Novatek HDAL Design Specification - hd_videoenc

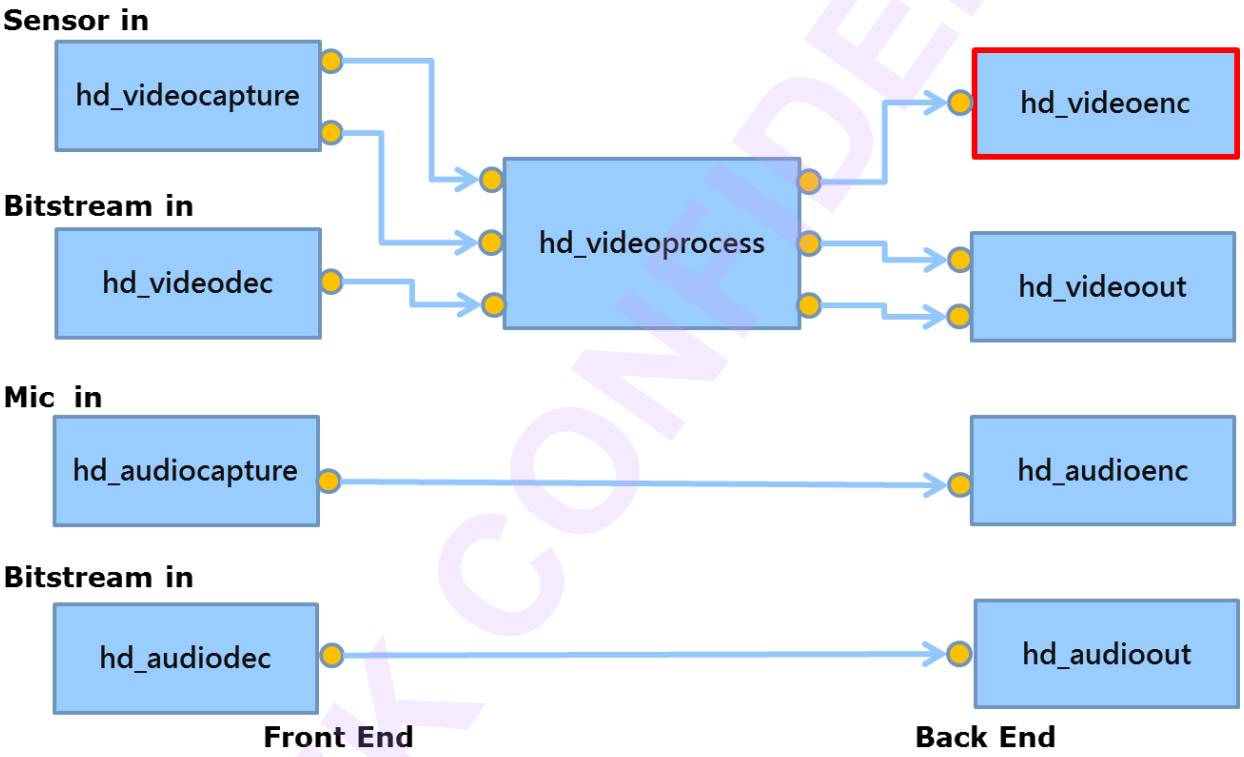
Table of Content

Difference Table (for IPC only)

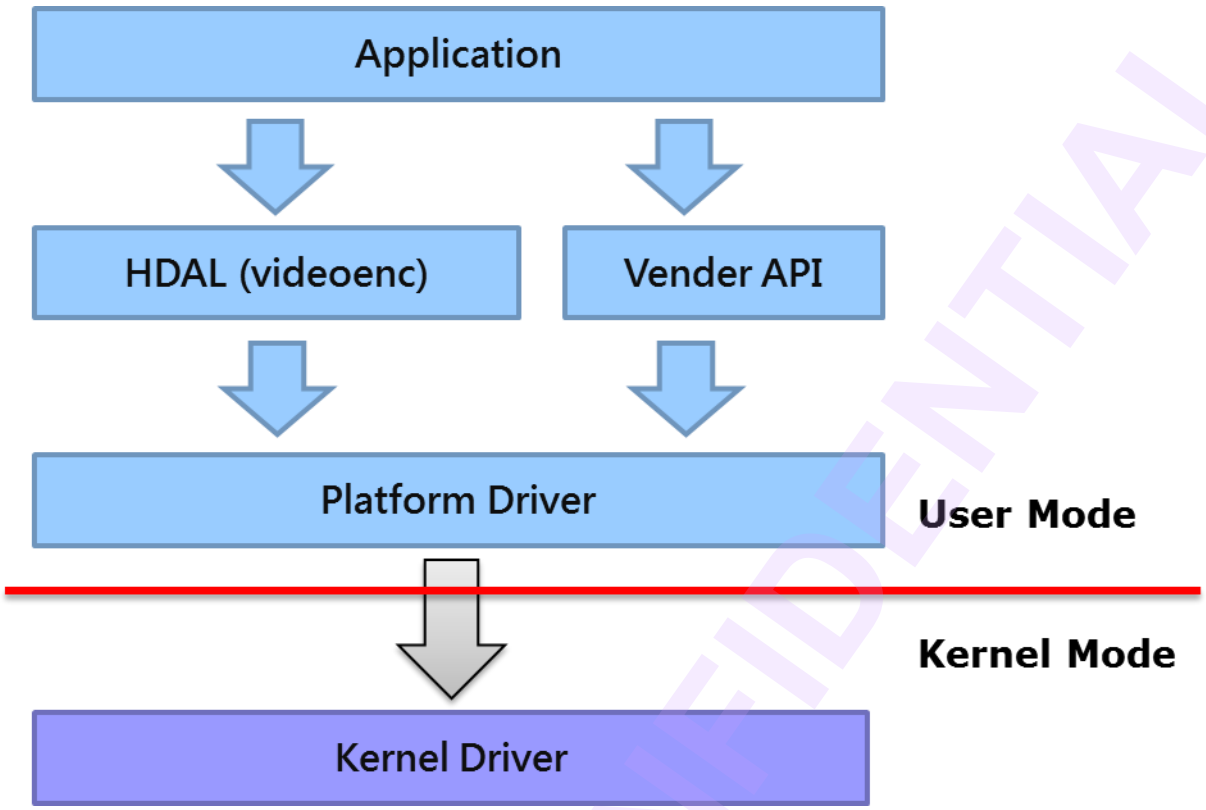
Item	NT9668X	NT9852X
HD_VIDEOENC_PATH_CONFIG in_func = ONEBUFF	Not supported.	Support. Configure to enable ONEBUFF mode
HD_VIDEOENC_PATH_CONFIG in_func = LOWLATENCY	Not supported.	Support. Configure to enable LOWLATENCY mode
HD_VIDEOENC_PATH_CONFIG data_pool. ddr_id	Not supported	Support. Configure which ddr to work & output bitstream
HD_H26XENC_DEBLOCK dis_ilf_idc-> across_tile_en	Not supported	Support. Configure if deblocking should reference across tile
HD_VIDEOENC_PARAM_IN_STAMP_ATTR alpha	Only for rgb565	For argb4444、árgb1555、rgb565

1 Introduction

The major purpose of hd_videoenc is to get YUV raw data from upper unit, and controls the video encoder to encode the YUV data then return the bitstream data which can be used for saving video files / online streaming. This document will talk about the red block in the following diagram. The device driver is not the main point in this document.

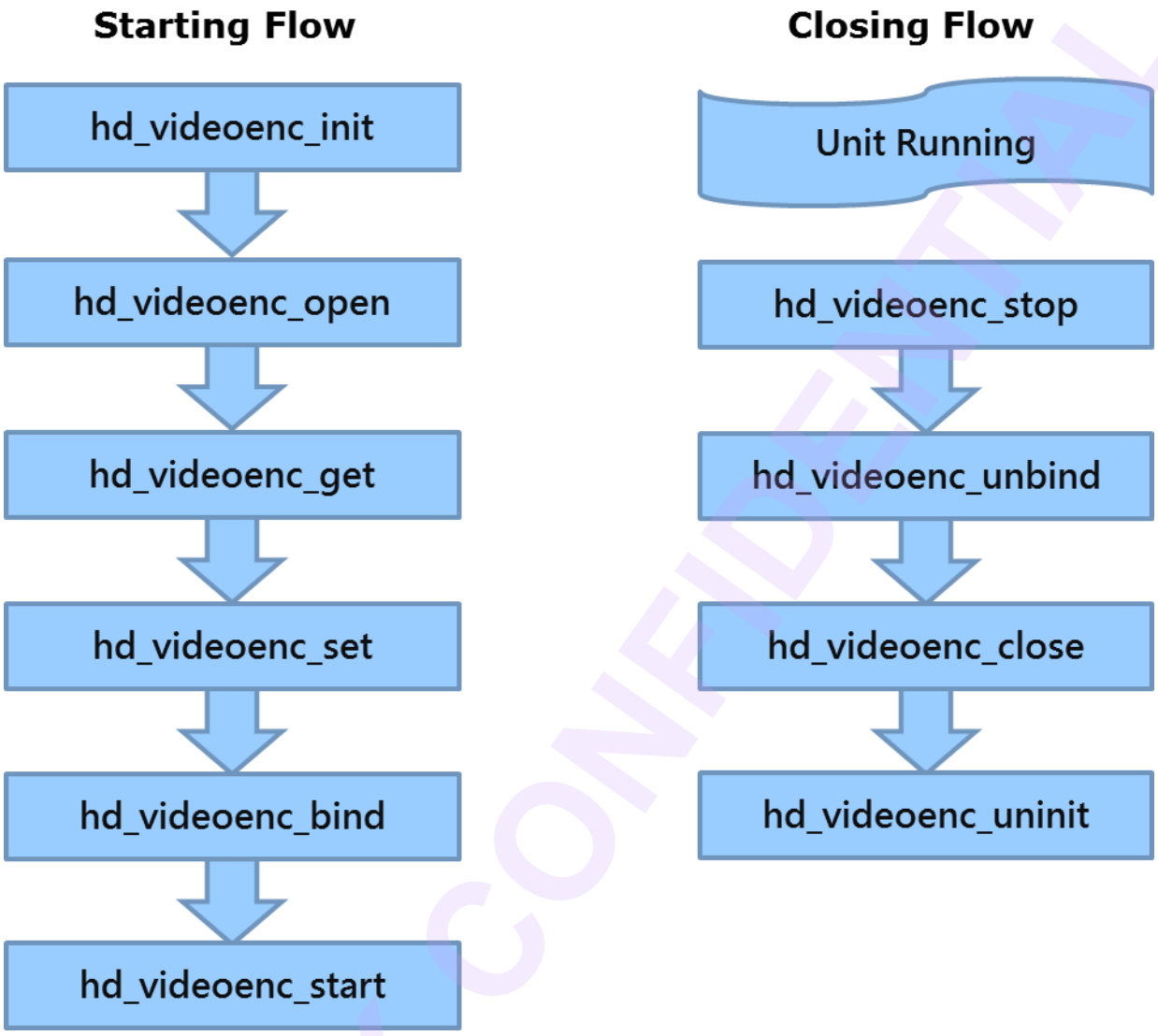


Module diagram is shown as below:



1.1 Basic Flow

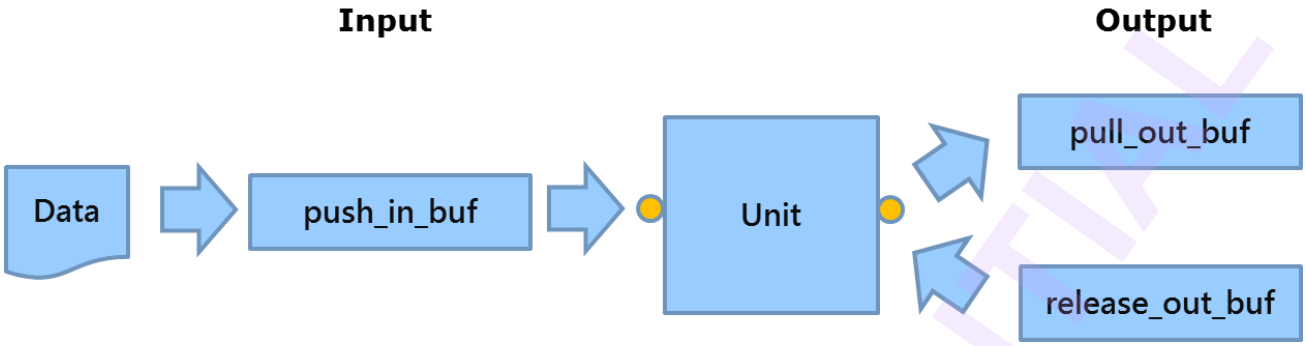
The call sequence is needed to be done correctly for the unit. The standard starting flows of most modules are init, open, get, set, bind and start. The standard closing flows of most modules are stop, unbind, close and uninit. The basic flow is shown as below.



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

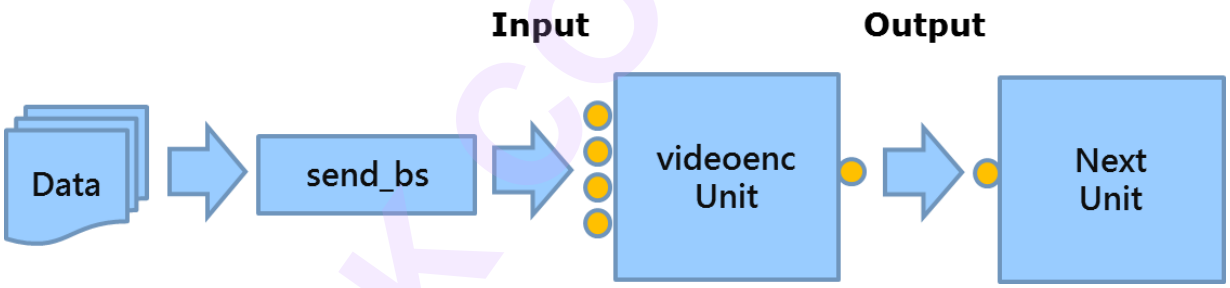
1.2 Single Trigger Operation

Single trigger operation is used to trigger the unit to do one job, such as to grab one YUV frame from video capture; then, 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 push; the sequence for output port is pull and release. The flow is shown as below.



1.3 Multi Channel Operation

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



2 Parameter IDs and data structure definition

The videoenc provides the following parameter IDs:

- HD_VIDEOENC_PARAM_DEVCOUNT
  - NVR/IPC. support get with ctrl path
  - using HD_DEVCOUNT struct (device id max count)
- HD_VIDEOENC_PARAM_SYSCAPS
  - NVR/IPC. support get with ctrl path
  - using HD_VIDEOENC_SYSCAPS struct (system capabilitiy)
- HD_VIDEOENC_PARAM_PATH_CONFIG
  - NVR/IPC. support get/set with i/o path
  - using HD_VIDEOENC_PATH_CONFIG struct
- HD_VIDEOENC_PARAM_BUFINFO
  - IPC only. support get with i/o path
  - using HD_VIDEOENC_BUFINFO struct
- HD_VIDEOENC_PARAM_IN
  - NVR/IPC. support get/set with i/o path
  - using HD_VIDEOENC_IN struct
- HD_VIDEOENC_PARAM_OUT_ENC_PARAM
  - NVR/IPC. support get/set with i/o path
  - using HD_VIDEOENC_OUT struct
- HD_VIDEOENC_PARAM_OUT_VUI
  - NVR/IPC. support get/set with i/o path
  - using HD_H26XENC_VUI struct
- HD_VIDEOENC_PARAM_OUT_DEBLOCK
  - NVR/IPC. support get/set with i/o path
  - using HD_H26XENC_DEBLOCK struct
- HD_VIDEOENC_PARAM_OUT_RATE_CONTROL

- NVR/IPC. support get/set with i/o path
- using HD_H26XENC_RATE_CONTROL struct

- HD_VIDEOENC_PARAM_OUT_USR_QP
  - NVR/IPC. support get/set with i/o path
  - using HD_H26XENC_USR_QP struct

- HD_VIDEOENC_PARAM_OUT_SLICE_SPLIT
  - NVR/IPC. support get/set with i/o path
  - using HD_H26XENC_SLICE_SPLIT struct

- HD_VIDEOENC_PARAM_OUT_ENC_GDR
  - NVR/IPC. support get/set with i/o path
  - using HD_H26XENC_GDR struct

- HD_VIDEOENC_PARAM_OUT_ROI
  - NVR/IPC. support get/set with i/o path
  - using HD_H26XENC_ROI struct

- HD_VIDEOENC_PARAM_OUT_ROW_RC
  - NVR/IPC. support get/set with i/o path
  - using HD_H26XENC_ROW_RC struct

- HD_VIDEOENC_PARAM_OUT_AQ
  - NVR/IPC. support get/set with i/o path
  - using HD_H26XENC_AQ struct

- HD_VIDEOENC_PARAM_OUT_REQUEST_IFRAME
  - NVR/IPC. support set with i/o path
  - using HD_H26XENC_REQUEST_IFRAME struct

- HD_VIDEOENC_PARAM_OUT_TRIG_SNAPSHOT
  - IPC only support set with i/o path
  - using HD_H26XENC_TRIG_SNAPSHOT struct

- HD_VIDEOENC_PARAM_IN_STAMP_BUF
  - NVR/IPC. support set with i/stamp path
  - using HD_OSG_STAMP_BUF struct (stamp buffer parameter)

- HD_VIDEOENC_PARAM_IN_STAMP_IMG

- NVR/IPC. support set with i/stamp path
- using HD_OSG_STAMP_IMG struct (stamp image parameter)

- HD_VIDEOENC_PARAM_IN_STAMP_ATTR
  - NVR/IPC. support get/set with i/stamp path
  - using HD_OSG_STAMP_ATTR struct (stamp display attribute)

- HD_VIDEOENC_PARAM_IN_MASK_ATTR
  - NVR/IPC. support get/set with i/mask path
  - using HD_OSG_MASK_ATTR struct (mask display attribute)

- HD_VIDEOENC_PARAM_IN_MOSAIC_ATTR
  - NVR/IPC. support get/set with i/mask path
  - using HD_OSG_MOSAIC_ATTR struct (mosaic display attribute)

- HD_VIDEOENC_PARAM_IN_PALETTE_TABLE
  - NVR only support get/set with i path
  - using HD_OSG_PALETTE_TBL struct

2.1 General function

2.1.1 hd_videoenc_init

[Description]
Initialize the unit

[Syntax]
HD_RESULT hd_videoenc_init(VOID);

[Parameter]

Value	Description
VOID	Not available

[Return Value]

Value	Description

HD_OK	Success
HD_ERR_NG	Failure

2.1.2 hd_videoenc_open

[Description]
Open the unit

[Syntax]
HD_RESULT hd_videoenc_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

[Note]
For OSG:

1. There are two kinds of OSG : ext and non-ext. ext poses less position limitation but consumes more CPU/DMA. ext is ideal for OSG with small resolution and high position flexibility.

2.1.3 hd_videoenc_get

[Description]
Get parameters from unit by path id

[Syntax]

HD_RESULT hd_videoenc_get(HD_PATH_ID path_id, HD_VIDEOENC_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_videoenc_set

[Description]
Set parameters to unit by path id

[Syntax]
HD_RESULT hd_videoenc_set(HD_PATH_ID path_id, HD_VIDEOENC_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_videoenc_bind

[Description]
Bind this unit with destination unit

[Syntax]
HD_RESULT hd_videoenc_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_videoenc_start

[Description]
Start the unit

[Syntax]
HD_RESULT hd_videoenc_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_videoenc_stop

[Description]
Stop the unit

[Syntax]
HD_RESULT hd_videoenc_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_videoenc_unbind

[Description]
Unbind the unit

[Syntax]
HD_RESULT hd_videoenc_unbind(HD_OUT_ID out_id);

[Parameter]

Value	Description
out_id	id of output port.

[Return Value]

Value	Description
HD_OK	Success

HD_ERR_NG	Failure

2.1.9 hd_videoenc_close

[Description]
Close the unit

[Syntax]
HD_RESULT hd_videoenc_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

[Note]
For OSG:
1. OSGs will keep registered buffer until they are closed. Only after this API returns can application safely access/reclaim the buffer.

2.1.10 hd_videoenc_uninit

[Description]
Uninitialize the unit

[Syntax]
HD_RESULT hd_videoenc_uninit(VOID);

[Parameter]

Value	Description

VOID	Not available

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.11 hd_videoenc_push_in_buf

[Description]
Push the video frame buffer to unit

[Syntax]
HD_RESULT hd_videoenc_push_in_buf(HD_PATH_ID path_id, HD_VIDEO_FRAME* p_in_video_frame, HD_VIDEOENC_BS* p_user_out_videoenc_bs, INT32 wait_ms);

[Parameter]

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

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

[Note]
p_user_out_videoenc_bs is optional. If this value is set, videoenc module uses it as the output buffer for bitstream, and the buffer should be released by user after using it; otherwise, if the value is NULL, videoenc will allocate the buffer internally, and user need to call hd_videoenc_release_out_buf API to release the buffer finally.

[Note]
If videoproc is NOT binding to videoenc, that is, SDK user call pull_out YUV from videoproc and push_in to videoenc to encode. Be sure to check YUV weight/height is equal to encoder weight/height.

If videoproc is binding to videoenc, the SDK will automatically check YUV & encoder setting. If the YUV weight/height is not correctly for encoding, SDK will automatically drop YUV.

