Novatek HDAL Design Specification - hd_videocapture

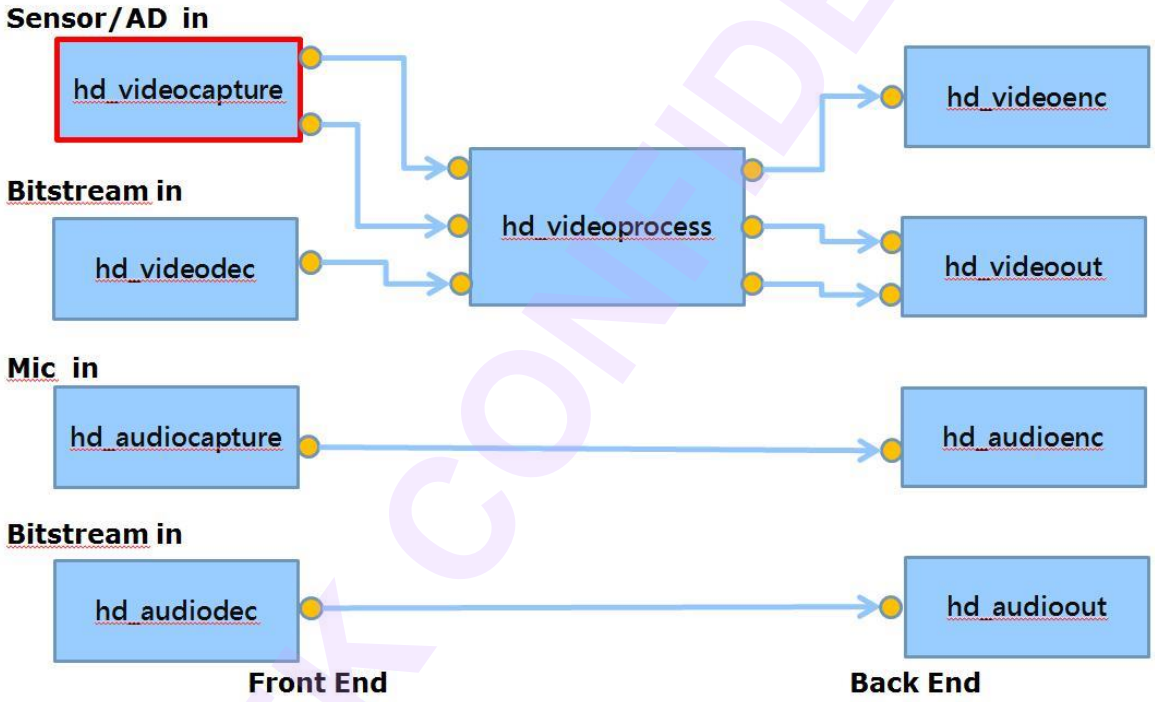
Table of Content

Difference Table (for IPC only)

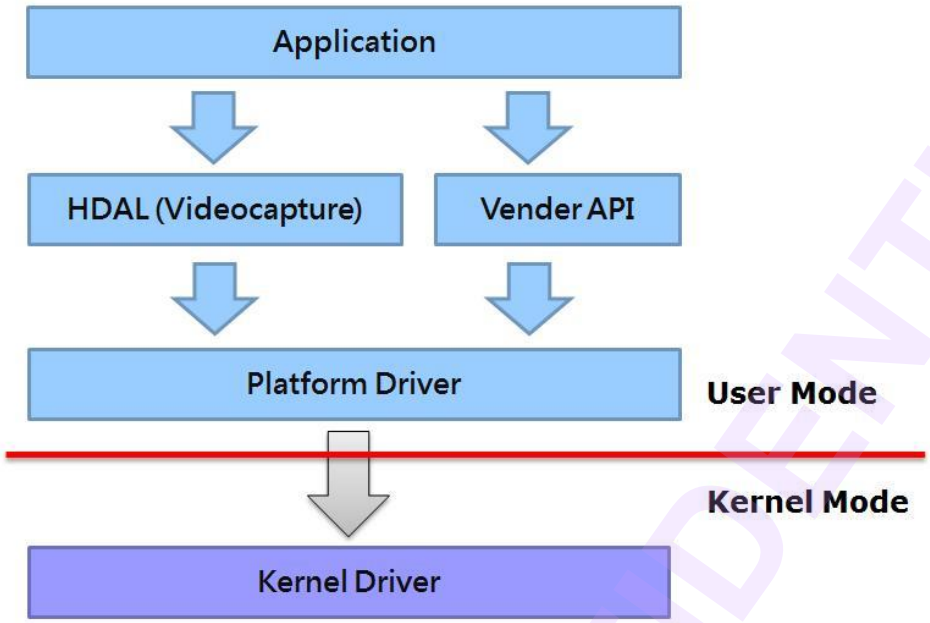
Item	NT9668X	NT9852X
MAX DEVICE NUM	8	3
HDR Sensor	Support at most two HDR sensors, using VCAP0~3.	Support at most one HDR sensor, using VCAP0 and VCAP1.
Output Directly to HD_VIDEOPRC	Not supported.	Yes, refer to HD_VIDEOCAP_PATH_CONFIG. out_func.
Output Scale Down	Supported, referring to HD_VIDEOCAP_OUT. dim.	Not supported.

1 Introduction

The major purpose of hd_videocapture.c is to get video signal from Sensor or Camera, and controls the video capture engine to process image and scaling, and then return the YUV frame data which can be used for displaying and recording. This document will talk about the red block in the following diagram. The device driver is not the main point in this document.



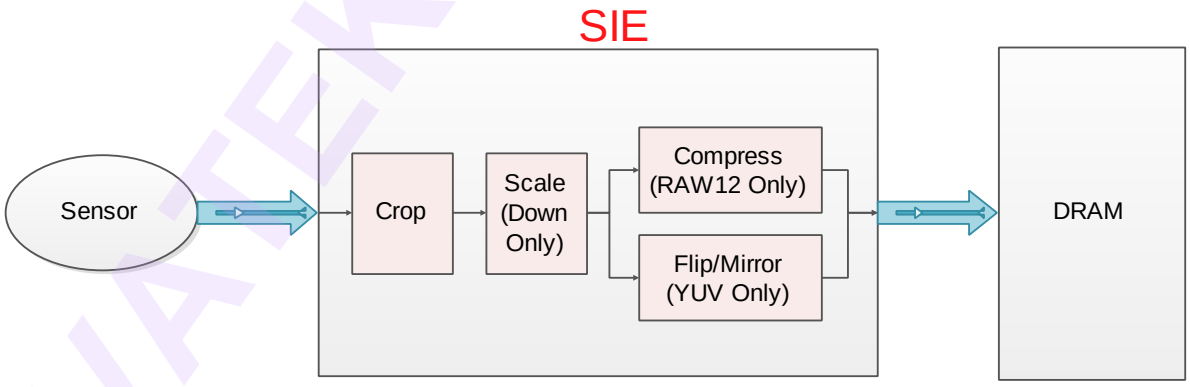
Module diagram is shown as below:



1.1 Block Diagram

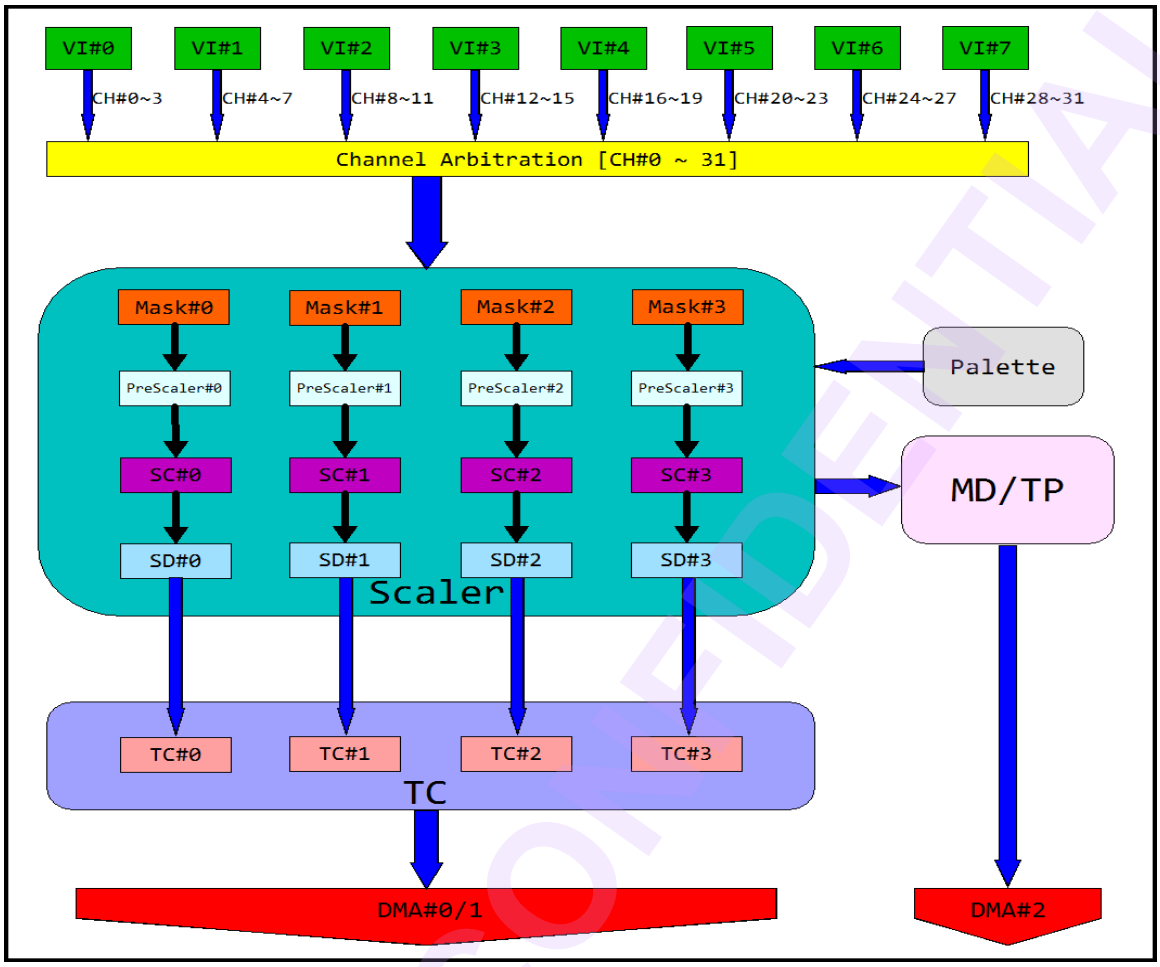
1.1.1 IPC Block Diagram

The block diagram of video capture engine is shown as below:



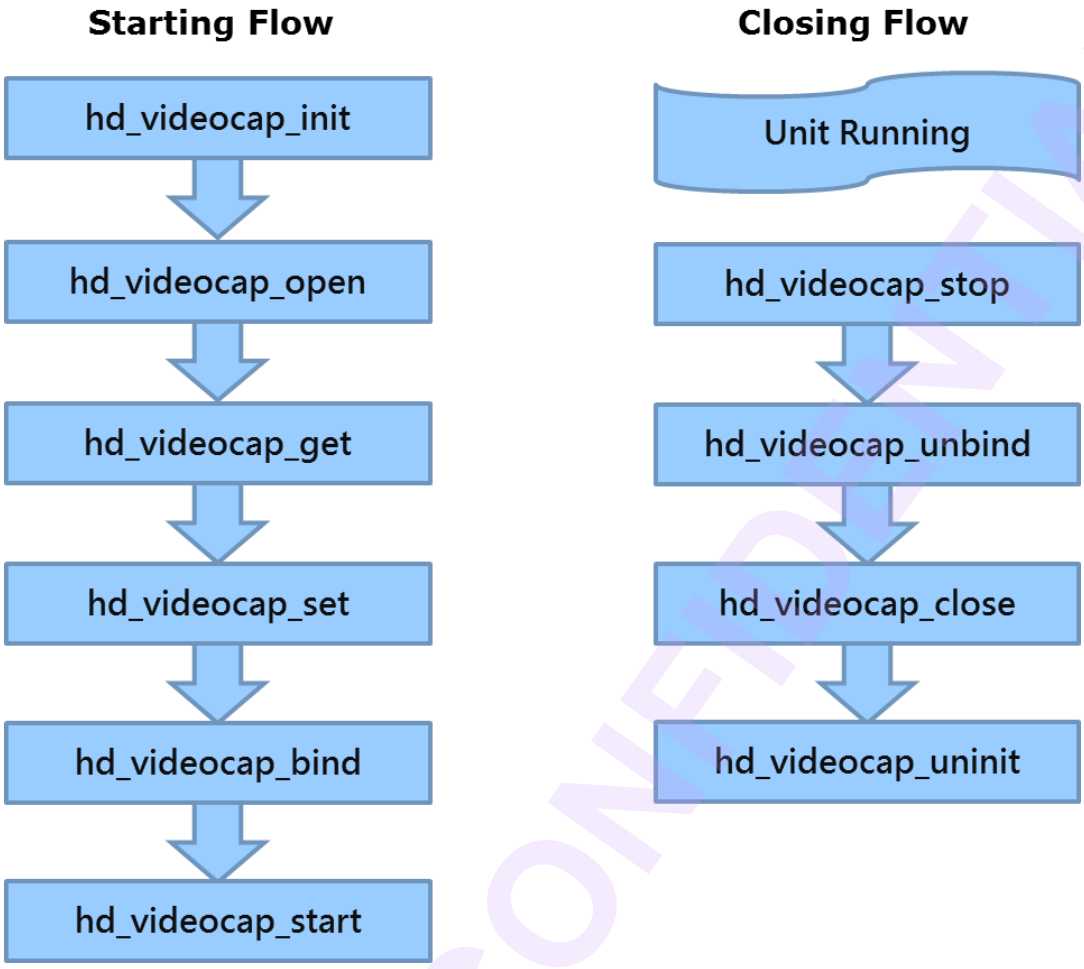
1.1.2 NVR Block Diagram

The block diagram of video capture engine 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.

