cmd RDYSYNC
[ 4183.748315] hd: "vdodec".out[0]: bind end
[ 4183.753391]
[ 4183.753391] hd: "vdodec".out[0]: start begin, state=1
[ 4183.761045] hd: "vdodec".out[0]: cmd RDYSYNC
[ 4183.771498] hd: "vdodec".out[0]: cmd START
[ 4183.782689] hd: "vdodec".out[0]: start end, state=2
Enter q to exit
bs file: [/mnt/sd/video_bs_640_480_jpeg.dat]
bslen file: [/mnt/sd/video_bs_640_480_jpeg.len]


q
[ 4188.409136]
[ 4188.409136] hd: "vdodec".out[0]: stop begin, state=2
[ 4188.416733] hd: "vdodec".out[0]: cmd STOP


[ 4188.429400] hd: "vdodec".out[0]: stop end, state=1
[ 4188.443260]
[ 4188.443260] hd: "vdodec".out[0]: unbind begin, ("vdoprc0".in[0])
[ 4188.451877] hd: "vdodec".out[0]: cmd DISCONNECT
[ 4188.457451] hd: "vdodec".out[0]: unbind end
[ 4188.462694] hd: "vdodec".out[0]: set param(08000a05)=0
[ 4188.468879]
[ 4188.468879] hd: "vdodec".out[0]: close begin, state=1
[ 4188.476533] hd: "vdodec".out[0]: cmd CLOSE
[ 4188.484832] hd: "vdodec".out[0]: close end, state=0
```

4.2.3 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 mffff > /proc/hdal/vdec/cmd
```

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

4.2.4 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/vdec/cmd
```

this probe command could print per-data status

```
[ 5310.074418] hd: "vdodec".out[0] - NEW - new -- h=94001fc0 size=00070800 addr=94002000 OK
[ 5310.085787] hd: "vdodec".out[0] - PUSH - data -- h=94001fc0 t=000000013dd1c406 (YUV:
640x480.520c0420 94002000 9404d000 640 640)
[ 5310.113972] hd: "vdodec".out[0] - NEW - new -- h=9407afc0 size=00070800 addr=9407b000 OK
[ 5310.125334] hd: "vdodec".out[0] - PUSH - data -- h=9407afc0 t=000000013dd25e8e (YUV:
640x480.520c0420 9407b000 940c6000 640 640)
[ 5310.153493] hd: "vdodec".out[0] - NEW - new -- h=940f3fc0 size=00070800 addr=940f4000 OK
[ 5310.164831] hd: "vdodec".out[0] - PUSH - data -- h=940f3fc0 t=000000013dd2f8ef (YUV:
640x480.520c0420 940f4000 9413f000 640 640)
[ 5310.192974] hd: "vdodec".out[0] - NEW - new -- h=940f3fc0 size=00070800 addr=940f4000 OK
[ 5310.204320] hd: "vdodec".out[0] - PUSH - data -- h=940f3fc0 t=000000013dd39329 (YUV:
640x480.520c0420 940f4000 9413f000 640 640)
[ 5310.232463] hd: "vdodec".out[0] - NEW - new -- h=9407afc0 size=00070800 addr=9407b000 OK
[ 5310.243803] hd: "vdodec".out[0] - PUSH - data -- h=9407afc0 t=000000013dd42d6b (YUV:
640x480.520c0420 9407b000 940c6000 640 640)
[ 5310.271956] hd: "vdodec".out[0] - NEW - new -- h=94001fc0 size=00070800 addr=94002000 OK
[ 5310.283296] hd: "vdodec".out[0] - PUSH - data -- h=94001fc0 t=000000013dd4c7ac (YUV:
640x480.520c0420 94002000 9404d000 640 640)
```

4.2.5 perf command

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

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

this perf command could print data count per second

```
[ 5390.325791] hd: "vdodec".in[0] - perf -- (VideoBs) 825 KByte/sec
[ 5391.330332] hd: "vdodec".in[0] - perf -- (VideoBs) 825 KByte/sec
[ 5392.336753] hd: "vdodec".in[0] - perf -- (VideoBs) 819 KByte/sec
[ 5393.345119] hd: "vdodec".in[0] - perf -- (VideoBs) 820 KByte/sec
```

4.2.6 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/vdec/cmd
```