But if videoproc is NOT binding to videoenc, the SDK will NOT automatically check YUV & encoder setting, SDK user should take responsibility for checking YUV weight/height and encoder setting. This is essential to check YUV & encoder settings while changing resolution, because the two module will NOT change to new setting at the same time. If YUV from videoproc is NOT match videoenc setting, the YUV should be dropped instead of pushing to videoenc.

2.1.12 hd_videoenc_pull_out_buf

[Description]
Pull the video bitstream buffer from unit

[Syntax]
HD_RESULT hd_videoenc_pull_out_buf (HD_PATH_ID path_id, HD_VIDEOENC_BS* p_videoenc_bs, INT32 wait_ms);

[Parameter]

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

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.13 hd_videoenc_release_out_buf

[Description]
Release the video bitstream buffer which is get from unit

[Syntax]
HD_RESULT hd_videoenc_release_out_buf (HD_PATH_ID path_id, HD_VIDEOENC_BS* p_videoenc_bs)

[Parameter]

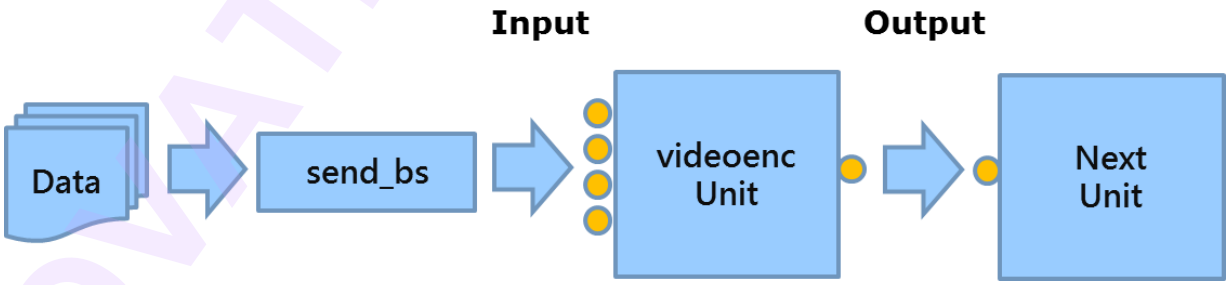
Value	Description
path_id	the path id
p_videoenc_bs	pointer of the output video bitstream buffer

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.2 Multi List Operation

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



2.2.1 hd_videoenc_start_list

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

[Syntax]
HD_RESULT hd_videoenc_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.

2.2.2 hd_videoenc_stop_list

[Description]
Stop sending multi bitstream data to the unit

[Syntax]
HD_RESULT hd_videoenc_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.

2.2.3 hd_videoenc_poll_list

[Description]

Query the bitstream status of all specifying channels

[Syntax]

HD_RESULT hd_videoenc_poll_list(HD_VIDEOENC_POLL_LIST *p_poll, UINT32 num, INT32 wait_ms);

[Parameter]

Value	Description
p_poll	The path information of multi channels
num	Number of bitstream paths
wait_ms	The timeout value in millisecond while polling

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

[Difference]

Chip	Description
IPC	NOT supported.

NVR	All functions are supported.

2.2.4 hd_videoenc_recv_list

[Description]
Receive bitstream data for all channels

[Syntax]
HD_RESULT hd_videoenc_recv_list(HD_VIDEOENC_RECV_LIST *p_videoenc_list, UINT32 num);

[Parameter]

Value	Description
p_videoenc_list	An array of bitstream structure to be filled for multi channels
num	The number of channels to retrieve bitstream

[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

2.3.1 HD_VIDEOENC_PARAM_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, using HD_DEVICE_CAPS
in_caps	capability of input, using HD_VIDEO_CAPS
out_caps	capability of output, using HD_VIDEOENC_CAPS
max_in_stamp	max input stamp
max_in_stamp_ex	max input stamp_ex
max_in_mask	max input mask
max_in_mask_ex	max input mask_ex

2.3.2 HD_VIDEOENC_PARAM_PATH_CONFIG

[Description]
Path configure

[Parameter]

Value	Description
max_mem	IPC only. maximum memory information. Using HD_VIDEOENC_MAXMEM struct
isp_id	IPC only. ISP id. range: 0~7 or 0xffffffff = ignore

data_pool	NVR only. pool memory information. User can specify the size/count of buffers for encoder's out-buffer.

[Apply Require]

stop -> close -> open -> set -> start

2.3.3 HD_VIDEOENC_PARAM_BUFINFO

[Description]
Buffer information

[Parameter]

Value	Description
buf_info	IPC only. physical addr/size of bitstream buffer, for user space to mmap

2.3.4 HD_VIDEOENC_PARAM_IN

[Description]
Input parameters

[Parameter]

Value	Description
dim	encode width/height, using HD_DIM struct
pxl_fmt	source format, using HD_VIDEO_PXLFMT struct NVR: the valid value H.264/H.265: HD_VIDEO_PXLFMT_YUV420 / HD_VIDEO_PXLFMT_YUV420_NVX3 JPEG: HD_VIDEO_PXLFMT_YUV420_MB
dir	input direction, using HD_VIDEO_DIR struct

[Apply Require]

stop -> set -> start

[Note]
If videoproc is binding to videoenc, and HD_VIDEOPROC_OUT is set to default=0 (auto-sync parameters from videoenc), then remember to call start for videoproc

[videoenc] stop -> set -> start [videoproc] start

2.3.5 HD_VIDEOENC_PARAM_OUT_ENC_PARAM

[Description]
Input frame

[Parameter]

Value	Description
codec_type	codec type, using HD_VIDEO_CODEC struct
h26x	H26x config, using HD_H26X_CONFIG struct
jpeg	Jpeg config, using HD_JPEG_CONFIG struct

[Apply Require]

stop -> set -> start

2.3.6 HD_VIDEOENC_PARAM_OUT_VUI

[Description]
H26x vui settings

[Parameter]

Value	Description
vui_en	enable vui. default: 0, range: 0~1 (0: disable, 1: enable)
sar_width	Horizontal size of the sample aspect ratio. default: 0, range: 0~65535
sar_height	Vertical size of the sample aspect rat. default:

	0, range: 0~65535
matrix_coef	Matrix coefficients are used to derive the luma and Chroma signals from green, blue, and red primaries. default: 2, range: 0~255
transfer_characteristics	The opto-electronic transfers characteristic of the source pictures. default: 2, range: 0~255
colour_primaries	Chromaticity coordinates the source primaries. default: 2, range: 0~255
video_format	Indicate the representation of pictures. default: 5, range: 0~7
color_range	Indicate the black level and range of the luma and Chroma signals. default: 0, range: 0~1 (0: Not full range, 1: Full range)
timing_present_flag	timing info present flag. default: 0, range: 0~1 (0: disable, 1: enable)

[Apply Require]

stop -> set -> start

2.3.7 HD_VIDEOENC_PARAM_OUT_DEBLOCK

[Description]

H26x deblock settings

[Parameter]

Value	Description
dis_ilf_idc	Disable loop filter in slice header. default: 0, range: 0~2 (0: Filter, 1: No Filter, 2: Slice Mode)
db_alpha	Alpha & C0 offset. default: 0, range: -12~12
db_beta	Beta offset. default: 0, range: -12~12

[Apply Require]

stop -> set -> start

2.3.8 HD_VIDEOENC_PARAM_OUT_RATE_CONTROL

[Description]
H26x rate control settings

[Parameter]

Value	Description
rc_mode	rate control mode. default: 1, range: 1~4 (1: CBR, 2: VBR, 3: FixQP, 4: EVBR), using HD_VIDEOENC_RC_MODE struct
cbr	parameter of rate control mode CBR, using HD_H26XENC_CBR struct
vbr	parameter of rate control mode VBR, using HD_H26XENC_VBR struct
fixqp	parameter of rate control mode FixQP, using HD_H26XENC_FIXQP struct
evbr	parameter of rate control mode EVBR, using HD_H26XENC_EVBR struct

[Apply Require]

set -> start

2.3.9 HD_VIDEOENC_PARAM_OUT_USR_QP

[Description]
H26x user qp settings

[Parameter]

Value	Description
Enable	NVR/IPC. enable user qp. default: 0, range: 0~1 (0: disable, 1: enable)
qp_map_addr	NVR/IPC. buffer address of user qp map. IPC: one byte per cu16 (bit[0:7] qp value. default: 26, range: 0~51)

	NVR: two byte per cu16. (bit[0:5] qp value (default: 0; if qp mode is 3 then qp value means fixed qp [range: 0~51], otherwise qp value means delta qp [range: -32~31])) (bit[6:7] qp mode (default: 0; 0: delta qp, 1: reserved, 2: delta qp [disable AQ], 3: fixed qp)

[Apply Require]

set -> start

2.3.10 HD_VIDEOENC_PARAM_OUT_SLICE_SPLIT

[Description]

H26x slice split

[Parameter]

Value	Description
enable	enable multiple slice. default: 0, range: 0~1 (0: disable, 1: enable)
slice_row_num	number of macroblock/ctu rows occupied by a slice, range: 1 ~ number of macroblock/ctu row

[Apply Require]

set -> start

2.3.11 HD_VIDEOENC_PARAM_OUT_ENC_GDR

[Description]

H26x GDR settins

[Parameter]

Value	Description

enable	enable gdr. default: 0, range: 0~1 (0: disable, 1: enable)
period	intra refresh period. default: 0, range: 0~0xFFFFFFFF (0: always refresh, others: intra refresh frame period)
number	intra refresh row number. default: 1, range: 1 ~ number of macroblock/ctu row

[Apply Require]

set -> start

2.3.12 HD_VIDEOENC_PARAM_OUT_ROI

[Description]

H26x ROI settings

[Parameter]

Value	Description
roi_qp_mode	IPC only. roi qp mode for all windows. available value: HD_VIDEOENC_QPMODE_FIXED_QP(default) / HD_VIDEOENC_QPMODE_DELTA, using HD_VIDEOENC_QPMODE struct
st_roi	NVR/IPC. roi window settings. ROIs can be overlaid, and the priority of the ROIs is based on index number, index 0 is highest priority and index 9 is lowest. Using HD_H26XENC_ROI_WIN struct

[Apply Require]

set -> start

2.3.13 HD_VIDEOENC_PARAM_OUT_ROW_RC

[Description]

H26x row rc settings

[Parameter]

Value	Description
enable	NVR/IPC. enable row rc. default: 1, range: 0~1 (0: disable, 1: enable)
i_qp_range	NVR/IPC. IPC. qp range of I&P frame for row-level rata control. default: 2, range: 0~15 NVR. qp range of I frame for row-level rata control. default: 2, range: 0~15
i_qp_step	NVR/IPC. IPC. qp step of I&P frame for row-level rata control. default: 1, range: 0~15 NVR. qp step of I frame for row-level rata control. default: 1, range: 0~15
p_qp_range	NVR only. qp range of P frame for row-level rata control. default: 4, range: 0~15
p_qp_step	NVR only. qp step of P frame for row-level rata control. default: 1, range: 0~15
min_i_qp	NVR only. min qp of I frame for row-level rata control. default: 1, range: 0~51
max_i_qp	NVR only. max qp of I frame for row-level rata control. default: 51, range: 0~51
min_p_qp	NVR only. min qp of P frame for row-level rata control. default: 1, range: 0~51
max_p_qp	NVR only. max qp of P frame for row-level rata control. default: 51, range: 0~51