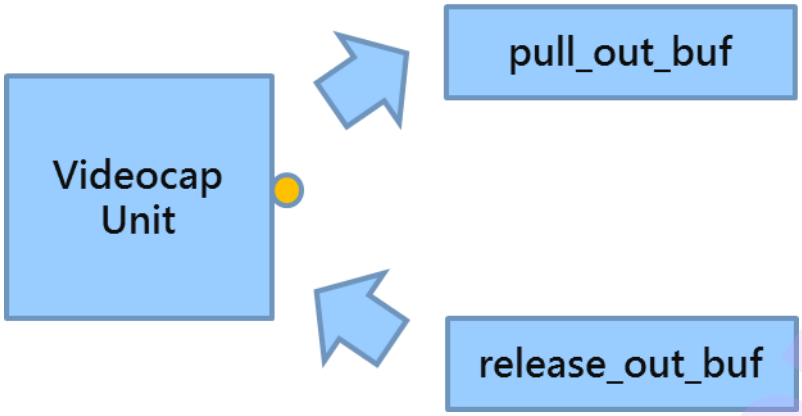


Now, below section in this chapter is mainly about what things to do in those functions above.

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; the sequence for output port is pull and then releases it. The flow is shown as below.

Output



2 Function and data structure definition

2.1 General function

2.1.1 hd_videocap_init

[Description]
Initialize the unit

[Syntax]
HD_RESULT hd_videocap_init(VOID);

[Parameter]

Value	Description
VOID	Not available

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.2 hd_videocap_open

[Description]
Open the unit

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

[Parameter]

Value	Description

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

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.3 hd_videocap_get

[Description]
Get parameters from unit by path id

[Syntax]
HD_RESULT hd_videocap_get(HD_PATH_ID path_id, HD_VIDEOCAP_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_videocap_set

[Description]
Set parameters to unit by path id

[Syntax]
HD_RESULT hd_videocap_set(HD_PATH_ID path_id, HD_VIDEOCAP_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_videocap_drv_get (for NVR only)

[Description]
Get parameters from unit HW config

[Syntax]
HD_RESULT hd_videocap_get(HD_VIDEOCAP_DRV_PARAM_ID id, VOID* p_param)

[Parameter]

Value	Description
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

2.1.6 hd_videocap_drv_set (for NVR only)

[Description]
Set parameters to unit HW config

[Syntax]
HD_RESULT hd_videocap_drv_set(HD_VIDEOCAP_DRV_PARAM_ID id, VOID* p_param)

[Parameter]

Value	Description
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

2.1.7 hd_videocap_bind

[Description]
Bind this unit with destination unit

[Syntax]
HD_RESULT hd_videocap_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

2.1.8 hd_videocap_start

[Description]
Start the unit

[Syntax]
HD_RESULT hd_videocap_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

2.1.9 hd_videocap_stop

[Description]
Stop the unit

[Syntax]
HD_RESULT hd_videocap_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

2.1.10 hd_videocap_unbind

[Description]
Unbind the unit

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

[Parameter]

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

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.11 hd_videocap_close

[Description]
Close the unit

[Syntax]
HD_RESULT hd_videocap_close(HD_PATH_ID path_id)

[Parameter]

Value	Description
path_id	pointer of the path id

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.12 hd_videocap_uninit

[Description]
Uninitialize the unit

[Syntax]
HD_RESULT hd_videocap_uninit(VOID);

[Parameter]

Value	Description
VOID	Not available

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.13 hd_videocap_pull_out_buf

[Description]
Pull the video buffer from unit

[Syntax]
HD_RESULT hd_videocap_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

2.1.14 hd_videocap_release_out_buf

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

[Syntax]
HD_RESULT hd_videocap_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

2.2 Multi-list function (for NVR only)

Multi-list operation is used to multiple start/stop funciton simultaneously, it is very efficiency in the multi channels case.

2.2.1 hd_videocap_start_list

[Description]

Start to send multi video data to the next unit

[Syntax]

HD_RESULT hd_videocap_start_list(HD_PATH_ID *path_id, UINT num);

[Parameter]

Value	Description
path_id	the path id
Num	number of path id

[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_videocap_stop_list

[Description]

Stop to send multi video data to the next unit

[Syntax]

HD_RESULT hd_videocap_stop_list(HD_PATH_ID *path_id, UINT num);

[Parameter]

Value	Description
path_id	the path id
num	number of path id

[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 of general configure

The function hd_videocap_get and hd_videocap_set provides the following parameter IDs:

HD_VIDEOCAP_PARAM_DEVCOUNT

NVR/IPC. NVR/IPC. support get with ctrl path, using HD_DEVCOUNT struct.

HD_VIDEOCAP_PARAM_SYSCAPS

NVR/IPC. support get with ctrl path, using HD_VIDEOCAP_SYSCAPS struct (system capabilitiy)

HD_VIDEOCAP_PARAM_SYSINFO

NVR/IPC. support get with ctrl path, using HD_VIDEOCAP_SYSINFO struct (system infomation)

HD_VIDEOCAP_PARAM_DRV_CONFIG

NVR/IPC. support set with ctrl path, using HD_VIDEOCAP_DRV_CONFIG struct (device device config)

HD_VIDEOCAP_PARAM_PATH_CONFIG

NVR/IPC. support get/set with i/o path, using HD_VIDEOCAP_PATH_CONFIG struct

HD_VIDEOCAP_PARAM_OUT_NOTIFY_HANDLER

NVR only. support get/set with i/o path, HD_VIDEOCAP_NOTIFY_HANDLER. struct

HD_VIDEOCAP_PARAM_OUT_MD_STATUS

NVR only. support get/set with i/o path, HD_VIDEOCAP_MD_STATUS struct.

HD_VIDEOCAP_PARAM_CTRL

IPC only. support get/set with ctrl path, using HD_VIDEOCAP_CTRL struct (effect of whole device)

HD_VIDEOCAP_PARAM_IN

IPC only. support get/set with i/o path, HD_VIDEOCAP_IN

2.3.1 HD_VIDEOCAP_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_VIDEOCAP_DEVCAPS
in_caps	capability of input, combine caps of HD_VIDEO_CAPS and HD_VIDEOCAP_INCAPS
out_caps	capability of output, combine caps of HD_VIDEO_CAPS and HD_VIDEOCAP_OUTCAPS
max_dim	max dimension width/height
max_frame_rate	max frame rate
max_w_scaleup	max scaling up ratio
pxlfmt	pixel format

2.3.2 HD_VIDEOCAP_SYSINFO

[Description]
System information

[Parameter]

Value	Description
dev_id	device id
cur_in_fps	current input fps
vd_count	IPC only. VD count.

2.3.3 HD_VIDEOCAP_PARAM_IN (for IPC only)

[Description]
Sensor mode setting

[Parameter]

Value	Description
sen_mode	IPC only. Referring to sensor driver, or set HD_VIDEOCAP_SEN_MODE_AUTO for AUTO selecting
frc	IPC only. frame rate
dim	IPC only. dim w,h. only valid when sen_mode is HD_VIDEOCAP_SEN_MODE_AUTO
pxlfmt	IPC only. pixel format. only valid when sen_mode is HD_VIDEOCAP_SEN_MODE_AUTO
out_frame_num	IPC only. Sensor output frame number, 1 for linear mode and 2/3/4 for sensor HDR mode.

2.3.4 HD_VIDEOCAP_PARAM_OUT

[Description]
Output frame

[Parameter]

Value	Description
dim	NVR/IPC. output dim w,h
pxlfmt	IPC only. output pixel format.
frc	NVR only. output frame-control or frame rate
dir	IPC only. output direction, like mirror/flip

2.3.5 HD_VIDEOCAP_PARAM_IN_CROP

[Description]
Input crop

[Parameter]

Value	Description
mode	NVR/IPC. cropping mode, referring to #_HD_CROP_MODE
win	NVR/IPC. set region while the cropping mode is HD_CROP_ON, referring to #_HD_VIDEO_CROP
auto_info	IPC only. only valid while the cropping mode is HD_CROP_AUTO, referring to #_HD_VIDEOCAP_CROP_AUTO_INFO
align	IPC only. alignment, must be a multiple of 4

2.3.6 HD_VIDEOCAP_PARAM_PATH_CONFIG

[Description]
Path configuration

[Parameter]

Value	Description
data_pool	pool memory information
out_func	IPC only. additional function of out (bit-wise mask)

2.3.7 HD_VIDEOCAP_PARAM_CTRL (for IPC only)

[Description]
Control function

[Parameter]

Value	Description
Func	IPC only, additional function of ctrl path (whole device) (bit-wise mask)

2.3.8 HD_VIDEOCAP_DRV_CONFIG (for IPC only)

[Description]
Driver interface configuration

[Parameter]

Value	Description
sen_cfg	IPC only. sensor config

2.3.9 HD_VIDEOCAP_PARAM_OUT_MD_STATUS (for NVR only)

[Description]
Motion detection status configuration

[Parameter]

Value	Description
enabled	NVR only. Enable/Disable ouytput motion meta data.

2.4 Data structure definition of driver HW configure (for NVR Only)

The function hd_videocap_drv_get hd_videocap_drv_set provides the following parameter IDs and it's NVR only:

HD_VIDEOCAP_DRV_PARAM_INIT_HOST

NVR only. Support set, videocap host init, to specify vidoecap system vi usage and prepare requirement memory, using HD_VIDEOCAP_HOST structure.

HD_VIDEOCAP_DRV_PARAM_UNINIT_HOST

NVR only. support set, uninit videocap host to force clear setting, using HD_VIDEOCAP_HOST_ID structure

HD_VIDEOCAP_DRV_PARAM_REGISTER_VI

NVR only. support set, VCAP VI Register, using HD_VIDEOCAP_VI structure.

HD_VIDEOCAP_DRV_PARAM_DEREGISTER_VI

NVR only. support set, deregister all vcap vi to force clear setting, using HD_VIDEOCAP_VI_ID structure.

HD_VIDEOCAP_DRV_PARAM_GET_VI

NVR only. support get, vcap vi info, using HD_VIDEOCAP_VI structure

HD_VIDEOCAP_DRV_PARAM_VI_VPORT

NVR only. support get/set, vcap vi vport info, using HD_VIDEOCAP_VI_VPORT structure.

HD_VIDEOCAP_DRV_PARAM_VI_CH_PARAM

NVR only. support set/get HD_VIDEOCAP_VI_CH_PARAM item, using HD_VIDEOCAP_VI_CH_PARAM structure.

HD_VIDEOCAP_DRV_PARAM_VI_CH_NORM

NVR only. support set/get video norm, using HD_VIDEOCAP_VI_CH_NORM structure.

