



**Horizon
Robotics**

MU-2520-6-X3J3 Platform System Software

AUTO Media Interface Manual

v1.5

2021-02

Horizon Robotics
版权所有 禁止转载 算法工具部 杨柳军 2021-04-01 10:02:46

©2020 Horizon Robotics

All rights reserved

This document contains the information related to the product under development. Horizon Robotics reserves the right to alter or discontinue the product without prior notice.

Disclaimer

This document is only used to help system and software users to use products of Horizon Robotics. It does not expressly or implicitly authorize others to design or manufacture any integrated circuit based on the information in this document.

The information in this document is subject to change without notice. Although this document ensures the accuracy of the content as much as possible, all statements, information and suggestions in this document do not constitute any express or implied warranties, representations or statements.

All information in this document is provided "as is". Horizon Robotics makes no express or implied warranties, representations or statements with respect to the merchantability, suitability and non-infringement of any third-party intellectual property rights of its products for any particular purpose. Horizon Robotics shall not be liable for any liabilities arising from the use of the product, including but not limited to direct or indirect damages.

The buyer and other parties (hereinafter referred to as "users") who perform the development based on the products of Horizon Robotics, understand and agree that users shall bear the responsibility of independent analysis, evaluation and judgment in the design of product applications. Users shall take full responsibility for the safety of their applications (including all products of Horizon Robotics used for their applications) and ensure that they meet the requirements of all applicable regulations, laws, and other regulations. The "typical" parameters provided in product introductions and product specifications of Horizon Robotics may vary in different applications, moreover, actual performance may vary over time. All working parameters (including "typical" parameters) must be verified by user for each user application.

Users agree to indemnify Horizon Robotics and its representatives in full for any claims, damages, costs, losses and/or liabilities caused by users' unauthorized use of products of Horizon Robotics or non-compliance with the terms of these instructions.

Copyright © 2020

Horizon Robotics

<https://www.horizon.ai>

Revision Record

The revision record lists the major changes that have occurred between versions of each document. The following table lists the technical content of each document update.

Version	Revision Date	Description
0.1	2020-3-24	First creation.
0.5	2020-4-15	add new camera mclk api 2.1.11,2.1.12
0.6	2020-4-16	add new gdc cfg bin api 4.7.3,4.7.4,4.7.5
0.7	2020-6-2	add new osd draw api 4.5.7
0.8	2020-6-23	add code api 5.2.59~5.2.64 &5.1.8
1.0	2020-8-3	Description of adding lpwm to camera configuration file
1.1	2020-8-24	add sensor dynamic mode for camera Improve interface definition Remove untested interfaces
1.2	2020-8-24	Fix MediaCodec/MediaMuxer/MediaRecorder description
1.3	2020-12-16	Add description about decoder mode
1.4	2020-12-21	update vio data flow and new api (gdc), osd removed.
1.5	2021-1-5	Add more jpeg striction and more mediaCode API

Table of Content

Revision Record.....	ii
Table of Content.....	iii
1 Video System Initialization.....	1
1.1 VIO Sys API.....	1
1.1.1 hb_vio_init.....	1
1.1.2 hb_vio_deinit	5
1.1.3 hb_vio_start.....	10
1.1.4 hb_vio_stop.....	15
1.1.5 hb_vio_start_pipeline	19
1.1.6 hb_vio_stop_pipeline	24
2 Camera Input	30
2.1 Camera Application API.....	30
2.1.1 hb_cam_init.....	30
2.1.2 hb_cam_deinit	31
2.1.3 hb_cam_start	32
2.1.4 hb_cam_stop.....	32
2.1.5 hb_cam_start_all.....	33
2.1.6 hb_cam_stop_all.....	33
2.1.7 hb_cam_power_on.....	34
2.1.8 hb_cam_power_off.....	34
2.1.9 hb_cam_reset	35
2.1.10 hb_cam_set_mclk.....	35
2.1.11 hb_cam_enable_mclk.....	36
2.1.12 hb_cam_disable_mclk	37
2.2 Camera Control API.....	38
2.2.1 hb_cam_i2c_read.....	38
2.2.2 hb_cam_i2c_read_byte	40
2.2.3 hb_cam_i2c_write.....	40
2.2.4 hb_cam_i2c_write_byte	41

2.2.5	hb_cam_i2c_block_write	41
2.2.6	hb_cam_i2c_block_read.....	42
2.2.7	hb_cam_dynamic_switch_fps	43
2.2.8	hb_cam_dynamic_switch_mode.....	43
2.3	SIF API.....	44
2.3.1	hb_vio_raw_dump.....	44
2.3.2	hb_vio_raw_feedback.....	44
2.4	Camera Return Value	47
2.5	Camera Configuration File Description.....	48
2.6	Camera Configuration File (Ver. 2.0).....	49
2.7	Camera Configurable Environment Variable	50
3	Audio Input and Output.....	51
3.1	Audio Service Description.....	51
3.1.1	API Usage Background	51
3.1.2	API Provision Scheme	51
3.1.3	Sample code and usage of API	51
3.2	API Definition.....	51
3.2.1	pcm_open	51
3.2.2	pcm_write	52
3.2.3	pcm_read	53
3.2.4	pcm_close	55
3.2.5	mixer_open.....	55
3.2.6	mixer_get_ctl.....	56
3.2.7	mixer_ctl_set_value.....	57
3.2.8	mixer_close.....	58
3.3	Definition and Description of Data Structure.....	58
4	Video Processing.....	60
4.1	VPS API Procedure (X3/J3)	60
4.1.1	VPS supports single input.....	60
4.1.2	VPS supports multi-channel data input.....	65
4.1.3	Fillback interface procedure	66
4.2	VPS API	69

4.2.1	hb_vio_get_info.....	69
4.2.2	hb_vio_get_info_conditional.....	76
4.2.3	hb_vio_set_info.....	81
4.2.4	hb_vio_free_info.....	86
4.2.5	hb_vio_free.....	90
4.2.6	hb_vio_src_free.....	95
4.2.7	hb_vio_mult_free.....	100
4.2.8	hb_vio_mult_src_free.....	104
4.2.9	hb_vio_set_param.....	109
4.2.10	hb_vio_get_param.....	114
4.2.11	hb_vio_get_data.....	119
4.2.12	hb_vio_get_data_conditional.....	127
4.2.13	hb_vio_set_callbacks.....	132
4.3	ISP API.....	140
4.3.1	hb_isp_get_ae_statistics	140
4.3.2	hb_isp_release_ae_statistics	141
4.3.3	hb_isp_get_awb_statistics	142
4.3.4	hb_isp_release_awb_statistics	142
4.3.5	hb_isp_set_context.....	144
4.3.6	hb_isp_get_context	145
4.3.7	hb_isp_dev_init	146
4.3.8	hb_isp_dev_deinit.....	146
4.3.9	hb_isp_iridix_ctrl.....	147
4.3.10	ISP Parameter Description	149
4.4	IPU API.....	150
4.4.1	hb_vio_free_ipubuf	150
4.5	PYM API.....	154
4.5.1	hb_vio_pym_process.....	154
4.5.2	hb_vio_mult_pym_process.....	161
4.5.3	hb_vio_run_pym	166
4.5.4	hb_vio_free_pymbuf.....	175
4.6	GDC API.....	179

4.6.1	hb_vio_run_gdc.....	179
4.6.2	hb_vio_run_gdc_opt.....	184
4.6.3	hb_vio_free_gdcbuf.....	189
4.6.4	hb_vio_gen_gdc_cfg.....	194
4.6.5	hb_vio_free_gdc_cfg.....	199
4.6.6	hb_vio_set_gdc_cfg.....	203
4.6.7	hb_vio_set_gdc_cfg_opt.....	208
4.7	VIO Parameter Descriptions.....	213
4.7.1	X2/J2 main parameters	213
4.7.2	X3/J3 main parameters	214
4.8	VIO Return Code	218
4.8.1	X2/J2 return code.....	218
4.8.2	X3/J3 return code.....	219
4.9	VIO Configuration File.....	223
4.9.1	X2/J2 configuration file descriptions	223
4.9.2	X3/J3 configuration file descriptions	233
4.10	VIO Accessable Environment Variable.....	244
4.10.1	X3/J3 environment variable.....	244
5	Encoding and Decoding.....	245
5.1	MediaCodec Interface Descriptions	245
5.1.1	GOP.....	246
5.1.2	Long-term Reference Frame	252
5.1.3	Intra Refresh.....	253
5.1.4	Bitrate Control.....	253
5.1.5	ROI	253
5.1.6	Smart background encoding.....	254
5.1.7	Frame Skip Settings.....	255
5.1.8	JPEG codec limit.....	255
5.2	MediaCodec API.....	256
5.2.1	hb_mm_mc_get_descriptor	256
5.2.2	hb_mm_mc_get_default_context.....	256
5.2.3	hb_mm_mc_initialize	257

5.2.4 hb_mm_mc_set_callback.....	263
5.2.5 hb_mm_mc_set_vlc_buffer_listener	264
5.2.6 hb_mm_mc_configure.....	264
5.2.7 hb_mm_mc_start.....	265
5.2.8 hb_mm_mc_stop	266
5.2.9 hb_mm_mc_pause	266
5.2.10 hb_mm_mc_flush.....	267
5.2.11 hb_mm_mc_release.....	268
5.2.12 hb_mm_mc_get_state.....	268
5.2.13 hb_mm_mc_get_status.....	269
5.2.14 hb_mm_mc_queue_input_buffer.....	270
5.2.15 hb_mm_mc_dequeue_input_buffer.....	275
5.2.16 hb_mm_mc_queue_output_buffer.....	276
5.2.17 hb_mm_mc_dequeue_output_buffer.....	276
5.2.18 hb_mm_mc_get_longterm_ref_mode	277
5.2.19 hb_mm_mc_set_longterm_ref_mode.....	290
5.2.20 hb_mm_mc_get_intra_refresh_config	290
5.2.21 hb_mm_mc_set_intra_refresh_config.....	291
5.2.22 hb_mm_mc_get_rate_control_config	292
5.2.23 hb_mm_mc_set_rate_control_config	292
5.2.24 hb_mm_mc_get_deblk_filter_config.....	293
5.2.25 hb_mm_mc_set_deblk_filter_config	293
5.2.26 hb_mm_mc_get_sao_config.....	294
5.2.27 hb_mm_mc_set_sao_config	295
5.2.28 hb_mm_mc_get_entropy_config.....	295
5.2.29 hb_mm_mc_set_entropy_config	296
5.2.30 hb_mm_mc_get_vui_timing_config.....	297
5.2.31 hb_mm_mc_set_vui_timing_config	297
5.2.32 hb_mm_mc_get_slice_config.....	298
5.2.33 hb_mm_mc_set_slice_config	299
5.2.34 hb_mm_mc_insert_user_data.....	299
5.2.35 hb_mm_mc_request_idr_frame	300

5.2.36 hb_mm_mc_skip_pic.....	301
5.2.37 hb_mm_mc_get_smart_bg_enc_config.....	301
5.2.38 hb_mm_mc_set_smart_bg_enc_config.....	302
5.2.39 hb_mm_mc_get_pred_unit_config.....	303
5.2.40 hb_mm_mc_set_pred_unit_config.....	303
5.2.41 hb_mm_mc_get_transform_config.....	304
5.2.42 hb_mm_mc_set_transform_config.....	305
5.2.43 hb_mm_mc_get_roi_config	305
5.2.44 hb_mm_mc_set_roi_config.....	306
5.2.45 hb_mm_mc_get_mode_decision_config.....	307
5.2.46 hb_mm_mc_set_mode_decision_config	307
5.2.47 hb_mm_mc_get_user_data	308
5.2.48 hb_mm_mc_release_user_data.....	309
5.2.49 hb_mm_mc_get_mjpeg_config.....	309
5.2.50 hb_mm_mc_set_mjpeg_config	310
5.2.51 hb_mm_mc_get_jpeg_config.....	311
5.2.52 hb_mm_mc_set_jpeg_config.....	311
5.2.53 hb_mm_mc_get_fd	312
5.2.54 hb_mm_mc_close_fd.....	317
5.2.55 hb_mm_mc_set_camera	318
5.2.56 hb_mm_mc_get_vui_config.....	319
5.2.57 hb_mm_mc_set_vui_config	319
5.2.58 hb_mm_mc_get_3dnr_enc_config.....	320
5.2.59 hb_mm_mc_set_3dnr_enc_config	321
5.2.60 hb_mm_mc_request_idr_header	321
5.2.61 hb_mm_mc_enable_idr_frame.....	322
5.2.62 hb_mm_mc_register_audio_encoder.....	323
5.2.63 hb_mm_mc_unregister_audio_encoder	323
5.2.64 hb_mm_mc_register_audio_decoder.....	324
5.2.65 hb_mm_mc_unregister_audio_decoder	324
5.2.66 hb_mm_mc_get_explicit_header_config	325
5.2.67 hb_mm_mc_set_explicit_header_config	326

5.2.68 hb_mm_mc_get_roi_avg_qp.....	326
5.2.69 hb_mm_mc_set_roi_avg_qp.....	327
5.3 Descriptions of MediaCodec Main Parameters	327
5.4 MediaMuxer Interface Descriptions.....	336
5.5 MediaMuxer API	336
5.5.1 hb_mm_mx_get_default_context.....	336
5.5.2 hb_mm_mx_initialize.....	340
5.5.3 hb_mm_mx_add_stream	341
5.5.4 hb_mm_mx_start.....	341
5.5.5 hb_mm_mx_stop	342
5.5.6 hb_mm_mx_write_stream.....	343
5.5.7 hb_mm_mx_get_state.....	343
5.6 Descriptions of MediaMuxer Main Parameters.....	344
5.7 MediaRecorder Interface Descriptions.....	345
5.8 MediaRecorder API.....	346
5.8.1 hb_mm_mr_get_default_context	346
5.8.2 hb_mm_mr_initialize	354
5.8.3 hb_mm_mr_set_listener	355
5.8.4 hb_mm_mr_get_mr_video_source	356
5.8.5 hb_mm_mr_set_mr_video_source	356
5.8.6 hb_mm_mr_get_mr_audio_source.....	357
5.8.7 hb_mm_mr_set_mr_audio_source	358
5.8.8 hb_mm_mr_set_camera	358
5.8.9 hb_mm_mr_set_camera_pym.....	359
5.8.10 hb_mm_mr_configure.....	360
5.8.11 hb_mm_mr_prepare.....	360
5.8.12 hb_mm_mr_start.....	361
5.8.13 hb_mm_mr_stop	362
5.8.14 hb_mm_mr_release	362
5.8.15 hb_mm_mr_get_state	363
5.9 Descriptions of MediaRecorder Main Parameters	364
5.10 Media Return Code.....	366

6	Display Output System.....	367
6.1	Display Subsystem Description.....	367
6.2	DISP API Definition	368
6.2.1	hb_disp_init_cfg	368
6.2.2	hb_disp_init.....	369
6.2.3	hb_disp_start	371
6.2.4	hb_disp_stop.....	371
6.2.5	hb_disp_close.....	371
6.2.6	hb_disp_layer_on.....	372
6.2.7	hb_disp_layer_off	374
6.2.8	hb_disp_set_video_channel.....	374
6.2.9	hb_disp_set_video_display_ddr_layer.....	375
6.2.10	hb_disp_set_vio_channel	375
6.2.11	hb_disp_set_lcd_backlight	376
6.2.12	hb_disp_get_output_cfg	376
6.2.13	hb_disp_set_output_cfg	380
6.2.14	hb_disp_get_upscaling_cfg	381
6.2.15	hb_disp_set_upscaling_cfg.....	381
6.2.16	hb_disp_get_channel_cfg	381
6.2.17	hb_disp_set_channel_cfg	382
6.2.18	hb_set_layer_cfg	382
6.2.19	hb_set_video_bufaddr	383
6.2.20	hb_check_video_bufaddr_valid	383
6.2.21	hb_disp_set_timing.....	384
6.2.22	hb_disp_out_upscale.....	384
6.2.23	hb_disp_get_gamma_cfg	384
6.2.24	hb_disp_set_gamma_cfg	385
6.2.25	hb_disp_wb_start	385
6.2.26	hb_get_disp_done.....	385
6.2.27	hb_disp_wb_stop.....	386
6.2.28	hb_disp_get_screen_frame.....	386
6.2.29	hb_disp_release_screen_frame	386

6.2.30	hb_disp_wb_setcfg.....	387
6.3	DISP Parameter Descriptions	388
6.4	DISP Configuration File Descriptions	394
7	Appendix.....	398

Horizon Robotics
版权所有 禁止转载 算法工具部 杨帆 2021-04-01 10:02:46

1 Video System Initialization

1.1 VIO Sys API

1.1.1 hb_vio_init

[Function Declaration]

```
int hb_vio_init( const char *cfg_file)
```

[Parameter Description]

- [IN]const char *cfg_file: VIO module configuration files

[Return Value]

- Success: Returns HB_OK
- Failure: Returns negative error code (refer to [X3/J3 main](#) parameters)

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
}
```

```

    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DS0_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK,// 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
}

```

```

HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]      addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output

typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

```

```

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment

typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; ///< Output window position and size
    transformation_t transform; ///< Used transformation
    rect_t input_roi_r;
    int32_t pan; ///< Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; ///< Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; ///< Target zoom dimensionless coefficient (must not be bigger
    than zero)
    double strength; ///< Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double strengthY; ///< Dimensionless non-negative parameter defining the strength
    of transformation along X axis
}

```

```

    double angle;           ///< Angle of main projection axis rotation around itself in
degrees

    // universal transformation

    double elevation;      ///< Angle in degrees which specify the main projection axis

    double azimuth;        ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)

    int32_t keep_ratio;    ///< Keep the same stretching strength in both horizontal and
vertical directions

    double FOV_h;          ///< Size of output field of view in vertical dimension in
degrees

    double FOV_w;          ///< Size of output field of view in horizontal dimension in
degrees

    double cylindricity_y;  ///< Level of cylindricity for target projection shape in
vertical direction

    double cylindricity_x;  ///< Level of cylindricity for target projection shape in
horizontal direction

    char custom_file[128];  ///< File name of the file containing custom transformation
description

    custom_transformation_t custom;  ///< Parsed custom transformation structure

    double trapezoid_left_angle;    ///< Left Acute angle in degrees between trapezoid base and leg

    double trapezoid_right_angle;   ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;
}

```

● VIO Return Code)

[Function Description]

Initialize the VIO module and enable it to match the CFG configuration. The configuration file for the multi-channel scenario contains the multi-channel configuration.

[Compatibility]

System version: 1.1 and above. Note: The input configuration files may vary in different versions (1.1/1.2).

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 4.2.11hb_vio_get_data

1.1.2 hb_vio_deinit

[Function Declaration]

int hb_vio_deinit()

[Parameter Description]

None

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to X3/J3 main parameters)
- System version: 2.0 and above.
- typedef struct address_info_s {

```

        uint16_t width;// Image data width
        uint16_t height;// Image data width
        uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
        char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
        uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.

    } address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
}

```

```

HB_VIO_IPU_US_DATA, //ipu us channel data type
HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
HB_VIO_PYM_DATA,// Pyramid output data
HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
HB_VIO_GDC_DATA,//gdc output data type
HB_VIO_IARWB_DATA,//iar display data type
HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
HB_VIO_PYM_LAYER_DATA,//pym all layer data
HB_VIO_MD_DATA,//motion detect data
HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
HB_VIO_PYM_COMMON_DATA,//pym base layer data
HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

```

```

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

```

```

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

```

```

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,

```

```

HB_VIO_IPU_DS4_IMG_INFO,
HB_VIO_PYM_IMG_INFO, //pym info
HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
HB_VIO_INFO_MAX

} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]          size[num]          addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)]  size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]      addr[0(y),1(c)]
 * pym buf : fd[0(all channel)]  size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

```

```

/**
 * _gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/** 
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double angle; // Angle of main projection axis rotation around itself in
degrees
    // universal transformation
    double elevation; // Angle in degrees which specify the main projection axis
    double azimuth; // Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio; // Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h; // Size of output field of view in vertical dimension in
degrees
    double FOV_w; // Size of output field of view in horizontal dimension in
degrees
}

```

```

        double cylindricity_y;           ///// Level of cylindricity for target projection shape in
vertical direction
        double cylindricity_x;           ///// Level of cylindricity for target projection shape in
horizontal direction
        char custom_file[128];          ///// File name of the file containing custom transformation
description
        custom_transformation_t custom;  ///// Parsed custom transformation structure
        double trapezoid_left_angle;    ///// Left Acute angle in degrees between trapezoid base and leg
        double trapezoid_right_angle;   ///// Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code

[Function Description]

Release various resources, memory and so forth used in the initialization.

[Compatibility]

System version: 1.1 and above

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 4.2.11hb_vio_get_data

1.1.3 hb_vio_start

[Function Declaration]

int hb_vio_start()

[Parameter Description]

None

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to X3/J3 main parameters)
- System version: 2.0 and above.
- typedef struct address_info_s {

```

        uint16_t width;// Image data width
        uint16_t height;// Image data width
        uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
        char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
        uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

```

```

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */

typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
}

```

```

HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX

} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO, //isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO, //not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

```

```

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]    addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */

// normal capture buffer type, one image data output

typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;

/***
 * Window definition and transformation parameters
*/

```

```

* For parameters meaning read GDC guide
*/
typedef struct {
    rect_t out_r;           ///< Output window position and size
    transformation_t transform;   ///< Used transformation
    rect_t input_roi_r;
    int32_t pan;           ///< Target shift in horizontal direction from centre of the
output image in pixels
    int32_t tilt;           ///< Target shift in vertical direction from centre of the output
image in pixels
    double zoom;           ///< Target zoom dimensionless coefficient (must not be bigger
than zero)
    double strength;        ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double strengthY;       ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double angle;           ///< Angle of main projection axis rotation around itself in
degrees
    // universal transformation
    double elevation;        ///< Angle in degrees which specify the main projection axis
    double azimuth;          ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio;      ///< Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h;           ///< Size of output field of view in vertical dimension in
degrees
    double FOV_w;           ///< Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y;   ///< Level of cylindricity for target projection shape in
vertical direction
    double cylindricity_x;   ///< Level of cylindricity for target projection shape in
horizontal direction
    char custom_file[128];    ///< File name of the file containing custom transformation
description
    custom_transformation_t custom;  ///< Parsed custom transformation structure
    double trapezoid_left_angle;  ///< Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle;  ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code

[Function Description]

Start all channel works of VIO, corresponding to the data channels in the initialization.

[Compatibility]

System version: 1.1 and above

[Sample Code] Refer to [4.2.1 hb_vio_get_info Sample Code](#)

1.1.4 hb_vio_stop

[Function Declaration]

```
int hb_vio_stop()
```

[Parameter Description]

None

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to X3/J3 main parameters)
- System version: 2.0 and above.
- typedef struct address_info_s {

```

        uint16_t width;// Image data width
        uint16_t height;// Image data width
        uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
        char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
        uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.

    } address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
}
```

```

    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DS0_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK,// 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
}

```

```

HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]      addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output

typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

```

```

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment

typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
    than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
    of transformation along X axis
}

```

```

    double angle;           ///< Angle of main projection axis rotation around itself in
degrees

    // universal transformation

    double elevation;      ///< Angle in degrees which specify the main projection axis

    double azimuth;        ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)

    int32_t keep_ratio;    ///< Keep the same stretching strength in both horizontal and
vertical directions

    double FOV_h;          ///< Size of output field of view in vertical dimension in
degrees

    double FOV_w;          ///< Size of output field of view in horizontal dimension in
degrees

    double cylindricity_y;  ///< Level of cylindricity for target projection shape in
vertical direction

    double cylindricity_x;  ///< Level of cylindricity for target projection shape in
horizontal direction

    char custom_file[128];  ///< File name of the file containing custom transformation
description

    custom_transformation_t custom;  ///< Parsed custom transformation structure

    double trapezoid_left_angle;    ///< Left Acute angle in degrees between trapezoid base and leg

    double trapezoid_right_angle;   ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code

[Function Description]

Stop all channel works of VIO, corresponding to the data channels in the initialization.

[Compatibility]

System version: 1.1 and above

Hardware: X2/J2; X3/J3

[Sample Code] Refer to [4.2.1 hb_vio_get_info Sample Code](#)

1.1.5 hb_vio_start_pipeline

[Function Declaration]

```
int hb_vio_start_pipeline(uint32_t pipeline_id)
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Represents the software data channel to be set, which should correspond to the enabling, with a range of 0 ~ 7

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to X3/J3 main parameters)
- System version: 2.0 and above.
- typedef struct address_info_s {

```

        uint16_t width;// Image data width
        uint16_t height;// Image data width
        uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
        char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
        uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.

} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES];// Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type;//Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
}

```

```

HB_VIO_PYM_DATA,// Pyramid output data
HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
HB_VIO_GDC_DATA,//gdc output data type
HB_VIO_IARWB_DATA,//iar display data type
HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
HB_VIO_PYM_LAYER_DATA,//pym all layer data
HB_VIO_MD_DATA,//motion detect data
HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
HB_VIO_PYM_COMMON_DATA,//pym base layer data
HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

```

```

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

```

```

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
}

```

```

HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]   size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**

```

```

* gdc Common parameters
*/
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
*/
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
of transformation along Y axis
    double angle; // Angle of main projection axis rotation around itself in
degrees
    // universal transformation
    double elevation; // Angle in degrees which specify the main projection axis
    double azimuth; // Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio; // Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h; // Size of output field of view in vertical dimension in
degrees
    double FOV_w; // Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y; // Level of cylindricity for target projection shape in
vertical direction

```

```

    double cylindricity_x;           ///// Level of cylindricity for target projection shape in horizontal direction
    char custom_file[128];          ///// File name of the file containing custom transformation description
    custom_transformation_t custom;  ///// Parsed custom transformation structure
    double trapezoid_left_angle;    ///// Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle;   ///// Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code

[Function Description]

Start the work of VIO pipe channel, corresponding to the data channel in the initialization.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to 4.2.11hb_vio_get_data

1.1.6 hb_vio_stop_pipeline

[Function Declaration]

```
int hb_vio_stop_pipeline(uint32_t pipeline_id)
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Represents the software data channel to be set, which should correspond to the enabling

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to X3/J3 main parameters)
- System version: 2.0 and *above*.
- typedef struct address_info_s {

```

    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;
/* info :

```

```

* y,uv           2 plane
* raw            1 plane
* raw, raw       2 plane(dol2)
* raw, raw, raw  3 plane(dol3)
*/
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
}

```

```

HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX

} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK

} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO, //isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO, //not used in XJ3
    HB_VIO_INFO_MAX

} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

```

```

/*
 * buf type    fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]   size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */

// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;

/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide

```

```

/*
typedef struct {
    rect_t out_r;           ///< Output window position and size
    transformation_t transform;  ///< Used transformation
    rect_t input_roi_r;
    int32_t pan;           ///< Target shift in horizontal direction from centre of the
output image in pixels
    int32_t tilt;           ///< Target shift in vertical direction from centre of the output
image in pixels
    double zoom;            ///< Target zoom dimensionless coefficient (must not be bigger
than zero)
    double strength;        ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double strengthY;       ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double angle;           ///< Angle of main projection axis rotation around itself in
degrees
    // universal transformation
    double elevation;       ///< Angle in degrees which specify the main projection axis
    double azimuth;          ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio;     ///< Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h;           ///< Size of output field of view in vertical dimension in
degrees
    double FOV_w;           ///< Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y;   ///< Level of cylindricity for target projection shape in
vertical direction
    double cylindricity_x;   ///< Level of cylindricity for target projection shape in
horizontal direction
    char custom_file[128];   ///< File name of the file containing custom transformation
description
    custom_transformation_t custom; // < Parsed custom transformation structure
    double trapezoid_left_angle; // < Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle; // < Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code

[Function Description]

Stop the pipe id channel works of VIO, corresponding to the data channels in the initialization.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to 4.2.11hb_vio_get_data

Horizon Robotics
版权所有 禁止转载 算法工具部 杨柳军 2021-04-01 10:02:46

2 Camera Input

2.1 Camera Application API

2.1.1 hb_cam_init

[Function Declaration]

```
int hb_cam_init(uint32_t cfg_index, const char *cfg_file)
```

[Parameter Description]

- [IN]uint32_t cfg_index: The sensor to be initialized, which cfg_index corresponds to the configuration file config_*
- const char *setting_file: The path to the json configuration file (the absolute path that is accessible, in the format described in the configuration file)

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [Camera Return Value](#))

[Function Description]

Initialize sensor;

Each board has a configuration file. For example, the configuration file of x3dev board is hb_x3dev.json. By parsing the configuration file hb_x3dev.json and using the passed cfg_index to match which cam, and then initialize the cam.

[Compatibility]

System version: 1.1 and above. Note: The input configuration files may vary in different versions (1.1/1.2).

Hardware: X2/J2; X3/J3

[Sample Code]

```
int ret = 0, port = 0;  
  
int cam_index = 1;  
  
char *cam_cfg_file = "/etc/cam/hb_x3dev.json";  
ret = hb_cam_init(cam_index, cam_cfg_file);  
  
if(ret < 0){  
    printf("cam init fail\n");  
    return -1;  
}  
  
ret = hb_cam_start(port);  
if(ret < 0){
```

```

printf("cam start fail, do cam_deinit\n");
hb_cam_deinit(cam_index);
return -1;
}

ret = hb_cam_power_on(port);
if(ret < 0){
    printf("hb_cam_power_on fail, do cam_deinit\n");
    hb_cam_stop(port);
    hb_cam_deinit(cam_index);
    return -1;
}

ret = hb_cam_power_off(port);
if(ret < 0){
    printf("hb_cam_power_off fail, do cam_deinit\n");
    hb_cam_stop(port);
    hb_cam_deinit(cam_index);
    return -1;
}

ret = hb_cam_reset(port);
if(ret < 0){
    printf("hb cam reset fail, do cam_deinit\n");
    hb_cam_stop(port);
    hb_cam_deinit(cam_index);
    return -1;
}

hb_cam_stop(port);
hb_cam_deinit(cam_index);

```

2.1.2 hb_cam_deinit

[Function Declaration]

```
int hb_cam_deinit(uint32_t cfg_index)
```

[Parameter Description]

- [IN]uint32_t cfg_index: Sensor to be released
- [IN]cfg_index corresponds to the configuration file config_*

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [Camera Return Value](#))

[Function Description]

Release the camera module resources, corresponding to hb_cam_init.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 2.1.1hb_cam_init Example

2.1.3 hb_cam_start

[Function Declaration]

```
int hb_cam_start(uint32_t port)
```

[Parameter Description]

- [IN] uint32_t port: The sensor to be opened. The port corresponds to the configuration file port_*

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [Camera Return Value](#))

[Function Description]

Start the sensor data stream. Check for hs signal by using mipi.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 2.1.1hb_cam_init Example

2.1.4 hb_cam_stop

[Function Declaration]

```
int hb_cam_stop(uint32_t port)
```

[Parameter Description]

- [IN] uint32_t port: The sensor to be closed. The port corresponds to the configuration file port_*

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [Camera Return Value](#))

[Function Description]

Close the sensor data stream and execute mipi stop.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 2.1.1hb_cam_init Example

2.1.5 hb_cam_start_all

[Function Declaration]

```
int hb_cam_start_all()
```

[Parameter Description]

None

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [Camera Return Value](#))

[Function Description]

Start the multi-channel sensor during the multi-channel access.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] As with hb_cam_start, it replaces hb_cam_start during the multi-channel access. hb_cam_start can be called repeatedly during the multi-channel access to start each data stream.

2.1.6 hb_cam_stop_all

[Function Declaration]

```
int hb_cam_stop_all()
```

[Parameter Description]

None

[Return Value]

- Success: Returns HB_OK 0

- Failure: Returns negative error code (refer to [Camera Return Value](#))

[Function Description]

When multi-channel sensors are connected, stop multi-sensors.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] As with hb_cam_stop, it replaces hb_cam_stop during the multi-channel access. hb_cam_start can be called repeatedly during the multi-channel access to start each data stream.

2.1.7 hb_cam_power_on

[Function Declaration]

```
int hb_cam_power_on(uint32_t port)
```

[Parameter Description]

- [IN] uint32_t port: The camera to be powered on. The port corresponds to the configuration file port*.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [Camera Return Value](#))

[Function Description]

Power on the camera. At present, the sensor on the X3 does not have GPIO pin to control the power-on and off.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to 2.1.1hb_cam_init Example

2.1.8 hb_cam_power_off

[Function Declaration]

```
int hb_cam_power_off(uint32_t port)
```

[Parameter Description]

- [IN] uint32_t port: The camera to be powered off. The port corresponds to the configuration file port*.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [Camera Return Value](#))

[Function Description]

Power off the camera.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to 2.1.1hb_cam_init Example

2.1.9 hb_cam_reset

[Function Declaration]

```
int hb_cam_reset(uint32_t port)
```

[Parameter Description]

- [IN] uint32_t port: The camera to be powered on. The port corresponds to the configuration file port*.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [Camera Return Value](#))

[Function Description]

Reset the camera, if there is the pin used to control the reset.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to 2.1.1hb_cam_init Example

2.1.10 hb_cam_set_mclk

[Function Declaration]

```
int hb_cam_set_mclk(uint32_t entry_num, uint32_t mclk)
```

[Parameter Description]

- [IN] uint32_t entry_num: The corresponding mipi_host.
- [IN] uint32_t mclk: mclk value (Hz)

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [Camera Return Value](#))

[Function Description]

Camera sets sensor_mclk, which needs to be called when the sensor sub-board has no crystal oscillation.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
int entry_num = 1;
int cam_index = 3;
int port = 0;
char * cam_cfg_file = "/etc/cam/hb_x3dev.json"

hb_cam_set_mclk(entry_num, 24000000);
hb_cam_enable_mclk(1);
ret = hb_cam_init(cam_index, cam_cfg_file);
if(ret < 0){
    printf("cam init fail\n");
    return -1;
}
ret = hb_cam_start(port);
if(ret < 0){
    printf("cam start fail, do cam deinit\n");
    hb_cam_deinit(cam_index);
    return -1;
}
hb_cam_stop(port);
hb_cam_deinit(cam_index);
hb_cam_disable_mclk(0);
```

2.1.11 hb_cam_enable_mclk

[Function Declaration]

```
int hb_cam_enable_mclk(uint32_t entry_num)
```

[Parameter Description]

- [IN] uint32_t entry_num: The corresponding mipi_host.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [Camera Return Value](#))

[Function Description]

Camera sets sensor_mclk, which needs to be called when the sensor sub-board has no crystal oscillation.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to 2.1.10hb_cam_set_mclk Example

2.1.12 hb_cam_disable_mclk

[Function Declaration]

```
int hb_cam_disable_mclk(uint32_t entry_num)
```

[Parameter Description]

- [IN] uint32_t entry_num: The corresponding mipi_host.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [Camera Return Value](#))

[Function Description]

Camera sets sensor_mclk, which needs to be called when the sensor sub-board has no crystal oscillation.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to 2.1.10hb_cam_set_mclk Example

2.2 Camera Control API

2.2.1 hb_cam_i2c_read

[Function Declaration]

```
int hb_cam_i2c_read(uint32_t port, uint32_t reg_addr)
```

[Parameter Description]

- [IN]uint32_t port: The sensor to be accessed. The port corresponds to the configuration file port_*.
- [IN]uint32_t reg_addr: The register address to be read

[Return Value]

- Success: Returns the value read
- Failure: Returns negative error code (refer to 错误!未找到引用源。)

[Function Description]

Access to the sensor through the i2c.

[Compatibility]

System version: 2.1 and above

Hardware: X2/J2; X3/J3

[Sample Code]

```
int ret = 0, port = 0;  
int cam_index = 1, value;  
char read_buffer[4] = {0};  
char buffer = {0x36, 0x37, 0x38, 0x39};  
uint32_t size = 4;  
char *cam_cfg_file = "/etc/cam/hb_x3dev.json";  
ret = hb_cam_init(cam_index, cam_cfg_file);  
if(ret < 0){  
    printf("cam init fail\n");  
    return -1;  
}  
ret = hb_cam_start(port);  
if(ret < 0){  
    printf("cam start fail, do cam deinit\n");  
    hb_cam_deinit(cam_index);  
    return -1;  
}
```

```
value = hb_cam_i2c_read(port, 0x3018);
if(value < 0){
    printf("hb_cam_i2c_read fail, do cam_deinit\n");
    hb_cam_deinit(cam_index);
    return -1;
}

value = hb_cam_i2c_read_byte(port, 0x3018);
if(value < 0){
    printf("hb_cam_i2c_read_byte fail, do cam_deinit\n");
    hb_cam_deinit(cam_index);
    return -1;
}

ret = hb_cam_i2c_block_read (port, 0, 0x3018, read_buffer, size);
if(ret < 0){
    printf("hb_cam_i2c_block_read fail, do cam_deinit\n");
    hb_cam_deinit(cam_index);
    return -1;
}

ret = hb_cam_i2c_write(port, 0x3018, 0x36);
if(ret < 0){
    printf("hb_cam_i2c_write fail, do cam_deinit\n");
    hb_cam_deinit(cam_index);
    return -1;
}

ret = hb_cam_i2c_write_byte(port, 0x3018, 0x36);
if(ret < 0){
    printf("hb_cam_i2c_write_byte fail, do cam_deinit\n");
    hb_cam_deinit(cam_index);
    return -1;
}

ret = hb_cam_i2c_block_write (port, 0, 0x3018, buffer, size);
if(ret < 0){
    printf("hb_cam_i2c_block_write fail, do cam_deinit\n");
    hb_cam_deinit(cam_index);
    return -1;
}

ret = hb_cam_dynamic_switch_fps(port, 10);
if(ret < 0){
    printf("hb_cam_dynamic_switch_fps fail, do cam_deinit\n");
    hb_cam_deinit(cam_index);
    return -1;
}

ret = hb_cam_dynamic_switch_mode(port, 1);
if(ret < 0){
```

```
    printf("hb_cam_dynamic_switch_mode fail, do cam_deinit\n");
    hb_cam_deinit(cam_index);
    return -1;
}
hb_cam_stop(port);
hb_cam_deinit(cam_index);
```

2.2.2 hb_cam_i2c_read_byte

[Function Declaration]

```
int hb_cam_i2c_read_byte(uint32_t port, uint32_t reg_addr)
```

[Parameter Description]

- [IN]uint32_t port: The sensor to be accessed. The port corresponds to the configuration file port_*.
- [IN]uint32_t reg_addr: The register address read

[Return Value]

- Success: Returns the value read
- Failure: Returns negative error code (refer to 错误!未找到引用源。)

[Function Description]

Access to the sensor through the i2c and read by bytes.

[Compatibility]

System version: 2.1 and above

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 2.2.1hb_cam_i2c_read Example

2.2.3 hb_cam_i2c_write

[Function Declaration]

```
int hb_cam_i2c_write(uint32_t port, uint32_t reg_addr, uint16_t value)
```

[Parameter Description]

- [IN]uint32_t port: The sensor to be accessed. The port corresponds to the configuration file port_*.
- [IN]uint32_t reg_addr: The written register address
- [IN]uint16_t value: The written value

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to 错误!未找到引用源。)

[Function Description]

Write the sensor register through the i2c.

[Compatibility]

System version: 2.1 and above

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 2.2.1hb_cam_i2c_read Example

2.2.4 hb_cam_i2c_write_byte

[Function Declaration]

```
int hb_cam_i2c_write_byte(uint32_t port, uint32_t reg_addr, uint16_t value)
```

[Parameter Description]

- [IN]uint32_t port: The sensor to be accessed. The port corresponds to the configuration file port_*.
- [IN]uint32_t reg_addr: The written register address
- [IN]uint16_t value: The written value

[Return Value]

Success: Returns HB_OK 0

Failure: Returns negative error code (refer to 错误!未找到引用源。)

[Function Description]

By using the i2c , write the sensor in bytes, and write the incoming value.

[Compatibility]

System version: 2.1 and above

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 2.2.1hb_cam_i2c_read Example

2.2.5 hb_cam_i2c_block_write

[Function Declaration]

```
int hb_cam_i2c_block_write(uint32_t port, uint32_t subdev, uint32_t reg_addr, char *buffer,  
                           uint32_t size)
```

[Parameter Description]

- [IN]uint32_t port: The sensor to be accessed. The port corresponds to the configuration file port_*.
- [IN] uint32 subdev: The access device type. Currently, sensor, external ISP, eeprom in the sensor, or IMU are supported.
- [IN]uint32reg_addr: The register address to be operated.
- [IN]char *buffer written value
- [IN] uint32 size: The written data length

[Return Value]

Success: Returns HB_OK 0

Failure: Returns negative error code (refer to 错误!未找到引用源。)

[Function Description]

By using the large block of the i2c, write the sensor or other devices inside the sensor.

[Compatibility]

System version: 2.1 and above

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 2.2.1hb_cam_i2c_read Example

2.2.6 hb_cam_i2c_block_read

[Function Declaration]

```
int hb_cam_i2c_block_read(uint32_t port, uint32_t subdev, uint32_t reg_addr, char *buffer,  
uint32_t size)
```

[Parameter Description]

- [IN]uint32_t port: The sensor to be accessed. The port corresponds to the configuration file port_*.
- [IN] uint32 subdev: The access device type. Currently, sensor, external ISP, eeprom in the sensor, or IMU are supported.
- [IN]uint32 reg_addr: The register address to be read
- [IN]char *buffer read value
- [IN]uint32 size: The read data length

[Return Value]

- Success: Returns the read value

- Failure: Returns negative error code (refer to 错误!未找到引用源。)

[Function Description]

By using the large block of the i2c, write the sensor or other devices inside the sensor.

[Compatibility]

System version: 2.1 and above

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 2.2.1hb_cam_i2c_read Example

2.2.7 hb_cam_dynamic_switch_fps

[Function Declaration]

```
int hb_cam_dynamic_switch_fps(uint32_t port, uint32_t fps);
```

[Parameter Description]

- [IN]uint32_t port: The sensor to be accessed. The port corresponds to the configuration file port_*.
- [IN]uint32 fps: The frame rate to be switched.

[Return Value]

Success: Returns HB_OK 0

Failure: Returns negative error code (refer to 错误!未找到引用源。)

[Function Description]

Frame rate switching interface.

[Compatibility]

System version: 2.1 and above

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 2.2.1hb_cam_i2c_read Example

2.2.8 hb_cam_dynamic_switch_mode

[Function Declaration]

```
int hb_cam_dynamic_switch_mode(uint32_t port, uint32_t mode)
```

[Parameter Description]

- [IN]uint32_t port: The sensor to be accessed. The port corresponds to the configuration file port_*.
- [IN]uint32 mode: The mode to be switched.

[Return Value]

Success: Returns HB_OK 0

Failure: Returns negative error code (refer to 错误!未找到引用源。)

[Function Description]

Sensor mode switching interface.

[Compatibility]

System version: 2.1 and above

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 2.2.1hb_cam_i2c_read Example

2.3 SIF API

2.3.1 hb_vio_raw_dump

[Function Declaration]

```
int hb_vio_raw_dump(uint32_t pipeline_id, hb_vio_buffer_t * raw_img,  
                    hb_vio_buffer_t * isp_yuv)
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Software channel identification of the corresponding data.
- [OUT]hb_vio_buffer_t *raw_img: Sif RAW image buffer obtained. See X3/J3 main parameters for the structure.
- [OUT]hb_vio_buffer_t *isp_yuv: ISP YUV image buffer obtained.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to VIO Return Code)

[Function Description]

After initializing the video system, the image information of SIF RAW and ISP YUV corresponding to the camera data stream is obtained and used for debugging. It is not used together with RAW fallback.

This debug api only support in sepecial work mode(all on the fly) in vio.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to hb_vio_raw_feedback

2.3.2 hb_vio_raw_feedback

[Function Declaration]

```
int hb_vio_raw_feedback(uint32_t pipeline_id, hb_vio_buffer_t * feedback_src, hb_vio_buffer_t  
*isp_dst_yuv)
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Software channel identification of the corresponding data.
- [IN]hb_vio_buffer_t * feedback_src: The buffer pointer that needs to be filled back. See X3/J3 main parameters for the structure.
- [IN] hb_vio_buffer_t *isp_dst_yuv: The YUV data generated after the ISP processing

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to VIO Return Code)

[Function Description]

After initializing the video system, fill the external RAW image back to the ISP and get the the ISP output data, which is used for debugging. It is not used together with RAW fillback.

This debug api only support in sepecial work mode(ddr in to isp) in vio.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "hb_vio_interface.h"
#include "hb_cam_interface.h "
int main(int argc, char *argv[])
{
    int ret = 0;
    FILE *inputFp = NULL;
    int need_raw_dump = 1;
    hb_vio_buffer_t sif_raw = {0};//for raw dump
    hb_vio_buffer_t isp_yuv = {0};//for raw dump
    hb_vio_buffer_t sif_raw_src = {0};//for raw feedback
    hb_vio_buffer_t isp_yuv_dst = {0};//for raw feedback
    char *cam_cfg_file = "/etc/cam/hb_x3dev.json";
    ret = hb_vio_init("/cfg/imx327_raw_12bit_1952x1097_online_Pipeline.json");
    if (ret < 0) {
        printf("vio init fail\n");
        return -1;
    }
    ret = hb_cam_init(0, cam_cfg_file);
    if (ret < 0) {
        printf("cam init fail\n");
        hb_vio_deinit();
    }
}
```

```

        return -1;
    }

ret = hb_vio_start_pipeline(0);
if (ret < 0) {
    printf("vio start fail, do cam&vio deinit.\n");
    hb_cam_deinit(0);
    hb_vio_deinit();
    return -1;
}

ret = hb_cam_start(0);
if (ret < 0) {
    printf("cam start fail, do cam&vio deinit.\n");
    goto err;
}

if (int need_raw_dump == 1) {
    ret = hb_vio_raw_dump(0, &sif_raw, &isp_yuv);
} else {
    ret = hb_vio_get_data(0, HB_VIO_SIF_FEEDBACK_SRC_DATA,
                          &sif_raw_src);
    if (ret < 0) {
        printf("pipe 0 get raw fb src data failed!\n");
        goto err;
    }

    inputFp = fopen("/raw_feedback_src.raw", "rb");
    int count = fread(sif_raw_src.img_addr.addr[0], 1,
                      sif_raw_src.img_info.size[0], inputFp);
    fclose(inputFp);
    ret = hb_vio_raw_feedback(0, &sif_raw_src, &isp_yuv_dst);
}
if (ret < 0) {
    printf("pipe 0 get data failed!\n");
    goto err;
}

hb_cam_stop(0);
hb_vio_stop_pipeline(0);
hb_cam_deinit(0);
hb_vio_deinit();
return 0;

err:
    hb_cam_stop(0);
    hb_cam_deinit(0);
    hb_vio_deinit();
    return -1;
}

```

}

2.4 Camera Return Value

Return Value: 20/21/22 compatible with the system version 1.2 only.

```
//libcam: vin | cam

#define RET_OK 0
#define RET_ERROR 1

enum HB_CAM_ERROR_CODE {
    HB_CAM_PARSE_BOARD_CFG_FAIL = 2,
    HB_CAM_PARSE_MIPI_CFG_FAIL,
    HB_CAM_DOPEN_LIBRARY_FAIL,
    HB_CAM_INIT_FAIL,
    HB_CAM_DEINIT_FAIL,
    HB_CAM_START_FAIL,
    HB_CAM_STOP_FAIL,
    HB_CAM_I2C_WRITE_FAIL,
    HB_CAM_I2C_WRITE_BYTE_FAIL,
    HB_CAM_I2C_WRITE_BLOCK_FAIL,
    HB_CAM_I2C_READ_BLOCK_FAIL,
    HB_CAM_DYNAMIC_SWITCH_FAIL,
    HB_CAM_DYNAMIC_SWITCH_FPS_FAIL,
    HB_CAM_S954_POWERON_FAIL,
    HB_CAM_S954_CONFIG_FAIL,
    HB_CAM_S954_STREAM_ON_FAIL,
    HB_CAM_S954_STREAM_OFF_FAIL,
    HB_CAM_SENSOR_POWERON_FAIL,
    HB_CAM_SENSOR_POWEROFF_FAIL,
    HB_CAM_START_PHYSICAL_FAIL,
    HB_CAM_SPI_WRITE_BLOCK_FAIL,
    HB_CAM_SPI_READ_BLOCK_FAIL,
    HB_CAM_INVALID_PARAM,
    HB_CAM_SET_EX_GAIN_FAIL,
    HB_CAM_SET_AWB_FAIL,
    HB_CAM_I2C_READ_FAIL,
    HB_CAM_I2C_READ_BYTE_FAIL,
    HB_CAM_RESET_FAIL,
    HB_CAM_OPS_NOT_SUPPORT,
    HB_CAM_ISP_POWERON_FAIL,
    HB_CAM_ISP_POWEROFF_FAIL,
    HB_CAM_ISP_RESET_FAIL,
    HB_CAM_ENABLE_CLK_FAIL,
    HB_CAM_DISABLE_CLK_FAIL,
```

```
    HB_CAM_SET_CLK_FAIL
};
```

2.5 Camera Configuration File Description

At present, each board has a configuration file, which contains all the camera information used on the board. For example, for the xj3dev board, the configuration file used is Hb_Xj3 dev.json 。 The following example is compatible with system version 2.0, please refer to: [Camera Configuration File \(Ver. 2.0\)](#).

```
{
    "config_number":4,                                     /* The total number of sensors supported on the current development board
    */
    "board_name":"x3dev",                                /* Current development board name*/
    "config_0":{                                         /* Configuration serial number */
        "interface_type": "mipi",                         /* Data interface types: mipi, dvp, bt, sdio, net, etc.*/
        "deserial_num": 1,                                /* Number of deserializer */
        "deserial_0": {                                    /* Serial number of deserializer */
            "deserial_name": "s954",                      /* Deserializer name*/
            "deserial_addr": "0x3d",                      /* Deserializer address */
            "bus_type":0,                                 /* The bus type where the current deserializer resides*/
            "bus_number":0,                               /* The bus number of the current deserializer */
            "physical_entry": 0,                          /* Which entry belongs to. Entry stands for mipi host*/
            "power_mode": 1,                            /* Whether the power on is controllable. Some boards do not need to be powered on, and some boards need GPIO
power-on or reset.*/
            "gpio_pin": [73],                           /* GPIO pins for operation */
            "gpio_level": [0],                           /* Initial effective value, for example, the pin needs to be pulled down first and then pulled up, and the value is 0. If
first pulled up and then pulled down, the value is 1.*/
        },
        "port_number":1,                                /* Which sensor is used by users */
        "port_0":{                                       /* Channel of the current sensor */
            "bus_type":0,                                /* Bus type of current sensor */
            "bus_number":0,                               /* Bus number of the current sensor */
            "sensor_addr": "0x21",                        /*sensor address*/
            "serial_addr": "0x18",                        /* The string address inside the sensor */
            "sensor_name": "imx390",                      /*sensor name*/
            "power_mode": 1,                            /* Whether the sensor power-on is controllable */
            "extra_mode": 0
            /* Some sensor libraries can be shared by several sensors, where this mark bit can be used to distinguish them. For
example, if extra_mode is 0, it means x2dev binocular. If 1, it means 96board binocular. This field can be used as some candidate
functions, but it is not necessary for some sensors.*/
            "reg_width": 16,                            /* Register address width of sensor */
            "entry_num": 0,                            /* The mipi host that the sensor currently belongs to */
        }
    }
}
```

```

        "fps": 27,           /*sensor frame rate*/
        "resolution": 1080,   /* Sensor resolution */
        "width": 1280,       /* Image width, special for vcam */
        "height": 720,       /* Image height, special for vcam */
        "format": 24,         /*Image format (0x18: yuv420), special for vcam */
        "power_delay":20,
        /* The delay time usleep (sensor_info->power_delay *1000) required to operate some pins when the sensor is
powered on */

        "gpio_pin": [62,63],    /* GPIO pin used by sensor */
        "gpio_level": [0,0],
        /* Initial effective value, for example, the pin needs to be pulled down first and then pulled up, and the value is 0. If
first pulled up and then pulled down, the value is 1. */

        "deserial_index": 0,      /* The deserial that the sensor belongs to */
        "deserial_port": 0,       /* The port of which deserial that sensor belongs */
        "config_path": "/etc/cam/hb_mipi_imx390_raw_%dfps_%dP.json"
        /* MIPI configuration file path which the sensor corresponds to */
    }
}
}
}

```

Introduction:

To add a sensor:

- Modify the json configuration file of the corresponding board, and add the corresponding port information and deserial information.
- Add ***_utility.c to hbre/camera/utility/sensor (*** is the library name, which is also the sensor name). After compilation, the corresponding .so file will be generated. After launching the system, it will be copied to /lib/sensorlib directory.
- According to the characteristics of each sensor, implement ***_utility.c, the power on / power off of the sensor, sensor configuration, and the stream on/stream off of the sensor.