[Apply Require]

set -> start

2.3.14 HD_VIDEOENC_PARAM_OUT_AQ

[Description]

H26x aq settings

[Parameter]

Value	Description
enable	NVR/IPC. AQ enable. default: 0, range: 0~1 (0: disable, 1: enable)
i_str	NVR/IPC. aq strength of I frame. default: 3, range: 1~8
p_str	NVR/IPC. aq strength of P frame. default: 1, range: 1~8
max_delta_qp	NVR/IPC. IPC. max delta qp of aq. default: 6, range: 0~8 NVR. max delta qp of aq. default: 6, range: 0~15
min_delta_qp	NVR/IPC. IPC. min delta qp of aq. default: -6, range: -8~0 NVR. min delta qp of aq. default: -6, range: -15~0
depth	NVR only. AQ depth. default: 2, range(H.264): 2, range(H.265): 0~2 (0: cu64, 1: cu32, 2: cu16)
thd_table	NVR only. non-linear AQ mapping table. range: -512~511, default: {-120,-112,-104, -96,

	-88, -80, -72, -64, -56, -48, -40, -32, -24, -16, -8, 7, 15, 23, 31, 39,47, 55, 63, 71, 79, 87, 95, 103, 111, 119} for (dqp = -15; dqp < 15; dqp++) if (Cu.RelativeTextureComplexity(x_str) <= thd_table[dqp+15]) break; Cu.DeltaQP_AQ = MIN (MAX (min_delta_qp, dqp), max_delta_qp);

[Apply Require]

set -> start

2.3.15 HD_VIDEOENC_PARAM_OUT_REQUEST_IFRAME

[Description]

H26x request I-frame

[Parameter]

Value	Description
enable	request i-frame enable. default: 0, range: 0~1 (0: disable, 1: enable)

[Apply Require]

set -> start

2.3.16 HD_VIDEOENC_PARAM_OUT_TRIG_SNAPSHOT

[Description]

H26x trigger snapshot

[Parameter]

Value	Description

phy_addr	[w]physical address of encoded data (user provide memory space to put JPEG result)
size	[w]user buffer size provided [r]real size of encoded data

[Apply Require]

set

2.3.17 HD_VIDEOENC_PARAM_IN_STAMP_BUF

[Description]

Stamp buffer settings

[Parameter]

Value	Description
type	NVR/IPC. ping pong buffer or single buffer, using HD_OSG_BUF_TYPE
size	NVR/IPC. buffer's size in byte
p_addr	NVR/IPC. buffer's physical address
ddr_id	NVR only. p_addr's ddr id

[Note]

For IPCam:

2. Different OSGs can share the same buffer to save memory
3. Double buffer requires "2 * max OSG resolution * sizeof(short)" while single buffer requires only "max OSG resolution* sizeof(short)". But single buffer suffers from blinking when image is updated.
4. The starting address and length should be 4bytes aligned.

2.3.18 HD_VIDEOENC_PARAM_IN_STAMP_IMG

[Description]

Stamp image settings

[Parameter]

Value	Description
fmt	NVR/IPC. RGB565/ARGB1555/ARGB4444/ARGB8888/ Using HD_VIDEO_PXLFMT struct
dim	NVR/IPC. image's width and height, using HD_DIM struct
p_addr	NVR/IPC. image's bitmap content
ddr_id	NVR only. p_addr's ddr id

[Note]

For IPCam:

1. Only RGB565/ARGB1555/ARGB4444 are supported
2. Image width and height are best to be multiple of 2 for best compatibility.
3. In addition to the whole image width and height, every color area(e.g. timestamp and border)'s width and height should be multiple of 2.
4. hd_videoenc_get retrieves free buffer(not accessed by hardware) for OSG of ping pong buffer

2.3.19 HD_VIDEOENC_PARAM_IN_STAMP_ATTR

[Description]

Stamp attr settings

[Parameter]

Value	Description
align_type	NVR only. to which corner is stamp aligned Using HD_OSG_ALIGN_TYPE struct
alpha	NVR/IPC. (DISP)alpha value
position	NVR/IPC. (DISP)stamp's x,y position, using HD_IPOINT struct
colorkey_en	IPC only. is colorkey used to filter background
colorkey_val	IPC only. filtered background color
qp_en	IPC only. does stamp have its own qp
qp_fix	IPC only. qp_val is fixed or relative to

	streaming qp
qp_val	IPC only. qp value
layer	IPC only. set layer attribute for videoenc
region	IPC only. set region attribute for videoenc
gcac_enable	NVR only. (GCAC)gcac enable
gcac_blk_width	NVR only. unit width of GCAC
gcac_blk_height	NVR only. unit height of GCAC. Note: OSG dim / (gcac_blk_width* gcac_blk_height) must less than 64

[Note]

For IPCam:

1. align_type, gcac_* are not supported
2. For ARGB4444, alpha field is not applicable. For ARGB1555. alpha[3..0] is for pixels of A = 0 and alpha[7..4] is for pixels of A = 1.
3. X y are best to be multiple of 2 for best compatibility. For h264, any 16*16 macro block can have only one OSG. For h265, any 64*64 macro block can have only one OSG.
4. If two OSGs are inside a macro block or even overlapped, they must be in different layer(currently, there are only two layers : 0 and 1. Layer 0 will be rendered above layer 1.)
5. region is a serial number(each layer has 16 regions : 0 ~ 15)
6. qp_* are used to resolve conflicting qp value between streaming and OSG.

2.3.20 HD_VIDEOENC_PARAM_IN_MASK_ATTR

[Description]

Mask attribute settings

[Parameter]

Value	Description
type	NVR/IPC. mask is solid or hollow. Using HD_OSG_MASK_TYPE
color	IPC. mask color in rgb, NVR.mask palette index
alpha	NVR/IPC. mask transparency

position	NVR/IPC. 4 vertices' position, using HD_UPOINT struct
thickness	IPC only. border width for hollow mask

[Note]

For IPCam:

1. position[0] should be the top left. Others should be in clockwise order.
2. thickness should be multiple of 2
3. Hollow mask takes more time to complete than solid mask. Don't set over 4 hollow masks in a path.

2.3.21 HD_VIDEOENC_PARAM_IN_MOSAIC_ATTR

[Description]

Mosaic attribute settings

[Parameter]

Value	Description
Type	NVR only. mask is solid or inversion. Using HD_OSG_MASK_TYPE struct
Alpha	NVR only. mask alpha blending. range: 0 ~ 256 (0: foreground, 256: background)
mosaic_blk_w	NVR/IPC. witdh of internal block
mosaic_blk_h	NVR/IPC. height of internal block
position	NVR/IPC. 4 vertices' position, using HD_UPOINT struct

[Note]

For IPCam:

1. Mosaic is not supported

2.3.22 HD_VIDEOENC_PARAM_IN_PALETTE_TABLE

[Description]

Palette table settings

[Parameter]

Value	Description
pal_y	NVR mask only. palette colors y. range: 0 ~ 255
pal_cb	NVR mask only. palette colors cb. range: 0 ~ 255
pal_cr	NVR mask only. palette colors cr. range: 0 ~ 255

2.3.23 HD_VIDEOENC_POLL_LIST

[Description]

The polling item including path information

Use this type to form an array in hd_videoenc_poll_list() to get the results for all paths.

[Parameter]

Value	Description
path_id	path ID
revent	The returned event value

[Difference]

Chip	Description
IPC	NOT supported.
NVR	Supported.

2.3.24 HD_VIDEOENC_USER_BS

[Description]

Video bitstream data and relative information

[Parameter]

Value	Description
sign	signature = MAKEFOURCC('V','S','T','M')

p_next	pointer to next meta
vcodec_format	Encoded format of video frame
pack_num	Pack number in video frame
timestamp	Encode bs timestamp
frame_type	The frame type
svc_layer_type	svc layer type
video_pack	Pack array of encoded data
psnr_info	The PSNR information
blk_info	The block partition information
newbs_flag	Flag notification of new seting
qp	The qp value
slice_offset	multi-slice offset 0~VENC_USER_SLICE_MAX
p_user_buf	Bitstream buffer pointer
user_buf_size	AP provide bs_buf max size

[Difference]

Chip	Description
IPC	NOT supported.
NVR	Supported.

2.3.25 HD_VIDEOENC_RECV_LIST

[Description]

The video bitstream item including path information

Use this type to form an array in hd_videoenc_recv_list() to get the video bitstreams for all paths.

[Parameter]

Value	Description
path_id	path ID
user_bs	video encode user bitstream
retval	less than 0: recv bistream fail.

[Difference]

Chip	Description
IPC	NOT supported.
NVR	Supported.

3 Trouble shooting

The hd_videoenc provides a useful feature to debug, it is called debug menu.

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 “8” to open VIDEOENC debug menu

```
==============================
 VIDEOENC
------------------------------
 01 : dump status
```

```
02 : enc info
03 : rc info ON
04 : rc info OFF
------------------------------
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 videoenc

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

------------------------- VIDEOENC 0  PATH & BIND -----------------------------
in    out   state bind_src            bind_dest
0     0     START VIDEOPROC_0_OUT_0     (null)
------------------------- VIDEOENC 0  PATH CONFIG -----------------------------
in    out   max_w max_h  svc  ltr  rotate  bitrate   enc_ms
0     0     1920  1080   2    1     0     2097152   3000
------------------------- VIDEOENC 0  IN FRAME --------------------------------
in    w     h     pxlfmt  frc   dir
0     1920  1080  YUV420  1/1   ....
------------------------- VIDEOENC 0  OUT BS ----------------------------------
--- [H26x] ---
out   codec  gop  ltr_int  ltr_ref  gray  src_out  profile  level  svc  entropy
0     H265   15   0        0        0     0        MAIN     150    0    CABAC
------------------------- VIDEOENC 0  VUI -------------------------------------
out   vui_en  sar_w  sar_h  mat_c  tran_c  col_prim  vid_fmt  col_rng  time_pre
0     0       .....  .....  .....  ......  ........  .......  .......  ........
------------------------- VIDEOENC 0  DEBLOCK ---------------------------------
out   dis_ilf_idc  alpha  beta
0     0            0      0
```

```
------------------------- VIDEOENC 0  RC -------------------------------------
out   mode  bitrate  fr  I(int/min/max)  P(int/min/max)  sta  ip_w
0     CBR   2097152  30   ( 26/ 10/ 45)   ( 26/ 10/ 45)  4    0
------------------------- VIDEOENC 0  USER QP --------------------------------
out   en  map_addr
0     0   ........
------------------------- VIDEOENC 0  SLICE SPLIT ----------------------------
out   en  row_num
0     0   .......
------------------------- VIDEOENC 0  GDR ------------------------------------
out   en  period  row_num
0     0   ......  .......
------------------------- VIDEOENC 0  ROI ------------------------------------
out   qp_mode  win  qp  rect(x,y,w,h)
------------------------- VIDEOENC 0  ROW RC ---------------------------------
out   en  qp_rng  qp_step
0     1   2      1
------------------------- VIDEOENC 0  AQ -------------------------------------
out   en  i_str  p_str  max_delta  min_delta
0     0   .....  .....  .........  .........
------------------------- VIDEOENC 0  IN WORK STATUS -------------------------
in    PUSH  drop  wrn   err   PROC  drop  wrn   err   REL
0     30    0     0     0     30    0     0     0     30
------------------------- VIDEOENC 0  OUT WORK STATUS ------------------------
out   NEW   drop  wrn   err   PROC  drop  wrn   err   PUSH  drop  wrn   err
0      30   0     0     0     30    0     0     0     30    0     0     0
------------------------- VIDEOENC 0  USER WORK STATUS -----------------------
out   PULL   drop  wrn   err   REL
0     30     0     0     0     30
```

As above, the debug menu shows the path & bind information, path_config , input frame / output bitstream information.