2.4.1 HD_VIDEOCAP_HOST

[Description],
System VI Parameter and system Motion Detection Parameter

[Parameter]

Value	Description
host	host index
nr_of_vi	number of video interface
vi.chip	chip index
vi.vcap	vcap index
vi.vi	Vi index
vi.mode	vi mode, value as HD_VIDEOCAP_VI_MODE
md.enable	enable/disable capture motion/tamper detection support
md.mb_x_num_max	specify horizontial motion block max number, 0 ~ 128
md.mb_y_num_max	specify vertical motion block max number, 0 ~ 128
md.buf_src	motion detection buffer allocate source, 0:driver 1:library

2.4.2 HD_VIDEOCAP_HOST_ID

[Description]
Host index

[Parameter]

Value	Description
host	host index

2.4.3 HD_VIDEOCAP_VI_ID

[Description]
vi index

[Parameter]

Value	Description
chip	chip index
vcap	vcap index
vi	vi index

2.4.4 HD_VIDEOCAP_VI

[Description]

vi Global Parameter and VI VPort Parameter

[Parameter]

Value	Description
chip	chip index
vcap	vcap index
vi	vi index
global.src	vi input source
global.format	vi input source format
global.tdm	vi time division multiplexed mode
global.id_extract	vi channel id extract mode for 2CH_MUX or 4CH_MUX
global.latch_edge	vi data latch edge for 2CH_DualEdge or 4CH_MUX_DualEdge
vport.data_swap	vport data swap control
vport.clk_pin	vport pixel clock pin source
vport.clk_inv	vport pixel clock invertion, 0:disable 1:enable
vport.clk_dly	vport pixel clock delay
vport.clk_pdly	vport pixel clock polarity delay

2.4.5 HD_VIDEOCAP_VI_VPORT

[Description]

vi port parameter

[Parameter]

Value	Description
chip	chip index
vcap	vcap index
vi	vi index
vport	vport index
pid	vport parameter index, value as HD_VIDEOCAP_VI_VPORT_PARAM_ID
value	vport parameter get/set value

[HD_VIDEOCAP_VI_VPORT_PARAM_ID]

Value	Description
HD_VIDEOCAP_VI_VPORT_PARAM_CLK_INV	vi vport pixel clock invertion, parameter value as 0:disable 1:enable
HD_VIDEOCAP_VI_VPORT_PARAM_CLK_DELAY	vi vport pixel clock delay, parameter value as 0 ~ 0xff
HD_VIDEOCAP_VI_VPORT_PARAM_CLK_PDELAY	vi vport pixel clock polarity delay, parameter value as 0 ~ 0xff, not support in GM8220
HD_VIDEOCAP_VI_VPORT_PARAM_DATA_SWAP	vi vport data swap control, parameter value as HD_VIDEOCAP_VI_VPORT_DATA_SWAP

2.4.6 HD_VIDEOCAP_VI_CH_PARAM

[Description]

System capability

[Parameter]

Value	Description
chip	chip index
vcap	vcap index
vi	vi index
ch	ch index

pid	vi channel parameter index, value as HD_VIDEOCAP_VI_CH_PARAM_ID
value	vi channel parameter get/set value

[Parameter]

Value	Description
HD_VIDEOCAP_VI_CH_PARAM_DATA_RANGE	channel output data range, parameter value as HD_VIDEOCAP_VI_CH_DATA_RATE_T
HD_VIDEOCAP_VI_CH_PARAM_YC_SWAP	channel output ycbcr swap, parameter value as HD_VIDEOCAP_VI_CH_SWAP_T
HD_VIDEOCAP_VI_CH_PARAM_FPS	channel source frame rate, parameter value must > 0
HD_VIDEOCAP_VI_CH_PARAM_TIMEOUT_MS	channel capture frame timeout ms, parameter value must > 0
HD_VIDEOCAP_VI_CH_PARAM_VCH_ID	channel video index => mapping to physical connector index, means VCH index, parameter value as int
HD_VIDEOCAP_VI_CH_PARAM_VLOS	channel video loss status, 0:video present 1:video loss

2.4.7 HD_VIDEOCAP_VI_CH_NORM

[Description]

System capability

[Parameter]

Value	Description
chip	chip index
vcap	vcap index
vi	vi index
ch	ch index
cap_width	channel capture width
cap_height	channel capture height
org_width	channel original width, video source width

org_height	channel original height, video source height
fps	channel frame rate, must > 0
format	channel format
prog	channel progressive/interlace
data_rate	channel data rate, for specify byte duplicate mode
data_latch	channel data latch mode
horiz_dup	channel horizontal pixel duplicate mode

3 Usage

3.1 user_videocap (for NVR only)

The user_videocap demonstrates how to use the single trigger operation to process the output image.

```
/**
     @brief This sample demonstrates the usage of videocap functions.

     @file user_videocap.c

   @ingroup hdal

     @note Nothing.

     Copyright Novatek Microelectronics Corp. 2018.  All rights reserved.
*/

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

```
#include <asm/ioctl.h>
#include "hdal.h"

HD_VIDEO_FRAME videocap_out_buffer;
VOID  *videocap_out_buffer_va = NULL;
HD_PATH_ID video_cap;

#define MAX_FRAME_WIDTH    1920
#define MAX_FRAME_HEIGHT   1080

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 main(INT argc, CHAR *argv[])
{
     INT raw_frame_size;
     CHAR filename[40];
     HD_RESULT ret;
     HD_COMMON_MEM_VB_BLK blk;
     HD_COMMON_MEM_DDR_ID ddr_id = DDR_ID0;
     UINT64 pool = HD_COMMON_MEM_COMMON_POOL;

     /* Initialization */
     ret = hd_common_init(1);
     if (ret != HD_OK) {
          printf("hd_common_init fail\n");
          goto exit;
     }
     ret = hd_videocap_init();
```

```
if (ret != HD_OK) {
     printf("hd_videocap_init fail\n");
     goto exit;
}
ret =  hd_videocap_open(HD_VIDEOCAP_IN(0, 0), HD_VIDEOCAP_OUT(0, 0), &video_cap);
if (ret != HD_OK) {
     printf("hd_videocap_open fail\n");
     goto exit;
}

/* Allocate out buffer */
memset(&videocap_out_buffer, 0, sizeof(videocap_out_buffer));
videocap_out_buffer.ddr_id = ddr_id;
videocap_out_buffer.dim.w = MAX_FRAME_WIDTH;
videocap_out_buffer.dim.h = MAX_FRAME_HEIGHT;
videocap_out_buffer.pxlfmt = HD_VIDEO_PXLFMT_YUV420;
raw_frame_size = MAX_FRAME_WIDTH * MAX_FRAME_HEIGHT * 3 / 2;
blk = hd_common_mem_get_block(pool, raw_frame_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;
}
videocap_out_buffer.phy_addr[0] = hd_common_mem_blk2pa(blk);
if (videocap_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;
}
videocap_out_buffer_va = hd_common_mem_mmap(HD_COMMON_MEM_MEM_TYPE_NONCACHE,
                                            videocap_out_buffer.phy_addr[0],
                                            raw_frame_size);
printf("Allocate videocap_out_buffer pa(%#lx) va(%p)\n",
          videocap_out_buffer.phy_addr[0], videocap_out_buffer_va);
/* Pull out buffer */
ret = hd_videocap_pull_out_buf(video_cap, &videocap_out_buffer, 500);
if (ret != HD_OK) {
     printf("hd_videocap_pull_out_buf fail\n");
```

```
            goto exit;
      } else {
            sprintf(filename, "user_cap_%ldx%ld_YUV420.yuv", videocap_out_buffer.dim.w,
videocap_out_buffer.dim.h);
            save_output(filename, videocap_out_buffer_va, raw_frame_size);
      }

      /* Release out buffer */
      hd_common_mem_munmap(videocap_out_buffer_va, raw_frame_size);
      hd_common_mem_release_block((HD_COMMON_MEM_VB_BLK)videocap_out_buffer.phy_addr[0]);

exit:
      ret = hd_videocap_close(video_cap);
      if (ret != HD_OK) {
            printf("hd_videocap_close fail\n");
            return -1;
      }
      ret = hd_videocap_uninit();
      if (ret != HD_OK) {
            printf("hd_videocap_uninit fail\n");
            return -1;
      }
      ret = hd_common_uninit();
      if (ret != HD_OK) {
            printf("hd_common_uninit fail\n");
            return -1;
      }

      return 0;
}
```

3.2 videocap_module_init (for NVR only)

The videocap_module_init demonstrates how to set the AD and videocapture H/W config.

```
HD_RESULT videocap_module_init(CHAR *ad_dev_name)
{
    HD_RESULT ret = HD_OK;
    INT i, j;
    INT notify_polling = 0;
    HD_VIDEOCAP_HOST_ID vcap_host_id;
    HD_VIDEOCAP_HOST vcap_host;
    HD_VIDEOCAP_VI vcap_vi;
    HD_VIDEOCAP_VI_ID vcap_vi_id;
    HD_VIDEOCAP_VI_CH_PARAM ch_param;
    HD_VIDEOCAP_VI_CH_NORM ch_norm;
    VENDOR_AD_TP28XX_DEVICE_INFO dev_info;
    VENDOR_AD_TP28XX_VIDEO_NORM video_norm;
    VENDOR_AD_TP28XX_VIDEO_LOSS ch_loss;

      ret = vendor_ad_init(ad_dev_name);
    if (ret != HD_OK)
      goto exit;

    /* get tp28xx device info */
      ret = vendor_ad_get(VENDOR_AD_PARAM_TP28XX_DEVICE_INFO, &dev_info);
    if (ret != HD_OK)
      goto exit;

    /* deregister all vcap vi to force clear previous setting */
    for (i=0; i<HD_VIDEOCAP_VI_MAX; i++) {
        vcap_vi_id.chip    = i/8;      ///< GM8220 support 8   VI for each chip, GM8296 support
4   VI for each chip
        vcap_vi_id.vcap    = i/4;      ///< GM8220 support 2 VCAP for each chip, GM8296 support
1 VCAP for each chip
        vcap_vi_id.vi      = i%4;      ///< GM8220 and GM8296 support 4 VI for each VCAP
        ret = hd_videocap_drv_set(HD_VIDEOCAP_DRV_PARAM_DEREGISTER_VI, &vcap_vi_id);
        if (ret != HD_OK)
            goto exit;
    }

    /* uninit vcap host to force clear previous setting */
```

```
    vcap_host_id.host    = 0;
      ret = hd_videocap_drv_set(HD_VIDEOCAP_DRV_PARAM_UNINIT_HOST, &vcap_host_id);

    /* vcap host init, to specify vcap system vi usage and prepare requirement memory */
    memset(&vcap_host, 0, sizeof(vcap_host));
    vcap_host.host          = 0;
    vcap_host.md.enable      = 1;
    vcap_host.md.mb_x_num_max = 128;
    vcap_host.md.mb_y_num_max = 64;
    for (i = 0; i< dev_info.dev_num; i++) {
        for(j = 0; j< VENDOR_AD_VOUT_MAX; j++) {
            if(!dev_info.dev[i].vout[j].xcap || !dev_info.dev[i].vout[j].vi)  ///< VOUT#
Connect to X_CAP# and Grab VI#
                continue;

            vcap_host.vi[vcap_host.nr_of_vi].chip = (dev_info.dev[i].vout[j].vi - 1)/8;   ///<
GM8220 support 8   VI for each chip, GM8296 support 4   VI for each chip
            vcap_host.vi[vcap_host.nr_of_vi].vcap = (dev_info.dev[i].vout[j].vi - 1)/4;   ///<
GM8220 support 2 VCAP for each chip, GM8296 support 1 VCAP for each chip
            vcap_host.vi[vcap_host.nr_of_vi].vi   = (dev_info.dev[i].vout[j].vi - 1)%4;   ///<
GM8220 and GM8296 support 4 VI for each VCAP
            switch (dev_info.dev[i].vout_mode) {
                case AD_TP28XX_VOUT_MODE_SDR_2CH_DUAL_EDGE:
                case AD_TP28XX_VOUT_MODE_SDR_2CH_MUX:
                case AD_TP28XX_VOUT_MODE_DDR_2CH_DUAL_EDGE:
                    vcap_host.vi[vcap_host.nr_of_vi].mode = HD_VIDEOCAP_VI_MODE_2CH;
                    break;
                case AD_TP28XX_VOUT_MODE_DDR_4CH_MUX_DUAL_EDGE:
                    vcap_host.vi[vcap_host.nr_of_vi].mode = HD_VIDEOCAP_VI_MODE_4CH_2P;
                    break;
                case AD_TP28XX_VOUT_MODE_SDR_1CH_BYPASS:
                case AD_TP28XX_VOUT_MODE_DDR_1CH_BYPASS:
                case AD_TP28XX_VOUT_MODE_DDR_1CH_16BIT_BYPASS:
                default:
                    vcap_host.vi[vcap_host.nr_of_vi].mode = HD_VIDEOCAP_VI_MODE_1CH;
                    break;
            }
```

```
            vcap_host.nr_of_vi++;
            if (vcap_host.nr_of_vi > HD_VIDEOCAP_VI_MAX) {
                ret = -1;
                goto exit;
            }
        }
    }
    if (!vcap_host.nr_of_vi)
        goto exit;

      ret = hd_videocap_drv_set(HD_VIDEOCAP_DRV_PARAM_INIT_HOST, &vcap_host);
    if (ret != HD_OK)
      goto exit;