2.6 Camera Configuration File (Ver. 2.0)

```

{
    "config_number":4,           /* Current sensor scenarios supported by the development board */
    "board_name":"x3dev",        /* Development board name*/
    "config_0":{                  /* Configuration serial number */
        "port_number":1,          /* The channel of the sensor used by users */
        "port_0":{                /* The channel of the current sensor */
            "bus_type":0,          /* Bus type of sensor */
            "bus_number":0,         /* Bus number of sensor */
            "sensor_addr":"0x36",   /* I2C address of sensor */
            "sensor_name":"imx327", /*sensor name*/
            "power_mode":1,         /* Whether the power on of the sensor is controllable */
            "config_path":"/etc/cam/hb_mipi_*.json"
        }
    }
}

```

```
        /*Mipi configuration file path corresponding to the sensor*/  
    }  
}  
}
```

2.7 Camera Configurable Environment Variable

Environment variable: CAM_CONFIG_FILE_PATH、CAM_INDEX

[Function Description]

```
hb_cam_init(uint32_t cfg_index, const char *cfg_file)
```

When the interface parameter cfg_file is empty, use the values of the environment variable CAM_CONFIG_FILE_PATH and CAM_INDEX. If CAM_INDEX is 0, then cfg_index is 0.

```
export CAM_CONFIG_FILE_PATH="/etc/cam/hb_x3dev.json" //Set the environment variable CAM_CONFIG_FILE_PATH  
export CAM_INDEX=3 // Set the environment variable CAM_INDEX  
echo $CAM_CONFIG_FILE_PATH // Display environment variable value, whether it is set successfully  
echo $CAM_INDEX // Display environment variable value, whether it is set successfully
```

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2, X3/J3

Environment Variable: CAM_PORT

[Function Description]

When the input parameter port is greater than the maximum number of the ports supported, the value of the environment variable CAM_PORT will be obtained. If cam_ If the value of CAM_PORT is valid, use the value of CAM_PORT for initialization.

```
export CAM_PORT=0 // Set the environment variable CAM_PORT  
echo $CAM_PORT // Display environment variable value, whether it is set successfully
```

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2, X3/J3

3 Audio Input and Output

3.1 Audio Service Description

3.1.1 API Usage Background

The API is based on the improvement of tinyalsa on Android system. Because the code of tinyalsa is more concise, the definition and use of API is simpler. CP audio is mainly used to realize the input and output of pcm audio stream, so tinyalsa completely replaces the alsalib of the Linux standard.

3.1.2 API Provision Scheme

Use header file +lib(so) for APP.

3.1.3 Sample code and usage of API

Use the tinyalsa command line and the utility sample code.

3.2 API Definition

3.2.1 pcm_open

[Function Declaration]

```
struct pcm *pcm_open(unsigned int card, unsigned int device, unsigned int flags, struct pcm_config *config)
```

[Parameter Description]

- [IN]card: sound card number
- [IN]device: Device serial number of sound card
- [IN]flag: sound card flag
- [IN]config: User's audio configuration

[Return Value]

- Success: Returns PCM structure pointer
- Failure: Returns NULL

[Function Description]

Turn on the sound card device.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2, X3/J3

[Sample Code] Refer to [3.2.2pcm_write\(\) Sample Code](#)

3.2.2 pcm_write

[Function Declaration]

```
int pcm_write(struct pcm *pcm, const void *data, unsigned int count)
```

[Parameter Description]

- [IN]pcm: The pcm pointer returned from pcm_open
- [IN]data: The address which the data is written to
- [IN]count: Numbers to read

[Return Value]

- Success: Returns 0.
- Failure: Returns error code.

[Function Description]

Writes the data to the sound card device.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2, X3/J3

[Sample Code]

```
#include <tinysalsa/pcm.h>

int main(int argc, char **argv) {
    int ret = -1;
    int size = 0;
    int cardId = 0;
    int deviceId = 1;
    int flags = PCM_OUT;
    FILE *inputFile;
    struct pcm_config config;
    config.channels = 2;
    config.rate = 48000;
    config.format = PCM_FORMAT_S16_LE;
    config.period_size = 1024;
    config.period_count = 2;
    struct pcm *pcm;
    pcm = pcm_open(cardId, deviceId, flags, &config);
    if (pcm == NULL) {
```

```

        printf("failed to open PCM\n");
        pcm_close(pcm);
        return ret;
    }

    size = pcm->buffer_size * config.channels *
        (pcm_format_to_bits(pcm->config.format) >> 3));
    void *frames = malloc(size);
    if (!frames) {
        printf("failed to allocate memory for frames\n");
        free(frames);
        pcm_close(pcm);
        return ret;
    }

    inputFile = fopen("audio.pcm", "rb");
    if (!inputFile) {
        printf("failed to open file for writing\n");
        free(frames);
        pcm_close(pcm);
        fclose(inputFile);
        return ret;
    }

    fread(frames, 1, size, inputFile);
    ret = pcm_write(pcm, frames, size);
    if (ret < 0) {
        printf("error playing\n");
        free(frames);
        pcm_close(pcm);
        fclose(inputFile);
        return ret;
    }

    pcm_close(pcm);
    free(frames);
    fclose(inputFile);
}

```

3.2.3 pcm_read

[Function Declaration]

```
int pcm_read(struct pcm *pcm, const void *data, unsigned int count)
```

[Parameter Description]

- [IN]pcm: The pcm pointer returned from pcm_open
- [IN]data: The address to save

- [IN]count: Numbers to read

[Return Value]

- Success: Returns 0.
- Failure: Returns error code.

[Function Description]

Read the data from the sound card device.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2, X3/J3

[Sample Code]

```
#include <tinysalsa/pcm.h>

int main(int argc, char **argv) {
    int ret = -1;
    int cardId = 0;
    int deviceld = 0;
    int flags = PCM_IN;
    void *frames;
    int size;
    FILE *outputFile;
    struct pcm_config config;
    config.channels = 2;
    config.rate = 16000;
    config.format = PCM_FORMAT_S16_LE;
    config.period_size = 1024;
    config.period_count = 4;
    struct pcm *pcm;
    pcm = pcm_open(cardId, deviceld, flags, &config);
    if (pcm == NULL) {
        printf("failed to open PCM\n");
        pcm_close(pcm);
        return ret;
    }
    size = pcm->buffer_size * config.channels *
        (pcm_format_to_bits(config.format) >> 3);
    frames = malloc(size);
    if (frames == NULL) {
        printf("failed to allocate frames\n");
        free(frames);
        pcm_close(pcm);
    }
}
```

```
        return ret;
    }

    outputFile = fopen("audio.pcm", "wb");
    if (!outputFile) {
        printf("failed to open file for writing\n");
        free(frames);
        pcm_close(pcm);
        fclose(outputFile);
        return ret;
    }

    int read_cound = pcm_read(pcm, frames, size);
    fwrite(frames, 1, size, outputFile);
    free(frames);
    pcm_close(pcm);
    fclose(outputFile);
}
```

3.2.4 pcm_close

[Function Declaration]

```
int pcm_close(struct pcm *pcm)
```

[Parameter Description]

- [IN]pcm: The pcm pointer returned from pcm_open

[Return Value]

- Success: Returns 0.
- Failure: Returns error code.

[Function Description]

Turn off the sound card device.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2, X3/J3

[Sample Code]Refer to [3.2.2pcm_write\(\) Example Code](#)

3.2.5 mixer_open

[Function Declaration]

```
mixer * mixer_open(unsigned int card)
```

[Parameter Description]

- [IN]card: Sound card serial number

[Return Value]

- Success: Returns mixer structure pointer.
- Failure: Returns NULL

[Function Description]

Open the control device of the sound card.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2, X3/J3

[Sample Code]

```
#include <tinysalsa/mixer.h>
int main(int argc, char **argv) {
    int ret = -1;
    int id = 4;
    int volume = 120;
    struct mixer *mixer;
    struct mixer_ctl *ctl;
    int cardId = 0;
    mixer = mixer_open(cardId);
    if (!mixer) {
        printf("failed to open mixer\n");
        mixer_close(mixer);
        return ret;
    }
    ctl = mixer_get_ctl(mixer, "set");
    ret = mixer_ctl_set_value(ctl, id, volume);
    if (ret) {
        printf("set channel digital volume failed\n");
        mixer_close(mixer);
        return ret;
    }
    mixer_close(mixer);
}
```

3.2.6 mixer_get_ctl

[Function Declaration]

```
mixer_ctl * mixer_get_ctl(struct mixer *mixer, unsigned int id)
```

[Parameter Description]

- [IN]mixer: mixer pointer returned from mixer_open
- [IN]id: control id

[Return Value]

- Success: Returns mixer_ctl structure pointer
- Failure: Returns NULL

[Function Description]

Returns the mixer_ctl structure corresponding to the ID.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2, X3/J3

[Sample Code]Refer to [3.2.5mixer_open\(\) Example Code](#)

3.2.7 mixer_ctl_set_value

[Function Declaration]

```
int mixer_ctl_set_value(struct mixer_ctl *ctl, unsigned int id, int value)
```

[Parameter Description]

- [IN]mixer: : Mixer pointer returned from mixer_open
- [IN]id: Control ID
- [IN]value:Control value

[Return Value]

- Success: Returns 0
- Failure: Returns error code

[Function Description]

Set the value using the ctl id.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2, X3/J3

[Sample Code]Refer to [3.2.5mixer_open\(\) Sample Code](#)

3.2.8 mixer_close

[Function Declaration]

```
void mixer_close(struct mixer *mixer)
```

[Parameter Description]

- mixer: Mixer pointer returned from mixer_open

[Return Value]

None

[Function Description]

Turn off the control device.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2, X3/J3

[Sample Code]Refer to [3.2.5mixer_open\(\) Sample Code](#)

3.3 Definition and Description of Data Structure

```
struct pcm_config {
    unsigned int channels;// Number of channels per frame
    unsigned int rate;// Sampling rate
    unsigned int period_size;// The number of frames each time when the hardware interrupt processes the audio data.
    unsigned int period_count;// The number of the hardware interrupts required per transmission cycle
    enum pcm_format format;//pcm format
    unsigned int start_threshold;// Minimum number of frames to start the PCM transmission. No need to specify it.
    unsigned int stop_threshold;// Minimum number of frames to stop the PCM transmission. No need to specify it.
    unsigned int silence_threshold;// Minimum number of frames required to mute the PCM. No need to specify it.
};

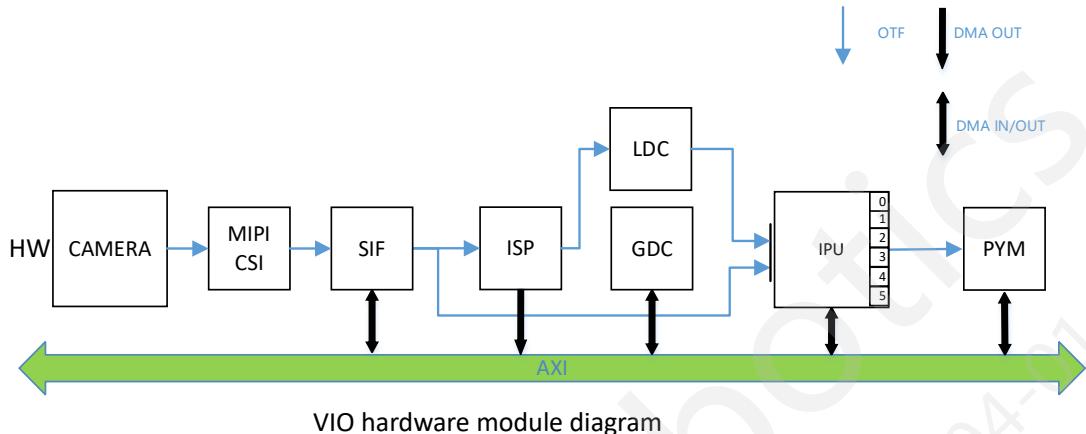
struct mixer_ctl {
    struct mixer *mixer;//The structure that the mixer control belongs to
    struct snd_ctl_elem_info info;// Controler value information, such as types and the number of values.
    char **ename;// Chart of enumeration values
};

struct snd_ctl_elem_info {
    struct snd_ctl_elem_id id; //controler ID
    snd_ctl_elem_type_t type; //controler type
    unsigned int access; // Access permissions
    unsigned int count; // The number of values
    __kernel_pid_t owner;
};
```

```
union {
    struct {
        long min;      /* R: minimum value */
        long max;      /* R: maximum value */
        long step;     /* R: step (0 variable) */
    } integer;// 32-bit integer
    struct {
        long long min;    /* R: minimum value */
        long long max;    /* R: maximum value */
        long long step;   /* R: step (0 variable) */
    } integer64;//64-bit integer
    struct {
        unsigned int items; /* R: number of items */
        unsigned int item;  /* W: item number */
        char name[64];    /* R: value name */
        __u64 names_ptr;  /* W: names list (ELEM_ADD only) */
        unsigned int names_length;
    } enumerated;
    unsigned char reserved[128];
} value;//value
union {
    unsigned short d[4];    /* dimensions */
    unsigned short *d_ptr;  /* indirect - obsoleted */
} dimen;// Direction, not commonly used
unsigned char reserved[64-4*sizeof(unsigned short)];
};
```

4 Video Processing

4.1 VPS API Procedure (X3/J3)



There are different working modes about the input and output of VIO hardware IP. Some IP support OTF (online) and DMA (offline), some IP can only work in online mode, and some IP can only work in offline mode. The overall data path is shown in the figure above.

The specific patterns supported by each hardware IP are shown in the following table:

Input type	IP	Output type
OTF	MIPI	OTF
OTF/DMA	SIF	OTF/DMA
OTF	ISP	OTF/DMA
OTF	LDC	OTF
DMA	GDC	DMA
OTF/DMA	IPU	OTF/DMA
OTF/DMA	PYM	DMA

Note: The DMA Read function in the SIF can only be output to the ISP for use.

(In the current design, the DMA Read of the ISP is on the SIF, so that the data can be transmitted in M2M mode from the SIF to the ISP.)

Based on the X3/J3 chip, the VIO software organizes IP as a hardware path as required and provide functions for upper applications via this hardware path. The VIO software covers basic application scenarios that includes single channel, dual channels, and four channels.

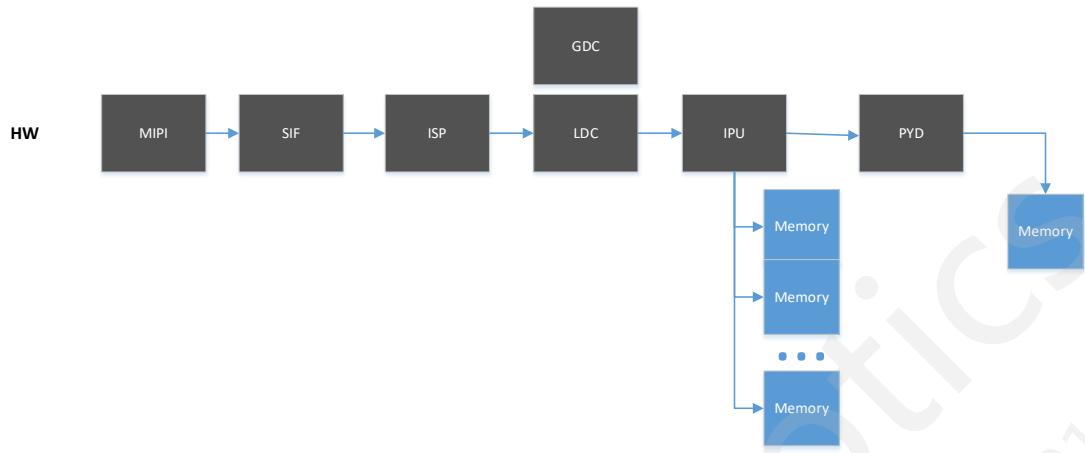
4.1.1 VPS supports single input

1) Single Input (pym online)

Note: In single channel scenario, the PIPE_ID of the API interface is usually 0, which indicates the

pipe channel of the first configuration.

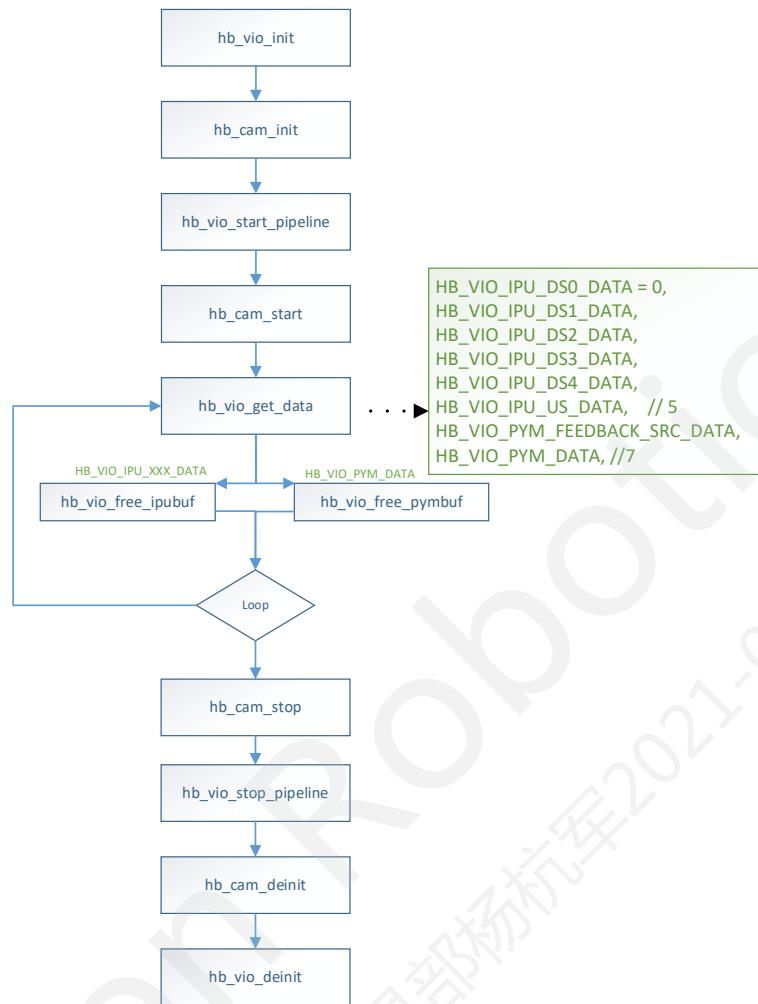
1. Connection status: VIO single scenario sif->isp->ldc->ipu->pym (All on the fly connect)



Single online scenario diagram

API procedure:

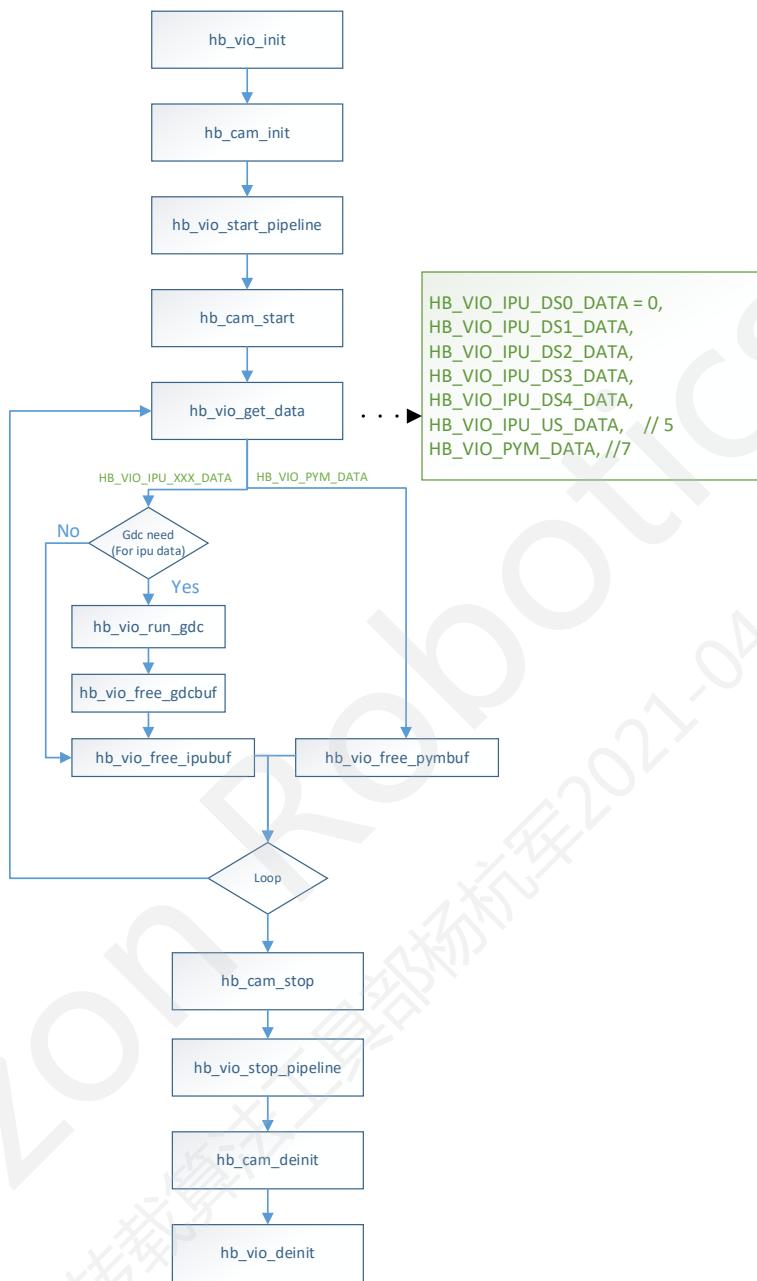
- The IPU data of each channel is obtained directly from the interface, and the PYM data is obtained from the interface, too. The data types of the API parameters are different. In this scenario, the PYM hardware is directly connected to the previous node and unable to be filled back (all hardware parameters are configured by using the configuration file during initialization).



VIO calling procedure diagram

- b. The IPU data of each channel is obtained directly from the interface, and the PYM data is obtained from the interface, too. The data types of the API parameters are different. In this scenario, the PYM hardware is directly connected to the previous node and unable to be filled back (all hardware parameters are configured by using the configuration file during initialization).