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

```
[ 5471.047725] save i/o begin: "vdodec".in[0] count=1
[ 5473.414755] hd: "vdodec".in[0] - save -- h=9416dfc0 t=00000001478e2434 (VSTM: 1296e000 25874)
[ 5473.443425] hd: "vdodec".in[0] - save -- //mnt//sd//isf_  vdodec_in[0]_c0.bsv ok

[ 5473.451859] save port end
```

4.3 Proc Command of NVR

4.3.1 dump setting

User can cat info file to dump module's status.

```
------------------------ VIDEODEC 0  PATH & BIND ----------------------------
in     out    state   bind_src              bind_dest
0      0      START   -                    VIDEOPROC_4_IN_0
0      1      START   -                    VIDEOPROC_5_IN_0
0      2      START   -                    VIDEOPROC_6_IN_0
0      3      START   -                    VIDEOPROC_7_IN_0
------------------------ VIDEODEC 0  PATH POOL ----------------------------
out    pool    ddr_id  count   max_count
0      0      0      3.5    3.0
0      1      0      3.5    3.0
1      0      0      3.5    3.0
1      1      0      3.5    3.0
2      0      0      3.5    3.0
2      1      0      3.5    3.0
3      0      0      3.5    3.0
3      1      0      3.5    3.0
```

[PATH & BIND]

Status	Description	Value
in	input id of path	0
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)

4.4 Debug Menu for NVR

4.4.1 dump info

After enter debug menu, select 09 to enter this module's sub-menu.

User can select 01 to dump module's status shown as below.

```
------------------------ VIDEODEC 0  PATH & BIND ----------------------------
in     out    state   bind_src              bind_dest
0      0      START   -                     VIDEOPROC_4_IN_0
0      1      START   -                     VIDEOPROC_5_IN_0
0      2      START   -                     VIDEOPROC_6_IN_0
0      3      START   -                     VIDEOPROC_7_IN_0
------------------------ VIDEODEC 0  PATH POOL ----------------------------
out    pool    ddr_id  count   max_count
0      0       0       3.5     3.0
0      1       0       3.5     3.0
1      0       0       3.5     3.0
1      1       0       3.5     3.0
2      0       0       3.5     3.0
2      1       0       3.5     3.0
3      0       0       3.5     3.0
3      1       0       3.5     3.0
```