    /* VCAP VI Register */
    for (i=0; i<dev_info.dev_num; i++) {
        for (j=0; j<VENDOR_AD_VOUT_MAX; j++) {
            if(!dev_info.dev[i].vout[j].xcap || !dev_info.dev[i].vout[j].vi)    ///< VOUT#
Connect to X_CAP# and Grab VI#
                continue;

            memset(&vcap_vi, 0, sizeof(vcap_vi));

            vcap_vi.chip    = (dev_info.dev[i].vout[j].vi - 1)/8;              ///< GM8220 support
8   VI for each chip, GM8296 support 4   VI for each chip
            vcap_vi.vcap    = (dev_info.dev[i].vout[j].vi - 1)/4;              ///< GM8220 support
2 VCAP for each chip, GM8296 support 1 VCAP for each chip
            vcap_vi.vi      = (dev_info.dev[i].vout[j].vi - 1)%4;               ///< GM8220 and
GM8296 support 4 VI for each VCAP

            vcap_vi.global.src       = dev_info.dev[i].vout[j].xcap - 1;
            vcap_vi.global.format    = HD_VIDEOCAP_VI_FMT_BT656;
            vcap_vi.global.id_extract = HD_VIDEOCAP_VI_CHID_EAV_SAV;
            switch (dev_info.dev[i].vout_mode) {
                case AD_TP28XX_VOUT_MODE_SDR_2CH_DUAL_EDGE:
                case AD_TP28XX_VOUT_MODE_DDR_2CH_DUAL_EDGE:
                    vcap_vi.global.tdm       = HD_VIDEOCAP_VI_TDM_2CH_DUALEDGE;
                    vcap_vi.global.latch_edge = HD_VIDEOCAP_VI_LATCH_EDGE_DUAL;
```

```
                break;
            case AD_TP28XX_VOUT_MODE_SDR_2CH_MUX:
                vcap_vi.global.tdm       = HD_VIDEOCAP_VI_TDM_2CH_MUX;
                vcap_vi.global.latch_edge = HD_VIDEOCAP_VI_LATCH_EDGE_SINGLE;
                break;
            case AD_TP28XX_VOUT_MODE_DDR_4CH_MUX_DUAL_EDGE:
                vcap_vi.global.tdm       = HD_VIDEOCAP_VI_TDM_4CH_2P_MUX;
                vcap_vi.global.latch_edge = HD_VIDEOCAP_VI_LATCH_EDGE_DUAL;
                break;
            case AD_TP28XX_VOUT_MODE_SDR_1CH_BYPASS:
            case AD_TP28XX_VOUT_MODE_DDR_1CH_BYPASS:
            case AD_TP28XX_VOUT_MODE_DDR_1CH_16BIT_BYPASS:
            default:
                vcap_vi.global.tdm       = HD_VIDEOCAP_VI_TDM_1CH_BYPASS;
                vcap_vi.global.latch_edge = HD_VIDEOCAP_VI_LATCH_EDGE_SINGLE;
                break;
        }
        vcap_vi.vport[0].clk_inv   = dev_info.dev[i].vout[j].clk_inv;
        vcap_vi.vport[0].clk_dly   = dev_info.dev[i].vout[j].clk_dly;
        vcap_vi.vport[0].clk_pdly  = dev_info.dev[i].vout[j].clk_pdly;
        vcap_vi.vport[0].clk_pin   = dev_info.dev[i].vout[j].clk_pin;
        vcap_vi.vport[0].data_swap = dev_info.dev[i].vout[j].data_swap;   ///<
VI-P0(rising)
        if (vcap_vi.global.latch_edge == HD_VIDEOCAP_VI_LATCH_EDGE_DUAL)
            vcap_vi.vport[1].data_swap = vcap_vi.vport[0].data_swap;     ///<
VI-P1(falling)
             ret = hd_videocap_drv_set(HD_VIDEOCAP_DRV_PARAM_REGISTER_VI, &vcap_vi);
            if (ret != HD_OK)
              goto exit;
      }
   }

   /* VCH ID */
   for (i=0; i<dev_info.dev_num; i++) {
      for (j=0; j<VENDOR_AD_CHANNELS_PER_CHIP; j++) {
         if (!dev_info.dev[i].vin[j].active)
            continue;
```

```
        memset(&ch_param, 0, sizeof(ch_param));

        ch_param.chip    = dev_info.dev[i].vin[j].chip;
        ch_param.vcap    = dev_info.dev[i].vin[j].vcap;
        ch_param.vi      = dev_info.dev[i].vin[j].vi;
        ch_param.ch      = dev_info.dev[i].vin[j].ch;
        ch_param.value   = dev_info.dev[i].vin[j].vch_id;
        ch_param.pid     = HD_VIDEOCAP_VI_CH_PARAM_VCH_ID;
             ret = hd_videocap_drv_set(HD_VIDEOCAP_DRV_PARAM_VI_CH_PARAM, &ch_param);
           if (ret != HD_OK)
             goto exit;
    }
}

/* Polling channel video norm to notify vcap channel norm switch */
do {
    for (i=0; i<dev_info.dev_num; i++) {
        for (j=0; j<VENDOR_AD_CHANNELS_PER_CHIP; j++) {
            if (!dev_info.dev[i].vin[j].active)
                continue;

            /* get video norm */
            memset(&video_norm, 0, sizeof(video_norm));
            video_norm.dev_id = i;
            video_norm.vin_id = j;
                   ret = vendor_ad_get(VENDOR_AD_PARAM_TP28XX_VIDEO_NORM, &video_norm);
                 if (ret != HD_OK)
                   goto chk_loss;

            /* check video norm */
            if (memcmp(&video_norm, &g_ch_norm[i][j], sizeof(video_norm)) == 0)
                goto chk_loss;

            ch_norm.chip      = dev_info.dev[i].vin[j].chip;
            ch_norm.vcap      = dev_info.dev[i].vin[j].vcap;
            ch_norm.vi        = dev_info.dev[i].vin[j].vi;
            ch_norm.ch        = dev_info.dev[i].vin[j].ch;
            ch_norm.org_width  = video_norm.src_width;
```

```
            ch_norm.org_height = video_norm.src_height;
            ch_norm.fps       = video_norm.src_fps;
            ch_norm.prog      = video_norm.src_prog;
            ch_norm.cap_width  = video_norm.out_width;
            ch_norm.cap_height = video_norm.out_height;
            ch_norm.format    = video_norm.out_fmt;
            ch_norm.data_rate  = video_norm.out_data_rate;
            ch_norm.data_latch = video_norm.out_data_latch;
            ch_norm.horiz_dup  = video_norm.out_horiz_dup;
                  ret = hd_videocap_drv_set(HD_VIDEOCAP_DRV_PARAM_VI_CH_NORM, &ch_norm);
                if (ret != HD_OK)
                  goto chk_loss;

            memcpy(&g_ch_norm[i][j], &video_norm, sizeof(video_norm));

            fprintf(stdout, "tp28xx#%d-vin#%d norm switch to %lux%lu%s@%lu\n",
                  i, j,
                  video_norm.src_width,
                  video_norm.src_height,
                  video_norm.src_prog ? "P" : "I",
                  video_norm.src_fps);

chk_loss:
            /* get video loss */
            ch_loss.chip = i;
            ch_loss.ch   = j;
                  ret = vendor_ad_get(VENDOR_AD_PARAM_TP28XX_VIDEO_LOSS, &ch_loss);
                if (ret != HD_OK)
                  continue;

            /* check video loss */
            if (ch_loss.is_lost == g_ch_loss[i][j])
               continue;

            ch_param.chip    = dev_info.dev[i].vin[j].chip;
            ch_param.vcap    = dev_info.dev[i].vin[j].vcap;
            ch_param.vi      = dev_info.dev[i].vin[j].vi;
            ch_param.ch      = dev_info.dev[i].vin[j].ch;
```

```
            ch_param.value   = ch_loss.is_lost;
            ch_param.pid     = HD_VIDEOCAP_VI_CH_PARAM_VLOS;
                  ret = hd_videocap_drv_set(HD_VIDEOCAP_DRV_PARAM_VI_CH_PARAM, &ch_param);
                if (ret != HD_OK)
                  goto chk_loss;

            g_ch_loss[i][j] = ch_loss.is_lost;

            fprintf(stdout, "tp28xx#%d-vin#%d video %s\n",
                  i, j,
                  ch_loss.is_lost ? "loss" : "present");
         }
      }
      usleep(500000);
   } while (notify_polling);

exit:
     ret = vendor_ad_uninit();
   return ret;
}
```

3.3 Sensor configuration (for IPC only)

The settings related to the HDAL layer are mainly based on the pinmux and interface of the IC itself. Others related to the sensor itself need to refer to the sensor driver settings. Videocapture is mainly based on the following settings:

- sensor driver name: It is used to set which sensor driver is used in the bottom layer of HDAL. Before opening, you must confirm whether the sensor driver has been inserted. You can refer to SDK S10_SysInit2, for example: Insmod /lib/modules/4.1.0/hdal/sen_imx290/nvt_sen_imx290.ko
- interface: According to hardware design, select interface type, referring to HD_COMMON_VIDEO_IN_TYPE.
- pinmux: Set the value which is defined in \BSP\linux-kernel\arch\arm\mach-nvt-na51000\include\mach\ top.h corresponding to sensor_pinmux/ serial_if_pinmux/ cmd_if_pinmux by hardware

circuit. For example, if ***sensor_pinmux*** uses PIN_SENSOR_CFG_MIPI and PIN_SENSOR_CFG_MCLK, the value will be 0x220. If ***serial_if_pinmux*** uses PIN_MIPI_LVDS_CFG_CLK0 | PIN_MIPI_LVDS_CFG_DAT0|PIN_MIPI_LVDS_CFG_DAT1, the value will be0x301. Please note that the Enum used by sensor_pinmux are related to the deice ID of videocapture. For instance, videocapture1 (the second sensor) should use PIN_SENSOR2_CFG_MIPI. When sensor command interface is I2C, ***cmd_if_pinmux*** is referred to PIN_I2C_CFG. In addition, the I2C pinmux used by sensor should not be enabled in the system default setting (nvt-na51000-top.dtsi) to prevent form the unstable hardware signal causing I2C engine to be in the unknown state. Fox example, if the second I2C channel, PIN_I2C_CFG_CH2, is used for sensor, then the dts setting for for I2C CH2 should be zero, such as i2c{pinmux = <0x10001>;}. The rightmost hex is for I2C CH1 while the leftmost of 0x10001 is for CH5.

- shdr_map: Only used for sensor HDR. Two videocaptures are required for an HDR sensor with 2 frame mode. For example, sensor with CSI0 can use any two of videocaptures, from ID0 to ID3. However, sensor with CSI1 can only use videocapture_1 and vidrocapture_3. Apart from the HD_VIDEOCAP_PARAM_DRV_CONFIG setting, the out_frame_num of HD_VIDEOCAP_PARAM_IN should also be changed to the corresponding frame number such as HD_VIDEOCAP_SEN_FRAME_NUM_2 for 2-framed mode and the func of HD_VIDEOCAP_PARAM_CTRL should add HD_VIDEOCAP_FUNC_SHDR. Meanwhile, both HD_VIDEOPROC_DEV_CONFIG's ctrl_max.func and HD_VIDEOPROC_CTRL's func also need to add HD_VIDEOPROC_FUNC_SHDR in HD_VIDEOPROCESS. In HDR with 2-framed mode, the memory block count required for RAW is 2 times that of the original linear, 3-framed mode is 3 times, and so on.

3.3.1 Single sensor (linear mode)

The following sample code demonstrates how to set sensor IMX291 to videocapture0 and enable IQ function of AE/AWB.