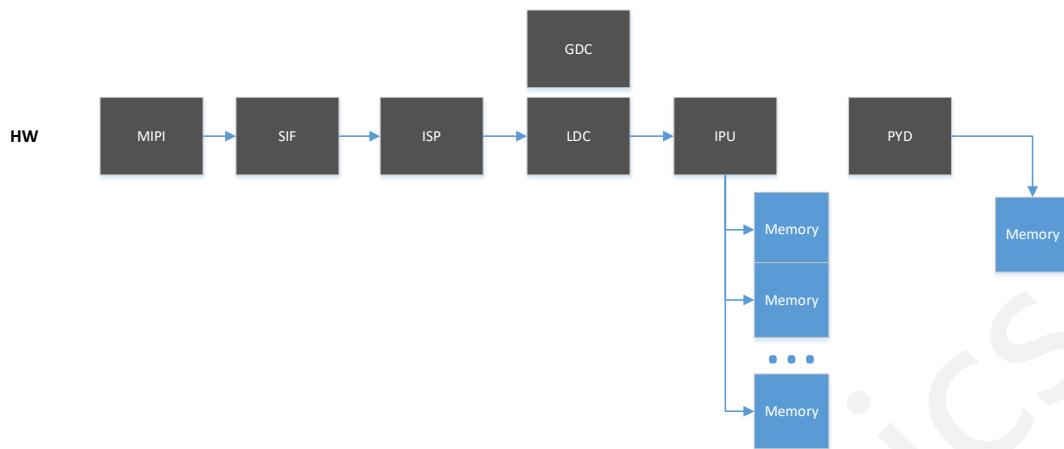
GDC processing: The IPU data can be used after being processed by GDC (GDC processing is carried out in offline mode). The process is as below (the size of the IPU output data shall be consistent with the processing size set by GDC):



VIO pym online calling procedure diagram

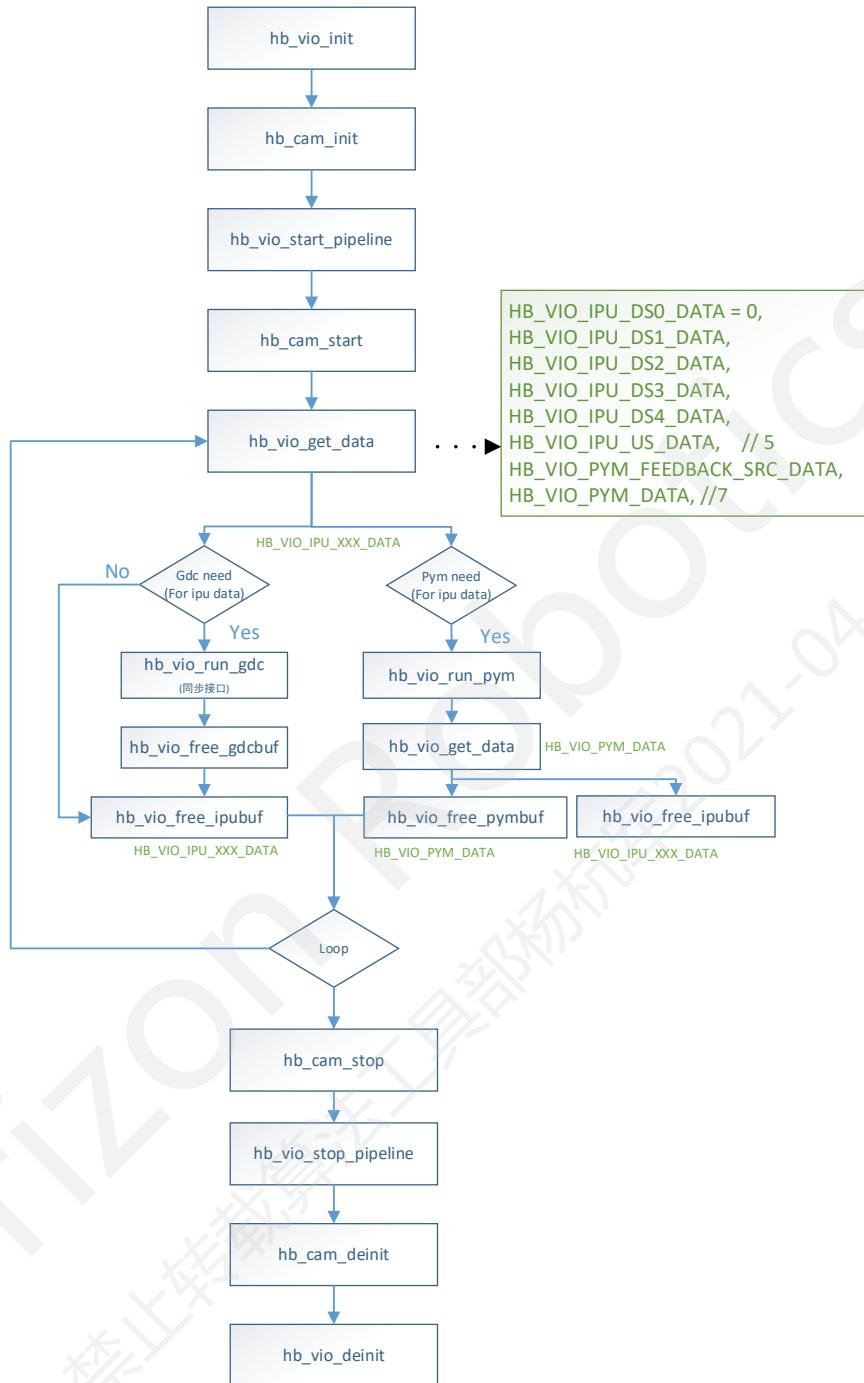
2) Single Input (pym offline)

Connection status: VIO single scenario sif->isp->ldc->ipu pym



Single offline scenario diagram

- a. In this single channel scenario, the hardware processing of the PYM module is carried out in the offline mode (which is mainly used for external fillback, see b for details). If a channel of the IPU needs to process the PYM, the PYM processing interface will be called independently to get the pyramid data (If the settings are configured by using the configuration file during initialization, the size of the IPU output data size should be equal to that of PYM input data).



VIO pym offline calling procedure diagram

4.1.2 VPS supports multi-channel data input

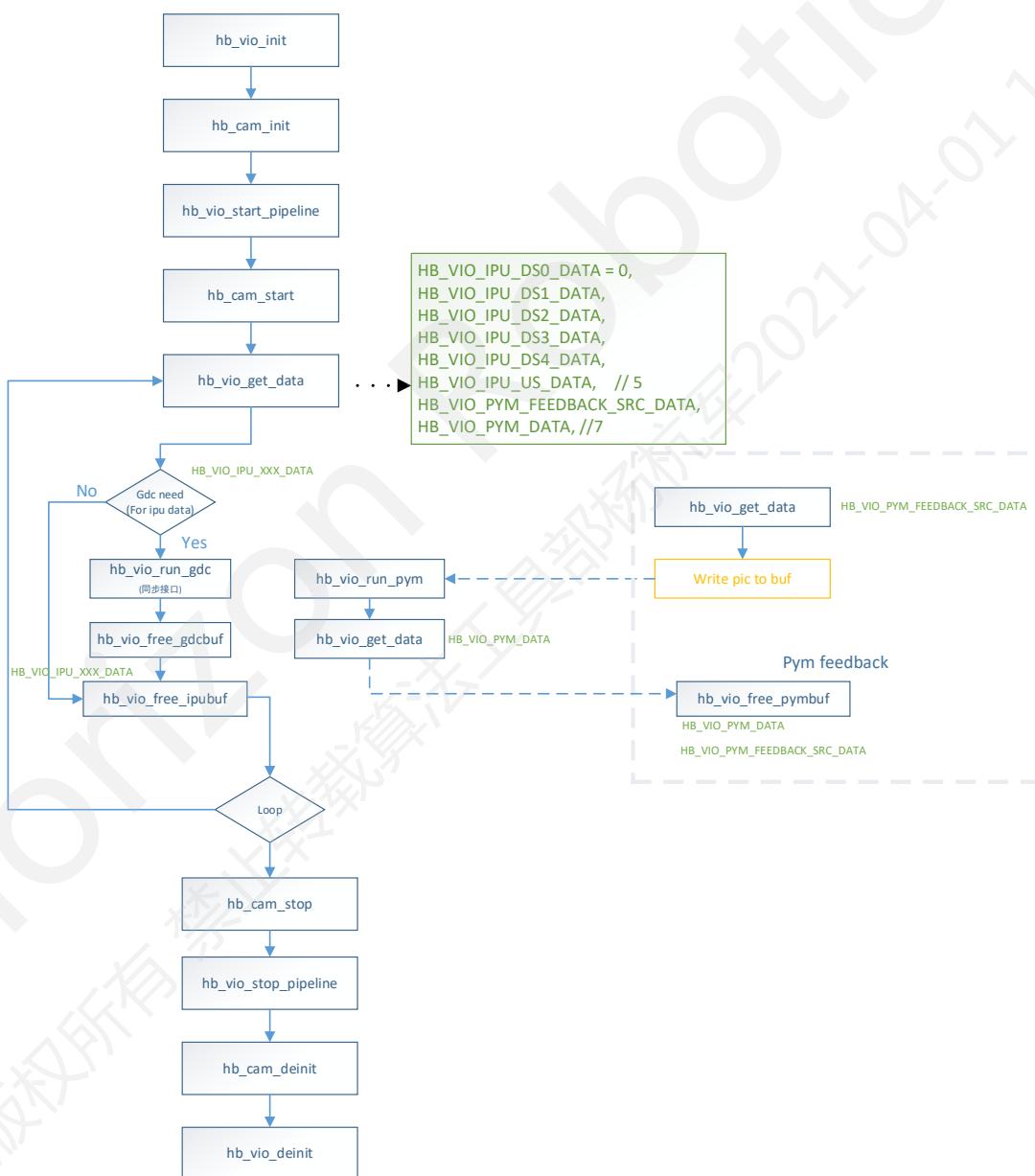
X3/J3 supports multi-channel input (8-channel input at most). The VIO should be initialized by using the json files of two or other channels. The VIO APIs should be used according to different pipe_id. Wait for IPU/PYM results. The corresponding calling process is similiar to the single channel scenario previously described.

4.1.3 Fillback interface procedure

The PYM fillback is used in the PYM offline scenario and it cannot work in online mode.

1) Fillback mode

Get a buf from the buf pool of the fillback source by using HB_VIO_PYM_FEEDBACK_SRC_DATA. Fill back a frame (yuv420sp) to memory by using the obtained address. After the fillback, send the corresponding buf to the pyramid process and execute it. Wait until the PYM process finishes by using HB_VIO_PYM_DATA and you get the PYM results of the fillback data. After this, use hb_vio_free_pymbuf to release the corresponding buf of HB_VIO_PYM_DATA and HB_VIO_PYM_FEEDBACK_SRC_DATA. The IPU data you can process it with GDC module.

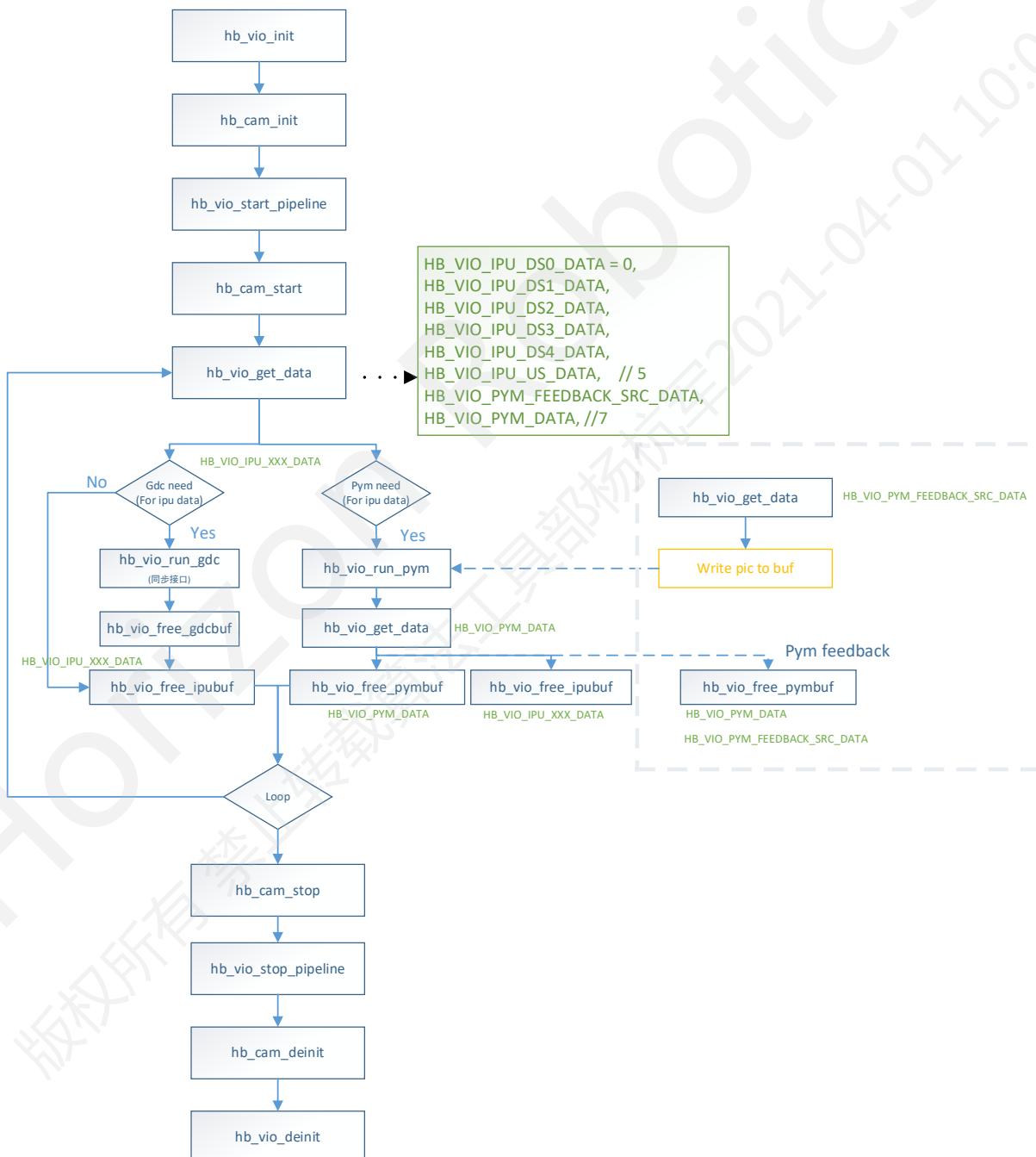


VIO fillback calling procedure diagram

2) Single channel fillback + single channel video streaming

Get a buf from the buf pool of the fillback source by using HB_VIO_PYM_FEEDBACK_SRC_DATA. Fill back a frame (yuv420sp) to memory by using the obtained address. After the fillback, send the corresponding buf to the pyramid process and execute it. Wait until the PYM process finishes by using HB_VIO_PYM_DATA and you get the PYM results of the fillback data. After this, use hb_vio_free_pymbuf to release the corresponding buf of HB_VIO_PYM_DATA and HB_VIO_PYM_FEEDBACK_SRC_DATA.

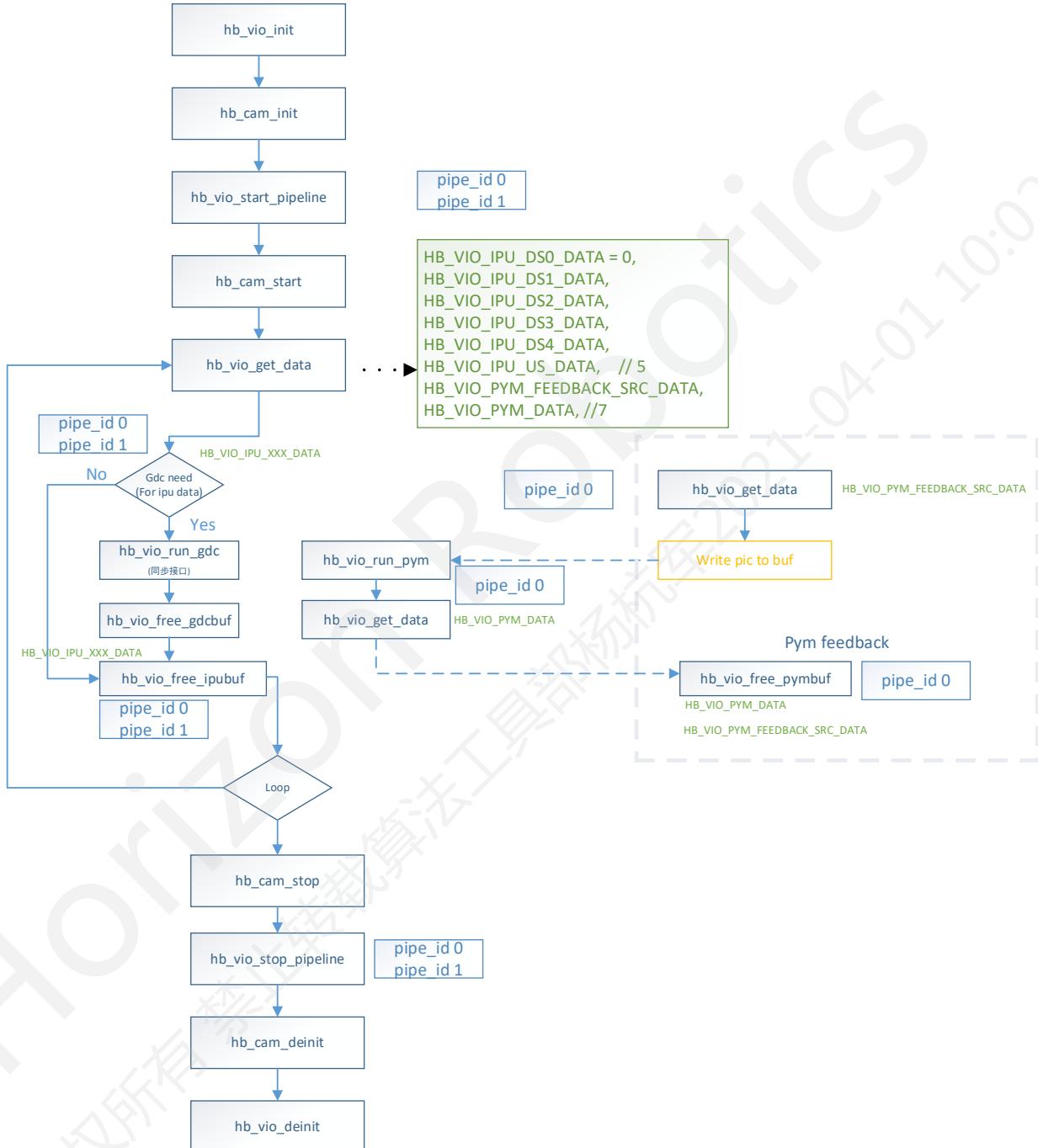
The process of external fallback in this part is consistent with that of fallback mode. In addition, the image data generated by each channel of the IPU (the required size is consistent with the PYM input settings) can also be directly processed offline by PYM. The process is as follows:



VIO video streaming + fallback calling procedure diagram

3) dual channel video + single channel fillback

The dual channel video and single channel fillback are executed by multi-threading. The VIO initialization uses the configuration file of the VIO dual channel(config two pipes in a json file). For the execution process, refer to the single channel scenario. The specific API operations are distinguished by pipe_id.



VIO dual channel video streaming + fillback calling procedure diagram

S

Note: In the case of more channels, the calling process is the same, except that pipe_id is used to distinguish the API calling actions.

4.2 VPS API

4.2.1 hb_vio_get_info

[Function Declaration]

```
int hb_vio_get_info(uint32_t info_type, void* data)
```

[Parameter Description]

- [IN]uint32_t info_type: The type of the message it's waiting.
- [OUT]void *data: The data it gets. Detailed data structure is distinguished as per info_type.

[Return Value]

- Success: Returns HB_OK 0.
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
   * y,uv          2 plane
   * raw           1 plane
   * raw, raw      2 plane(dol2)
   * raw, raw, raw 3 plane(dol3)
   */
}

typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
}
```

```

    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DS0_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, //6
    HB_VIO_DIS_CALLBACK,// Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
}

```

```

} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

}

/*
 * buf type   fd[num]          size[num]          addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)]  size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]      addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,

```

```

// but all channel output alloc one ion buffer avoid buf fragment

typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;

/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
    than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
    of transformation along Y axis
    double angle; // Angle of main projection axis rotation around itself in
    degrees
}

```

```

// universal transformation
double elevation;           ///< Angle in degrees which specify the main projection axis
double azimuth;             ///< Angle in degrees which specify the main projection axis,
                           //counted clockwise from North direction (positive to East)
int32_t keep_ratio;         ///< Keep the same stretching strength in both horizontal and
                           //vertical directions
double FOV_h;               ///< Size of output field of view in vertical dimension in
                           //degrees
double FOV_w;               ///< Size of output field of view in horizontal dimension in
                           //degrees
double cylindricity_y;       ///< Level of cylindricity for target projection shape in
                           //vertical direction
double cylindricity_x;       ///< Level of cylindricity for target projection shape in
                           //horizontal direction
char custom_file[128];        ///< File name of the file containing custom transformation
                           //description
custom_transformation_t custom; ///< Parsed custom transformation structure
double trapezoid_left_angle;  ///< Left Acute angle in degrees between trapezoid base and leg
double trapezoid_right_angle; ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

Perform different operations as per the type and obtain the data structure of that particular type, detailed as below:

[X2/J2, X3/J3 Compatible Functions]:

HB_VIO_PYM_INFO

Single channel input. Obtains the processing results from pym.

HB_VIO_FEEDBACK_SRC_INFO

Retrieve fillback address. Used only in fillback.

[X2/J2 Exclusive Functions]:

HB_VIO_SRC_INFO

Single channel input. Obtains the original data (effective only in ddr mode for now).

HB_VIO_PYM_MULT_INFO

Supports multi-channel input. Obtains the processing results of multiple channel data from pym (X2 supports only 2 channels at most for now)

HB_VIO_FEEDBACK_FLUSH

Fillback memory cache and separately execute the flush operation for once.

HB_VIO_FEEDBACK_SRC_MULT_INFO

Multi-channel fillback. Fillback buff need to be obtained first.

Info types that are to be supported in future.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2; X3/J3

Note: X3/J3 is not recommended. Instead, we recommend hb_vio_get_data().

[Sample Code]

```
#include <stdio.h>
#include "hb_cam_interface.h "
#include "hb_vio_interface.h"

int main(int argc, char *argv[])
{
    int ret = 0;
    img_info_t pym;
    const char *vio_cfg_file = "/etc/vio/imx327_raw_12bit_1952x1097_online_Pipeline.json";
    char *cam_cfg_file = "/etc/cam/hb_xj3dev.json";
    int cam_index = 1;
    int condition_time = 0;
    int need_time_condition = 0;

    // vio cam init.
    ret = hb_vio_init(vio_cfg_file);
    if(ret < 0){
        printf("vio init error %d\n", ret);
        return -1;
    }
    ret = hb_cam_init(cam_index, cam_cfg_file);
    if(ret < 0){
        printf("cam init error %d\n", ret);
        hb_vio_deinit();
        return -1;
    }

    // vio cam start.
```

```
ret = hb_cam_start(0);
if(ret < 0){
    printf("cam start error %d\n", ret);
    hb_cam_deinit(cam_index);
    hb_vio_deinit();
    return -1;
}
ret = hb_vio_start();
if(ret < 0){
    printf("vio start error %d\n", ret);
    hb_cam_stop(0);
    hb_cam_deinit(cam_index);
    hb_vio_deinit();
    return -1;
}

if (need_time_condition == 1)
    ret = hb_vio_get_info_conditional(HB_VIO_PYM_INFO_CONDITIONAL, &pym, condition_time);
else
    ret = hb_vio_get_info(HB_VIO_PYM_INFO, &pym);

if(ret < 0){
    printf("vio get info error %d\n", ret);
} else {
    printf("vio get pym slot=%d id=%d ts=%lld\n",
           pym.slot_id, pym.frame_id, pym.timestamp);
    if (need_time_condition == 1)
        ret = hb_vio_free_info(HB_VIO_PYM_INFO_CONDITIONAL, &pym);
    else
        ret = hb_vio_free_info(HB_VIO_PYM_INFO, &pym);
}

// vio cam stop deinit.
hb_vio_stop();
hb_vio_deinit();
hb_cam_stop(0);
hb_cam_deinit(cam_index);

return ret;
}
```

4.2.2 hb_vio_get_info_conditional

[Function Declaration]

```
int hb_vio_get_info_conditional(uint32_t info_type, void *data, int time);
```

[Parameter Description]

- [IN]uint32_t info_type The info type that is set.
- [OUT]void *data Parameters that are set. Detailed data structure is distinguished as per info_type
- [IN]int time Frame data that meets the time requirements. The obtained frames always meet:
time_need >= time_curr - diff
 - =0 Clears the queue immediately and waits for the next frame.
 - >0 Searches in the queue for the earliest frame with the time larger than or equal to the Current Time-time.
 - <0 Clears the queue immediately and waits for the next frame with the time larger than or equal to the Current Time+|time|.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
   * y,uv          2 plane
   * raw           1 plane
   * raw, raw      2 plane(dol2)
   * raw, raw, raw 3 plane(dol3)
 */

typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
}
```

```

int img_format;
int fd[HB_VIO_BUFFER_MAX_PLANES];      //ion buf fd
uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
uint32_t planeCount;// The number of planes in image
uint32_t dynamic_flag;
uint32_t water_mark_line;//pre-interrupt, not support in XJ3
VIO_DATA_TYPE_E data_type; //Data type of images
buffer_state_e state; // buf status, which is the user status at the user layer.

} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DS0_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
}

```

```

HB_VIO_IPU_US_CALLBACK,
HB_VIO_PYM_CALLBACK, // 6
HB_VIO_DIS_CALLBACK, // Not supported yet
HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO, //isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO, //not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {

```

```

image_info_t img_info;
address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
    than zero)
}

```

```

        double strength;           ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis

        double strengthY;         ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis

        double angle;             ///< Angle of main projection axis rotation around itself in
degrees

        // universal transformation

        double elevation;          ///< Angle in degrees which specify the main projection axis

        double azimuth;            ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)

        int32_t keep_ratio;        ///< Keep the same stretching strength in both horizontal and
vertical directions

        double FOV_h;              ///< Size of output field of view in vertical dimension in
degrees

        double FOV_w;              ///< Size of output field of view in horizontal dimension in
degrees

        double cylindricity_y;      ///< Level of cylindricity for target projection shape in
vertical direction

        double cylindricity_x;      ///< Level of cylindricity for target projection shape in
horizontal direction

        char custom_file[128];       ///< File name of the file containing custom transformation
description

        custom_transformation_t custom; //parsed custom transformation structure
        double trapezoid_left_angle;   ///< Left Acute angle in degrees between trapezoid base and leg
        double trapezoid_right_angle;  ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

Conditionally execute corresponding read operation as per the info_type and time.
 Only HB_VIO_PYM_INFO_CONDITIONAL is supported for now.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2; X3/J3

Note: X3/J3 is not recommended. Instead, we recommend [hb_vio_get_data_conditional\(\)](#).

[Sample Code] Refer to [4.2.1 hb_vio_get_info Sample Code](#).

4.2.3 hb_vio_set_info

[Function Declaration]

```
int hb_vio_set_info(uint32_t info_type, void *data)
```

[Parameter Description]

- [IN]uint32_t info_type: Message type that is set.
- [IN]void *data: Parameters that are set. Detailed data structure is distinguished as per info_type

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 *   * y,uv          2 plane
 *   * raw           1 plane
 *   * raw, raw      2 plane(dol2)
 *   * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
}
```

```

} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DS0_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK,// 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {

```

```

HB_VIO_CALLBACK_ENABLE = 0,
HB_VIO_CALLBACK_DISABLE,
HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
HB_VIO_IPU_US_IMG_INFO, //us channel info
HB_VIO_IPU_DS0_IMG_INFO,
HB_VIO_IPU_DS1_IMG_INFO,
HB_VIO_IPU_DS2_IMG_INFO,
HB_VIO_IPU_DS3_IMG_INFO,
HB_VIO_IPU_DS4_IMG_INFO,
HB_VIO_PYM_IMG_INFO, //pym info
HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
HB_VIO_INFO_MAX

} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
}

```

```

address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
//pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
address_info_t us[6];// Six us channels of pyramid output data address information.
char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym paddr
uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double angle; // Angle of main projection axis rotation around itself in
degrees
    // universal transformation
    double elevation; // Angle in degrees which specify the main projection axis
}

```

```

    double azimuth;           ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio;      ///< Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h;            ///< Size of output field of view in vertical dimension in
degrees
    double FOV_w;            ///< Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y;    ///< Level of cylindricity for target projection shape in
vertical direction
    double cylindricity_x;    ///< Level of cylindricity for target projection shape in
horizontal direction
    char custom_file[128];     ///< File name of the file containing custom transformation
description
    custom_transformation_t custom; ///< Parsed custom transformation structure
    double trapezoid_left_angle;   ///< Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle;  ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;
}

```

- VIO Return Code).

[Function Description]

Perform corresponding set operations as per the info_type, detailed as below:

[X2/J2, X3/J3 Compatible Functions]:

HB_VIO_FEEDBACK_FLUSH

Refresh fillback original image data and make sure they are written to DDR.

[X2/J2 Exclusive Functions]:

HB_VIO_BYPASS_CTRL_INFO

Set bypass ctrl info. Channel & enable can be configured.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2; X3/J3

Note: Should be used together with hb_vio_get_info(). Not recommended being used on X3/J3.

[Sample Code] Refer to 4.5.1 hb_vio_pym_process Sample Code.

4.2.4 hb_vio_free_info

[Function Declaration]

```
int hb_vio_free_info(uint32_t info_type, void *data)
```

[Parameter Description]

- [IN]uint32_t info_type: Message type that is set (corresponding to get info).
- [IN]void *data: [IN]void *data: Parameters that are set. Detailed data structure is distinguished as per info_type

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
}
```

```

        buffer_state_e state; // buf status, which is the user status at the user layer.

} image_info_t;

typedef enum VIO_DATA_TYPE_S {

    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {

    HB_VIO_IPU_DS0_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
    get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

```

```

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]   size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {

```

```

image_info_t pym_img_info;// Pyramid data information
address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
//pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
address_info_t us[6];// Six us channels of pyramid output data address information.
char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
    than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
    of transformation along Y axis
    double angle; // Angle of main projection axis rotation around itself in
    degrees
    // universal transformation
    double elevation; // Angle in degrees which specify the main projection axis
}

```

```

    double azimuth;           ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio;     ///< Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h;           ///< Size of output field of view in vertical dimension in
degrees
    double FOV_w;           ///< Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y;   ///< Level of cylindricity for target projection shape in
vertical direction
    double cylindricity_x;   ///< Level of cylindricity for target projection shape in
horizontal direction
    char custom_file[128];   ///< File name of the file containing custom transformation
description
    custom_transformation_t custom; //parsed custom transformation structure
    double trapezoid_left_angle;  ///< Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle; ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;
}

```

- VIO Return Code).

[Function Description]

Frees the slot of the corresponding info. info_type shall match get info.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2; X3/J3(only for info: HB_VIO_PYM_INFO/HB_VIO_PYM_INFO_CONDITIONAL)

Note: Should be used together with [hb_vio_get_info\(\)](#)/[hb_vio_get_info_conditional\(\)](#). On X3/J3, only single channel type is supported and it's not recommended.

[Sample Code] Refer to [4.2.1 hb_vio_get_info Sample Code](#).

4.2.5 hb_vio_free

[Function Declaration]

```
int hb_vio_free(img_info_t *img_info)
```

[Parameter description]

- [IN]img_info_t *img_info: pym buffer needs to be released

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
}

```

```

HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
HB_VIO_GDC_DATA,//gdc output data type
HB_VIO_IARWB_DATA,//iar display data type
HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
HB_VIO_PYM_LAYER_DATA,//pym all layer data
HB_VIO_MD_DATA,//motion detect data
HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
HB_VIO_PYM_COMMON_DATA,//pym base layer data
HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
}

```

```

HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * gdc Common parameters

```

```

/*
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
    than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
    of transformation along Y axis
    double angle; // Angle of main projection axis rotation around itself in
    degrees
    // universal transformation
    double elevation; // Angle in degrees which specify the main projection axis
    double azimuth; // Angle in degrees which specify the main projection axis,
    counted clockwise from North direction (positive to East)
    int32_t keep_ratio; // Keep the same stretching strength in both horizontal and
    vertical directions
    double FOV_h; // Size of output field of view in vertical dimension in
    degrees
    double FOV_w; // Size of output field of view in horizontal dimension in
    degrees
    double cylindricity_y; // Level of cylindricity for target projection shape in
    vertical direction
    double cylindricity_x; // Level of cylindricity for target projection shape in
    horizontal direction
}

```

```

    char custom_file[128];           ///<-- File name of the file containing custom transformation
description

    custom_transformation_t custom;  ///<-- Parsed custom transformation structure
    double trapezoid_left_angle;    ///<-- Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle;   ///<-- Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

During single channel input, release img_info of this frame after its data is used up.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2

4.2.6 hb_vio_src_free

[Function Declaration]

```
int hb_vio_src_free (src_img_info_t *src_img_info)
```

[Parameter Description]

- [IN]src_img_info_t *src_img_info: Source image buffer needs to be released.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [X3/J3 main](#) parameters

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane

```

```

* raw           1 plane
* raw, raw     2 plane(dol2)
* raw, raw, raw 3 plane(dol3)
*/
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type;//Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
}

```

```

HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO, //isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO, //not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*

```

```

* buf type  fd[num]           size[num]           addr[num]
*
* sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
* sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
* sif dol3: fd[0(raw),1(raw),2(raw)]  size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
* ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]    addr[0(y),1(c)]
* pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
*/
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * _gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
*/

```

```

typedef struct {
    rect_t out_r;           ///< Output window position and size
    transformation_t transform;   ///< Used transformation
    rect_t input_roi_r;
    int32_t pan;           ///< Target shift in horizontal direction from centre of the
                           output image in pixels
    int32_t tilt;           ///< Target shift in vertical direction from centre of the output
                           image in pixels
    double zoom;            ///< Target zoom dimensionless coefficient (must not be bigger
                           than zero)
    double strength;        ///< Dimensionless non-negative parameter defining the strength
                           of transformation along X axis
    double strengthY;       ///< Dimensionless non-negative parameter defining the strength
                           of transformation along X axis
    double angle;           ///< Angle of main projection axis rotation around itself in
                           degrees
    // universal transformation
    double elevation;       ///< Angle in degrees which specify the main projection axis
    double azimuth;          ///< Angle in degrees which specify the main projection axis,
                           counted clockwise from North direction (positive to East)
    int32_t keep_ratio;     ///< Keep the same stretching strength in both horizontal and
                           vertical directions
    double FOV_h;           ///< Size of output field of view in vertical dimension in
                           degrees
    double FOV_w;           ///< Size of output field of view in horizontal dimension in
                           degrees
    double cylindricity_y;  ///< Level of cylindricity for target projection shape in
                           vertical direction
    double cylindricity_x;  ///< Level of cylindricity for target projection shape in
                           horizontal direction
    char custom_file[128];   ///< File name of the file containing custom transformation
                           description
    custom_transformation_t custom; // < Parsed custom transformation structure
    double trapezoid_left_angle; // < Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle; // < Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

During single channel input, release buff of the original image cache (ddr mode only).

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2; X3/J3(Free src & pym buf together)

Note: Should be used together with [hb_vio_get_info\(\)](#). Not recommended being used on X3/J3.

[Sample Code] Refer to [4.5.1 hb_vio_pym_process Sample Code.](#)

4.2.7 hb_vio_mult_free

[Function Declaration]

```
int hb_vio_mult_free (mult_img_info_t *mult_img_info)
```

[Parameter Description]

- [IN]mult_img_info_t *mult_img_info: Cache of the original image needs to be released.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [X3/J3 main](#) parameters)

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
}
```

```

        uint32_t planeCount;// The number of planes in image
        uint32_t dynamic_flag;
        uint32_t water_mark_line;//pre-interrupt, not support in XJ3
        VIO_DATA_TYPE_E data_type; //Data type of images
        buffer_state_e state; // buf status, which is the user status at the user layer.
    } image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DS0_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK,// 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
}

```

```

HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]      addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output

typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

```

```

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment

typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; ///< Output window position and size
    transformation_t transform; ///< Used transformation
    rect_t input_roi_r;
    int32_t pan; ///< Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; ///< Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; ///< Target zoom dimensionless coefficient (must not be bigger
    than zero)
    double strength; ///< Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double strengthY; ///< Dimensionless non-negative parameter defining the strength
    of transformation along X axis
}

```

```

    double angle;           ///< Angle of main projection axis rotation around itself in
degrees

    // universal transformation

    double elevation;      ///< Angle in degrees which specify the main projection axis

    double azimuth;        ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)

    int32_t keep_ratio;    ///< Keep the same stretching strength in both horizontal and
vertical directions

    double FOV_h;          ///< Size of output field of view in vertical dimension in
degrees

    double FOV_w;          ///< Size of output field of view in horizontal dimension in
degrees

    double cylindricity_y;  ///< Level of cylindricity for target projection shape in
vertical direction

    double cylindricity_x;  ///< Level of cylindricity for target projection shape in
horizontal direction

    char custom_file[128];   ///< File name of the file containing custom transformation
description

    custom_transformation_t custom;  ///< Parsed custom transformation structure

    double trapezoid_left_angle;    ///< Left Acute angle in degrees between trapezoid base and leg

    double trapezoid_right_angle;   ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

During multiple channel input, release the cache buffer of the result and original image.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2

4.2.8 hb_vio_mult_src_free

[Function Declaration]

```
int hb_vio_mult_src_free (mult_src_info_t *mult_src_info)
```

[Parameter Description]

- [IN]mult_src_info_t *mult_src_info: Original image needs to be released.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
}

```

```

HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
HB_VIO_GDC_DATA,//gdc output data type
HB_VIO_IARWB_DATA,//iar display data type
HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
HB_VIO_PYM_LAYER_DATA,//pym all layer data
HB_VIO_MD_DATA,//motion detect data
HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
HB_VIO_PYM_COMMON_DATA,//pym base layer data
HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
}

```

```

HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * gdc Common parameters

```

```

/*
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
    than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
    of transformation along Y axis
    double angle; // Angle of main projection axis rotation around itself in
    degrees
    // universal transformation
    double elevation; // Angle in degrees which specify the main projection axis
    double azimuth; // Angle in degrees which specify the main projection axis,
    counted clockwise from North direction (positive to East)
    int32_t keep_ratio; // Keep the same stretching strength in both horizontal and
    vertical directions
    double FOV_h; // Size of output field of view in vertical dimension in
    degrees
    double FOV_w; // Size of output field of view in horizontal dimension in
    degrees
    double cylindricity_y; // Level of cylindricity for target projection shape in
    vertical direction
    double cylindricity_x; // Level of cylindricity for target projection shape in
    horizontal direction
}

```

```

    char custom_file[128];           ///<-- File name of the file containing custom transformation
description

    custom_transformation_t custom;  ///<-- Parsed custom transformation structure
    double trapezoid_left_angle;    ///<-- Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle;   ///<-- Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

During multiple channel input, release the cache buffer of the original image.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2

4.2.9 hb_vio_set_param

[Function Declaration]

```
int hb_vio_set_param(uint32_t pipeline_id, VIO_INFO_TYPE_E info_type, void *info);
```

[Parameter Description]

- [IN]uint32_t pipeline_id: : Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [IN]VIO_INFO_TYPE_E info_type: Message type set by info_type. For the structure, refer to X3/J3 main parameters.
- [IN]void* info: Parameters that has been set. The data structure of the specific parameter is distinguished by info_type.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [X3/J3 main](#) parameters)

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
}

```

```

        uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.

} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */

typedef struct image_info_s {

    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {

    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
}

```

```

HB_VIO_PYM_COMMON_DATA, //pym base layer data
HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO, //isp info, not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO, //not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
}

```

```

        BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]         addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]     size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;

```

```

/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r;           ///< Output window position and size
    transformation_t transform;   ///< Used transformation
    rect_t input_roi_r;
    int32_t pan;           ///< Target shift in horizontal direction from centre of the
output image in pixels
    int32_t tilt;           ///< Target shift in vertical direction from centre of the output
image in pixels
    double zoom;           ///< Target zoom dimensionless coefficient (must not be bigger
than zero)
    double strength;        ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double strengthY;       ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double angle;           ///< Angle of main projection axis rotation around itself in
degrees
    // universal transformation
    double elevation;        ///< Angle in degrees which specify the main projection axis
    double azimuth;          ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio;      ///< Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h;           ///< Size of output field of view in vertical dimension in
degrees
    double FOV_w;           ///< Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y;    ///< Level of cylindricity for target projection shape in
vertical direction
    double cylindricity_x;    ///< Level of cylindricity for target projection shape in
horizontal direction
    char custom_file[128];    ///< File name of the file containing custom transformation
description
    custom_transformation_t custom;  ///< Parsed custom transformation structure
    double trapezoid_left_angle;  ///< Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle;  ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

Execute corresponding set operation as per info_type, detailed as below:

- HB_VIO_CALLBACK_ENABLE, Enables data callback.
- HB_VIO_CALLBACK_DISABLE, Disables data callback.

Supports enabling/disabling the callback. Other info types are not supported.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to hb_vio_set_callbacks.

4.2.10 hb_vio_get_param

[Function Declaration]

```
int hb_vio_get_param(uint32_t pipeline_id, VIO_INFO_TYPE_E data_type, void *info);
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [IN]VIO_INFO_TYPE_E info_type: : Message type set by info_type. For the structure, refer to refer to X3/J3 main parameters.
- [OUT]void* info : Data it has been waited for. The data structure of the specific parameter is distinguished by info_type.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [X3/J3 main](#) parameters)

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width; // Image data width
    uint16_t height; // Image data width
    uint16_t stride_size; // Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;
/* info :
 * y,uv           2 plane
```

```

* raw           1 plane
* raw, raw     2 plane(dol2)
* raw, raw, raw 3 plane(dol3)
*/
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type;//Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
}

```

```

HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO, //isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO, //not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*

```

```

* buf type  fd[num]           size[num]           addr[num]
*
* sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
* sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
* sif dol3: fd[0(raw),1(raw),2(raw)]  size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
* ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]    addr[0(y),1(c)]
* pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
*/
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * _gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
*/

```

```

typedef struct {
    rect_t out_r;           ///< Output window position and size
    transformation_t transform;   ///< Used transformation
    rect_t input_roi_r;
    int32_t pan;           ///< Target shift in horizontal direction from centre of the
                           output image in pixels
    int32_t tilt;           ///< Target shift in vertical direction from centre of the output
                           image in pixels
    double zoom;            ///< Target zoom dimensionless coefficient (must not be bigger
                           than zero)
    double strength;        ///< Dimensionless non-negative parameter defining the strength
                           of transformation along X axis
    double strengthY;       ///< Dimensionless non-negative parameter defining the strength
                           of transformation along X axis
    double angle;           ///< Angle of main projection axis rotation around itself in
                           degrees
    // universal transformation
    double elevation;       ///< Angle in degrees which specify the main projection axis
    double azimuth;          ///< Angle in degrees which specify the main projection axis,
                           counted clockwise from North direction (positive to East)
    int32_t keep_ratio;     ///< Keep the same stretching strength in both horizontal and
                           vertical directions
    double FOV_h;           ///< Size of output field of view in vertical dimension in
                           degrees
    double FOV_w;           ///< Size of output field of view in horizontal dimension in
                           degrees
    double cylindricity_y;   ///< Level of cylindricity for target projection shape in
                           vertical direction
    double cylindricity_x;   ///< Level of cylindricity for target projection shape in
                           horizontal direction
    char custom_file[128];   ///< File name of the file containing custom transformation
                           description
    custom_transformation_t custom; // < Parsed custom transformation structure
    double trapezoid_left_angle; // < Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle; // < Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

Execute different operations as per type and obtain the data structure of that particular type corresponding to the information supported by set_info.

- HB_VIO_CALLBACK_ENABLE = 0, Gets the current status of callback control
- HB_VIO_CALLBACK_DISABLE, Gets the current status of callback control
- HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
- HB_VIO_IPU_US_IMG_INFO, Gets ipu u channel info (ref to chn_img_info_t)
- HB_VIO_IPU_DS0_IMG_INFO, Gets ipu ds0 channel info
- HB_VIO_IPU_DS1_IMG_INFO, Gets ipu ds1 channel info
- HB_VIO_IPU_DS2_IMG_INFO, Gets ipu ds2 channel info
- HB_VIO_IPU_DS3_IMG_INFO, Gets ipu ds3 channel info
- HB_VIO_IPU_DS4_IMG_INFO, Gets ipu ds4 channel info
- HB_VIO_PYM_IMG_INFO, Gets pym all channels info

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to hb_vio_set_callbacks.

4.2.11 hb_vio_get_data

[Function Declaration]

```
int hb_vio_get_data(uint32_t pipeline_id, VIO_DATA_TYPE_E data_type, void *data);
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [IN] VIO_DATA_TYPE_E data_type: Different retrievable data type. For the structure, refer to X3/J3 main parameters.
- [OUT]void* data : Data it has been waited for. The data structure of the specific parameter is distinguished by info_type.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
```

```

} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 **/


typedef struct image_info_s {

    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {

    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
}

```

```

HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX

} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK,// 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK

} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DSO_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
    HB_VIO_INFO_MAX

} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
}

```

```

} buffer_state_e;

/*
 * buf type    fd[num]          size[num]          addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)]  size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]      addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */

// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***

```

```

* Window definition and transformation parameters
* For parameters meaning read GDC guide
*/
typedef struct {
    rect_t out_r;           ///< Output window position and size
    transformation_t transform;  ///< Used transformation
    rect_t input_roi_r;
    int32_t pan;           ///< Target shift in horizontal direction from centre of the
output image in pixels
    int32_t tilt;           ///< Target shift in vertical direction from centre of the output
image in pixels
    double zoom;           ///< Target zoom dimensionless coefficient (must not be bigger
than zero)
    double strength;        ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double strengthY;       ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double angle;           ///< Angle of main projection axis rotation around itself in
degrees
    // universal transformation
    double elevation;       ///< Angle in degrees which specify the main projection axis
    double azimuth;          ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio;      ///< Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h;           ///< Size of output field of view in vertical dimension in
degrees
    double FOV_w;           ///< Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y;   ///< Level of cylindricity for target projection shape in
vertical direction
    double cylindricity_x;   ///< Level of cylindricity for target projection shape in
horizontal direction
    char custom_file[128];    ///< File name of the file containing custom transformation
description
    custom_transformation_t custom;  ///< Parsed custom transformation structure
    double trapezoid_left_angle;  ///< Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle;  ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

Execute different operations to obtain data structure of corresponding type.

- HB_VIO_IPU_DS0_DATA = 0, Gets ipu ds0 channel data
- HB_VIO_IPU_DS1_DATA, Gets ipu ds1 channel data
- HB_VIO_IPU_DS2_DATA, Gets ipu ds2 channel data
- HB_VIO_IPU_DS3_DATA, Gets ipu ds3 channel data
- HB_VIO_IPU_DS4_DATA, Gets ipu ds4 channel data
- HB_VIO_IPU_US_DATA, Gets ipu us channel data
- HB_VIO_PYM_FEEDBACK_SRC_DATA, Gets pym fillback source buf
- HB_VIO_PYM_DATA, Gets pyramid processing data
- HB_VIO_PYM_COMMON_DATA, Gets pyramid processing data (only base layer data, not used together with HB_VIO_PYM_DATA)

Other types cannot be retrieved from this interface.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
#include "hb_vio_interface.h"
#include "hb_cam_interface.h"

typedef struct work_info_s {
    uint32_t group_id;
    VIO_DATA_TYPE_E data_type;
    pthread_t thid;
    int running;
} work_info_t;
work_info_t work_info[GROUP_MAX][20];

int vio_md_func(work_info_t * info)
{
    int ret = 0;
    int pipeId = info->pipe_id;

    printf("hb_vio_motion_detect begin!!! ret %d \n", ret);
    ret = hb_vio_motion_detect(pipeId);
    if (ret < 0) {
        printf("hb_vio_motion_detect error!!! ret %d \n", ret);
    } else {
        printf("md_func success!!! ret %d \n", ret);
        hb_vio_disable_md(pipeId);
    }
}
```

```
        printf("hb_vio_disable_md success!!! ret %d \n", ret);
    }
    return ret;
}
void *work_md_func_thread(void* arg)
{
    printf("work_md_func_thread begin\n");
    work_info_t *info = (work_info_t *) arg;
    while (check_end()) {
        vio_md_func(info);
    }
    printf("work_md_func_thread end\n");
    return NULL;
}
void work_info_init(void)
{
    for (int i = 0; i < PIPE_MAX; i++) {
        for (int j = 0; j < HB_VIO_DATA_TYPE_MAX; j++) {
            work_info[i][j].pipe_id = i;
            work_info[i][j].data_type = j;
            work_info[i][j].running = 0;
        }
    }
}
int main(int argc, char *argv[])
{
    int ret = 0;
    int condition_time = 0;
    int need_time_condition = 0;
    hb_vio_buffer_t buf = {0};
    char *cam_cfg_file = "/etc/cam/hb_xj3dev.json";
    work_info_init();
    ret = hb_vio_init("/etc/vio/imx327_raw_12bit_1952x1097_online_Pipeline.json");
    if (ret < 0) {
        printf("vio init fail\n");
        return -1;
    }
    ret = hb_cam_init(0, cam_cfg_file);
    if (ret < 0) {
        printf("cam init fail\n");
        hb_vio_deinit();
        return -1;
    }
    ret = hb_vio_start_pipeline(0);
```

```

    if (ret < 0) {
        printf("vio start fail, do cam&vio_deinit.\n");
        hb_cam_deinit(0);
        hb_vio_deinit();
        return -1;
    }

    ret = hb_vio_enable_md(0);
    if (ret < 0) {
        printf("cam start fail, do cam&vio_deinit.\n");
    }

    ret = pthread_create(&work_info[0][HB_VIO_MD_DATA].thid,
                         NULL,
                         (void *)work_md_func_thread,
                         (void *)&work_info[0][HB_VIO_MD_DATA]);
    printf("pipe(%d)Test work md_data thread running.\n", 0);

    ret = hb_cam_start(0);
    if (ret < 0) {
        printf("cam start fail, do cam&vio_deinit.\n");
        goto err;
    }

    if (need_time_condition == 1) {
        ret = hb_vio_get_data_conditional(0,
                                         HB_VIO_IPU_US_DATA, &buf, condition_time);
    } else {
        ret = hb_vio_get_data(0, HB_VIO_IPU_US_DATA, &buf);
    }

    if (ret < 0) {
        printf("pipe 0 get us data failed!\n");
        goto err;
    }

    ret = hb_vio_free_ipubuf(0, &buf);
    if (ret < 0) {
        printf("pipe 0 free us data failed!\n");
    }

    pthread_join(work_info[0][HB_VIO_MD_DATA].thid, NULL);
    hb_cam_stop(0);
    hb_vio_stop_pipeline(0);
    hb_cam_deinit(0);
    hb_vio_deinit();
    return 0;

err:
    hb_cam_stop(0);
    hb_cam_deinit(0);
}

```

```

    hb_vio_deinit();
    return -1;
}

```

4.2.12 hb_vio_get_data_conditional

[Function Declaration]

```
int hb_vio_get_data_conditional(uint32_t pipeline_id, VIO_DATA_TYPE_E data_type, void *data, int time);
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [IN] VIO_DATA_TYPE_E data_type: Different retrievable data type. For the structure, refer to X3/J3 main parameters.
- [OUT]void* data: Data it has been waited for. The data structure of the specific parameter is distinguished by info_type.
- [IN]int time: Time limit for marking the required data.

time = 0: Clear the currently completed data and wait for the next frame of data.

time < 0: Clear the currently completed data and wait for the frame that meets the requirement:

Frame Time > (Current Time – time)

time > 0: Wait for the frame that meets the requirement: Frame Time > (Current Time – time)

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to [X3/J3 main](#) parameters)

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
   * y,uv          2 plane
   * raw           1 plane
   * raw, raw      2 plane(dol2)
   * raw, raw, raw 3 plane(dol3)

```

```
*/
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;
```

```

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DS0_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO, //isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO, //not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type fd[num]          size[num]          addr[num]
 *
 * sif buf : fd[0(raw)]      size[0(raw)]      addr[0(raw)]
 */

```

```

* sif dol2: fd[0(raw),1(raw)]           size[0(raw),1(raw)]      addr[0(raw),1(raw)]
* sif dol3: fd[0(raw),1(raw),2(raw)]    size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
* ipu buf : fd[0(y),1(c)]              size[0(y),1(c)]        addr[0(y),1(c)]
* pym buf : fd[0(all channel)]        size[0(all output)]     addr[0(y),1(c)] * 24
* */

// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in;       // input frame resolution
    resolution_t out;      // output frame resolution
    int32_t x_offset;      // center offset for input x coordinate
    int32_t y_offset;      // center offset for input y coordinate
    int32_t diameter;      // diameter of equivalent 180 field of view in pixels
    double fov;            // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r;          ///< Output window position and size
    transformation_t transform; // Used transformation
}

```

```

rect_t input_roi_r;
int32_t pan;                                ///< Target shift in horizontal direction from centre of the
output image in pixels
int32_t tilt;                               ///< Target shift in vertical direction from centre of the output
image in pixels
double zoom;                                ///< Target zoom dimensionless coefficient (must not be bigger
than zero)
double strength;                            ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
double strengthY;                           ///< Dimensionless non-negative parameter defining the strength
of transformation along Y axis
double angle;                               ///< Angle of main projection axis rotation around itself in
degrees
// universal transformation
double elevation;                          ///< Angle in degrees which specify the main projection axis
double azimuth;                            ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
int32_t keep_ratio;                         ///< Keep the same stretching strength in both horizontal and
vertical directions
double FOV_h;                               ///< Size of output field of view in vertical dimension in
degrees
double FOV_w;                               ///< Size of output field of view in horizontal dimension in
degrees
double cylindricity_y;                     ///< Level of cylindricity for target projection shape in
vertical direction
double cylindricity_x;                     ///< Level of cylindricity for target projection shape in
horizontal direction
char custom_file[128];                      ///< File name of the file containing custom transformation
description
custom_transformation_t custom; //parsed custom transformation structure
double trapezoid_left_angle;    ///< Left Acute angle in degrees between trapezoid base and leg
double trapezoid_right_angle;   ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

Execute different operations to obtain data structure of corresponding type as per type and corresponding time parameter.