5 Sample Codes

5.1 Video_playback (IPC)

```
/* Allocate common buffer*/
// config common pool (main)
mem_cfg.pool_info[0].type = HD_COMMON_MEM_COMMON_POOL;
mem_cfg.pool_info[0].blk_size = DBGINFO_BUFSIZE()+VDO_YUV_BUFSIZE(ALIGN_CEIL_64(VDO_SIZE_W),
                                ALIGN_CEIL_64(VDO_SIZE_H), HD_VIDEO_PXLFMT_YUV420);  // align
                                to 16 for rotate buffer
mem_cfg.pool_info[0].blk_cnt = 3;
mem_cfg.pool_info[0].ddr_id = DDR_ID0;
// config common pool for bs pushing in
mem_cfg.pool_info[1].type = HD_COMMON_MEM_USER_POOL_BEGIN;
mem_cfg.pool_info[1].blk_size = BS_BLK_SIZE;
mem_cfg.pool_info[1].blk_cnt = 1;
mem_cfg.pool_info[1].ddr_id = DDR_ID0;
ret = hd_common_mem_init(&mem_cfg);

/* Set dec path configuration */
video_path_cfg.max_mem.codec_type = dec_type;
video_path_cfg.max_mem.dim.w = p_max_dim->w;
video_path_cfg.max_mem.dim.h = p_max_dim->h;
ret = hd_videodec_set(video_dec_path, HD_VIDEODEC_PARAM_PATH_CONFIG, &video_path_cfg);
if (ret != HD_OK) { return ret; }

/* Set dec parameter */
video_in_param.codec_type = dec_type;
video_in_param.dim.w = p_dim->w;
video_in_param.dim.h = p_dim->h;
ret = hd_videodec_set(video_dec_path, HD_VIDEODEC_PARAM_IN, &video_in_param);
if (ret != HD_OK) { return ret; }
```

```
/* Set proc configuration */
ret = hd_videoproc_open(0, HD_VIDEOPROC_0_CTRL, &video_proc_ctrl); //open this for device control
if (p_max_dim != NULL) {
     video_cfg_param.pipe = HD_VIDEOPROC_PIPE_SCALE;
     video_cfg_param.isp_id = 0;
     video_cfg_param.ctrl_max.func = 0;
     video_cfg_param.in_max.func = 0;
     video_cfg_param.in_max.dim.w = p_max_dim->w;
     video_cfg_param.in_max.dim.h = p_max_dim->h;
     video_cfg_param.in_max.pxlfmt = HD_VIDEO_PXLFMT_YUV420;
     video_cfg_param.in_max.frc = HD_VIDEO_FRC_RATIO(1,1);
     ret = hd_videoproc_set(video_proc_ctrl, HD_VIDEOPROC_PARAM_DEV_CONFIG, &video_cfg_param);
     if (ret != HD_OK) {
          return HD_ERR_NG;
     }
}
video_ctrl_param.func = 0;
ret = hd_videoproc_set(video_proc_ctrl, HD_VIDEOPROC_PARAM_CTRL, &video_ctrl_param);
if (ret != HD_OK) { return ret;}

/* Set proc parameter */
video_out_param.func = 0;
video_out_param.dim.w = p_dim->w;
video_out_param.dim.h = p_dim->h;
video_out_param.pxlfmt = HD_VIDEO_PXLFMT_YUV420;
video_out_param.dir = HD_VIDEO_DIR_NONE;
video_out_param.frc = HD_VIDEO_FRC_RATIO(1,1);
ret = hd_videoproc_set(video_proc_path, HD_VIDEOPROC_PARAM_OUT, &video_out_param);
if (ret != HD_OK) { return ret;}

/* Set out configuration */
ret = hd_videoout_open(0, HD_VIDEOOUT_0_CTRL, &video_out_ctrl); //open this for device control
switch(out_type){
case 0:
     videoout_mode.output_type = HD_COMMON_VIDEO_OUT_CVBS;
     videoout_mode.input_dim = HD_VIDEOOUT_IN_AUTO;
```

```
        videoout_mode.output_mode.cvbs= HD_VIDEOOUT_CVBS_NTSC;
break;
case 1:
        videoout_mode.output_type = HD_COMMON_VIDEO_OUT_LCD;
        videoout_mode.input_dim = HD_VIDEOOUT_IN_AUTO;
        videoout_mode.output_mode.lcd = HD_VIDEOOUT_LCD_0;
break;
case 2:
        videoout_mode.output_type = HD_COMMON_VIDEO_OUT_HDMI;
        videoout_mode.input_dim = HD_VIDEOOUT_IN_AUTO;
        videoout_mode.output_mode.hdmi= hdmi_id;
break;
default:
        printf("not support out_type\r\n");
break;
}
ret = hd_videoout_set(video_out_ctrl, HD_VIDEOOUT_PARAM_MODE, &videoout_mode);
if (ret != HD_OK) { return ret;}

/* Set out parameter */
video_out_param.dim.w = p_dim->w;
video_out_param.dim.h = p_dim->h;
video_out_param.pxlfmt = HD_VIDEO_PXLFMT_YUV420;
video_out_param.dir = HD_VIDEO_DIR_NONE;
ret = hd_videoout_set(video_out_path, HD_VIDEOOUT_PARAM_IN, &video_out_param);
if (ret != HD_OK) { return ret;}
memset((void *)&video_out_param,0,sizeof(HD_VIDEOOUT_IN));
ret = hd_videoout_get(video_out_path, HD_VIDEOOUT_PARAM_IN, &video_out_param);
if (ret != HD_OK) { return ret;}

/* Bind modules */
hd_videodec_bind(HD_VIDEODEC_0_OUT_0, HD_VIDEOPROC_0_IN_0);
hd_videoproc_bind(HD_VIDEOPROC_0_OUT_0, HD_VIDEOOUT_0_IN_0);

/* start modules */
hd_videodec_start(stream[0].dec_path);
```

```
hd_videoproc_start(stream[0].proc_path);
hd_videoout_start(stream[0].out_path);

/* 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);
video_bs.phy_addr[0] = pa;
video_bs.size = length;
ret = hd_videodec_push_in_buf(p_stream0->dec_path, &video_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);
```