```
HD_RESULT set_cap_cfg(void)
{
    HD_RESULT ret = HD_OK;
    HD_VIDEOCAP_DRV_CONFIG cap_cfg = {0};
    HD_PATH_ID video_cap_ctrl = 0;
```

```
    HD_VIDEOCAP_CTRL iq_ctl = {0};


    snprintf(cap_cfg.sen_cfg.sen_dev.driver_name, HD_VIDEOCAP_SEN_NAME_LEN-1,
"nvt_sen_imx291");
    cap_cfg.sen_cfg.sen_dev.if_type = HD_COMMON_VIDEO_IN_MIPI_CSI;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.pinmux.sensor_pinmux =  0x220; //PIN_SENSOR_CFG_MIPI |
PIN_SENSOR_CFG_MCLK
    cap_cfg.sen_cfg.sen_dev.pin_cfg.pinmux.serial_if_pinmux =
0xf04;//PIN_MIPI_LVDS_CFG_CLK2 | PIN_MIPI_LVDS_CFG_DAT0|PIN_MIPI_LVDS_CFG_DAT1 |
PIN_MIPI_LVDS_CFG_DAT2 | PIN_MIPI_LVDS_CFG_DAT3
    cap_cfg.sen_cfg.sen_dev.pin_cfg.pinmux.cmd_if_pinmux = 0x10;//PIN_I2C_CFG_CH2
    cap_cfg.sen_cfg.sen_dev.pin_cfg.clk_lane_sel = HD_VIDEOCAP_SEN_CLANE_SEL_CSI0_USE_C2;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[0] = 0;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[1] = 1;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[2] = HD_VIDEOCAP_SEN_IGNORE;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[3] = HD_VIDEOCAP_SEN_IGNORE;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[4] = HD_VIDEOCAP_SEN_IGNORE;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[5] = HD_VIDEOCAP_SEN_IGNORE;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[6] = HD_VIDEOCAP_SEN_IGNORE;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[7] = HD_VIDEOCAP_SEN_IGNORE;
    ret = hd_videocap_open(0, HD_VIDEOCAP_0_CTRL, &video_cap_ctrl); //open this for device
control
    if (ret != HD_OK) {
        return ret;
    }
    ret |= hd_videocap_set(video_cap_ctrl, HD_VIDEOCAP_PARAM_DRV_CONFIG, &cap_cfg);
    iq_ctl.func = HD_VIDEOCAP_FUNC_AE | HD_VIDEOCAP_FUNC_AWB;
    ret |= hd_videocap_set(video_cap_ctrl, HD_VIDEOCAP_PARAM_CTRL, &iq_ctl);
    return ret;
}
```

3.3.2 Single sensor (HDR mode)

The following sample code demonstrates how to set sensor IMX290 to videocapture0 and videocapture2. Please note that HD_VIDEOCAP_PARAM_DRV_CONFIG should be set prior to HD_VIDEOCAP_PARAM_CTRL in HDR mode.

```
HD_RESULT set_cap_shdr_cfg(void)
```

```
{
    HD_RESULT ret = HD_OK;
    HD_VIDEOCAP_DRV_CONFIG cap_cfg = {0};
    HD_PATH_ID video_cap0_ctrl = 0;
    HD_VIDEOCAP_CTRL iq_ctl = {0};

    snprintf(cap_cfg.sen_cfg.sen_dev.driver_name, HD_VIDEOCAP_SEN_NAME_LEN-1,
"nvt_sen_imx290");
    cap_cfg.sen_cfg.sen_dev.if_type = HD_COMMON_VIDEO_IN_MIPI_CSI;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.pinmux.sensor_pinmux =  0x220; //PIN_SENSOR_CFG_MIPI |
PIN_SENSOR_CFG_MCLK
    cap_cfg.sen_cfg.sen_dev.pin_cfg.pinmux.serial_if_pinmux =
0xf04;//PIN_MIPI_LVDS_CFG_CLK2 | PIN_MIPI_LVDS_CFG_DAT0|PIN_MIPI_LVDS_CFG_DAT1 |
PIN_MIPI_LVDS_CFG_DAT2 | PIN_MIPI_LVDS_CFG_DAT3
    cap_cfg.sen_cfg.sen_dev.pin_cfg.pinmux.cmd_if_pinmux = 0x10;//PIN_I2C_CFG_CH2
    cap_cfg.sen_cfg.sen_dev.pin_cfg.clk_lane_sel = HD_VIDEOCAP_SEN_CLANE_SEL_CSI0_USE_C2;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[0] = 0;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[1] = 1;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[2] = 2;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[3] = 3;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[4] = HD_VIDEOCAP_SEN_IGNORE;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[5] = HD_VIDEOCAP_SEN_IGNORE;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[6] = HD_VIDEOCAP_SEN_IGNORE;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[7] = HD_VIDEOCAP_SEN_IGNORE;
    ret = hd_videocap_open(0, HD_VIDEOCAP_0_CTRL, &video_cap0_ctrl); //open this for device
control
    if (ret != HD_OK) {
        return ret;
    }
    cap_cfg.sen_cfg.shdr_map = HD_VIDEOCAP_SHDR_MAP(HD_VIDEOCAP_HDR_SENSOR1,
(HD_VIDEOCAP_0|HD_VIDEOCAP_1));
    ret |= hd_videocap_set(video_cap0_ctrl, HD_VIDEOCAP_PARAM_DRV_CONFIG, &cap_cfg);
    if (ret != HD_OK) {
        return ret;
    }
    iq_ctl.func = HD_VIDEOCAP_FUNC_AE | HD_VIDEOCAP_FUNC_AWB | HD_VIDEOCAP_FUNC_SHDR;
    ret |= hd_videocap_set(video_cap0_ctrl, HD_VIDEOCAP_PARAM_CTRL, &iq_ctl);
    return ret;
```

```
}
```

3.3.3 Dual sensor (HDR mode)

The first HDR sensor is the same as the above example, using videocapture_0 and videocapture_2 and the second HDR sensor uses CSI1 with videocapture_1 and videocapture_3 as follows:

```
HD_RESULT set_cap2_shdr_cfg(void)
{
    HD_RESULT ret = HD_OK;
    HD_VIDEOCAP_DRV_CONFIG cap_cfg = {0};
    HD_PATH_ID video_cap1_ctrl = 0;
    HD_VIDEOCAP_CTRL iq_ctl = {0};

    snprintf(cap_cfg.sen_cfg.sen_dev.driver_name, HD_VIDEOCAP_SEN_NAME_LEN-1,
"nvt_sen_imx290");
    cap_cfg.sen_cfg.sen_dev.if_type = HD_COMMON_VIDEO_IN_MIPI_CSI;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.pinmux.sensor_pinmux =  0x20; //PIN_SENSOR2_CFG_MIPI
    cap_cfg.sen_cfg.sen_dev.pin_cfg.pinmux.serial_if_pinmux = 0x3010;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.pinmux.cmd_if_pinmux = 0x100;//PIN_I2C_CFG_CH3
    cap_cfg.sen_cfg.sen_dev.pin_cfg.clk_lane_sel = HD_VIDEOCAP_SEN_CLANE_SEL_CSI1_USE_C4;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[0] = 0;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[1] = 1;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[2] = 2;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[3] = 3;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[4] = HD_VIDEOCAP_SEN_IGNORE;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[5] = HD_VIDEOCAP_SEN_IGNORE;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[6] = HD_VIDEOCAP_SEN_IGNORE;
    cap_cfg.sen_cfg.sen_dev.pin_cfg.sen_2_serial_pin_map[7] = HD_VIDEOCAP_SEN_IGNORE;
    ret = hd_videocap_open(0, HD_VIDEOCAP_0_CTRL, &video_cap1_ctrl); //open this for device
control
    if (ret != HD_OK) {
        return ret;
    }
    cap_cfg.sen_cfg.shdr_map = HD_VIDEOCAP_SHDR_MAP(HD_VIDEOCAP_HDR_SENSOR2,
(HD_VIDEOCAP_1|HD_VIDEOCAP_3));
```

```
    ret |= hd_videocap_set(video_cap1_ctrl, HD_VIDEOCAP_PARAM_DRV_CONFIG, &cap_cfg);
    if (ret != HD_OK) {
        return ret;
    }
    iq_ctl.func = HD_VIDEOCAP_FUNC_AE | HD_VIDEOCAP_FUNC_AWB | HD_VIDEOCAP_FUNC_SHDR;
    ret |= hd_videocap_set(video_cap1_ctrl, HD_VIDEOCAP_PARAM_CTRL, &iq_ctl);
    return ret;
}
```

3.3.4 Multi-sensor (linear mode)

Similar to single sensor (linear mode), each videocapture must set the complete information. Just note that the pinmux Enum used for each videocapture ID and isp ID for each videoprocess may be different. For example, videocapture_1 (the second sensor) uses PIN_SENSOR2_CFG_MIPI and videoprocess_1 uses isp_id = 1.

3.3.5 Multi-sensor (slave mode)

Similar to multi-sensor (linear mode), just add TGE-related settings.
If all slave sensors need synchronized signal, vcap_sync_set should contain all videocaptures which are connect to slave sensors.
For example, if two slave sensors use videocpature_0 and videocapture_1, both of their vcap_sync_set should be (HD_VIDEOCAP_0|HD_VIDEOCAP_1).
When videocaptures are in synchronized mode, all of them should be opened before any of them be started and one with smaller ID has to be prior to the others.
On the contrary, stop all before closing any.
For instance:
open vcap0 -> open vcap1 -> start vcap0 -> start vcap1
stop vcap0 -> stop vcap1 -> close vcap0 -> close vcap1

The followings only show TGE-related setting:

```
HD_RESULT set_cap_cfg(void)
{
    HD_RESULT ret = HD_OK;
    HD_VIDEOCAP_DRV_CONFIG cap_cfg = {0};
    …
    cap_cfg.sen_cfg.sen_dev.if_cfg.tge.tge_en = TRUE;
```

```
    cap_cfg.sen_cfg.sen_dev.if_cfg.tge.swap = FALSE;
    cap_cfg.sen_cfg.sen_dev.if_cfg.tge.vcap_vd_src = HD_VIDEOCAP_SEN_TGE_CH1_VD_TO_VCAP0;
    cap_cfg.sen_cfg.sen_dev.if_cfg.tge.vcap_sync_set = (HD_VIDEOCAP_0|HD_VIDEOCAP_1);
    …
}
HD_RESULT set_cap2_cfg(void)
{
    HD_RESULT ret = HD_OK;
    HD_VIDEOCAP_DRV_CONFIG cap_cfg = {0};
    …
    cap_cfg.sen_cfg.sen_dev.if_cfg.tge.tge_en = TRUE;
    cap_cfg.sen_cfg.sen_dev.if_cfg.tge.swap = FALSE;
    cap_cfg.sen_cfg.sen_dev.if_cfg.tge.vcap_vd_src = HD_VIDEOCAP_SEN_TGE_CH1_VD_TO_VCAP0;
    cap_cfg.sen_cfg.sen_dev.if_cfg.tge.vcap_sync_set = (HD_VIDEOCAP_0|HD_VIDEOCAP_1);
    …
}
```

3.4 Sensor mode and videocapture out (for IPC only)

- The settings related to the sensor mode are mainly HD_VIDEOCAP_PARAM_IN, referring to HD_VIDEOCAP_IN, you can use auto (sen_mode = HD_VIDEOCAP_SEN_MODE_AUTO) or directly specify a mode in the sensor driver. When using auto mode, videocapture will pass the value of frc/dim/pxlfmt/out_frame_num of HD_VIDEOCAP_IN to the sensor driver, and the sensor driver will automatically select the most suitable senor mode.
- The output of videocapture is related to HD_VIDEOCAP_PARAM_IN_CROP and HD_VIDEOCAP_PARAM_OUT. Sensor data will be cropped (if crop mode is not HD_CROP_OFF) and then scale to HD_VIDEOCAP_OUT dimension (only support scaling down or no scaling). When crop mode is HD_CROP_OFF and the dim of HD_VIDEOCAP_OUT is set to 0, the output resolution is directly based on the resolution of the sensor output.

3.4.1 Full frame output

Select sensor mode by AUTO and outputs the original sensor out resolution.

```
HD_RESULT set_cap_param(HD_PATH_ID video_cap_path, HD_DIM *p_dim)
{
     HD_RESULT ret = HD_OK;
     {//select sensor mode, manually or automatically
          HD_VIDEOCAP_IN video_in_param = {0};
          video_in_param.sen_mode = HD_VIDEOCAP_SEN_MODE_AUTO; //auto select sensor mode by the
parameter of HD_VIDEOCAP_PARAM_OUT
          video_in_param.frc = HD_VIDEO_FRC_RATIO(30,1);
          video_in_param.dim.w = 1920;
          video_in_param.dim.h = 1080;
          video_in_param.pxlfmt = HD_VIDEO_PXLFMT_RAW12;
          video_in_param.out_frame_num = HD_VIDEOCAP_SEN_FRAME_NUM_1;
          ret = hd_videocap_set(video_cap_path, HD_VIDEOCAP_PARAM_IN, &video_in_param);
          //printf("set_cap_param MODE=%d\r\n", ret);
          if (ret != HD_OK) {
               return ret;
          }
     }
     //no crop, full frame
     {
          HD_VIDEOCAP_CROP video_crop_param = {0};

          video_crop_param.mode = HD_CROP_OFF;
          ret = hd_videocap_set(video_cap_path, HD_VIDEOCAP_PARAM_OUT_CROP,
&video_crop_param);
          //printf("set_cap_param CROP NONE=%d\r\n", ret);
     }
     {
          HD_VIDEOCAP_OUT video_out_param = {0};

          //without setting dim for no scaling, using original sensor out size
          video_out_param.pxlfmt = HD_VIDEO_PXLFMT_RAW12;
          video_out_param.dir = HD_VIDEO_DIR_NONE;
          ret = hd_videocap_set(video_cap_path, HD_VIDEOCAP_PARAM_OUT, &video_out_param);
          //printf("set_cap_param OUT=%d\r\n", ret);
```

```
    }
    return ret;
}
```

3.4.2 Crop and scale down output

Select sensor mode by AUTO. Crop and scale down to target output resolution.