The detail for each count value is as following,

[IN]

IN	PUSH	=>	get YUV from previous unit / user push_in
		err	(1) module is NOT start (2) for auto-bind mod, check YUV lineoffset is wrong
		wrn	(1) YUV ping-pong buffer if full (2) bitstream buffer is full, drop this YUV
		drop	(1) module is stop, drop any YUV that is not processed
	PROC	=>	prepare to encode
		err	(1) encode fail
		wrn	(1) N/A
		drop	(1) N/A
	REL	=>	release YUV

[OUT]

OUT	NEW	=>	search for bitstream buffer to put encode result
		err	(1) N/A
		wrn	(1) bitstream buffer is full, could not encode this YUV
		drop	(1) N/A
	PROC	=>	prepare to encode
		err	(1) encode fail
		wrn	(1) N/A
		drop	(1) N/A
	PUSH	=>	push encoded bitstream to pull queue (for user pull later)
		err	(1) N/A
		wrn	(1) pull queue is full
		drop	(1) N/A

[USER]

USER	PULL	=>	user call hd_videoenc_pull_out_buf() to get bitstream
		err	(1) N/A
		wrn	(1) pull fail due to timeout
		drop	(1) N/A
	REL	=>	user call hd_videoenc_release_out_buf() to release bitstream

3.1.2 enc info

Enter “2” and then enter which path to show encoder information

```
Run: 02 : enc info

Please enter which path (0~15)  =>

: 0

[ 2855.827904] [VDOENC][0] Codec = H265, RC Mode = CBR, W = 1920, H = 1080, BitRate = 2097152, Fps
= 30, Gop = 15, SVC = 0, IQP = (26, 10, 45), PQP = (26, 10, 45), Static = 4, Weight = 0, RowRc =
(1, 2, 1), SmartRoi = 0, LTR = (0, 0), DB = (0, 0, 0), VUI = (0, 1, 1, 2, 2, 2, 5, 0, 0), 3DNR Callback
= 0x7ed68f20, AQ = (0, 3, 1, 36, 6, -6, 0), Rotate = 0, GDR = (0, 0, 1), SLICE = (0, 1), QPMap =
(0, 80000000), Enc(Drop, In, Out, Re-Enc, Err) = (0, 8444, 8444, 0, 0)
```

As above, the debug menu shows the encoder settings for certain path.

3.1.3 rc info on

Enter “3” and then enter which path to START dump rate control information

```
Run: 03 : rc info ON

Please enter which path (0~15)  =>

: 0


[ 3034.766550] h265Enc_DumpRcInfo:mode CBR, frame rate 30/1, gop 15, GOP bitrate 1048576, QP (I
10/45, P 10/45), cur qp 23 (23)
[ 3035.795426] dal_h265enc_encodeone:[H265ENC][0]
[ 3035.799832] h265Enc_DumpRcInfo:mode CBR, frame rate 30/1, gop 15, GOP bitrate 1048576, QP (I
10/45, P 10/45), cur qp 24 (24)
[ 3036.828812] dal_h265enc_encodeone:[H265ENC][0]
[ 3036.833193] h265Enc_DumpRcInfo:mode CBR, frame rate 30/1, gop 15, GOP bitrate 1048576, QP (I
10/45, P 10/45), cur qp 26 (26)
[ 3037.862142] dal_h265enc_encodeone:[H265ENC][0]
[ 3037.866552] h265Enc_DumpRcInfo:mode CBR, frame rate 30/1, gop 15, GOP bitrate 1048576, QP (I
10/45, P 10/45), cur qp 24 (24)
```

As above, the debug menu START dump rate control information.

3.1.4 rc info off

Enter “4” and then enter which path to STOP dump rate control information

```
Run: 04 : rc info OFF

Please enter which path (0~15)  =>

: 0
```

As above, the debug menu STOP dump rate control information.

3.2 proc command for IPC

3.2.1 dump status

```
[dump info]
Cat /proc/hdal/venc/info
```

the result is exactly the same as 3.1.1 Dump status

```
root@NVTEVM:~$ cat /proc/hdal/venc/info
HDAL_VERSION: 00010001:00010001


------------------------- VIDEOENC 0  PATH & BIND -----------------------------
in    out   state bind_src            bind_dest
0     0     START VIDEOPROC_0_OUT_0     (null)
------------------------- VIDEOENC 0  PATH CONFIG -----------------------------
in    out   max_w max_h  svc  ltr  rotate  bitrate   enc_ms
0     0     1920  1080   2    1     0     2097152   3000
------------------------- VIDEOENC 0  IN FRAME --------------------------------
in    w     h     pxlfmt  frc   dir
0     1920  1080  YUV420  1/1   ....
------------------------- VIDEOENC 0  OUT BS ----------------------------------
--- [H26x] ---
out   codec  gop  ltr_int  ltr_ref  gray  src_out  profile  level  svc  entropy
0     H265   15   0        0        0     0        MAIN     150    0    CABAC
------------------------- VIDEOENC 0  VUI -------------------------------------
out   vui_en  sar_w  sar_h  mat_c  tran_c  col_prim  vid_fmt  col_rng  time_pre
0     0       .....  .....  .....  ......  ........  .......  .......  ........
------------------------- VIDEOENC 0  DEBLOCK ---------------------------------
out   dis_ilf_idc  alpha  beta
0     0            0      0
------------------------- VIDEOENC 0  RC --------------------------------------
out   mode  bitrate  fr  I(int/min/max)  P(int/min/max)  sta  ip_w
0     CBR   2097152  30   ( 26/ 10/ 45)   ( 26/ 10/ 45)  4    0
------------------------- VIDEOENC 0  USER QP ---------------------------------
```

```
out   en  map_addr
0     0   ........
------------------------- VIDEOENC 0  SLICE SPLIT -----------------------------
out   en  row_num
0     0   .......
------------------------- VIDEOENC 0  GDR -------------------------------------
out   en  period  row_num
0     0   ......  .......
------------------------- VIDEOENC 0  ROI -------------------------------------
out   qp_mode  win  qp  rect(x,y,w,h)
------------------------- VIDEOENC 0  ROW RC ----------------------------------
out   en  qp_rng  qp_step
0     1   2       1
------------------------- VIDEOENC 0  AQ --------------------------------------
out   en  i_str  p_str  max_delta  min_delta
0     0   .....  .....  .........  .........
------------------------- VIDEOENC 0  IN WORK STATUS --------------------------
in    PUSH  drop  wrn   err   PROC  drop  wrn   err   REL
0     30    0     0     0     30    0     0     0     30
------------------------- VIDEOENC 0  OUT WORK STATUS -------------------------
out   NEW   drop  wrn   err   PROC  drop  wrn   err   PUSH  drop  wrn   err
0     30    0     0     0     30    0     0     0     30    0     0     0
------------------------- VIDEOENC 0  USER WORK STATUS ------------------------
out   PULL   drop  wrn   err   REL
0     30     0     0     0     30
```

3.2.2 debug command

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

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

this debug command can show more debug log on console

```
root@NVTEVM:/mnt/sd$ hd_video_record
[ 4376.440341] hd_reset - begin
[ 4376.445192] hd_reset - end
HDAL_VERSION: 00010001:00010001
[ 4376.467841]
[ 4376.467841]  "vdoenc".out[0]: open begin, state=0
[ 4376.475047]  "vdoenc".out[0]: cmd OPEN
[ 4376.479852] _ISF_VdoEnc_ImgCap_Open:bIs_NMI_ImgCap_ON = 0
[ 4376.485364]  "vdoenc".out[0]: open end, state=1
[ 4376.490942]  "vdoenc".out[0]: set param(0000f000)=2
[ 4376.496720]  "vdoenc".out[0]: set param(0000f005)=3000
[ 4376.502745]  "vdoenc".out[0]: set param(0000f039)=0
[ 4376.508512]  "vdoenc".out[0]: set param(0000f004)=2
[ 4376.514284] [VDOENC][0] Set max alloc size, codec = 2, w = 1920, h = 1080, byterate = 262144,
recformat = 7, rotate = 0, svc = 2, ltr = 1, snapshot size = 0, codec size = 9581988, enc buf size
= 786432
[ 4376.532280]  "vdoenc".out[0]: set vdo-size(1920,1080) vdo-format(520C0420) vdo-dir(0)
[ 4376.540977]  "vdoenc".out[0]: set vdo-framerate(1,1)
enc_type=0
[ 4377.711187]  "vdoenc".out[0]: set param(0000f015)=0
[ 4377.717186]  "vdoenc".out[0]: set param(0000f018)=1
[ 4377.722982]  "vdoenc".out[0]: set param(0000f01c)=0
[ 4377.728742]  "vdoenc".out[0]: set param(0000f023)=0
[ 4377.734515]  "vdoenc".out[0]: set param(0000f03d)=0
[ 4377.740272]  "vdoenc".out[0]: set param(0000f03e)=0
[ 4377.748686]  "vdoenc".out[0]: set param(0000f03f)=0
[ 4377.754662]  "vdoenc".out[0]: set param(0000f040)=0
[ 4377.760429]  "vdoenc".out[0]: set param(0000f041)=0
[ 4377.767241]  "vdoenc".out[0]: set param(0000f000)=3
[ 4377.773002]  "vdoenc".out[0]: set param(0000f001)=1
[ 4377.779520]  "vdoenc".out[0]: set param(0000f005)=3000
[ 4377.785535]  "vdoenc".out[0]: set param(0000f008)=0
[ 4377.791287]  "vdoenc".out[0]: set param(0000f009)=15
[ 4377.797124]  "vdoenc".out[0]: set param(0000f00a)=7
[ 4377.802874]  "vdoenc".out[0]: set param(0000f014)=0
```

```
[ 4377.808624]  "vdoenc".out[0]: set param(0000f030)=0
[ 4377.815039]  "vdoenc".out[0]: set param(0000f035)=0
[ 4377.820793]  "vdoenc".out[0]: set param(0000f03a)=150
[ 4377.826717]  "vdoenc".out[0]: set param(0000f03b)=1
[ 4377.832467]  "vdoenc".out[0]: set param(0000f03c)=0
[ 4377.839343]
[ 4377.839343]  "vdoenc".out[0]: start begin, state=1
[ 4377.846625]  "vdoenc".out[0]: cmd RDYSYNC
[ 4377.851524]  "vdoenc".out[0]: cmd START
[ 4377.856325] [VDOENC][0] Action = 0, Codec = 3, Mode = 7, Size = (1920, 1080), Fr = 30, Trig =
1, Alloc size = (0, 3239692, 786432), Enc Buf = (3000 ms, 0 ms, 0x95575f0c, 0x95635f08, 0x95635f08,
786428), Min Size(I, P) = (393216, 262144)
[ 4377.877363] dal_h265enc_init:[H265ENC][0] Init Codec = (0x9525f000, 3239692), W = 1920, H = 1080,
Prof = 1, Br = 2097152, Fps = 30, Gop = 15, SVC = 0, IQP = (26, 10, 45), PQP = (26, 10, 45), Sta
= 4, Wei = 0, RowRc = (1, 2, 1), LTR = (0, 0), MultiT = 0, Rot = 0, FastSr = 0, ColorR = 0, Mode
= 0, SEI = 0
[ 4377.904386] h26x_open:H26X Version = 0x20161211
[ 4377.909023] _h265enc_setratecontrol:[H265ENC][0] Set CBR
[ 4377.914457]  "vdoenc".out[0]: start end, state=2
[ 4377.920980]  "vdoenc".out[0]: get param(0000f037)=329641984
[ 4377.927623]  "vdoenc".out[0]: get param(0000f038)=10384808
Enter q to exit
[ 4377.936965] dal_h265enc_encodeone:[H265ENC][0] Reset I OK, idx = 0, Frame Type = 3, Frame =
(0x95575f0c, 258106), Buf = (0x95575f0c, 0x95635f08), t = 103465566 us


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


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

3.2.3 trace command

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

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

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

3.2.4 probe command

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

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

this probe command could print per-data status