- HB_VIO_IPU_DS0_DATA = 0, Get ipu ds0 channel data
- HB_VIO_IPU_DS1_DATA, Get ipu ds1 channel data

- HB_VIO_IPU_DS2_DATA, Get ipu ds2 channel data
- HB_VIO_IPU_DS3_DATA, Get ipu ds3 channel data
- HB_VIO_IPU_DS4_DATA, Get ipu ds4 channel data
- HB_VIO_IPU_US_DATA, Get ipu us channel data
- HB_VIO_PYM_DATA, Get pyramid processing data
- HB_VIO_PYM_COMMON_DATA, Gets pyramid processing data (only base layer data, not used together with HB_VIO_PYM_DATA)

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to 4.2.11hb_vio_get_data.

4.2.13 hb_vio_set_callbacks

[Function Declaration]

```
int hb_vio_set_callbacks(uint32_t pipeline_id, VIO_CALLBACK_TYPE_E type, data_cb cb);
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [IN]VIO_CALLBACK_TYPE_T type: Data type needs to be called back. For the structure, refer to X3/J3 main parameters.
- [IN]data_cb cb; Data callback interface.

[Return Value]

- Success: Returns HB_OK 0
- Failure: Returns negative error code (refer to to *X3/J3 main* parameters)

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;
/* info :
 * y,uv          2 plane
```

```

* raw           1 plane
* raw, raw     2 plane(dol2)
* raw, raw, raw 3 plane(dol3)
*/
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type;//Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
}

```

```

HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO, //isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO, //not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*

```

```

* buf type  fd[num]           size[num]           addr[num]
*
* sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
* sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
* sif dol3: fd[0(raw),1(raw),2(raw)]  size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
* ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]    addr[0(y),1(c)]
* pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
*/
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * _gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
*/

```

```

typedef struct {
    rect_t out_r;           ///< Output window position and size
    transformation_t transform;   ///< Used transformation
    rect_t input_roi_r;
    int32_t pan;           ///< Target shift in horizontal direction from centre of the
                           output image in pixels
    int32_t tilt;           ///< Target shift in vertical direction from centre of the output
                           image in pixels
    double zoom;            ///< Target zoom dimensionless coefficient (must not be bigger
                           than zero)
    double strength;        ///< Dimensionless non-negative parameter defining the strength
                           of transformation along X axis
    double strengthY;       ///< Dimensionless non-negative parameter defining the strength
                           of transformation along X axis
    double angle;           ///< Angle of main projection axis rotation around itself in
                           degrees
    // universal transformation
    double elevation;       ///< Angle in degrees which specify the main projection axis
    double azimuth;          ///< Angle in degrees which specify the main projection axis,
                           counted clockwise from North direction (positive to East)
    int32_t keep_ratio;     ///< Keep the same stretching strength in both horizontal and
                           vertical directions
    double FOV_h;           ///< Size of output field of view in vertical dimension in
                           degrees
    double FOV_w;           ///< Size of output field of view in horizontal dimension in
                           degrees
    double cylindricity_y;  ///< Level of cylindricity for target projection shape in
                           vertical direction
    double cylindricity_x;  ///< Level of cylindricity for target projection shape in
                           horizontal direction
    char custom_file[128];   ///< File name of the file containing custom transformation
                           description
    custom_transformation_t custom; // < Parsed custom transformation structure
    double trapezoid_left_angle; // < Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle; // < Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

Set the callback function for each data channel. Multiple calls are needed for multi-channel input. Pym callback are used in online mode while NULL is set in offline mode.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
#include "hb_vio_interface.h"
#include "x2_camera.h"
#include "../utils/list.h"

typedef struct callback_list_s {
    struct list_head cb_process;
    pthread_t cb_pid;
    pthread_mutex_t cb_mutex;
    int run_enable;
} callback_list_t;
callback_list_t cb_list;

void *callback_data_work_thread()
{
    struct list_head *this, *next;
    callback_data_t *cb_data;
    int cnt = 0;

    while (cb_list.run_enable) {
        pthread_mutex_lock(&cb_list.cb_mutex);

        list_for_each_safe(this, next, &cb_list.cb_process) {
            cb_data = (callback_data_t *)this;
            vio_list_del(&cb_data->node);

            if (cb_data->cb_type >= HB_VIO_IPU_DS0_CALLBACK &&
                cb_data->cb_type <= HB_VIO_IPU_US_CALLBACK) {
                hb_vio_free_ipubuf(cb_data->pipe_id, &cb_data->ipu_buffer);
                printf("callback ipu buffer free done\n");
            } else if (cb_data->cb_type == HB_VIO_PYM_CALLBACK) {
                hb_vio_free_pymbuf(cb_data->pipe_id, HB_VIO_PYM_DATA,
                    &cb_data->pym_buffer);
                printf("callback pym buffer free done\n");
            } else {
                printf("error callback data type\n");
            }
            free(cb_data);
            cnt++;
        }
    }
}
```

```

    }

    pthread_mutex_unlock(&cb_list.cb_mutex);
    usleep(20000);
}

printf("callback work thread exit\n");
pthread_exit((void *)1);
}

static void vio_data_receive(uint32_t pipe_id, uint32_t info, void *data,
                            void *userdata)
{
    callback_data_t *cb_data;
    printf("pipe_id(%u), callback type(%u) data receiverd.\n", pipe_id, info);

    switch (info) {
        case HB_VIO_IPU_US_CALLBACK:
        case HB_VIO_IPU_DS0_CALLBACK:
        case HB_VIO_IPU_DS1_CALLBACK:
        case HB_VIO_IPU_DS2_CALLBACK:
        case HB_VIO_IPU_DS3_CALLBACK:
        case HB_VIO_IPU_DS4_CALLBACK:
            if(data != NULL) {
                cb_data = malloc(sizeof(callback_data_t));
                memcpy(&cb_data->ipu_buffer, (hb_vio_buffer_t *)data,
                       sizeof(hb_vio_buffer_t));
                cb_data->cb_type = info;
                vio_list_add_tail(&cb_data->node, &cb_list.cb_process);
            }
            break;
        case HB_VIO_PYM_CALLBACK:
            if(data != NULL) {
                cb_data = malloc(sizeof(callback_data_t));
                memcpy(&cb_data->pym_buffer, (pym_buffer_t *)data,
                       sizeof(pym_buffer_t));
                cb_data->cb_type = info;
                vio_list_add_tail(&cb_data->node, &cb_list.cb_process);
            }
            break;
        default:
            printf("invaild callback type.\n");
            break;
    }
}
}

```

```
int main(int argc, char *argv[])
{
    int ret = 0;
    int cb_mesg = -1;
    int cb_info_type = -1;
    int cb_en_status = -1;
    char *cam_cfg_file = "/etc/cam/hb_xj3dev.json";
    ret = hb_vio_init("/etc/vio/imx327_raw_12bit_1952x1097_online_Pipeline.json");
    if (ret < 0) {
        printf("vio init fail\n");
        return -1;
    }
    ret = hb_cam_init(0, cam_cfg_file);
    if (ret < 0) {
        printf("cam init fail\n");
        hb_vio_deinit();
        return -1;
    }
    ret = hb_vio_start_pipeline(0);
    if (ret < 0) {
        printf("vio start fail, do cam&vio_deinit.\n");
        hb_cam_deinit(0);
        hb_vio_deinit();
        return -1;
    }
    ret = hb_cam_start(0);
    if (ret < 0) {
        printf("cam start fail, do cam&vio_deinit.\n");
        goto err;
    }
    cb_mesg = HB_VIO_IPU_DS0_CB_MSG;
    cb_info_type = HB_VIO_IPU_DS0_CALLBACK;

    pthread_mutex_init(&cb_list.cb_mutex, NULL);
    vio_init_list_head(&cb_list.cb_process);
    cb_list.run_enable = 1;
    ret = pthread_create(&cb_list.cb_pid, NULL,
                        (void *)callback_data_work_thread, NULL);
    if (ret < 0) {
        printf("callback thread start fail\n");
        goto err;
    }
    hb_vio_set_param(pipe_id, HB_VIO_CALLBACK_ENABLE, (void *)&cb_mesg);
    hb_vio_set_callbacks(pipe_id, cb_info_type, vio_data_receive);
```

```

hb_vio_get_param(pipe_id, HB_VIO_CALLBACK_ENABLE, (void *)&cb_en_status);
printf("callback set param done, cb_en_status = %d\n", cb_en_status);
chn_img_info_t info;
VIO_INFO_TYPE_E info_type = HB_VIO_IPU_DS0_IMG_INFO;
hb_vio_get_param(pipe_id, info_type, &info);
printf("chn info = %d\n", info.width, info.height);

VIO_INFO_TYPE_E info_type = HB_VIO_PYM_DATA;
chn_img_info_t pym_info[HB_VIO_PYM_MAX_LAYER];
hb_vio_get_param(pipe_id, info_type, &pym_info);
printf("pym chn0 info = %d\n", pym_info[0].width, pym_info[0].height);

usleep(2000000);
cb_list.run_enable = 0;
pthread_join(cb_list.cb_pid, &pret);

hb_cam_stop(0);
hb_vio_stop_pipeline(0);
hb_cam_deinit(0);
hb_vio_deinit();
return 0;

err:
hb_cam_stop(0);
hb_cam_deinit(0);
hb_vio_deinit();
return -1;
}

```

4.3 ISP API

4.3.1 hb_isp_get_ae_statistics

[Function Declaration]

```
int hb_isp_get_ae_statistics(uint32_t ctx_id, isp_statistics_t *isp_statistics, int time_out);
```

[Parameter Description]

- [IN] uint32_t ctx_id: Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [IN] int time_out: unit-Ms. Get data timeout value, timeout failure return.
- [OUT] isp_statistics_t *isp_statistics: Get the AE statistics.

[Return Value]

- Success: 0
- Failure: -1

[Function Description]

Get the AE statistics.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

4.3.2 [hb_isp_release_ae_statistics](#)

[Function Declaration]

```
int hb_isp_release_ae_statistics(uint32_t ctx_id, isp_statistics_t *isp_statistics);
```

[Parameter Description]

- [IN] uint32_t ctx_id: Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [IN] isp_statistics_t *isp_statistics: Release the AE statistics.

[Return Value]

- Success: 0
- Failure: -1

[Function Description]

Get the AE statistics.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
int i;
int ret = -1;
isp_statistics_t isp_statistics;
uint32_t ae[HB_ISP_FULL_HISTOGRAM_SIZE];
isp_statistics.crc_en = 1;
memset(ae, 0, sizeof(ae));
ret = hb_isp_get_ae_statistics(ctx_num - 1, &isp_statistics, 1000);
if (ret != 0) {
```

```

    printf("GET AWB METE DATA ERROR!\n");
    break;
}

memcpy(ae, isp_statistics.data, isp_statistics.len);
printf("\n--AE--\n");
for (i = 0; i < HB_ISP_FULL_HISTOGRAM_SIZE; i++) {
    printf("%-8d ", ae[i]);
    if ((i + 1) % 16 == 0)
        printf("\n");
}
hb_isp_release_awb_statistics(ctx_num - 1, &isp_statistics);

```

4.3.3 hb_isp_get_awb_statistics

[Function Declaration]

```
int hb_isp_get_awb_statistics(uint32_t ctx_id, isp_statistics_t *isp_statistics, int time_out);
```

[Parameter Description]

- [IN] uint32_t ctx_id: Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [IN] int time_out: unit-Ms. Get data timeout value, timeout failure return.
- [OUT] isp_statistics_t *isp_statistics: Get the AWB statistics.

[Return Value]

- Success: 0
- Failure: -1

[Function Description]

Get the AWB statistics.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

4.3.4 hb_isp_release_awb_statistics

[Function Declaration]

```
int hb_isp_release_awb_statistics(uint32_t ctx_id, isp_statistics_t *isp_statistics);
```

[Parameter Description]

- [IN] uint32_t ctx_id: Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [IN] isp_statistics_t *isp_statistics: Release the AWB statistics.

[Return Value]

- Success: 0
- Failure: -1

[Function Description]

Get the AWB statistics.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
int i = 0;
int ret = -1;
isp_statistics_t isp_statistics;
isp_awb_statistics_s awb[HB_ISP_MAX_AWB_ZONES];
isp_statistics.crc_en = 1;
memset(awb, 0, sizeof(awb));
ret = hb_isp_get_awb_statistics(ctx_num - 1, &isp_statistics, 1000);
if (ret != 0) {
    printf("GET AWB METE DATA ERROR!\n");
    break;
}
memcpy(awb, isp_statistics.data, isp_statistics.len);
printf("\n--AWB--\n");
for (i = 0; i < HB_ISP_MAX_AWB_ZONES; i++) {
    printf("%-4d", awb[i].rg);
    if ((i + 1) % 30 == 0)
        printf("\n");
}
printf("\n--rg end--\n");
for (i = 0; i < HB_ISP_MAX_AWB_ZONES; i++) {
    printf("%-4d", awb[i].bg);
    if ((i + 1) % 30 == 0)
        printf("\n");
}
printf("\n--bg end--\n");
for (i = 0; i < HB_ISP_MAX_AWB_ZONES; i++) {
    printf("%-5d", awb[i].sum);
```

```
if ((i + 1) % 24 == 0)
    printf("\n");
}

printf("\n---sum end---\n");
hb_isp_release_awb_statistics(ctx_num - 1, &isp_statistics);
```

4.3.5 hb_isp_set_context

[Function Declaration]

```
int hb_isp_set_context(uint32_t ctx_id, const isp_context_t *ptx);
```

[Parameter Description]

- [IN] uint32_t ctx_id: Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [OUT] isp_context_t *ptx: ISP context data.

[Return Value]

- Success: 0
- Failure: -1

[Function Description]

Set ISP context data.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
isp_context_t ptx;

ptx.ptr = calloc(CTX_SIZE, 1);
if (!ptx.ptr) {
    printf("alloc mem failed\n");
    return -1;
}

fp = fopen(file_name, "rb");
if (fp == NULL) {
    printf("isp ctx file(%s) open failed\n", file_name);
    return -1;
}
```

```

count = fread(&ptx.crc16, 1, sizeof(ptx.crc16), fp);
count = fread(ptx.ptr, 1, CTX_SIZE, fp);

printf("ctx data size %d, crc16 is 0x%08x \n", count, ptx.crc16);

fclose(fp);

ret = hb_isp_set_context(0, &ptx);
if (ret > 0)
    printf("isp context set success\n");

free(ptx.ptr);

```

4.3.6 hb_isp_get_context

[Function Declaration]

```
int hb_isp_get_context(uint32_t ctx_id, isp_context_t *ptx);
```

[Parameter Description]

- [IN] uint32_t ctx_id: Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [OUT] isp_context_t *ptx: ISP context data.

[Return Value]

- Success: 0
- Failure: -1

[Function Description]

Obtain ISP context data.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```

isp_context_t ptx;

ptx.ptr = calloc(CTX_SIZE, 1);
if (!ptx.ptr) {
    printf("alloc mem failed\n");
    return -1;
}

```

```
ptx.crc16 = crc16(~0, ptx.ptr, CTX_SIZE);

fp = fopen(FILE_NAME, "wb");
if (!fp) {
    printf("file %s open failed.\n", FILE_NAME);
    free(ptx.ptr);
    return -1;
}

ret = hb_isp_get_context(0, &ptx);
if (ret > 0) {
    fwrite(&ptx.crc16, sizeof(ptx.crc16), 1, fp);
    fwrite(ptx.ptr, CTX_SIZE, 1, fp);
}

free(ptx.ptr);
hb_isp_dev_deinit();
```

4.3.7 hb_isp_dev_init

[Function Declaration]

```
int hb_isp_dev_init(void);
```

[Parameter Description] None

[Return Value]

- Success: 0
- Failure: -1

[Function Description]

Initialize ISP parameter settings / the obtained character device.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] None

4.3.8 hb_isp_dev_deinit

[Function Declaration]

```
int hb_isp_dev_deinit(void);
```

[Parameter Description] None

[Return Value]

- Success: 0
- Failure: -1

[Function Description]

Deinitialize ISP parameter settings / the obtained character device.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] None

4.3.9 hb_isp_iridix_ctrl

[Function Declaration]

```
int hb_isp_iridix_ctrl(uint32_t provider_ctx_id, uint32_t user_ctx_id, uint8_t flag);
```

[Parameter Description]

- [IN] uint32_t provider_ctx_id: context id that give up iridix function.
- [IN] uint32_t user_ctx_id: context id that accept iridix function.
- [IN] uint8_t flag: reserved for future use, 0 default.

[Return Value]

- Success: 0
- Failure: -1

[Function Description]

Due to ISP hardware limitation, it can only support 4-way PWL. This interface is used to transfer the iridix function of provider_ctx_id to user_ctx_id, need to combine with hb_cam_dynamic_switch_mode interface.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
void *mode_switch_thread(void *arg)
{
    int ret = 0;
    int pipeid_a, pipeid_b;
```

```
char str[4];
char *linear_path = "/etc/cam/lib_ar0233_linear.so";
char *pwl_path = "/etc/cam/lib_ar0233_pwl.so";

camera_info_table_t linear;
camera_info_table_t pwl;

memset(&linear, 0, sizeof(linear));
memset(&pwl, 0, sizeof(pwl));

strcpy(linear.name, "linear");
linear.mode = 1;
linear.bit_width = 12;
linear.cfa_pattern = 0;
strcpy((char *)(linear.calib_path), linear_path);

strcpy(pwl.name, "pwl");
pwl.mode = 5;
pwl.bit_width = 12;
pwl.cfa_pattern = 1;
strcpy((char *)(pwl.calib_path), pwl_path);

while (check_end()) {

    printf("input switch cmd:>\n");

    scanf("%s", str);
    pipeid_a = str[0] - 48;
    pipeid_b = str[1] - 48;

    if (pipeid_a < 0 || pipeid_a > 8) {
        printf("pipeid_a %d is invalid.\n", pipeid_a);
        continue;
    }

    if (pipeid_b < 0 || pipeid_b > 8) {
        printf("pipeid_b %d is invalid.\n", pipeid_b);
        continue;
    }

    printf("pipe %d, pipe %d exchange mode\n", pipeid_a, pipeid_b);
    ret = hb_cam_dynamic_switch_mode(pipeid_a, &linear);
    if (ret < 0) {
        printf("pipe %d, switch to mode %d failed.\n", pipeid_a, linear.mode);
    }
}
```

```

        continue;
    }

    ret = hb_isp_iridix_ctrl(pipeid_a, pipeid_b, 1);
    if (ret < 0) {
        printf("pipe %d, pipe %d iridix ctrl failed.\n", pipeid_a, pipeid_b);
        continue;
    }

    ret = hb_cam_dynamic_switch_mode(pipeid_b, &pwl);
    if (ret < 0) {
        printf("pipe %d, switch to mode %d failed.\n", pipeid_a, pwl.mode);
    }

    usleep(50 * 1000);
}

return NULL;
}

```

4.3.10 ISP Parameter Description

Macros used in the sample code:

```
#define CTX_SIZE      (0x18e88 - 0xab6c + 0x4000)
#define HB_ISP_MAX_AWB_ZONES (33 * 33)
#define HB_ISP_FULL_HISTOGRAM_SIZE 1024
```

AWB statistic structure definition:

```
typedef struct{
    bool crc_en;
    void *data;
    uint32_t len;
    uint32_t frame_id;
    uint64_t timestamp;
    uint32_t buf_idx;
} isp_statistics_t;
```

ISP context data structure definition:

```
typedef struct {
    uint32_t frame_id;
    uint64_t timestamp;
    uint16_t crc16;
    void *ptr;
    uint32_t len;
} isp_context_t;
```

4.4 IPU API

4.4.1 hb_vio_free_ipubuf

[Function Declaration]

```
int hb_vio_free_ipubuf (uint32_t pipeline_id, hb_vio_buffer_t *dst_img_info);
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [IN]hb_vio_buffer_t *dst_img_info: Memory information to be released, see X3/J3 main parameters for the structure.

[Return Value]

- Success: Returns HB_OK 0.
- Failure: Returns negative error code (refer to *X3/J3 main* parameters

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
}
```

```

        uint32_t planeCount;// The number of planes in image
        uint32_t dynamic_flag;
        uint32_t water_mark_line;//pre-interrupt, not support in XJ3
        VIO_DATA_TYPE_E data_type; //Data type of images
        buffer_state_e state; // buf status, which is the user status at the user layer.
    } image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DS0_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK,// 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
}

```

```

HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]      addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output

typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

```

```

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment

typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; ///< Output window position and size
    transformation_t transform; ///< Used transformation
    rect_t input_roi_r;
    int32_t pan; ///< Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; ///< Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; ///< Target zoom dimensionless coefficient (must not be bigger
    than zero)
    double strength; ///< Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double strengthY; ///< Dimensionless non-negative parameter defining the strength
    of transformation along X axis
}

```

```

    double angle;           ///< Angle of main projection axis rotation around itself in
degrees

    // universal transformation

    double elevation;      ///< Angle in degrees which specify the main projection axis

    double azimuth;        ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)

    int32_t keep_ratio;    ///< Keep the same stretching strength in both horizontal and
vertical directions

    double FOV_h;          ///< Size of output field of view in vertical dimension in
degrees

    double FOV_w;          ///< Size of output field of view in horizontal dimension in
degrees

    double cylindricity_y;  ///< Level of cylindricity for target projection shape in
vertical direction

    double cylindricity_x;  ///< Level of cylindricity for target projection shape in
horizontal direction

    char custom_file[128];   ///< File name of the file containing custom transformation
description

    custom_transformation_t custom;  ///< Parsed custom transformation structure

    double trapezoid_left_angle;    ///< Left Acute angle in degrees between trapezoid base and leg

    double trapezoid_right_angle;   ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

Release the output buffer of the corresponding channel of the IPU module.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to 4.2.11hb_vio_get_data

4.5 PYM API

4.5.1 hb_vio_pym_process

[Function Declaration]

```
int hb_vio_pym_process(src_img_info_t *src_img_info)
```

[Parameter Description]

- [IN]src_img_info_t *src_img_info: The raw image information that needs to do the pym process (usually provided by info_type: HB_VIO_SRC_INFO).