5.2 video_decode_only (IPC)

```
/* Allocate common buffer*/
// config common pool (main)
mem_cfg.pool_info[0].type = HD_COMMON_MEM_COMMON_POOL;
mem_cfg.pool_info[0].blk_size = DBGINFO_BUFSIZE()+VDO_YUV_BUFSIZE(ALIGN_CEIL_64(VDO_SIZE_W),
                                ALIGN_CEIL_64(VDO_SIZE_H), HD_VIDEO_PXLFMT_YUV420);  // align
                                to 16 for rotate buffer
mem_cfg.pool_info[0].blk_cnt = 3;
mem_cfg.pool_info[0].ddr_id = DDR_ID0;
// config common pool for bs pushing in
mem_cfg.pool_info[1].type = HD_COMMON_MEM_USER_POOL_BEGIN;
mem_cfg.pool_info[1].blk_size = BS_BLK_SIZE;
mem_cfg.pool_info[1].blk_cnt = 1;
mem_cfg.pool_info[1].ddr_id = DDR_ID0;
ret = hd_common_mem_init(&mem_cfg);

/* Set dec path configuration */
```

```
video_path_cfg.max_mem.codec_type = dec_type;
video_path_cfg.max_mem.dim.w = p_max_dim->w;
video_path_cfg.max_mem.dim.h = p_max_dim->h;
ret = hd_videodec_set(video_dec_path, HD_VIDEODEC_PARAM_PATH_CONFIG, &video_path_cfg);
if (ret != HD_OK) { return ret; }

/* Set dec parameter */
video_in_param.codec_type = dec_type;
video_in_param.dim.w = p_dim->w;
video_in_param.dim.h = p_dim->h;
ret = hd_videodec_set(video_dec_path, HD_VIDEODEC_PARAM_IN, &video_in_param);
if (ret != HD_OK) { return ret; }

/* start modules */
hd_videodec_start(stream[0].dec_path);

/* 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);
video_bs.phy_addr[0] = pa;
video_bs.size = length;
ret = hd_videodec_push_in_buf(p_stream0->dec_path, &video_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_videodec_pull_out_buf(p_stream0->dec_path, &data_pull, -1);
if (ret == HD_OK) {
    hd_videodec_get(p_stream0->dec_path, HD_VIDEODEC_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_videodec_release_out_buf(p_stream0->dec_path, &data_pull);
```