```
[ 5692.621611] "vdoenc".out[0] - NEW - new -- h=9558697c size=00000000 addr=9558697c OK
[ 5692.631780] "vdoenc".out[0] - PUSH - rel -- h=9558697c (result=0) OK
[ 5692.652093] "vdoenc".out[0] - NEW - new -- h=9558779c size=00000000 addr=9558779c OK
[ 5692.662281] "vdoenc".out[0] - PUSH - rel -- h=9558779c (result=0) OK
[ 5692.682488] "vdoenc".out[0] - NEW - new -- h=95588474 size=00000000 addr=95588474 OK
[ 5692.691369] "vdoenc".out[0] - PUSH - rel -- h=95588474 (result=0) OK
[ 5692.718160] "vdoenc".out[0] - NEW - new -- h=95589084 size=00000000 addr=95589084 OK
[ 5692.729104] "vdoenc".out[0] - PUSH - rel -- h=95589084 (result=0) OK
[ 5692.749191] "vdoenc".out[0] - NEW - new -- h=95589e88 size=00000000 addr=95589e88 OK
[ 5692.758027] "vdoenc".out[0] - PUSH - rel -- h=95589e88 (result=0) OK
[ 5692.785621] "vdoenc".out[0] - NEW - new -- h=9558a9f0 size=00000000 addr=9558a9f0 OK
[ 5692.796616] "vdoenc".out[0] - PUSH - rel -- h=9558a9f0 (result=0) OK
```

3.2.5 perf command

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

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

this perf command could print data count per second

```
[   37.463860] "vdoenc".in[0] Perf! -- (Video) 30 Frame/sec
[   38.495429] "vdoenc".in[0] Perf! -- (Video) 30 Frame/sec
[   39.495233] "vdoenc".in[0] Perf! -- (Video) 30 Frame/sec
```

3.2.6 save command

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

[ Sample ]
echo perf d0 i0 1 > /proc/hdal/venc/cmd
```

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

```
[  140.135994] save i/o begin: "vdoenc".in[0] count=1
[  140.153779] "vdoenc".in[0] Save -- h=94f5bfc0 t=00000000099334e3 (YUV: 1920x1080.520c0420
94f5c000 95156400 1920 1920)
[  141.330243] "vdoenc".in[0] Save -- //mnt//sd//isf_
vdoenc_in[0]_520c0420_1920_1080_1920_c208.vdo ok

[  141.340271] save port end
```

3.3 OSG proc command

3.3.1 dump status

cat /proc/hdal/osg/info to show the status of OSG and focus on VIDEOENC

```
------------------------- VIDEOENC 0 BUFFER ----------------------------------
pid     type    fmt     w       h       addr        size        draw
0       pp      4444    1000    200     13a55000    400000      1
0       pp              0       0       13ab6a80    400000      0


------------------------- VIDEOENC 0 STAMP -----------------------------------
pid     start   x       y       alpha   cken    ckval   layer   rgn
0       1       0       0       255     0       0       0       0
------------------------- VIDEOENC 0 MASK ------------------------------------
pid     start   x       y       w       h       solid   thick   color           alpha
0       1       500     0       100     120     1       0       ff0000          255
```

As above, the debug menu shows buffer, stamp and mask configuration of all videoenc's OSGs. Most values are simply from hd_videoenc_set and self-explained. pid serves as an internal serial number and is mainly used to associate stamp and buffer information. start reflects if hd_videoenc_start/hd_videoenc_stop had been applied to that OSG.

3.3.2 change status

OSG attr can be changed through debug menu while buffer and image can't because buffer and image typically require a buffer which can't be created by shell console. To change an OSG's attr, echo *data* > /proc/hdal/osg/cmd. Below are the format of *data*:

1. For stamp: phase osg pid io start x y alpha cken ckval layer region
   example: to set the 5th stamp of output id 3 of videoenc to position[1024,512] and layer(1) region(8), run "echo videoenc stamp 5 3 1 1024 512 255 0 0 1 8"
2. For mask : phase osg pid io start x y w h solid thick color alpha
   example: to set the 5th green mask of output id 3 of videoenc to position[1024,512] and size 256x128, run "echo videoenc mask 5 3 1 1024 512 256 128 1 0 0x0FF00 255"

3.4 Debug menu for NVR

3.4.1 dump status

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

```
Run: 01 : dump status
------------------------- VIDEOENC 0  PATH & BIND -----------------------------
in     out    state   bind_src             bind_dest
0      0      START   VIDEOCAP_0_OUT_1      -
0      1      START   VIDEOCAP_1_OUT_1      -
0      2      START   VIDEOCAP_2_OUT_1      -
0      3      START   VIDEOCAP_3_OUT_1      -
0      4      START   VIDEOCAP_0_OUT_2      -
0      5      START   VIDEOCAP_1_OUT_2      -
0      6      START   VIDEOCAP_2_OUT_2      -
0      7      START   VIDEOCAP_3_OUT_2      -
------------------------- VIDEOENC 0  IN FRAME ---------------------------
in     w      h      pxlfmt
0      1920    1080    YUV420
1      1920    1080    YUV420
2      1920    1080    YUV420
3      1920    1080    YUV420
4      960     480     YUV420_NVX3
5      960     480     YUV420_NVX3
6      960     480     YUV420_NVX3
7      960     480     YUV420_NVX3
------------------------- VIDEOENC 0  OUT FRAME ----------------------------
in     codec   gop    profile svc    level   entropy
0      -       -      0001    -      -
1      -       -      0001    -      -
2      -       -      0001    -      -
3      -       -      0001    -      -
4      -       -      0001    -      -
5      -       -      0001    -      -
6      -       -      0001    -      -
7      -       -      0001    -      -
------------------------- VIDEOENC 0  RC -----------------------------
in     mode    base    incr    bitrate
0      CBR     0025    0001    4096Kbps
1      CBR     0025    0001    4096Kbps
2      CBR     0025    0001    4096Kbps
3      CBR     0025    0001    4096Kbps
4      CBR     0025    0001    1024Kbps
5      CBR     0025    0001    1024Kbps
6      CBR     0025    0001    1024Kbps
7      CBR     0025    0001    1024Kbps
```

3.5 proc command for NVR

3.5.1 dump status

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

the result is similar to 3.1.1 Dump status

```
root@NVTEVM:/$ cat /proc/videograph/hdal_setting
========================= VIDEOCAP 0  SYSCAP ==========================
w      h      fps    scaling
1920   1080   25     4088
------------------------- VIDEOCAP 0  PATH & BIND ----------------------------
in     out    state  bind_src             bind_dest
0      0      START  -                   VIDEOENC_0_IN_0
------------------------- VIDEOCAP 0  OUT FRAME ----------------------------
out    w      h      pxlfmt
0      1920   1080   YUV420_NVX3
------------------------- VIDEOCAP 0  PATH POOL -----------------------------
out    pool   ddr_id  count   max_count
0      0      0      4.0     4.0
------------------------- VIDEOENC 0  PATH & BIND ----------------------------
in     out    state  bind_src             bind_dest
0      0      START  VIDEOCAP_0_OUT_0       -
------------------------- VIDEOENC 0  IN FRAME -----------------------------
in     w      h      pxlfmt
0      1920   1080   YUV420_NVX3
------------------------- VIDEOENC 0  OUT FRAME -----------------------------
in     codec  gop    profile svc    level   entropy
0      -      -      1992728576   -       CABAC
------------------------- VIDEOENC 0  RC -----------------------------
in     mode   base   incr   bitrate
0      CBR    0030   0001   2048Kbps
```

4 Sample Codes

4.1 hd_videoenc_only (IPC sample)