```
HD_RESULT set_cap_param(HD_PATH_ID video_cap_path, HD_DIM *p_dim)
{
    HD_RESULT ret = HD_OK;
    {//select sensor mode, manually or automatically
        HD_VIDEOCAP_IN video_in_param = {0};
        video_in_param.sen_mode = HD_VIDEOCAP_SEN_MODE_AUTO; //auto select sensor mode by the
parameter of HD_VIDEOCAP_PARAM_OUT
        video_in_param.frc = HD_VIDEO_FRC_RATIO(30,1);
        video_in_param.dim.w = 1920;
        video_in_param.dim.h = 1080;
        video_in_param.pxlfmt = HD_VIDEO_PXLFMT_RAW12;
        video_in_param.out_frame_num = HD_VIDEOCAP_SEN_FRAME_NUM_1;
        ret = hd_videocap_set(video_cap_path, HD_VIDEOCAP_PARAM_IN, &video_in_param);
        //printf("set_cap_param MODE=%d\r\n", ret);
        if (ret != HD_OK) {
            return ret;
        }
    }
    {
        HD_VIDEOCAP_CROP video_crop_param = {0};
        video_crop_param.mode = HD_CROP_ON;
        video_crop_param.win.rect.x = 0;
        video_crop_param.win.rect.y = 0;
        video_crop_param.win.rect.w = 1920/2;
        video_crop_param.win.rect.h= 1080/2;
        video_crop_param.align.w = 4;
        video_crop_param.align.h = 4;
        ret = hd_videocap_set(video_cap_path, HD_VIDEOCAP_PARAM_OUT_CROP,
&video_crop_param);
        //printf("set_cap_param CROP ON=%d\r\n", ret);
```

```
    }
    {
        HD_VIDEOCAP_OUT video_out_param = {0};
        video_out_param.dim.w = 640;
        video_out_param.dim.h = 360;
        video_out_param.pxlfmt = HD_VIDEO_PXLFMT_RAW12;
        video_out_param.dir = HD_VIDEO_DIR_NONE;
        ret = hd_videocap_set(video_cap_path, HD_VIDEOCAP_PARAM_OUT, &video_out_param);
        //printf("set_cap_param OUT=%d\r\n", ret);
    }
    return ret;
}
```

3.5 Pattern Generation (for IPC only)

This function is mainly for debugging.
Setting HD_VIDEOCAP_SEN_PAT_GEN to the driver_name of HD_VIDEOCAP_DRV_CONFIG will activate pattern generation.
Example: color bar

```
HD_RESULT set_cap_cfg(void)
{
    HD_RESULT ret = HD_OK;
    HD_VIDEOCAP_DRV_CONFIG cap_cfg = {0};
    HD_PATH_ID video_cap_ctrl = 0;

    snprintf(cap_cfg.sen_cfg.sen_dev.driver_name, HD_VIDEOCAP_SEN_NAME_LEN-1,
HD_VIDEOCAP_SEN_PAT_GEN);
    ret = hd_videocap_open(0, HD_VIDEOCAP_0_CTRL, &video_cap_ctrl); //open this for device
control
    if (ret != HD_OK) {
        return ret;
    }
    ret |= hd_videocap_set(video_cap_ctrl, HD_VIDEOCAP_PARAM_DRV_CONFIG, &cap_cfg);
    return ret;
}
```

```
HD_RESULT set_cap_param(HD_PATH_ID video_cap_path, HD_DIM *p_dim)
{
     HD_RESULT ret = HD_OK;
     {
          HD_VIDEOCAP_IN video_in_param = {0};
          video_in_param.sen_mode = HD_VIDEOCAP_PATGEN_MODE(HD_VIDEOCAP_SEN_PAT_COLORBAR,
200);
          video_in_param.frc = HD_VIDEO_FRC_RATIO(30,1);
          video_in_param.dim.w = 1920;
          video_in_param.dim.h = 1080;
          ret = hd_videocap_set(video_cap_path, HD_VIDEOCAP_PARAM_IN, &video_in_param);
          //printf("set_cap_param MODE=%d\r\n", ret);
          if (ret != HD_OK) {
               return ret;
          }
     }
     //no crop, full frame
     {
          HD_VIDEOCAP_CROP video_crop_param = {0};

          video_crop_param.mode = HD_CROP_OFF;
          ret = hd_videocap_set(video_cap_path, HD_VIDEOCAP_PARAM_OUT_CROP,
&video_crop_param);
          //printf("set_cap_param CROP NONE=%d\r\n", ret);
     }
     //not support scaling in pattern generation mode
     {
          HD_VIDEOCAP_OUT video_out_param = {0};

          //without setting dim for no scaling, using original sensor out size
          video_out_param.pxlfmt = HD_VIDEO_PXLFMT_RAW12;
          video_out_param.dir = HD_VIDEO_DIR_NONE;
          ret = hd_videocap_set(video_cap_path, HD_VIDEOCAP_PARAM_OUT, &video_out_param);
          //printf("set_cap_param OUT=%d\r\n", ret);
     }
     return ret;
}
```

3.6 Output Directly to HD_VIDEOPRC (for IPC only)

It means that there is no need to reserve memory block for RAW(bayer pattern) data, but CA(for AWB) and LA(for AE) still needed. The following example shows how to enable this function and the most important is the sequence of invoking BIND/START/STOP between HD_VIDEOCAP and HD_VIDEOPRC.
Example: video_liveview_with_direct.c (only show the difference with non-direct mode)

```
D_RESULT set_cap_param(HD_PATH_ID video_cap_path, HD_DIM *p_dim)
{
     HD_VIDEOCAP_PATH_CONFIG video_path_param = {0};
     …
     …
     video_path_param.out_func = HD_VIDEOCAP_OUTFUNC_DIRECT;
     ret = hd_videocap_set(video_cap_path, HD_VIDEOCAP_PARAM_PATH_CONFIG, &video_path_param);
     return ret
}
HD_RESULT set_proc_cfg(HD_PATH_ID *p_video_proc_ctrl, HD_DIM* p_max_dim)
{
     HD_RESULT ret = HD_OK;
     HD_VIDEOPROC_DEV_CONFIG video_cfg_param = {0};
     HD_VIDEOPROC_CTRL video_ctrl_param = {0};
     HD_PATH_ID video_proc_ctrl = 0;
     …
     …
     {
          HD_VIDEOPROC_PATH_CONFIG video_path_param = {0};

          video_path_param.in_func = HD_VIDEOPROC_INFUNC_DIRECT; //direct NOTE: enable direct
          ret = hd_videoproc_set(video_proc_ctrl, HD_VIDEOPROC_PARAM_PATH_CONFIG,
&video_path_param);
          //printf("set_proc_param PATH_CONFIG=0x%X\r\n", ret);
     }
     …
```

```
    …
    return ret;
}
int main(int argc, char** argv)
{
    …
    // bind video_liveview modules (main)
    hd_videocap_bind(HD_VIDEOCAP_0_OUT_0, HD_VIDEOPROC_0_IN_0); //direct NOTE: ensure
videocap is binding to videoproc before they start
    hd_videoproc_bind(HD_VIDEOPROC_0_OUT_0, HD_VIDEOOUT_0_IN_0);

    // start video_liveview modules (main)
    hd_videoproc_start(stream[0].proc_path); //direct NOTE: ensure videoproc is start before
videocap
    hd_videocap_start(stream[0].cap_path); //direct NOTE: ensure videocap is start after
videoproc
    …
    …

    hd_videoproc_stop(stream[0].proc_path); //direct NOTE: ensure videoproc is stop before
videocap
    hd_videocap_stop(stream[0].cap_path); //direct NOTE: ensure videocap is stop after
videoproc
    …
    …
}
```

4 Debug command

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

4.1 Proc Command for IPC

4.1.1 cat /proc/hdal/vcap/info

The result will show the videocapture information by five parts.

1. **PATH & BIND**: bind status of hd_videocapture.
2. **DRV CONFIG**: sensor configuration, referring to HD_VIDEOCAP_SENSOR_DEVICE.
3. **CTRL**: ctrl function status, referring to HD_VIDEOCAP_CTRLFUNC.
4. **IN FRAME**: sensor mode configuration, referring to HD_VIDEOCAP_IN.
5. **OUT FRAME**: videocapture output setting, referring to HD_VIDEOCAP_CROP and HD_VIDEOCAP_OUT.

The parameters description of "**PATH & BIND**" can reference the below table.

Parameter	Description
in	Input port ID
out	Output port ID
state	Device status
bind_src	Input port binding source
bind_dest	Output port binding destination

The parameters description of "**DRV CONFIG**" can reference the below table.

Parameter	Description
driver_name	The file name of sensor driver
if_type	Interface type, referring to HD_COMMON_VIDEO_IN_TYPE
shdr_map	Sensor HDR mapping, referring to HD_VIDEOCAP_SEN_HDR_MAP
sensor_pinmux	Sensor pinmux, referring to top.h define (PIN_SENSOR_CFG/PIN_SENSOR2_CFG/.../PIN_SENSOR8_CFG)
serial_if_pinmux	Serial interface pinmux, referring to top.h define (PIN_MIPI_LVDS_CFG)
cmd_if_pinmux	Command interface pinmux, referring to top.h define (PIN_I2C_CFG/PIN_SIF_CFG)
clk_lane_sel	Clock lane selection, referring to

	HD_VIDEOCAP_SEN_CLANE_SEL
sen_2_serial_pin_map [0:7]	Serial (lvds/csi/slvsec) pin mapping
ccir_msblsb_switch	only for ccir sensor output HD mode (16 bits)
ccir_vd_hd_pin	CCIR601 need HW VD/HD output pin
vx1_tx241_cko_pin	ENABLE / DISABLE the THCV241's CKO pin output
vx1_tx241_cfg_2lane_mode	FALSE for 1 lane mode and TRUE for 2 lanes mode
vx1_en	Vx1 enabled or not, referring to HD_VIDEOCAP_SEN_IF_VX1
if_sel	Vx1 interface selection, referring to HD_VIDEOCAP_SEN_VX1_IF_SEL
ctl_sel	Vx1 controller selection, referring to HD_VIDEOCAP_SEN_VX1_CTL_SEL
tx_type	Vx1 tx module selection, referring to HD_VIDEOCAP_SEN_VX1_TXTYPE
tge_en	TGE enabled or not, referring to HD_VIDEOCAP_SEN_IF_TGE
swap	TGE information, swap CH1 & 5, or CH2 & 6, or CH3 & 7, or CH4 & 8.
vcap_vd_src	VIDEOCAP 0/2 VD/HD signal source, referring to HD_VIDEOCAP_SEN_TGE_CH_VD_TO_VCAP
optin_en	Option setting for sensor driver, referring to HD_VIDEOCAP_SEN_INIT_OPTION
sen_map_if	Sensor map interface, referring to HD_VIDEOCAP_SEN_MAP_IF
if_time_out	Sensor interface timeout, ms, default 1000ms if HD_VIDEOCAP_SEN_ENABLE_IF_TIMEOUT is not enabled

The parameters description of "**CTRL**" can reference the below table.

Parameter	Description
AE	AE is enabled or not, referring to HD_VIDEOCAP_CTRLFUNC
AWB	AWB is enabled or not, referring to HD_VIDEOCAP_CTRLFUNC
AF	AF is enabled or not, referring to HD_VIDEOCAP_CTRLFUNC
WDR	WDR is enabled or not, referring to HD_VIDEOCAP_CTRLFUNC
SHDR	SHDR is enabled or not, referring to HD_VIDEOCAP_CTRLFUNC
ETH	ETH is enabled or not, referring to HD_VIDEOCAP_CTRLFUNC

The parameters description of "**IN FRAME**" can reference the below table.

Parameter	Description
in	Input port ID

mode	Sensor mode selection, AUTO or manually indicate sensor mode.
w	Sensor output width, only valid when sen_mode is HD_VIDEOCAP_SEN_MODE_AUTO
h	Sensor output height, only valid when sen_mode is HD_VIDEOCAP_SEN_MODE_AUTO
pxlfmt	Sensor output pixel format, only valid when sen_mode is HD_VIDEOCAP_SEN_MODE_AUTO
frc	Sensor output frame rate
out_frame_num	Sensor output frame number, 1 for linear mode and 2/3/4 for sensor HDR mode.

The parameters description of "**OUT FRAME**" can reference the below table.

Parameter	Description
out	Output port ID
w	Videocapture output width
h	Videocapture output height
pxlfmt	Videocapture output pixel format
dir	Videocapture output direction, like mirror/flip, only valid when sensor is CCIR.
crop	Crop method and region.

The followings are common part of cat /proc/hdal/xxx/info

[OUT WORK STATUS]

Value	Description	value
out	output id	0~15
NEW	start job counter of output new stage (per second), also means VD count	
drop	dropping job counter of output new stage (per second)	
wrn	cancel by warning job counter of output new stage (per second)	
err	cancel by error job counter of output new stage (per second)	
PROC	start job counter of output process stage (per second), also means	

	VCAP get memory block successfully	
drop	dropping job counter of output process stage (per second)	
wrn	cancel by warning job counter of output process stage (per second)	
err	cancel by error job counter of output process stage (per second)	
PUSH	start job counter of output push stage (per second), also means VCAP output data to DRAM	
drop	dropping job counter of output push stage (per second)	
wrn	cancel by warning job counter of output push stage (per second)	
err	cancel by error job counter of output push stage (per second)	

[USER WORK STATUS]

Value	Description	value
out	output id	0~15
PULL	start job counter of user pull stage (per second)	
skip	skipping job counter of user pull stage (per second)	
wrn	cancel by warning job counter of user pull stage (per second)	
err	cancel by error job counter of user pull stage (per second)	
REL	finish job counter of user release stage (per second)	

Example:

```
-------------------------- VIDEOCAP 0  PATH & BIND -----------------------------
in     out     state   bind_src              bind_dest
```

```
0       0       START   (null)                  VIDEOPROC_0_IN_0
------------------------- VIDEOCAP 0  DRV CONFIG ------------------------------
driver_name     if_type shdr_map
nvt_sen_imx291  5       0
sensor_pinmux   serial_if_pinmux     cmd_if_pinmux clk_lane_sel
0x00000220      0x00000F04           0x00000010    0x00000001
sen_2_serial_pin_map[0:7]
0       1       -1      -1      -1      -1      -1      -1
ccir_msblsb_switch      ccir_vd_hd_pin
0                       0
vx1_tx241_cko_pin       vx1_tx241_cfg_2lane_mode
0                       0
vx1_en  if_sel  ctl_sel  tx_type
0       0       0        0
tge_en  swap    vcap_vd_src
0       0       0
optin_en        sen_map_if      if_time_out
0x0             0               0
------------------------- VIDEOCAP 0  CTRL -----------------------------------
AE      AWB     AF      WDR     SHDR    ETH
1       1       0       0       0       0
------------------------- VIDEOCAP 0  IN FRAME -------------------------------
in      mode    w       h       pxlfmt  frc     out_frame_num
0       AUTO    1920    1080    RAW12   30/1    1
------------------------- VIDEOCAP 0  OUT FRAME ------------------------------
out     w       h       pxlfmt  dir     crop
0       0       0       RAW12   ....    OFF
------------------------- VIDEOCAP 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
------------------------- VIDEOCAP 0  USER WORK STATUS -----------------------
out   PULL  drop  wrn   err   REL
0     0     0     0     0     0
```

4.1.2 debug command

```
[debug port]
```

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

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


[ Sample ]

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

this debug command can show more debug log on console

```
root@NVTEVM:~$ echo debug d0 o0 mfff > /proc/hdal/vcap/cmd

[  172.158130] debug i/o begin: "vdocap0".out[0], action mask=0xfff

root@NVTEVM:~$ hd_video_liveview 1

out_type 1

[  176.110944] hd_reset - begin

[  176.115701] hd_reset - end

HDAL_VERSION: 00010001:00010001

[  176.123359]  "vdocap0".ctrl: set param(80001011)=0

[  176.129074]  "vdocap0".ctrl: set param(80001013)=1601468014

[  176.135534]  "vdocap0".ctrl: set param(80001012)=768

out_type=1

[  176.194213]

[  176.194213]  "vdocap0".out[0]: open begin, state=0

[  176.201449]  "vdocap0".out[0]: cmd OPEN

[  176.222008]  "vdocap0".out[0]: open end, state=1

[  176.229018]  "vdocap0".ctrl: get param(8000101a)=@8be8ec00,size=16

[  176.236309]  "vdocap0".ctrl: set param(80001014)=-1


[  176.243171]  "vdocap0".out[0]: set vdo-winsize(0,0,0,0) vdo-aspect(92,-1947669504)

[  176.251775]  "vdocap0".ctrl: set param(80001017)=0

[  176.257445]  "vdocap0".out[0]: set vdo-size(0,0) vdo-format(410C0000) vdo-dir(0)

[  176.265731]  "vdocap0".out[0]: set param(8000101b)=0

[  176.271625]  "vdocap0".out[0]: set vdo-framerate(0,0)

##video_out_para[  176.277604]

[  176.277604]  "vdocap0".out[0]: bind begin, ("vdoprc0".in[0])

m w:960,h:240 52[  176.287057]  "vdocap0".out[0]: cmd CONNECT

0c0420 0

[  176.293393]  "vdocap0".out[0]: cmd RDYSYNC

[  176.299366]  "vdocap0".out[0]: bind end

[  176.304101]
```

```
[  176.304101]  "vdocap0".out[0]: start begin, state=1
[  176.311411]  "vdocap0".out[0]: cmd RDYSYNC
[  176.316393]  "vdocap0".out[0]: cmd START
[  176.488569]  "vdocap0".out[0]: start end, state=2
[  176.555368] ========== AE_CBMSG_PREVIEWSTABLE ==========
Enter q to exit
q
[  180.823296]
[  180.823296]  "vdocap0".out[0]: stop begin, state=2
[  180.830667]  "vdocap0".out[0]: cmd STOP
[  180.888649]  "vdocap0".out[0]: stop end, state=1
[  180.910634]
[  180.910634]  "vdocap0".out[0]: unbind begin, ("vdoprc0".in[0])
[  180.918908]  "vdocap0".out[0]: cmd DISCONNECT
[  180.924132]  "vdocap0".out[0]: unbind end
[  180.929054]
[  180.929054]  "vdocap0".out[0]: close begin, state=1
[  180.936363]  "vdocap0".out[0]: cmd CLOSE
[  180.954324]  "vdocap0".out[0]: close end, state=0
```

4.1.3 trace command

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

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

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

4.1.4 probe command

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

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

this probe command could print per-data status

```
root@NVTEVM:~$ echo probe d0 o0 mffff > /proc/hdal/vcap/cmd
[   29.504155] probe i/o begin: "vdocap0".out[0], action mask=0xffff
root@NVTEVM:~$ hd_video_liveview 1
out_type 1
[   32.135288] hd_reset - begin
[   32.140059] hd_reset - end
HDAL_VERSION: 00010001:00010001
out_type=1
##devcount 1
##video_out_param w:960,h:240 520c0420 0
[   32.342431] "vdocap0".out[0] - NEW - new -- h=94001fc0 size=002fc720 addr=94002000 OK
[   32.384469] "vdocap0".out[0] - NEW - new -- h=94324fc0 size=002fc720 addr=94325000 OK
[   32.417832] "vdocap0".out[0] - NEW - new -- h=94647fc0 size=00302720 addr=94648000 OK
[   32.451164] "vdocap0".out[0] - NEW - new -- h=94001fc0 size=00302720 addr=94002000 OK
[   32.460142] ========== AE_CBMSG_PREVIEWSTABLE ==========
[   32.484521] "vdocap0".out[0] - NEW - new -- h=94324fc0 size=00302720 addr=94325000 OK
[   32.517866] "vdocap0".out[0] - NEW - new -- h=94647fc0 size=00302720 addr=94648000 OK
[   32.551189] "vdocap0".out[0] - NEW - new -- h=94001fc0 size=00302720 addr=94002000 OK
[   32.584522] "vdocap0".out[0] - NEW - new -- h=94324fc0 size=00302720 addr=94325000 OK
[   32.617869] "vdocap0".out[0] - NEW - new -- h=94647fc0 size=00302720 addr=94648000 OK
[   32.651191] "vdocap0".out[0] - NEW - new -- h=94001fc0 size=00302720 addr=94002000 OK
[   32.684524] "vdocap0".out[0] - NEW - new -- h=94324fc0 size=00302720 addr=94325000 OK
[   32.717867] "vdocap0".out[0] - NEW - new -- h=94647fc0 size=00302720 addr=94648000 OK
[   32.751191] "vdocap0".out[0] - NEW - new -- h=94001fc0 size=00302720 addr=94002000 OK
[   32.784528] "vdocap0".out[0] - NEW - new -- h=94324fc0 size=00302720 addr=94325000 OK
[   32.817871] "vdocap0".out[0] - NEW - new -- h=94647fc0 size=00302720 addr=94648000 OK
[   32.851196] "vdocap0".out[0] - NEW - new -- h=94001fc0 size=00302720 addr=94002000 OK
[   32.884524] "vdocap0".out[0] - NEW - new -- h=94324fc0 size=00302720 addr=94325000 OK
[   32.917871] "vdocap0".out[0] - NEW - new -- h=94647fc0 size=00302720 addr=94648000 OK
[   32.951189] "vdocap0".out[0] - NEW - new -- h=94001fc0 size=00302720 addr=94002000 OK
[   32.984521] "vdocap0".out[0] - NEW - new -- h=94324fc0 size=00302720 addr=94325000 OK
[   33.017873] "vdocap0".out[0] - NEW - new -- h=94647fc0 size=00302720 addr=94648000 OK
[   33.051190] "vdocap0".out[0] - NEW - new -- h=94001fc0 size=00302720 addr=94002000 OK
```

```
[   33.084521] "vdocap0".out[0] - NEW - new -- h=94324fc0 size=00302720 addr=94325000 OK
[   33.117868] "vdocap0".out[0] - NEW - new -- h=94647fc0 size=00302720 addr=94648000 OK
[   33.151200] "vdocap0".out[0] - NEW - new -- h=94001fc0 size=00302720 addr=94002000 OK
[   33.184520] "vdocap0".out[0] - NEW - new -- h=94324fc0 size=00302720 addr=94325000 OK
[   33.217871] "vdocap0".out[0] - NEW - new -- h=94647fc0 size=00302720 addr=94648000 OK
q
[   33.251195] "vdocap0".out[0] - NEW - new -- h=94001fc0 size=00302720 addr=94002000 OK
[   33.284525] "vdocap0".out[0] - NEW - new -- h=94324fc0 size=00302720 addr=94325000 OK
[   33.317880] "vdocap0".out[0] - NEW - new -- h=94647fc0 size=00302720 addr=94648000 OK
[   33.351192] "vdocap0".out[0] - NEW - new -- h=94001fc0 size=00302720 addr=94002000 OK
[   33.384524] "vdocap0".out[0] - NEW - new -- h=94324fc0 size=00302720 addr=94325000 OK
Enter q to exit
[   33.417896] "vdocap0".out[0] - NEW - new -- h=94647fc0 size=00302720 addr=94648000 OK
```

4.1.5 perf command

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


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

this perf command could print data count per second

```
root@NVTEVM:~$ echo perf d0 o0 > /proc/hdal/vcap/cmd
[   31.246264] perf i/o begin: "vdocap0".out[0]
root@NVTEVM:~$ hd_video_liveview 1
out_type 1
[   35.007468] hd_reset - begin
[   35.012264] hd_reset - end
HDAL_VERSION: 00010001:00010001
out_type=1
##devcount 1
##video_out_param w:960,h:240 520c0420 0
[   35.288929] "vdocap0".out[0] Perf! -- (Video) 0 Frame/sec
[   35.323329] ========== AE_CBMSG_PREVIEWSTABLE ==========
Enter q to exit
[   36.289567] "vdocap0".out[0] Perf! -- (Video) 30 Frame/sec
```

```
[   37.289575] "vdocap0".out[0] Perf! -- (Video) 30 Frame/sec
[   38.289583] "vdocap0".out[0] Perf! -- (Video) 30 Frame/sec
[   39.289596] "vdocap0".out[0] Perf! -- (Video) 30 Frame/sec
[   40.289610] "vdocap0".out[0] Perf! -- (Video) 30 Frame/sec
[   41.289617] "vdocap0".out[0] Perf! -- (Video) 30 Frame/sec
```

4.1.6 save command

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

[ Sample ]
echo save d0 o0 2 > /proc/hdal/vcap/cmd
```

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

```
[  140.311213] save i/o begin: "vdocap0".out[0] count=1
[  140.340105] "vdocap0".out[0] Save -- h=94001fc0 t=0000000008827a01 (RAW:
1920x1080.410c0100)
[  140.379260] "vdocap0".out[0] Save --
//mnt//sd//isf_vdocap0_out[0]_410c0100_1920_1080_c79.raw ok
[  140.388954] save port end
```

4.2 Debug menu for IPC

The currently supported videocapture module debug menu is as below.

```
============================
 VIDEOCAP
----------------------------
 01 : dump info
----------------------------
```

User can choose the number to dump the status what you want. The dump result is just like the example shows on 4.1.