5.3 user_videodec (NVR)

```
/* Set parameters */
config.max_mem.dim.w = max_frame_width;
config.max_mem.dim.h = max_frame_height;
config.max_mem.frame_rate = 30;
ret = hd_videodec_set(path_id, HD_VIDEODEC_PARAM_PATH_CONFIG, &config);
if (ret != HD_OK) {
    printf("hd_videodec_set fail\n");
    goto exit;
}

/* Allocate in buffer */
bs_in_buffer.ddr_id = ddr_id;
bs_in_buffer.size = BS_BUF_SIZE;
blk = hd_common_mem_get_block(pool, bs_in_buffer.size, ddr_id);
if (HD_COMMON_MEM_VB_INVALID_BLK == blk) {
    printf("hd_common_mem_get_block fail\r\n");
    ret =  HD_ERR_NG;
    goto exit;
}
bs_in_buffer.phy_addr = hd_common_mem_blk2pa(blk);
if (bs_in_buffer.phy_addr == 0) {
```

```
    printf("hd_common_mem_blk2pa fail, blk = %#lx\r\n", blk);
    hd_common_mem_release_block(blk);
    return HD_ERR_NG;
}
bs_in_buffer_va = hd_common_mem_mmap(HD_COMMON_MEM_MEM_TYPE_NONCACHE,
                                     bs_in_buffer.phy_addr,
                                     bs_in_buffer.size);

/* Allocate out buffer */
blk = hd_common_mem_get_block(pool, frame_buf_size, ddr_id);
if (HD_COMMON_MEM_VB_INVALID_BLK == blk) {
    printf("hd_common_mem_get_block fail\r\n");
    ret =  HD_ERR_NG;
    goto exit;
}
dec_out_buffer.phy_addr[0] = hd_common_mem_blk2pa(blk);
if (dec_out_buffer.phy_addr[0] == 0) {
    printf("hd_common_mem_blk2pa fail, blk = %#lx\r\n", blk);
    hd_common_mem_release_block(blk);
    return HD_ERR_NG;
}
dec_out_buffer_va = hd_common_mem_mmap(HD_COMMON_MEM_MEM_TYPE_NONCACHE,
                                       dec_out_buffer.phy_addr[0],
                                       frame_buf_size);

/* Push in buffer */
fscanf(bs_len_fd, "%d\n", &length);
if (length > BS_BUF_SIZE) {
    printf("User bitstream is too large. bs_size(%d) > max(%d)\n", length, BS_BUF_SIZE);
    goto exit;
}
fseek(bs_fd, 0, SEEK_SET);
fread((void *)bs_in_buffer_va, 1, length, bs_fd);
bs_in_buffer.size = length;

ret = hd_videodec_push_in_buf(path_id, &bs_in_buffer, &dec_out_buffer, 500);
```

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

/* Pull out buffer */
ret = hd_videodec_pull_out_buf(path_id, &dec_out_buffer, 500);
if (ret != HD_OK) {
     printf("hd_videodec_pull_out_buf fail\n");
     goto exit;
} else {
     dec_out_buffer_va = hd_common_mem_mmap(HD_COMMON_MEM_MEM_TYPE_NONCACHE,
                                            dec_out_buffer.phy_addr[0],
                                            frame_buf_size);
     sprintf(filename, "user_dec_%ldx%ld_YUV420_16x2.yuv",
             dec_out_buffer.dim.w, dec_out_buffer.dim.h);
     save_output(filename, dec_out_buffer_va, frame_buf_size);
}

/* Release in buffer */
hd_common_mem_munmap(bs_in_buffer_va, frame_buf_size);
hd_common_mem_release_block((HD_COMMON_MEM_VB_BLK)bs_in_buffer.phy_addr);

/* Release out buffer */
hd_common_mem_munmap(dec_out_buffer_va, frame_buf_size);
hd_common_mem_release_block((HD_COMMON_MEM_VB_BLK)dec_out_buffer.phy_addr[0]);
```

6 Q & A

1. Why using hd_videodec_push_in_buf() but return error -19?
   Answer:
   - Check hd_videodec_start() is already called.
   - If hd_videodec_start() is called, please check the order of hd_videodec_set (HD_VIDEODEC_PARAM_IN_DESC) and hd_videodec_start(). Call hd_videodec_set(HD_VIDEODEC_PARAM_IN_DESC) first, and then call hd_videodec_start().

2. Why using hd_videoproc_pull_out_buf() to pull the yuv data from videodec, but return time out error -15?
   Answer:
   - Check the bitstream first. If the continuous incorrect frames exist in the bitstream, video decode error happened and decoder not send the error YUV to vproc in binding mode. It caused time out error -15 if timeout value is set.

3. What values of width and height should I set for each codecs?
   Answer:
   - H264: width is aligned to 64, height is aligned to 16.
   - H264 and JPEG: width is aligned to 64, height is aligned to 64.

4. Is it possible to push yuv data from vdodec to vdoout directly?
   Answer:
   - For Achieving video playback function, vdec, vproc and vout are necessary. The binding order is vdec, vproc, and then vout. Vproc scaled the resolution if the resolution of YUV is differ to display device.

5. Is it possible to change the resolution when video decode is processing?
   Answer:
   - No, please call hd_videodec_stop() to stop decoder, set the new width/height, and call hd_videodec_start() to start decoder again. Remember that config the maximum width/height at beginning.