[Return Value]

- Success: Returns HB_OK 0.
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
}

```

```

HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
HB_VIO_IPU_US_DATA, //ipu us channel data type
HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
HB_VIO_PYM_DATA,// Pyramid output data
HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
HB_VIO_GDC_DATA,//gdc output data type
HB_VIO_IARWB_DATA,//iar display data type
HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
HB_VIO_PYM_LAYER_DATA,//pym all layer data
HB_VIO_MD_DATA,//motion detect data
HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
HB_VIO_PYM_COMMON_DATA,//pym base layer data
HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

```

```

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

```

```

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

```

```

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,

```

```

HB_VIO_IPU_DS3_IMG_INFO,
HB_VIO_IPU_DS4_IMG_INFO,
HB_VIO_PYM_IMG_INFO, //pym info
HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]          size[num]          addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)]  size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]      addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
}

```

```

} pym_buffer_t;

/**
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double angle; // Angle of main projection axis rotation around itself in
degrees
    // universal transformation
    double elevation; // Angle in degrees which specify the main projection axis
    double azimuth; // Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio; // Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h; // Size of output field of view in vertical dimension in
degrees
    double FOV_w; // Size of output field of view in horizontal dimension in
degrees
}

```

```

        double cylindricity_y;           /////< Level of cylindricity for target projection shape in
vertical direction
        double cylindricity_x;           /////< Level of cylindricity for target projection shape in
horizontal direction
        char custom_file[128];          /////< File name of the file containing custom transformation
description
        custom_transformation_t custom;  /////< Parsed custom transformation structure
        double trapezoid_left_angle;    /////< Left Acute angle in degrees between trapezoid base and leg
        double trapezoid_right_angle;   /////< Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

Set the pym module and enable the pym process.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2, X3/J3

Note: Should be used together with [hb_vio_get_info\(\)](#). Not recommended being used on X3/J3.
Instead, we recommend [hb_vio_run_pym\(\)](#).

[Sample Code]

```

#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include "hb_cam_interface.h"
#include "hb_vio_interface.h"

int main(int argc, char *argv[])
{
    int ret = 0;
    int fd, size;
    src_img_info_t src;
    img_info_t pym;
    const char *vio_cfg_file = " /etc/vio/pattern_dual_sif_isp_ipu_offline_1952x1097.json";
    const char *fb_yuv_file = "feedback.nv12";

    // vio init.
    ret = hb_vio_init(vio_cfg_file);
    if(ret < 0){

```

```

        printf("vio init error %d\n", ret);
        return -1;
    }

// vio start.
ret = hb_vio_start();
if(ret < 0){
    printf("vio start error %d\n", ret);
    hb_vio_deinit();
    return -1;
}

printf("get ======%d %p %d\n",
       HB_VIO_FEEDBACK_SRC_INFO, &src, sizeof(src));
ret = hb_vio_get_info(HB_VIO_FEEDBACK_SRC_INFO, &src);
if(ret < 0){
    printf("vio get feed src info error %d\n", ret);
    goto out;
}
printf("done\n");
fflush(stdout);

printf("vio feed src width=%d height=%d y_vaddr=%p c_vaddr=%p\n",
       src.src_img.step, src.src_img.height,
       src.src_img.y_vaddr, src.src_img.c_vaddr);

fd = open(fb_yuv_file, O_RDONLY);
if (fd < 0) {
    printf("file %s open error %d\n", fb_yuv_file, fd);
    goto free_out;
}
size = src.src_img.step * src.src_img.height;
read(fd, src.src_img.y_vaddr, size);
read(fd, src.src_img.c_vaddr, size / 2);
//y,uv addr not phy continuous addr
close(fd);
hb_vio_set_info(HB_VIO_FEEDBACK_FLUSH, &src);
ret = hb_vio_pym_process(&src);
if (fd < 0) {
    printf("pym process error %d\n");
    goto free_out;
}
ret = hb_vio_get_info(HB_VIO_PYM_INFO, &pym);
if(ret < 0){
    printf("vio get info error %d\n", ret);
}

```

```

        goto free_out;
    } else {
        printf("vio feed pym slot=%d id=%d ts=%lld\n",
               pym.slot_id, pym.frame_id, pym.timestamp);
    }

free_out:
    hb_vio_src_free(&src);//free src & pyms
out:
    // vio stop deinit.
    hb_vio_stop();
    hb_vio_deinit();
    return ret;
}

```

4.5.2 hb_vio_mult_pym_process

[Function Declaration]

```
int hb_vio_mult_pym_process(mult_src_info_t *mult_src_info)
```

[Parameter Description]

- [IN]mult_src_info_t *mult_src_info: The raw image information that needs to do the pym process.

[Return Value]

- Success: Returns HB_OK 0.
- Failure: Returns negative error code (refer to *X3/J3 main* parameters

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
   * y,uv          2 plane
   * raw           1 plane
   * raw, raw      2 plane(dol2)
   * raw, raw, raw 3 plane(dol3)
   */
typedef struct image_info_s {

```

```

    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {

```

```

HB_VIO_IPU_DSO_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

HB_VIO_IPU_DS1_CALLBACK,
HB_VIO_IPU_DS2_CALLBACK,
HB_VIO_IPU_DS3_CALLBACK,
HB_VIO_IPU_DS4_CALLBACK,
HB_VIO_IPU_US_CALLBACK,
HB_VIO_PYM_CALLBACK, // 6
HB_VIO_DIS_CALLBACK, // Not supported yet
HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO, //isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO, //not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
*/

```

```

* ipu buf : fd[0(y),1(c)]           size[0(y),1(c)]           addr[0(y),1(c)]
* pym buf : fd[0(all channel)]      size[0(all output)]      addr[0(y),1(c)] * 24
* */

// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;

/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
}

```

```

    int32_t pan;           ///< Target shift in horizontal direction from centre of the
output image in pixels

    int32_t tilt;          ///< Target shift in vertical direction from centre of the output
image in pixels

    double zoom;           ///< Target zoom dimensionless coefficient (must not be bigger
than zero)

    double strength;       ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis

    double strengthY;      ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis

    double angle;          ///< Angle of main projection axis rotation around itself in
degrees

    // universal transformation

    double elevation;      ///< Angle in degrees which specify the main projection axis

    double azimuth;         ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)

    int32_t keep_ratio;     ///< Keep the same stretching strength in both horizontal and
vertical directions

    double FOV_h;           ///< Size of output field of view in vertical dimension in
degrees

    double FOV_w;           ///< Size of output field of view in horizontal dimension in
degrees

    double cylindricity_y;   ///< Level of cylindricity for target projection shape in
vertical direction

    double cylindricity_x;   ///< Level of cylindricity for target projection shape in
horizontal direction

    char custom_file[128];    ///< File name of the file containing custom transformation
description

    custom_transformation_t custom; // < Parsed custom transformation structure

    double trapezoid_left_angle;  ///< Left Acute angle in degrees between trapezoid base and leg

    double trapezoid_right_angle; ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;
}

```

- VIO Return Code).

[Function Description]

Set the pym module and enable the pym process.

[Compatibility]

System version: 1.1 and above.

Hardware: X2/J2.

4.5.3 hb_vio_run_pym

[Function Declaration]

```
int hb_vio_run_pym(uint32_t pipeline_id, hb_vio_buffer_t * src_img_info)
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Software data channel needs to be set. Needs to be the corresponding enabled channel.
- [IN]hb_vio_buffer_t * src_img_info: The raw image information that needs to do the pym process. For the structure, refer to X3/J3 main parameters.

[Return Value]

- Success: Returns HB_OK 0.
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
}
```

```

VIO_DATA_TYPE_E data_type; //Data type of images
buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0, //ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA, //ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA, //ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA, //ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA, //ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA, // Pyramid fillback source data type
    HB_VIO_PYM_DATA, // Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA, // Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA, // Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA, // Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA, //gdc output data type
    HB_VIO_IARWB_DATA, //iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA, //gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA, //pym all layer data
    HB_VIO_MD_DATA, //motion detect data
    HB_VIO_ISP_RAW_DATA, //isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA, //pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DS0_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

```

```

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment

```

```

typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
    than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double angle; // Angle of main projection axis rotation around itself in
    degrees
    // universal transformation
}

```

```

    double elevation;           ///< Angle in degrees which specify the main projection axis
    double azimuth;            ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio;        ///< Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h;              ///< Size of output field of view in vertical dimension in
degrees
    double FOV_w;              ///< Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y;     ///< Level of cylindricity for target projection shape in
vertical direction
    double cylindricity_x;     ///< Level of cylindricity for target projection shape in
horizontal direction
    char custom_file[128];      ///< File name of the file containing custom transformation
description
    custom_transformation_t custom; //;< Parsed custom transformation structure
    double trapezoid_left_angle;  ///< Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle; ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;
}

```

- VIO Return Code).

[Function Description]

Set the pym module and enable the pym process.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```

#include "hb_vio_interface.h"
#include "hb_cam_interface.h "
static int gcdc_cfg_bin_update(uint32_t pipe_id, char *layout_file)
{
    int ret = 0;
    uint64_t config_size = 0;
    ret = gcdc_cfg_bin_gen(layout_file, "./gdc.bin", &config_size);

    if (ret == 0) {
        printf("pipe(%u) cfg_buf(%p) size(%lu)\n",
               pipe_id, g_config_buffer, config_size);
    }
}

```

```

ret = hb_vio_set_gdc_cfg(pipe_id, g_config_buffer, config_size);
//also can use gdc 1: hb_vio_set_gdc_cfg_opt(pipe_id, 1, g_config_buffer, config_size);

if (g_config_buffer) {
    hb_vio_free_gdc_cfg(g_config_buffer);
printf("free config_buf %p size(%lu)\n", g_config_buffer, config_size);
    g_config_buffer = NULL;
}

if (ret < 0) {
    printf("gdc cfg bin set failed.\n");
    return -1;
}

} else {
    printf("gdc cfg bin gen failed.\n");
    return -1;
}

printf("pipe(%u)gdc bin update done.File %s size %lu\n",
       pipe_id, layout_file, config_size);

return ret;
}

static int gdc_feedback_func(work_info_t * info)
{
    int ret = 0;
    char file_name[100];
    hb_vio_buffer_t buf = {0};
    hb_vio_buffer_t dst_buf = {0};
printf("pipe(%u)gdc fb process start!\n", info->pipe_id);
    ret = gdc_cfg_bin_update(info->pipe_id, gdc_cfg);
    if (ret < 0) {
        return -1;
    }

    ret = hb_vio_get_data(info->pipe_id, HB_VIO_GDC_FEEDBACK_SRC_DATA, &buf);
    if (ret < 0) {
        printf("hb_vio_get_data GDC_FEEDBACK error!\n");
        return -1;
    }

printf("pipe(%u)gdc fb buf(%d)img_addr.w = %d, h = %d, stride = %d\n",
       info->pipe_id,
       buf.img_info.buf_index,
       buf.img_addr.width,
       buf.img_addr.height,
       buf.img_addr.stride_size);

    if (gdc_fb_pic == NULL) {

```

```

    hb_vio_free_gdcbuf(info->pipe_id, &buf);
    printf("pipe(%u)gdc_fb_pic null !\n", info->pipe_id);
    return -1;
}

int img_in_fd = open(gdc_fb_pic, O_RDWR | O_CREAT);
int size_0 = buf.img_addr.stride_size * buf.img_addr.height;
int size_1 = size_0 / 2;
printf("pipe(%u)gdc_fb size_0 = %d, size_1 = %d\n",
       info->pipe_id,
       size_0,
       size_1);
if(img_in_fd < 0) {
    printf("pipe(%u)open image failed !\n", info->pipe_id);
    return -1;
}
read(img_in_fd, buf.img_addr.addr[0], size_0);
usleep(10 * 1000);
read(img_in_fd, buf.img_addr.addr[1], size_1);
usleep(10 * 1000);
close(img_in_fd);

int i = info->data_type % 2;
printf("hb_vio_gdc_process begin use %d, rotate %d\n", i, need_gdc_fb);
pthread_mutex_lock(&g_gdc_lock[i]);
ret = hb_vio_run_gdc(info->pipe_id, &buf, &dst_buf, need_gdc_fb);
//also can use gcd 1: hb_vio_run_gdc_opt(info->pipe_id, 1, &buf, &dst_buf, need_gdc_fb);
pthread_mutex_unlock(&g_gdc_lock[i]);

if (ret < 0) {
    printf("hb_vio_gdc_process failed use %d\n", i);
    hb_vio_free_gdcbuf(info->pipe_id, &buf);
    return -1;
}

hb_vio_free_gdcbuf(info->pipe_id, &buf);
hb_vio_free_gdcbuf(info->pipe_id, &dst_buf);
printf("gdc_fb process done use %d\n", i);
return 0;
}

int main(int argc, char *argv[])
{
    int ret = 0;
}

```

```
int need_pym_process = 1;
int need_gdc_process = 0;
pthread_mutex_t g_pym_lock;
pthread_mutex_t g_gdc_lock;
hb_vio_buffer_t buf = {0};
pym_buffer_t pym_buf = {0};
hb_vio_buffer_t dst_buf = {0};
char *cam_cfg_file = "/etc/cam/hb_xj3dev.json";
ret = hb_vio_init("/etc/vio/imx327_raw_12bit_1952x1097_online_Pipeline.json");
if (ret < 0) {
    printf("vio init fail\n");
    return -1;
}
ret = hb_cam_init(0, cam_cfg_file);
if (ret < 0) {
    printf("cam init fail\n");
    hb_vio_deinit();
    return -1;
}
ret = gdc_cfg_bin_update(0, gdc_cfg);
if (ret < 0) {
    printf("gdc_cfg_bin_update failed\n");
    hb_vio_deinit();
    hb_cam_deinit(cam_index);
    return -1;
}
ret = hb_vio_start_pipeline(0);
if (ret < 0) {
    printf("vio start fail, do cam&vio_deinit.\n");
    hb_cam_deinit(0);hb_vio_deinit();
    return -1;
}
ret = hb_cam_start(0);
if (ret < 0) {
    printf("cam start fail, do cam&vio_deinit.\n");
    goto err;
}
ret = hb_vio_get_data(0, HB_VIO_IPU_US_DATA, &buf);
if (ret < 0) {
    printf("pipe 0 get us data failed!\n");
    goto err;
}
pthread_mutex_init(&g_pym_lock, NULL);
pthread_mutex_init(&g_gdc_lock, NULL);
```

```
if (need_pym_process) {
    pthread_mutex_lock(&g_pym_lock);
    ret = hb_vio_run_pym(0, &buf);
    if (ret < 0) {
        hb_vio_free_ipubuf (0, &buf);
        printf("hb_vio_run_pym failed.\n");
        pthread_mutex_unlock(&g_pym_lock);
        goto err;
    }
    ret = hb_vio_get_data(0, HB_VIO_PYM_DATA, &pym_buf);
    pthread_mutex_unlock(&g_pym_lock);
    if (ret < 0) {
        printf("hb_vio_get_data HB_VIO_PYM_DATA error\n");
        goto err;
    }

    hb_vio_free_pympbuf(0, HB_VIO_PYM_DATA, &pym_buf);
}

if (need_gdc_process) {
    printf("hb_vio_run_gdc begin\n");
    pthread_mutex_lock(&g_gdc_lock);
    ret = hb_vio_run_gdc(0, &buf, &dst_buf, 90);
    pthread_mutex_unlock(&g_gdc_lock);
    if (ret < 0) {
        printf("hb_vio_run_gdc failed\n");
        hb_vio_free_ipubuf (0, &buf);
        goto err;
    }
    hb_vio_free_gdcbuf(0, &dst_buf);
}

ret = hb_vio_free_ipubuf(0, &buf);

hb_cam_stop(0);
hb_vio_stop_pipeline(0);
hb_cam_deinit(0);
hb_vio_deinit();
return 0;

err:
    hb_cam_stop(0);
    hb_cam_deinit(0);
    hb_vio_deinit();
return -1;
}
```

4.5.4 hb_vio_free_pymbuf

[Function Declaration]

```
int hb_vio_free_pymbuf(uint32_t pipeline_id, VIO_DATA_TYPE_E data_type,void *img_info)
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Software channel identification of the corresponding data.
- [IN]VIO_DATA_TYPE_E data_type: pym fillback source buf or pym buf. For the structure, refer to X3/J3 main parameters.
- [IN]hb_vio_buffer_t *img_info: The pym fillback source buf or pym buffer information that needs to be released.

[Return Value]

- Success: Returns HB_OK 0.
- Failure: Returns negative error code (refer to refer to [X3/J3 main](#) parameters

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
}
```

```

    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type;//Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DS0_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
    get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK,// 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
}

```

```

HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]      addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output

typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

```

```

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment

typedef struct pym_buffer_s {

    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {

    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
    than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
    of transformation along X axis
}

```

```

    double angle;           ///< Angle of main projection axis rotation around itself in
degrees

    // universal transformation

    double elevation;      ///< Angle in degrees which specify the main projection axis
    double azimuth;        ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)

    int32_t keep_ratio;    ///< Keep the same stretching strength in both horizontal and
vertical directions

    double FOV_h;          ///< Size of output field of view in vertical dimension in
degrees

    double FOV_w;          ///< Size of output field of view in horizontal dimension in
degrees

    double cylindricity_y;  ///< Level of cylindricity for target projection shape in
vertical direction

    double cylindricity_x;  ///< Level of cylindricity for target projection shape in
horizontal direction

    char custom_file[128];   ///< File name of the file containing custom transformation
description

    custom_transformation_t custom;  ///< Parsed custom transformation structure
    double trapezoid_left_angle;    ///< Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle;   ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

Release the output buffer of the pym module or the source buf of the pym fillback.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to hb_vio_run_pym

4.6 GDC API

4.6.1 hb_vio_run_gdc

[Function Declaration]

```
int hb_vio_run_gdc(uint32_t pipeline_id, hb_vio_buffer_t *src_img_info, hb_vio_buffer_t
*dst_img_info, int rotate);
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Software channel identification of the corresponding data.
- [IN]hb_vio_buffer_t *src_img_info: Buffer information that needs to do the GDC process. For the structure, refer to X3/J3 main parameters.
- [OUT]hb_vio_buffer_t *dst_img_info: Image information processed by GDC.
- [IN] int rotate: Indicates the rotation angle to be processed, supporting 0°, 90°, 180°, and 270°.

[Return Value]

- Success: Returns HB_OK 0.
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

```

```

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0, //ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA, //ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA, //ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA, //ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA, //ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA, // Pyramid fillback source data type
    HB_VIO_PYM_DATA, // Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA, // Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA, // Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA, // Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA, //gdc output data type
    HB_VIO_IARWB_DATA, //iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA, //gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA, //pym all layer data
    HB_VIO_MD_DATA, //motion detect data
    HB_VIO_ISP_RAW_DATA, //isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA, //pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DS0_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
}

```

```

HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
HB_VIO_IPU_US_IMG_INFO, //us channel info
HB_VIO_IPU_DS0_IMG_INFO,
HB_VIO_IPU_DS1_IMG_INFO,
HB_VIO_IPU_DS2_IMG_INFO,
HB_VIO_IPU_DS3_IMG_INFO,
HB_VIO_IPU_DS4_IMG_INFO,
HB_VIO_PYM_IMG_INFO, //pym info
HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]          size[num]          addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)]  size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]      addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
}

```

```

//pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
address_info_t us[6];// Six us channels of pyramid output data address information.

char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size

} pym_buffer_t;

/***
* gcd Common parameters
*/
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
* Window definition and transformation parameters
* For parameters meaning read GDC guide
*/
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double angle; // Angle of main projection axis rotation around itself in
degrees
    // universal transformation
    double elevation; // Angle in degrees which specify the main projection axis
    double azimuth; // Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
}

```

```

    int32_t keep_ratio;           ///< Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h;               ///< Size of output field of view in vertical dimension in
degrees
    double FOV_w;               ///< Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y;      ///< Level of cylindricity for target projection shape in
vertical direction
    double cylindricity_x;      ///< Level of cylindricity for target projection shape in
horizontal direction
    char custom_file[128];       ///< File name of the file containing custom transformation
description
    custom_transformation_t custom; //parsed custom transformation structure
    double trapezoid_left_angle;  ///< Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle; ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;
}

```

- VIO Return Code).

[Function Description]

Enable the GDC process(default gcd 0).

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to hb_vio_run_pym.

4.6.2 hb_vio_run_gdc_opt

[Function Declaration]

```

int hb_vio_run_gdc_opt(uint32_t pipeline_id,
                       uint32_t gdc_id,
                       hb_vio_buffer_t *src_img_info,
                       hb_vio_buffer_t *dst_img_info,
                       int rotate) [Parameter Description]

```

- [IN]uint32_t pipeline_id: Software channel identification of the corresponding data.
- [IN]uint32_t gdc_id: gdc module identification (0 or 1)
- [IN]hb_vio_buffer_t *src_img_info: Buffer information that needs to do the GDC process. For the structure, refer to X3/J3 main parameters.
- [OUT]hb_vio_buffer_t *dst_img_info: Image information processed by GDC.
- [IN] int rotate: Indicates the rotation angle to be processed, supporting 0°, 90°, 180°, and 270°.

[Return Value]

- Success: Returns HB_OK 0.
- Failure: Returns negative error code (refer to [X3/J3 main](#) parameters)

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
}

```

```

HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
HB_VIO_PYM_DATA,// Pyramid output data
HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
HB_VIO_GDC_DATA,//gdc output data type
HB_VIO_IARWB_DATA,//iar display data type
HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
HB_VIO_PYM_LAYER_DATA,//pym all layer data
HB_VIO_MD_DATA,//motion detect data
HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
HB_VIO_PYM_COMMON_DATA,//pym base layer data
HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX

} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DS0_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
}

```

```

HB_VIO_PYM_IMG_INFO, //pym info
HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

```

```

/**
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
    than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
    of transformation along Y axis
    double angle; // Angle of main projection axis rotation around itself in
    degrees
    // universal transformation
    double elevation; // Angle in degrees which specify the main projection axis
    double azimuth; // Angle in degrees which specify the main projection axis,
    counted clockwise from North direction (positive to East)
    int32_t keep_ratio; // Keep the same stretching strength in both horizontal and
    vertical directions
    double FOV_h; // Size of output field of view in vertical dimension in
    degrees
    double FOV_w; // Size of output field of view in horizontal dimension in
    degrees
    double cylindricity_y; // Level of cylindricity for target projection shape in
    vertical direction
}

```

```

    double cylindricity_x;           ///// Level of cylindricity for target projection shape in horizontal direction
    char custom_file[128];          ///// File name of the file containing custom transformation description
    custom_transformation_t custom;  ///// Parsed custom transformation structure
    double trapezoid_left_angle;    ///// Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle;   ///// Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

Enable the GDC 0 or 1 to process.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to hb_vio_run_pym.

4.6.3 hb_vio_free_gdcbuf

[Function Declaration]

```
int hb_vio_free_gdcbuf (uint32_t pipeline_id,hb_vio_buffer_t *dst_img_info);
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Software channel identification of the corresponding data.
- [IN]hb_vio_buffer_t * dst_img_info: GDC buffer information that needs to be released. For the GDC buffer information, refer to X3/J3 main parameters.

[Return Value]

- Success: Returns HB_OK 0.
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width; // Image data width
    uint16_t height; // Image data width
    uint16_t stride_size; // Image data memory stride (Row width of actual memory storage)
}

```

```

char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.

} address_info_t;

/* info :
 * y,uv           2 plane
 * raw            1 plane
 * raw, raw       2 plane(dol2)
 * raw, raw, raw  3 plane(dol3)
 */

typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
}

```

```

HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
HB_VIO_PYM_COMMON_DATA,//pym base layer data
HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX

} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK,// 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK

} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
    HB_VIO_INFO_MAX

} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
}

```

```

    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */

// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
}

```

```

} param_t;

/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r;           ///< Output window position and size
    transformation_t transform; //;< Used transformation
    rect_t input_roi_r;
    int32_t pan;           ///< Target shift in horizontal direction from centre of the
output image in pixels
    int32_t tilt;           ///< Target shift in vertical direction from centre of the output
image in pixels
    double zoom;            ///< Target zoom dimensionless coefficient (must not be bigger
than zero)
    double strength;        ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double strengthY;       ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double angle;           ///< Angle of main projection axis rotation around itself in
degrees
    // universal transformation
    double elevation;       ///< Angle in degrees which specify the main projection axis
    double azimuth;          ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio;     ///< Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h;           ///< Size of output field of view in vertical dimension in
degrees
    double FOV_w;           ///< Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y;   ///< Level of cylindricity for target projection shape in
vertical direction
    double cylindricity_x;   ///< Level of cylindricity for target projection shape in
horizontal direction
    char custom_file[128];    ///< File name of the file containing custom transformation
description
    custom_transformation_t custom; //;< Parsed custom transformation structure
    double trapezoid_left_angle; //;< Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle; //;< Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

Release the output buffer of the GDC module.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to hb_vio_run_pym.