The **hd_videoenc_only** demonstrates how to use the single trigger operation to process the input image.

```
/* Allocate common buffer */
   mem_cfg.pool_info[0].type     = HD_COMMON_MEM_COMMON_POOL;
   mem_cfg.pool_info[0].blk_size = YUV_BLK_SIZE;
   mem_cfg.pool_info[0].blk_cnt  = MAX_YUV_BLK_CNT;
   mem_cfg.pool_info[0].ddr_id   = DDR_ID0;
   ret = hd_common_mem_init(&mem_cfg);

/* set enc path configuration */
   video_path_config.max_mem.codec_type = HD_CODEC_TYPE_H265;
   video_path_config.max_mem.max_dim.w  = 1920;
   video_path_config.max_mem.max_dim.h  = 1080;
   video_path_config.max_mem.bitrate    = 2 * 1024 * 1024; // 2 Mb/s = 512 MB/s
   video_path_config.max_mem.enc_buf_ms = 3000;
   video_path_config.max_mem.svc_layer  = HD_SVC_4X;
   video_path_config.max_mem.ltr       = TRUE;
   video_path_config.max_mem.rotate     = FALSE;
   video_path_config.max_mem.source_output   = FALSE;
   video_path_config.isp_id           = 0;
   ret = hd_videoenc_set(video_enc_path0, HD_VIDEOENC_PARAM_PATH_CONFIG, &video_path_config);

/* set enc parameters */
   //--- HD_VIDEOENC_PARAM_IN ---
```

```
video_in_param.dir     = HD_VIDEO_DIR_NONE;
video_in_param.pxl_fmt = HD_VIDEO_PXLFMT_YUV420;
video_in_param.dim.w   = main_dim.w;
video_in_param.dim.h   = main_dim.h;
ret = hd_videoenc_set(video_enc_path0, HD_VIDEOENC_PARAM_IN, &video_in_param);

//--- HD_VIDEOENC_PARAM_OUT_ENC_PARAM ---
video_out_param.codec_type        = HD_CODEC_TYPE_H264;
video_out_param.h26x.profile      = HD_H264E_HIGH_PROFILE;
video_out_param.h26x.level_idc    = HD_H264E_LEVEL_5_1;
video_out_param.h26x.gop_num      = 15;
video_out_param.h26x.ltr_interval = 0;
video_out_param.h26x.ltr_pre_ref  = 0;
video_out_param.h26x.gray_en      = 0;
video_out_param.h26x.source_output = 0;
video_out_param.h26x.svc_layer    = HD_SVC_DISABLE;
video_out_param.h26x.entropy_mode = HD_H264E_CABAC_CODING;
ret = hd_videoenc_set(video_enc_path0, HD_VIDEOENC_PARAM_OUT_ENC_PARAM, &video_out_param);

//--- HD_VIDEOENC_PARAM_OUT_RATE_CONTROL ---
rc_param.rc_mode            = HD_RC_MODE_CBR;
rc_param.cbr.bitrate        = 2 * 1024 * 1024;
rc_param.cbr.frame_rate_base = 30;
rc_param.cbr.frame_rate_incr = 1;
rc_param.cbr.init_i_qp      = 26;
rc_param.cbr.min_i_qp       = 10;
rc_param.cbr.max_i_qp       = 45;
rc_param.cbr.init_p_qp      = 26;
rc_param.cbr.min_p_qp       = 10;
rc_param.cbr.max_p_qp       = 45;
rc_param.cbr.static_time    = 4;
rc_param.cbr.ip_weight      = 0;
ret = hd_videoenc_set(video_enc_path0, HD_VIDEOENC_PARAM_OUT_RATE_CONTROL, &rc_param);

/* 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 = (UINT32)hd_common_mem_mmap(HD_COMMON_MEM_MEM_TYPE_CACHE, pa, blk_size);
fread((void *)va, 1, yuv_size, fd_yuv);
```

```
video_frame.sign       = MAKEFOURCC('V','F','R','M');
video_frame.p_next     = NULL;
video_frame.ddr_id     = ddr_id;
video_frame.pxlfmt     = HD_VIDEO_PXLFMT_YUV420;
video_frame.dim.w      = VDO_SIZE_W;
video_frame.dim.h      = VDO_SIZE_H;
video_frame.count      = 0;
video_frame.timestamp   = 0;
video_frame.loff[0]    = VDO_SIZE_W; // Y
video_frame.loff[1]    = VDO_SIZE_W; // UV
video_frame.phy_addr[0] = pa;                          // Y
video_frame.phy_addr[1] = pa + VDO_SIZE_W*VDO_SIZE_H;  // UV pack
video_frame.blk        = blk;
ret = hd_videoenc_push_in_buf(video_enc_path0, &video_frame, NULL, 0);
//--- release buffer ---
hd_common_mem_release_block(blk);
hd_common_mem_munmap((void *)va, blk_size);

/* pull out buffer */
ret = hd_videoenc_pull_out_buf(video_enc_path0, &data_pull, -1);
if (ret == HD_OK) {
     hd_videoenc_get(video_enc_path0, HD_VIDEOENC_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))
     for (j=0; j< data_pull.pack_num; j++) {
          va   = PHY2VIRT_MAIN(data_pull.video_pack[j].phy_addr);
          size = data_pull.video_pack[j].size;
          fwrite(va, 1, size, fd_bs);
     }
     //--- release buffer ---
```

```
        ret = hd_videoenc_release_out_buf(video_enc_path0, &data_pull);

        hd_common_mem_munmap((void *)vir_addr_main, phy_buf_main.buf_info.buf_size);

    }
```

4.2 user_videoenc (NVR sample)

The **user_videoenc** demonstrates how to use the single trigger operation to process the input image.

```
typedef struct {
   union {
        HD_VIDEO_FRAME frame;
        HD_VIDEOENC_BS bs;
   };
   UINT64 pool;
   int size;
   HD_COMMON_MEM_DDR_ID ddr_id;
   HD_COMMON_MEM_VB_BLK blk;
   void *va;
   UINT32 pa;
} app_buffer_t;

typedef struct _USER_VIDEOENC {
   app_buffer_t enc_in_buffer[MAX_IN_FRAME_COUNT];
   int mem_fd;
   HD_PATH_ID videoenc_path_id;
} USER_VIDEOENC;

void save_output(char *filename, void *data, int size)
{
   FILE *pfile;
   if ((pfile = fopen(filename, "wb")) == NULL) {
        printf("[ERROR] Open File %s failed!!\n", filename);
        exit(1);
   }
   fwrite(data, 1, size, pfile);
   fclose(pfile);
   printf("Write file: %s\n", filename);
}

int read_input(char *filename, void *buffer, int max_buf_size, int read_size)
{
   FILE *pfile;
   INT ret;
   if ((pfile = fopen(filename, "rb")) == NULL) {
        printf("[ERROR] Open File %s failed!!\n", filename);
        exit(1);
   }
```

```
    //if read_size is zero, read whole file.
    if (read_size == 0) {
        fseek(pfile, 0, SEEK_END);
        read_size = ftell(pfile);
          fseek(pfile, 0, SEEK_SET);
    }
    if (read_size > max_buf_size)
          read_size = max_buf_size;

    if (read_size > 0) {
          ret = fread(buffer, 1, read_size, pfile);
    }

    fclose(pfile);
    return ret;
}

HD_RESULT allocate_buffer(UINT64 pool, int size, HD_COMMON_MEM_DDR_ID ddr_id, app_buffer_t
*p_app_buf)
{
    HD_RESULT ret = HD_OK;
    HD_COMMON_MEM_VB_BLK blk;
    UINT32 pa, sign;
    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 = HD_ERR_NG;
          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 = HD_ERR_NG;
          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->pa = pa;

    sign = p_app_buf->frame.sign;
    if (sign == MAKEFOURCC('V','F','R','M')) {
          p_app_buf->frame.phy_addr[0] = pa;
    } else {
          p_app_buf->bs.video_pack[0].phy_addr = pa;
    }
```

```
exit:
   return ret;
}

HD_RESULT free_buffer(app_buffer_t *p_app_buf)
{
   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");
        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");
        goto exit;
   }
exit:
   return hd_ret;
}

HD_RESULT prepare_buffers(USER_VIDEOENC *p_usr_videoenc)
{
   INT i, x, y, raw_frame_size;
   UINT16 *pixel;
   HD_VIDEO_FRAME *p_frame;
   HD_RESULT ret = HD_OK;

   for (i = 0; i < MAX_IN_FRAME_COUNT; i++) {
        p_frame = &p_usr_videoenc->enc_in_buffer[i].frame;
        p_frame->sign = MAKEFOURCC('V','F','R','M');
        p_frame->ddr_id = 0;
        p_frame->dim.w = RAW_FRAME_WIDTH;
        p_frame->dim.h = RAW_FRAME_HEIGHT;
        p_frame->pxlfmt = HD_VIDEO_PXLFMT_YUV420;
        raw_frame_size = p_frame->dim.w * p_frame->dim.h * 2;
        ret = allocate_buffer(HD_COMMON_MEM_ENC_CAP_OUT_POOL, raw_frame_size, p_frame->ddr_id,
&p_usr_videoenc->enc_in_buffer[i]);
        if (ret != 0) {
             goto exit;
        }

        read_input(RAW_FRAME_FILE, p_usr_videoenc->enc_in_buffer[i].va, FRAME_BUF_SIZE, 0);

        // Set moving black block
        for (y = 300; y < 330; y ++)
        for (x = (i * 10 + 300); x < (i * 10 + 330); x ++) {
             pixel = (unsigned short *) (p_usr_videoenc->enc_in_buffer[i].va + ((y * p_frame->dim.w)
+ x) * 2);
             *pixel = 0x1080;
        }
   }
exit:
   return ret;
}
```

```
HD_RESULT return_buffers(USER_VIDEOENC *p_usr_videoenc)
{
   INT i;
   HD_RESULT ret = HD_OK;

   for (i = 0; i < MAX_IN_FRAME_COUNT; i++) {
        ret = free_buffer(&p_usr_videoenc->enc_in_buffer[i]);
        if (ret != HD_OK) {
             printf("free_buffer fail\n");
             break;
        }
   }
   return ret;
}


HD_RESULT set_param(USER_VIDEOENC *p_usr_videoenc)
{
   HD_RESULT ret = HD_OK;
   HD_VIDEOENC_IN video_in_param;
   HD_VIDEOENC_OUT enc_param;
   HD_H26XENC_RATE_CONTROL rc_param;

   //set main stream param
   video_in_param.dim.w = RAW_FRAME_WIDTH;
   video_in_param.dim.h = RAW_FRAME_HEIGHT;

   ret = hd_videoenc_set(p_usr_videoenc->videoenc_path_id, HD_VIDEOENC_PARAM_IN,
&video_in_param);
   if (ret != HD_OK) {
        printf("hd_videoenc_set(HD_VIDEOENC_PARAM_IN) fail\n");
        goto exit;
   }

   ret = hd_videoenc_get(p_usr_videoenc->videoenc_path_id, HD_VIDEOENC_PARAM_OUT_ENC_PARAM,
&enc_param);
   if (ret != HD_OK) {
        printf("hd_videoenc_get(HD_VIDEOENC_PARAM_OUT_ENC_PARAM) fail\n");
        goto exit;
   }
   enc_param.codec_type = HD_CODEC_TYPE_H264;
   enc_param.h26x.gop_num = 60;
   ret = hd_videoenc_set(p_usr_videoenc->videoenc_path_id, HD_VIDEOENC_PARAM_OUT_ENC_PARAM,
&enc_param);
   if (ret != HD_OK) {
        printf("hd_videoenc_set(HD_VIDEOENC_PARAM_OUT_ENC_PARAM) fail\n");
        goto exit;
   }

   ret = hd_videoenc_get(p_usr_videoenc->videoenc_path_id, HD_VIDEOENC_PARAM_OUT_RATE_CONTROL,
&rc_param);
   if (ret != HD_OK) {
        printf("hd_videoenc_get(HD_VIDEOENC_PARAM_OUT_RATE_CONTROL) fail\n");
        goto exit;
```

```
    }
    rc_param.rc_mode = HD_RC_MODE_CBR;
    rc_param.cbr.frame_rate_base = 30;   //fps = 30/1 = 30
    rc_param.cbr.frame_rate_incr = 1;
    rc_param.cbr.bitrate = 2048 * 1024;
    ret = hd_videoenc_set(p_usr_videoenc->videoenc_path_id, HD_VIDEOENC_PARAM_OUT_RATE_CONTROL,
&rc_param);
    if (ret != HD_OK) {
        printf("hd_videoenc_set(HD_VIDEOENC_PARAM_OUT_RATE_CONTROL) fail\n");
        goto exit;
    }
exit:
    return ret;
}

int encode_the_buffer(USER_VIDEOENC *p_usr_videoenc)
{
    INT i, pattern_idx, bs_max_size;
    FILE *bs_file;
    HD_VIDEOENC_BS video_bitstream;
    app_buffer_t bs_out_buffer;
    HD_RESULT ret = HD_OK;

    //open files for writing
    if ((bs_file = fopen(OUT_BS_FILE, "wb")) == NULL) {
        printf("[ERROR] Open File %s failed!!\n", OUT_BS_FILE);
        exit(1);
    }

    //prepare bitstream buffer
    bs_max_size = BS_BUF_SIZE;
    bs_out_buffer.bs.sign = MAKEFOURCC('V','S','T','M');
    bs_out_buffer.bs.video_pack[0].size = bs_max_size;
    bs_out_buffer.bs.ddr_id = 0;
    ret = allocate_buffer(HD_COMMON_MEM_ENC_CAP_OUT_POOL, bs_max_size, bs_out_buffer.bs.ddr_id,
&bs_out_buffer);
    if (ret != 0) {
        printf("allocate_buffer fail\n");
        goto exit;
    }

    //encode frames for LOOP_COUNT times
    pattern_idx = 0;
    for (i = 0; i < LOOP_COUNT; i++) {
        bs_out_buffer.bs.video_pack[0].size = bs_max_size;

        //trigger it
        ret = hd_videoenc_push_in_buf(p_usr_videoenc->videoenc_path_id,
&p_usr_videoenc->enc_in_buffer[pattern_idx++].frame, &(bs_out_buffer.bs), 500);
        if (ret != HD_OK) {
            printf("hd_videoenc_push_in_buf fail\n");
            goto exit;
        }

        //get the result
```

```
        ret = hd_videoenc_pull_out_buf(p_usr_videoenc->videoenc_path_id, &video_bitstream, 500);
        if (ret != HD_OK) {
             printf("hd_videoenc_pull_out_buf fail\n");
        } else {
             INT bs_size = 0, pack_num, pa_offset;
             for (pack_num = 0; pack_num < video_bitstream.pack_num; pack_num++) {
                  pa_offset = video_bitstream.video_pack[pack_num].phy_addr - bs_out_buffer.pa;
                  bs_size += fwrite(bs_out_buffer.va+pa_offset, 1,
video_bitstream.video_pack[pack_num].size, bs_file);
             }
             printf(" Encode output size: %d\n", bs_size);
        }
        if (pattern_idx >= MAX_IN_FRAME_COUNT)
             pattern_idx = 0;
   }

   //free bitstream buffer
   ret = free_buffer(&bs_out_buffer);
   if (ret != HD_OK) {
        printf("free_buffer fail\n");
   }

exit:
   if (bs_file)
        fclose(bs_file);

   return ret;
}


int main(int argc, char *argv[])
{
   HD_RESULT ret = HD_OK;
   USER_VIDEOENC usr_videoenc;

   memset(&usr_videoenc, 0, sizeof(usr_videoenc));
   usr_videoenc.mem_fd = -1;

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

   //init hdal and open modules
   ret = hd_common_init(1);
   if (ret != HD_OK) {
        printf("hd_common_init fail\n");
        goto exit;
   }
   ret = hd_videoenc_init();
   if (ret != HD_OK) {
        printf("hd_videoenc_init fail\n");
        goto exit;
   }
   ret = hd_videoenc_open(HD_VIDEOENC_0_IN_0, HD_VIDEOENC_0_OUT_0,
```

```
&usr_videoenc.videoenc_path_id);
   if (ret != HD_OK) {
        printf("hd_videoenc_open fail\n");
        goto exit;
   }

   //setup encoder parameters
   ret = set_param(&usr_videoenc);
   if (ret < 0) {
        printf("set_param fail\n");
        goto exit;
   }

   //generate source frame for encoding
   ret = prepare_buffers(&usr_videoenc);
   if (ret < 0) {
        printf("prepare_buffers fail\n");
        goto exit;
   }

   //encode the frames
   ret = encode_the_buffer(&usr_videoenc);
   if (ret < 0) {
        printf("encode_the_buffer fail\n");
        goto exit;
   }

   //release source frames
   ret = return_buffers(&usr_videoenc);
   if (ret < 0) {
        printf("return_buffers fail\n");
        goto exit;
   }

exit:
   //close modules, release resources and exit hdal
   ret = hd_videoenc_close(usr_videoenc.videoenc_path_id);
   if (ret != HD_OK) {
        printf("hd_videoenc_close fail\n");
        return -1;
   }
   ret = hd_videoenc_uninit();
   if (ret != HD_OK) {
        printf("hd_videoenc_uninit fail\n");
        return -1;
   }
   ret = hd_common_uninit();
   if (ret != HD_OK) {
        printf("hd_common_uninit fail\n");
        return -1;
   }
   if (usr_videoenc.mem_fd != -1) {
        close(usr_videoenc.mem_fd);
   }
   return 0;
```