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

Proc command	Debug menu
cat /proc/hdal/vcap/info	dump videocapture information

4.3 Proc Command for NVR

It includes the available device and scaling ability of capture.

Example.

```
root@NVTEVM:~$ cat /proc/videograph/capture_channels
Cap Info:
scM: scan method, 0:interlace, 1:progressive
scl_h_max: max scaling height output size
dn_w: max scaling down width ratio
dn_h: max scaling down height ratio
dn_w_qt: max scaling down width quality ratio
dn_h_qt: max scaling down height quality ratio
-----------------------------------------------------------
id  chip paths fps scM resolution scl_h_max dn_w dn_h dn_w_qt dn_h_qt
0   0    8     30  1   1920x1080  4088     16   16   8      4
1   0    8     30  1   1920x1080  4088     16   16   8      4
2   0    8     30  1   1920x1080  4088     16   16   8      4
3   0    8     30  1   1920x1080  4088     16   16   8      4
4   0    8     25  1   1920x1080  4088     16   16   8      4
5   0    8     25  1   1920x1080  4088     16   16   8      4
6   0    8     25  1   1920x1080  4088     16   16   8      4
7   0    8     25  1   1920x1080  4088     16   16   8      4
root@NVTEVM:~$
```

4.4 Debug menu for NVR

The currently supported videocapture module debug menu is as below.

```
==============================
 VIDEOCAP
------------------------------
 01 : dump status
------------------------------
 254 : Quit
 255 : Return
------------------------------
01
Run: 01 : dump status
========================= VIDEOCAP 0  SYSCAP ========================
w      h      fps     scaling
1920    1080    30      4088
------------------------- VIDEOCAP 0  PATH & BIND -----------------------------
in     out     state   bind_src              bind_dest
0      0       START   -                     VIDEOPROC_0_IN_0
0      1       OPEN    -                     -
0      2       OPEN    -                     -
------------------------- VIDEOCAP 0  OUT FRAME ------------------------------
out     w       h       pxlfmt
0       960     540     YUV420_NVX3
------------------------- VIDEOCAP 0  PATH POOL ------------------------------
out     pool    ddr_id  count   max_count
0       0       0       4.0     3.0
==============================
```

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

Proc command	Debug menu
cat /proc/videograph/hdal_setting	dump hdal all devices information

5 Trouble shooting

5.1 Error Code for IPC

[hd_videocap_open]

HD_RESULT	Description
HD_ERR_NOT_AVAIL	Sensor driver is not installed, inserted, or driver_name for HD_VIDEOCAP_PARAM_DRV_CONFIG not matched.
HD_ERR_FAIL	Error returning form sensor driver, please send the log to us.
HD_ERR_INV	Improper or not supported value for HD_VIDEOCAP_PARAM_DRV_CONFIG.
HD_ERR_ALREADY_OPEN	The path has already opened.

Example:

```
ret = hd_videocap_open(HD_VIDEOCAP_0_IN_0, HD_VIDEOCAP_0_OUT_0, &cap_path);
if (ret) {
     printf("hd_videocap_open fail(%d)\n", ret);
}
```

[hd_videocap_start/ hd_videocap_stop]

HD_RESULT	Description
HD_ERR_INV	Invalid parameter.
HD_ERR_NOBUF	VIDEOCAP output buffer is insufficient.
HD_ERR_FAIL	Error returning from drivers, please send the log to us.

[hd_videocap_set for HD_VIDEOCAP_PARAM_DRV_CONFIG]

HD_RESULT	Description
HD_ERR_NOT_AVAIL	nvt_ctl_sen.ko is not inserted
HD_ERR_FAIL	Versions of HDAL and k_flow are not matched.

HD_ERR_INV	Invalid or not supported parameter.

[hd_videocap_set for HD_VIDEOCAP_PARAM_CTRL]

HD_RESULT	Description
HD_ERR_NOT_AVAIL	Invalid or not supported parameter.

[hd_videocap_set for HD_VIDEOCAP_PARAM_IN]

HD_RESULT	Description
HD_ERR_NOT_AVAIL	Invalid or not supported parameter.

5.2 Error Log for IPC

log	Possible cause
_isf_vdocap_do_open:no sen driver	1. Sensor driver is not inserted in S10_SysInit2. 2. Driver name set to HD_VIDEOCAP_PARAM_DRV_CONFIG is not matched in sensor driver, referring to sen_init_xxx. E.g. ctl_sen_reg_sendrv("nvt_sen_imx290", &reg_obj) in sen_init_imx290().
_isf_vdocap_do_setportstructen init_cfg failed(-2)!	Insert to many sensor drivers in S10_SysInit2.
_isf_vdocap_do_start:failed(-9) _isf_vdocap_do_start:RAW buf insufficient(0x?????)	The size of memory pool for vcap is too small. Please check the blk_size in mem_init(), especially the definition of VDO_SIZE_W and VDO_SIZE_H.
_isf_base_unit_push: "vdocap0".out[0] push to "vdoprc0".in[0] is failed (-54)	The in_max.dim for HD_VIDEOPROC_PARAM_DEV_CONFIG should be larger or equal to the VCAP output size.

6 FAQ for IPC

6.1 VCAP open or start failed

Please check the returned error code or error log and refer to the Chapter 5 “Trouble shooting”. If the trouble shooting doesn’t help, just try to open debug log using the following command and send to us.

For 68x:

echo 7 > /sys/module/kflow_videocap/parameters/isf_vdocap_debug_level

For 52x:

echo 7 > /sys/module/kflow_videocapture/parameters/isf_vdocap_debug_level

6.2 Minimum buffer requirement

The memory block size is based on the output format, such as HD_VIDEO_PXLFMT_RAW8, HD_VIDEO_PXLFMT_RAW12 or HD_VIDEO_PXLFMT_NRX12, and the ALG function of AW/AWB. One VCAP needs at least 2 buffer count and the count should be multiplied by n for n-framed SHDR mode. Just refer to our HDAL sample code in mem_init(), here are some examples.

Example for single sensor linear mode:

```
#define DBGINFO_BUFSIZE()     (0x200)
//RAW
#define VDO_RAW_BUFSIZE(w, h, pxlfmt)   (ALIGN_CEIL_4((w) * HD_VIDEO_PXLFMT_BPP(pxlfmt) / 8)
* (h))
//NRX: RAW compress: Only support 12bit mode
#define RAW_COMPRESS_RATIO ((7/12)*100)
#define VDO_NRX_BUFSIZE(w, h)          (ALIGN_CEIL_4(ALIGN_CEIL_64(w) * 12 / 8 *
RAW_COMPRESS_RATIO / 100 * (h)))
//CA for AWB
#define VDO_CA_BUF_SIZE(win_num_w, win_num_h) ALIGN_CEIL_4((win_num_w * win_num_h << 3) << 1)
//LA for AE
#define VDO_LA_BUF_SIZE(win_num_w, win_num_h) ALIGN_CEIL_4((win_num_w * win_num_h << 1) << 1)
static HD_RESULT mem_init(void)
{
    HD_RESULT            ret;
```

```
    HD_COMMON_MEM_INIT_CONFIG mem_cfg = {0};


    // config common pool (cap)
    mem_cfg.pool_info[0].type = HD_COMMON_MEM_COMMON_POOL;
    mem_cfg.pool_info[0].blk_size = DBGINFO_BUFSIZE()+VDO_RAW_BUFSIZE(VDO_SIZE_W,
VDO_SIZE_H, CAP_OUT_FMT)
    +VDO_CA_BUF_SIZE(CA_WIN_NUM_W, CA_WIN_NUM_H)
    +VDO_LA_BUF_SIZE(LA_WIN_NUM_W, LA_WIN_NUM_H);
    mem_cfg.pool_info[0].blk_cnt = 2;
    mem_cfg.pool_info[0].ddr_id = DDR_ID0;
    // config common pool (main)
    …
    …
    ret = hd_common_mem_init(&mem_cfg);
    return ret;
}
```

Example for single sensor 2-framed SHDR mode:

```
…//the other macros are the same with above example
#define CAP_OUT_FMT      HD_VIDEO_PXLFMT_RAW12_SHDR2
static HD_RESULT mem_init(void)
{
    HD_RESULT            ret;
    HD_COMMON_MEM_INIT_CONFIG mem_cfg = {0};


    // config common pool (cap)
    mem_cfg.pool_info[0].type = HD_COMMON_MEM_COMMON_POOL;
    mem_cfg.pool_info[0].blk_size = DBGINFO_BUFSIZE()+VDO_RAW_BUFSIZE(VDO_SIZE_W,
VDO_SIZE_H, CAP_OUT_FMT)
    +VDO_CA_BUF_SIZE(CA_WIN_NUM_W, CA_WIN_NUM_H)
    +VDO_LA_BUF_SIZE(LA_WIN_NUM_W, LA_WIN_NUM_H);
    mem_cfg.pool_info[0].blk_cnt = 2*2;
    mem_cfg.pool_info[0].ddr_id = DDR_ID0;
    // config common pool (main)
    …
    …
```

```
    ret = hd_common_mem_init(&mem_cfg);
    return ret;
}
```

Example for single sensor linear direct mode (52x only):

```
//almost the same with non-direct mode, just don't need raw buffer size of VDO_RAW_BUFSIZE()
static HD_RESULT mem_init(void)
{
    HD_RESULT            ret;
    HD_COMMON_MEM_INIT_CONFIG mem_cfg = {0};

    // config common pool (cap)
    mem_cfg.pool_info[0].type = HD_COMMON_MEM_COMMON_POOL;
    mem_cfg.pool_info[0].blk_size = DBGINFO_BUFSIZE()+
    +VDO_CA_BUF_SIZE(CA_WIN_NUM_W, CA_WIN_NUM_H)
    +VDO_LA_BUF_SIZE(LA_WIN_NUM_W, LA_WIN_NUM_H);
    mem_cfg.pool_info[0].blk_cnt = 2;
    mem_cfg.pool_info[0].ddr_id = DDR_ID0;
    // config common pool (main)
    …
    …
    ret = hd_common_mem_init(&mem_cfg);
    return ret;
}
```

6.3 How to check if frame is dropped in VCAP

Using the following command and check “OUT WORK STATUS”.

cat /proc/hdal/vcap/info

case 1: no dropping in VCAP

```
------------------------ VIDEOCAP 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
```

The values of NEW, PROC, PUSH are all equal. If the value is not fit the target frame

rate, please check the sensor driver or ISP/AE setting.

case 2: frame dropped by getting memory block failed

```
------------------------ VIDEOCAP 0  OUT WORK STATUS -------------------------
out   NEW   drop  wrn   err   PROC  drop  wrn   err   PUSH  drop  wrn   err
0     30    0     10    0     20    0     0     0     20    0     0     0
```

This case shows there are 10 frames dropped in one second and the reason is VCAP can't get memory block. It means the memory block counts for VCAP are not enough or the memory pool for VCAP is occupied by other application. One can use "cat /proc/hdal/comm/info" to check.

```
----------------------- COMMON POOL-----------------------
PoolId     PoolType   DDR     PhyAddr      VirAddr      BlkSize     BlkCnt  Free  MinFree
   0          0x01     1  0x12802000   0x94002000   0x002FC800        2     0       0
    BlkId      BlkHdl      BlkAddr     WantSize   user vdocap vdoprc vdoout
      0    0x94001FC0   0x94002000   0x002FC740      1     1     0     0
…
…
-------------------------ERR STATUS ----------------------
Modules:         user   vdocap   vdoprc   vdoout
Get blk Fail:       0     237       0       0
Loc blk Fail:       0       0       0       0
Unl blk Fail:       0       0       0       0
```

The above sample shows user application occupied one block in Poold0 which is dedicated for VCAP.

case 3: frame dropped by VD timing change

```
------------------------ VIDEOCAP 0  OUT WORK STATUS -------------------------
out   NEW   drop  wrn   err   PROC  drop  wrn   err   PUSH  drop  wrn   err
0     30    0     0     0     30    0     4     0     26    0     0     0
```

This case shows there are 4 frames dropped in one second and the reason is RAW data timing NG. Please try to lock or disable AE and see if above dropping frames still happened. If no, it means AE causes frame dropped. If yes, maybe there is something wrong in sensor board.