5 Q&A

1. What is the differnece from HD_VIDEOENC_PARAM_PATH_CONFIG.max_mem and HD_VIDEOENC_PARAM_OUT_ENC_PARAM?
   Answer:
   - HD_VIDEOENC_PARAM_PATH_CONFIG.max_mem sets "Maximum requirement", while HD_VIDEOENC_PARAM_OUT_ENC_PARAM sets current encode settings.

   - For example, you may set PATH_CONFIG.max_mem.max_dim as 1920x1080. Then you can set HD_VIDEOENC_IN.dim as any resolution smaller than 1920x1080.
   - Another example, you may set PATH_CONFIG.max_mem.ltr as TRUE. Then you can set HD_VIDEOENC_PARAM_OUT_ENC_PARAM.h26x.ltr_interval > 0 to enable LTR, or, you can set ltr_interval = 0 to disable LTR.
   - But if you set PATH_CONFIG.max_mem.ltr as FALSE, then you can only set HD_VIDEOEC_PARAM_0UT_ENC_PARAM.h26x.ltr_interval = 0.

   - PATH_CONFIG set maximum requirement, if you don't need certain function, you should not set the function as TRUE because this will alloc extra memory.
     - svc_layer, ltr, source_output => each of those function will alloc extra YUV420 buffer to support function.

2. What control flow should I apply for each param for hd_videoenc_set()?
   - stop -> close -> open -> set -> start
     - HD_VIDEOENC_PARAM_PATH_CONFIG
       - because this will involve memory re-alloc.
   - stop -> set -> start
     - HD_VIDEOENC_PARAM_IN
       - If videoproc is binding to videoenc, and videoproc.out setting=0 (auto sync videoenc settings), you have to call videoproc_start to sync videoenc new settings.
     - HD_VIDEOENC_PARAM_OUT_ENC_PARAM
     - HD_VIDEOENC_PARAM_OUT_VUI
     - HD_VIDEOENC_PARAM_OUT_DEBLOCK

- set -> start
  - HD_VIDEOENC_PARAM_OUT_RATE_CONTROL
  - HD_VIDEOENC_PARAM_OUT_USR_QP
  - HD_VIDEOENC_PARAM_OUT_SLICE_SPLIT
  - HD_VIDEOENC_PARAM_OUT_ENC_GDR
  - HD_VIDEOENC_PARAM_OUT_ROI
  - HD_VIDEOENC_PARAM_OUT_ROW_RC
  - HD_VIDEOENC_PARAM_OUT_AQ
  - HD_VIDEOENC_PARAM_OUT_REQUEST_IFRAME
  - HD_VIDEOENC_PARAM_OUT_TRIG_SNAPSHOT
  - HD_VIDEOENC_PARAM_IN_FRC

3. Is there anything I should take care for codec JPG with HD_VIDEOENC_PARAM_IN.dir rotate90/270?
   - Yes : because JPG rotate depends on gximage library
     - You have to alloc extra YUV buffer. On the other word, hd_common_mem_init() for YUV buffer, set blk_cnt from 3 to 4
     - Also, the YUV buffer width/height MUST be ALIGN_CEIL_16.
4. Can an osd image shared between videoprocess, videoenc and videoout?
   - Yes : Just set the same p_addr value of HD_OSG_STAMP_BUF to videoprocess, videoenc and videoout pathid.
   - Subsequent upate through any pathid with automatically reflect on other pathid
5. Why an osd/mask disappear without error or warning message?
   - In any macro block, only an osd will be rendered in a layer.
   - Since there are only two layers, this means at most two images can be rendered in a macro block
   - In h264, a macro block is 16x16
   - In h265, a macro block is 64x64
   - Osd on layer 0 will be rendered above layer 1
6. Any constrain on osd image and buffer?
   - Width of an image is best to be 4 aligned
   - Height of an image is best to be 2 aligned
   - Buffer address is best to be 128 aligned
7. Ex stamp and mask consume much system resource. Any advice on the number?
   - It's hard to say. System with heavy loading has less margin for ex stamp and mask
   - For a 30fps system, 4 ex stamps or 4 ex masks are safe

8. How to set alpha for an osd image
   - This is conditional to image formats
   - Alpha of argb4444 is completely determined by pixel's 4-bits alpha value
   - Alpha of argb1555 is determined by pixel's 1-bit alpha value and the alpha field of HD_OSG_STAMP_ATTR. If pixel's alpha value is 0, bits 3 ~ bits 0 of HD_OSG_STAMP_ATTR's alpha field determines transparency. If pixel's alpha value is 1, bits 7 ~ bits 4 of HD_OSG_STAMP_ATTR's alpha field determines transparency
   - Alpha of rgb565 is completely determined by the alpha field of HD_OSG_STAMP_ATTR.
9. If an osd is configured with ping pong buffer. Is it possible to directly draw the free buffer?
   - Yes
   - Use HD_VIDEOENC_PARAM_IN_STAMP_IMG to get the physical address of the free buffer
   - Draw this free buffer
   - Apply the drawing by set HD_VIDEOENC_PARAM_IN_STAMP_IMG. The p_addr field of HD_OSG_STAMP_IMG should be the same physical address.