4.6.4 hb_vio_gen_gdc_cfg

[Function Declaration]

```
int    hb_vio_gen_gdc_cfg(param_t *gdc_parm, window_t *wnds, uint32_t wnd_num, void **cfg_buf,
                           uint64_t *cfg_size);
```

[Parameter Description]

- [IN] param_t *gdc_parm: Corresponding parameters of GDC, including resolution, format.
- [IN] window_t *wnds: gdc internal region parameters. For more information, refer to X3/J3 main parameters.
- [IN] uint32_t wnd_num: the num of window expected to be processed.
- [OUT] void **cfg_buf: the buffer contain gdc cfg bin.
- [OUT] uint64_t *cfg_size: size of gdc cfg bin.

[Return Value]

- Success: Returns HB_OK 0.
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
```

```

    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {

```

```

HB_VIO_IPU_DSO_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

HB_VIO_IPU_DS1_CALLBACK,
HB_VIO_IPU_DS2_CALLBACK,
HB_VIO_IPU_DS3_CALLBACK,
HB_VIO_IPU_DS4_CALLBACK,
HB_VIO_IPU_US_CALLBACK,
HB_VIO_PYM_CALLBACK, // 6
HB_VIO_DIS_CALLBACK, // Not supported yet
HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO, //isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO, //not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]        size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
*/

```

```

* ipu buf : fd[0(y),1(c)]           size[0(y),1(c)]           addr[0(y),1(c)]
* pym buf : fd[0(all channel)]      size[0(all output)]       addr[0(y),1(c)] * 24
* */

// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/***
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;

/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
}

```

```

    int32_t pan;           ///< Target shift in horizontal direction from centre of the
output image in pixels

    int32_t tilt;          ///< Target shift in vertical direction from centre of the output
image in pixels

    double zoom;           ///< Target zoom dimensionless coefficient (must not be bigger
than zero)

    double strength;       ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis

    double strengthY;      ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis

    double angle;          ///< Angle of main projection axis rotation around itself in
degrees

    // universal transformation

    double elevation;      ///< Angle in degrees which specify the main projection axis

    double azimuth;         ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)

    int32_t keep_ratio;     ///< Keep the same stretching strength in both horizontal and
vertical directions

    double FOV_h;           ///< Size of output field of view in vertical dimension in
degrees

    double FOV_w;           ///< Size of output field of view in horizontal dimension in
degrees

    double cylindricity_y;   ///< Level of cylindricity for target projection shape in
vertical direction

    double cylindricity_x;   ///< Level of cylindricity for target projection shape in
horizontal direction

    char custom_file[128];    ///< File name of the file containing custom transformation
description

    custom_transformation_t custom; // < Parsed custom transformation structure

    double trapezoid_left_angle;  ///< Left Acute angle in degrees between trapezoid base and leg

    double trapezoid_right_angle; // < Right Acute angle in degrees between trapezoid base and leg
} window_t;
}

```

- VIO Return Code).

[Function Description]

Generate the CFG bin file required by the GDC module.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to hb_vio_run_pym.

4.6.5 hb_vio_free_gdc_cfg

[Function Declaration]

```
void hb_vio_free_gdc_cfg(uint32_t* cfg_buf);
```

[Parameter Description]

- [IN] uint32_t* cfg_buf: the buffer contain gcd cfg bin.

[Return Value]

- Success: Returns HB_OK 0.
- Failure: Returns negative error code (refer to *X3/J3 main* parameters)

System version: 2.0 and above.

```
typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
}
```

```

        buffer_state_e state; // buf status, which is the user status at the user layer.

} image_info_t;

typedef enum VIO_DATA_TYPE_S {

    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {

    HB_VIO_IPU_DS0_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
    get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

```

```

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS,// Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]   size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {

```

```

image_info_t pym_img_info;// Pyramid data information
address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
//pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
address_info_t us[6];// Six us channels of pyramid output data address information.
char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; // Output window position and size
    transformation_t transform; // Used transformation
    rect_t input_roi_r;
    int32_t pan; // Target shift in horizontal direction from centre of the
    output image in pixels
    int32_t tilt; // Target shift in vertical direction from centre of the output
    image in pixels
    double zoom; // Target zoom dimensionless coefficient (must not be bigger
    than zero)
    double strength; // Dimensionless non-negative parameter defining the strength
    of transformation along X axis
    double strengthY; // Dimensionless non-negative parameter defining the strength
    of transformation along Y axis
    double angle; // Angle of main projection axis rotation around itself in
    degrees
    // universal transformation
    double elevation; // Angle in degrees which specify the main projection axis
}

```

```

    double azimuth;           ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio;     ///< Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h;           ///< Size of output field of view in vertical dimension in
degrees
    double FOV_w;           ///< Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y;   ///< Level of cylindricity for target projection shape in
vertical direction
    double cylindricity_x;   ///< Level of cylindricity for target projection shape in
horizontal direction
    char custom_file[128];   ///< File name of the file containing custom transformation
description
    custom_transformation_t custom; //parsed custom transformation structure
    double trapezoid_left_angle;  ///< Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle; ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;
}

```

- VIO Return Code).

[Function Description]

Release the buffer of the GDC bin.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to hb_vio_run_pym.

4.6.6 hb_vio_set_gdc_cfg

[Function Declaration]

```
int hb_vio_set_gdc_cfg(uint32_t pipeline_id, uint32_t* cfg_buf, uint64_t cfg_size);
```

[Parameter Description]

- [IN]uint32_t pipeline_id: Software channel identification of the corresponding data.
- [IN]uint32_t* cfg_buf: the buffer contain gcd cfg bin.
- [IN]uint64_t cfg_size: size of gcd cfg bin.

[Return Value]

- Success: Returns HB_OK 0.

- Failure: Returns negative error code (refer to [X3/J3 main](#) parameters)

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
 * y,uv          2 plane
 * raw           1 plane
 * raw, raw      2 plane(dol2)
 * raw, raw, raw 3 plane(dol3)
 */
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
}

```

```

HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
HB_VIO_GDC_DATA,//gdc output data type
HB_VIO_IARWB_DATA,//iar display data type
HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
HB_VIO_PYM_LAYER_DATA,//pym all layer data
HB_VIO_MD_DATA,//motion detect data
HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
HB_VIO_PYM_COMMON_DATA,//pym base layer data
HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX
} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0,//Callback data type, which is mutually and exclusively used with the interface that is used to
get data.
    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK,// Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO,//isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO,//not used in XJ3
}

```

```

HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]  size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * gdc Common parameters
 */

```

```

typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;
/***
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r; ///< Output window position and size
    transformation_t transform; ///< Used transformation
    rect_t input_roi_r;
    int32_t pan; ///< Target shift in horizontal direction from centre of the
output image in pixels
    int32_t tilt; ///< Target shift in vertical direction from centre of the output
image in pixels
    double zoom; ///< Target zoom dimensionless coefficient (must not be bigger
than zero)
    double strength; ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double strengthY; ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double angle; ///< Angle of main projection axis rotation around itself in
degrees
    // universal transformation
    double elevation; ///< Angle in degrees which specify the main projection axis
    double azimuth; ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio; ///< Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h; ///< Size of output field of view in vertical dimension in
degrees
    double FOV_w; ///< Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y; ///< Level of cylindricity for target projection shape in
vertical direction
    double cylindricity_x; ///< Level of cylindricity for target projection shape in
horizontal direction

```

```

    char custom_file[128];           ///<-- File name of the file containing custom transformation
description

    custom_transformation_t custom;  ///<-- Parsed custom transformation structure
    double trapezoid_left_angle;    ///<-- Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle;   ///<-- Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

set the GDC bin to gcd hw(default gcd 0).

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to hb_vio_run_pym.

4.6.7 hb_vio_set_gdc_cfg_opt

[Function Declaration]

```
int    hb_vio_set_gdc_cfg_opt(uint32_t pipeline_id, uint32_t gcd_id,
                           uint32_t* cfg_buf, uint64_t cfg_size) [Parameter Description]
```

- [IN]uint32_t pipeline_id: Software channel identification of the corresponding data.
- [IN]uint32_t gcd_id: gcd module identification(0 or 1)
- [IN]uint32_t* cfg_buf: the buffer contain gcd cfg bin.
- [IN]uint64_t cfg_size: size of gcd cfg bin.

[Return Value]

- Success: Returns HB_OK 0.
- Failure: Returns negative error code (refer to *X3/J3 main* parameters

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;
/* info :

```

```

* y,uv           2 plane
* raw            1 plane
* raw, raw       2 plane(dol2)
* raw, raw, raw  3 plane(dol3)
*/
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type; //Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
}

```

```

HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
HB_VIO_EMBED_DATA, //embed data, not used in XJ3
HB_VIO_DATA_TYPE_MAX

} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK

} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO, //isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO, //not used in XJ3
    HB_VIO_INFO_MAX

} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

```

```

/*
 * buf type    fd[num]           size[num]           addr[num]
 *
 * sif buf : fd[0(raw)]         size[0(raw)]        addr[0(raw)]
 * sif dol2: fd[0(raw),1(raw)]   size[0(raw),1(raw)]  addr[0(raw),1(raw)]
 * sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
 * ipu buf : fd[0(y),1(c)]      size[0(y),1(c)]     addr[0(y),1(c)]
 * pym buf : fd[0(all channel)] size[0(all output)]  addr[0(y),1(c)] * 24
 */

// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in; // input frame resolution
    resolution_t out; // output frame resolution
    int32_t x_offset; // center offset for input x coordinate
    int32_t y_offset; // center offset for input y coordinate
    int32_t diameter; // diameter of equivalent 180 field of view in pixels
    double fov; // field of view
} param_t;

/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide

```

```

/*
typedef struct {
    rect_t out_r;           ///< Output window position and size
    transformation_t transform;  ///< Used transformation
    rect_t input_roi_r;
    int32_t pan;           ///< Target shift in horizontal direction from centre of the
output image in pixels
    int32_t tilt;           ///< Target shift in vertical direction from centre of the output
image in pixels
    double zoom;           ///< Target zoom dimensionless coefficient (must not be bigger
than zero)
    double strength;        ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double strengthY;       ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
    double angle;           ///< Angle of main projection axis rotation around itself in
degrees
    // universal transformation
    double elevation;        ///< Angle in degrees which specify the main projection axis
    double azimuth;          ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
    int32_t keep_ratio;      ///< Keep the same stretching strength in both horizontal and
vertical directions
    double FOV_h;           ///< Size of output field of view in vertical dimension in
degrees
    double FOV_w;           ///< Size of output field of view in horizontal dimension in
degrees
    double cylindricity_y;   ///< Level of cylindricity for target projection shape in
vertical direction
    double cylindricity_x;   ///< Level of cylindricity for target projection shape in
horizontal direction
    char custom_file[128];    ///< File name of the file containing custom transformation
description
    custom_transformation_t custom; // Parsed custom transformation structure
    double trapezoid_left_angle;  ///< Left Acute angle in degrees between trapezoid base and leg
    double trapezoid_right_angle; // Right Acute angle in degrees between trapezoid base and leg
} window_t;

```

- VIO Return Code).

[Function Description]

set the GDC bin to gcd hw(0 or 1).

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code] Refer to hb_vio_run_pym.

4.7 VIO Parameter Descriptions

4.7.1 X2/J2 main parameters

System version: 1.1 and above.

```
typedef struct vio_channel_s{
    int ipu;
    int sif;
    int mipi_host;
    int mipi_dev;
    int isp;
}vio_channel;

Output parameters. Input parameters during VIO initialization and enabling. 0 indicates that channel does not need this module; 1 indicates that channel uses this module and indicates the compatibility and adaptation of multiple image processing modules later.

typedef struct addr_info_s{
    uint16_t width;
    uint16_t height;
    uint16_t step;
    uint8_t *y_paddr;
    uint8_t *c_paddr;
    uint8_t *y_vaddr;
    uint8_t *c_vaddr;
}addr_info_t;

/* for single output */

typedef struct src_img_info_s{
    int cam_id;
    int slot_id;          // getted slot buff
    int img_format;       // now only support yuv420sp
    int frame_id;         // for x2 may be 0 - 0xFFFF or 0x7FFF(used when multi-channel input)
    int64_t timestamp;    // BT from Hisi; mipi & dvp from kernel time
    addr_info_t src_img; // for x2 src img
}src_img_info_t;

Output parameters, raw image information structure;

typedef struct img_info_s{
    int slot_id;          // getted slot buff
    int frame_id;         // for x2 may be 0 - 0xFFFF or 0x7FFF(used when multi-channel input)
```

```

int64_t timestamp;           // BT from Hisi; mipi & dvp from kernel time
int img_format;              // now only support yuv420sp
int ds_pym_layer;            // get down scale layers
int us_pym_layer;            // get up scale layers
addr_info_t src_img;         // for x2 src img = crop img
addr_info_t down_scale[DOWN_SCALE_MAX]; // Currently, down_scale supports 0-23 layers at most, depending on the
configuration file ds_layer_en.

addr_info_t up_scale[UP_SCALE_MAX]; // Currently, up_scale supports 6 layers at most, depending on the configuration file
us_layer_en.

}img_info_t;

Output parameters, pym result structure;

/* for multiple output */

typedef struct mult_src_info_s{
    int src_num;
    src_img_info_t src_img_info[SRC_MAX];
}mult_src_info_t;

Output parameters. Output raw image parameters during multi-channel input.

int src_num; Explain which channel raw images to output, and the value is less than or equal to SRC_MAX;
rc_img_info_t src_img_info[SRC_MAX]; Corresponding to the output results of raw images of each channel;
typedef struct mult_img_info_s{
    int img_num;
    img_info_t img_info[PYM_MAX];
}mult_img_info_t;

Output parameters. Output pym result parameters during multi-channel input.

int img_num; Explain which channel results to output, and the value is less than PYM_MAX;
rc_img_info_t src_img_info[SRC_MAX]; Corresponding to the output results of raw images of each channel;

```

4.7.2 X3/J3 main parameters

System version: 2.0 and above.

```

typedef struct address_info_s {
    uint16_t width;// Image data width
    uint16_t height;// Image data width
    uint16_t stride_size;// Image data memory stride (Row width of actual memory storage)
    char *addr[HB_VIO_BUFFER_MAX_PLANES]; // Virtual address, stored according to the number of YUV plane.
    uint64_t paddr[HB_VIO_BUFFER_MAX_PLANES]; // Physical address, stored according to the number of YUV plane.
} address_info_t;

/* info :
   * y,uv          2 plane
   * raw           1 plane
   * raw, raw      2 plane(dol2)

```

```

* raw, raw, raw    3 plane(dol3)

*/
typedef struct image_info_s {
    uint16_t sensor_id;//sensor id
    uint16_t pipeline_id; // Corresponding data channel number
    uint32_t frame_id; // Data frame number
    uint64_t time_stamp; //HW time stamp
    struct timeval tv; //system time of hal get buf
    int buf_index; // Index of the obtained data buf
    int img_format;
    int fd[HB_VIO_BUFFER_MAX_PLANES]; //ion buf fd
    uint32_t size[HB_VIO_BUFFER_MAX_PLANES]; // Corresponding plane size
    uint32_t planeCount;// The number of planes in image
    uint32_t dynamic_flag;
    uint32_t water_mark_line;//pre-interrupt, not support in XJ3
    VIO_DATA_TYPE_E data_type;//Data type of images
    buffer_state_e state; // buf status, which is the user status at the user layer.
} image_info_t;

typedef enum VIO_DATA_TYPE_S {
    HB_VIO_IPU_DS0_DATA = 0,//ipu ds0 channel data type
    HB_VIO_IPU_DS1_DATA,//ipu ds1 channel data type
    HB_VIO_IPU_DS2_DATA,//ipu ds2 channel data type
    HB_VIO_IPU_DS3_DATA,//ipu ds3 channel data type
    HB_VIO_IPU_DS4_DATA,//ipu ds4 channel data type
    HB_VIO_IPU_US_DATA, //ipu us channel data type
    HB_VIO_PYM_FEEDBACK_SRC_DATA,// Pyramid fillback source data type
    HB_VIO_PYM_DATA,// Pyramid output data
    HB_VIO_SIF_FEEDBACK_SRC_DATA, //sif raw fillback source data type
    HB_VIO_SIF_RAW_DATA,// Internally defined sif raw data
    HB_VIO_SIF_YUV_DATA,// Internally defined sif yuv data
    HB_VIO_ISP_YUV_DATA,// Internally defined isp yuv data, which is got after fillback
    HB_VIO_GDC_DATA,//gdc output data type
    HB_VIO_IARWB_DATA,//iar display data type
    HB_VIO_GDC_FEEDBACK_SRC_DATA,//gdc feedback src buf
    HB_VIO_PYM_LAYER_DATA,//pym all layer data
    HB_VIO_MD_DATA,//motion detect data
    HB_VIO_ISP_RAW_DATA,//isp raw not used in XJ3
    HB_VIO_PYM_COMMON_DATA,//pym base layer data
    HB_VIO_PYM_DATA_V2, //next version pym data, not used in XJ3
    HB_VIO_CIM_RAW_DATA, //cim raw data, not used in XJ3
    HB_VIO_CIM_YUV_DATA, //cim yuv, not used in XJ3
    HB_VIO_EMBED_DATA, //embed data, not used in XJ3
    HB_VIO_DATA_TYPE_MAX
}

```

```

} VIO_DATA_TYPE_E;

typedef enum VIO_CALLBACK_TYPE {
    HB_VIO_IPU_DSO_CALLBACK = 0, //Callback data type, which is mutually and exclusively used with the interface that is used to
get data.

    HB_VIO_IPU_DS1_CALLBACK,
    HB_VIO_IPU_DS2_CALLBACK,
    HB_VIO_IPU_DS3_CALLBACK,
    HB_VIO_IPU_DS4_CALLBACK,
    HB_VIO_IPU_US_CALLBACK,
    HB_VIO_PYM_CALLBACK, // 6
    HB_VIO_DIS_CALLBACK, // Not supported yet
    HB_VIO_PYM_V2_CALLBACK, //not used in XJ3
    HB_VIO_MAX_CALLBACK
} VIO_CALLBACK_TYPE_E;

typedef enum VIO_INFO_TYPE_S {
    HB_VIO_CALLBACK_ENABLE = 0,
    HB_VIO_CALLBACK_DISABLE,
    HB_VIO_IPU_SIZE_INFO, //reserve for ipu size setting in dis
    HB_VIO_IPU_US_IMG_INFO, //us channel info
    HB_VIO_IPU_DS0_IMG_INFO,
    HB_VIO_IPU_DS1_IMG_INFO,
    HB_VIO_IPU_DS2_IMG_INFO,
    HB_VIO_IPU_DS3_IMG_INFO,
    HB_VIO_IPU_DS4_IMG_INFO,
    HB_VIO_PYM_IMG_INFO, //pym info
    HB_VIO_ISP_IMG_INFO, //isp info,not used in XJ3
    HB_VIO_PYM_V2_IMG_INFO, //not used in XJ3
    HB_VIO_INFO_MAX
} VIO_INFO_TYPE_E;

typedef enum buffer_state {
    BUFFER_AVAILABLE, // VIO available status (VIO internal status)
    BUFFER_PROCESS, // Hardware processing status (VIO internal status)
    BUFFER_DONE, // Data completion status (VIO internal status)
    BUFFER_REPROCESS, // Data reprocessing (VIO internal status)
    BUFFER_USER, // Obtained by users
    BUFFER_INVALID
} buffer_state_e;

/*
 * buf type   fd[num]          size[num]          addr[num]
 *

```

```

* sif buf : fd[0(raw)]           size[0(raw)]           addr[0(raw)]
* sif dol2: fd[0(raw),1(raw)]    size[0(raw),1(raw)]    addr[0(raw),1(raw)]
* sif dol3: fd[0(raw),1(raw),2(raw)] size[0(raw),1(raw),2(raw)]  addr[0(raw),1(raw),2(raw)]
* ipu buf : fd[0(y),1(c)]       size[0(y),1(c)]       addr[0(y),1(c)]
* pym buf : fd[0(all channel)]   size[0(all output)]   addr[0(y),1(c)] * 24
*/
// normal capture buffer type, one image data output
typedef struct hb_vio_buffer_s {
    image_info_t img_info;
    address_info_t img_addr;
} hb_vio_buffer_t;

// special buffer type, multi image data output,
// but all channel output alloc one ion buffer avoid buf fragment
typedef struct pym_buffer_s {
    image_info_t pym_img_info;// Pyramid data information
    address_info_t pym[6];// Pyramid base layer data address, corresponding to s0 s4 s8 s16 s20 s24
    address_info_t pym_roi[6][3];// Data address information of ROI layer associated with pyramid base layer
    //pym_roi[0]: [0] corresponds to s1 pym_roi[0] in s0, [1] corresponds to s2 in s0, pym_roi[0][2] corresponds to s3 in s0.
    address_info_t us[6];// Six us channels of pyramid output data address information.
    char *addr_whole[HB_VIO_BUFFER_MAX_PLANES];// whole pym vaddr
    uint64_t paddr_whole[HB_VIO_BUFFER_MAX_PLANES]; // whole pym paddr
    uint32_t layer_size[30][HB_VIO_BUFFER_MAX_PLANES];//pym layers size
} pym_buffer_t;

/**
 * gdc Common parameters
 */
typedef struct {
    frame_format_t format; // frame format
    resolution_t in;      // input frame resolution
    resolution_t out;     // output frame resolution
    int32_t x_offset;    // center offset for input x coordinate
    int32_t y_offset;    // center offset for input y coordinate
    int32_t diameter;   // diameter of equivalent 180 field of view in pixels
    double fov;          // field of view
} param_t;
/**
 * Window definition and transformation parameters
 * For parameters meaning read GDC guide
 */
typedef struct {
    rect_t out_r;          ///< Output window position and size

```

```

transformation_t transform;      ///< Used transformation
rect_t input_roi_r;
int32_t pan;                  ///< Target shift in horizontal direction from centre of the
output image in pixels
int32_t tilt;                 ///< Target shift in vertical direction from centre of the output
image in pixels
double zoom;                  ///< Target zoom dimensionless coefficient (must not be bigger
than zero)
double strength;              ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
double strengthY;             ///< Dimensionless non-negative parameter defining the strength
of transformation along X axis
double angle;                 ///< Angle of main projection axis rotation around itself in
degrees
// universal transformation
double elevation;             ///< Angle in degrees which specify the main projection axis
double azimuth;               ///< Angle in degrees which specify the main projection axis,
counted clockwise from North direction (positive to East)
int32_t keep_ratio;           ///< Keep the same stretching strength in both horizontal and
vertical directions
double FOV_h;                 ///< Size of output field of view in vertical dimension in
degrees
double FOV_w;                 ///< Size of output field of view in horizontal dimension in
degrees
double cylindricity_y;         ///< Level of cylindricity for target projection shape in
vertical direction
double cylindricity_x;         ///< Level of cylindricity for target projection shape in
horizontal direction
char custom_file[128];          ///< File name of the file containing custom transformation
description
custom_transformation_t custom; // < Parsed custom transformation structure
double trapezoid_left_angle;   ///< Left Acute angle in degrees between trapezoid base and leg
double trapezoid_right_angle;  ///< Right Acute angle in degrees between trapezoid base and leg
} window_t;
}

```

4.8 VIO Return Code

4.8.1 X2/J2 return code

System version: 1.1 and above.

```
//success
```

```

#define HB_OK 0

// libvio: sif|isp|ipu|pym

//sif fail

#define HB_VIO_SIF_INIT_FAIL      101
#define HB_VIO_SIF_DEINIT_FAIL    102
#define HB_VIO_SIF_START_FAIL     103
#define HB_VIO_SIF_STOP_FAIL      104

//ipu fail

#define HB_VIO_IPU_INIT_FAIL      201
#define HB_VIO_IPU_DEINIT_FAIL    202
#define HB_VIO_IPU_START_FAIL     203
#define HB_VIO_IPU_STOP_FAIL      204

#define HB_VIO_INIT_FAIL          301
#define HB_VIO_START_FAIL         302
#define HB_VIO_PAESER_FAIL        401
#define HB_VIO_OPEN_INT_FAIL       402
#define HB_VIO_EPOLL_CREATE_FAIL  403
#define HB_VIO_EPOLL_CTL_FAIL     404
#define HB_VIO_EPOLL_WAIT_FAIL    405
#define HB_VIO_EPOLL_WAIT_TIMEOUT 406
#define HB_VIO_PYM_IS_BUSY        407
#define HB_VIO_QUENE_IS_FULL      408
#define HB_VIO_QUENE_INIT_FAIL    409
#define HB_VIO_NOT_SUPPORT        900

```

Note:

The return value of the execution function is expressed in the form of negative result code.

For example: return -HB_VIN_CAM_POWER_ON_FAIL;

4.8.2 X3/J3 return code

System version: 2.0 and above.

```

//success

#define HB_VIO_NO_ERR              0

enum HB_VIO_ERROR_CODE {
    HB_VIO_PAESER_FAIL = 1,
    HB_VIO_OPEN_CFG_FAIL,
    HB_VIO_INIT_FAIL,
    HB_VIO_START_FAIL,
    HB_VIO_STOP_FAIL,
    HB_VIO_BAD_VALUE,
    HB_VIO_NULL_POINT,
    HB_VIO_TIME_OUT,

```

```
HB_VIO_INVALID_CONFIG,  
HB_VIO_INVALID_OPERATION,  
HB_VIO_BUF_MGR_FAIL,  
HB_VIO_ENABLE_MD_FAIL,  
HB_VIO_DISABLE_MD_FAIL  
};  
  
enum HB_VIO_ISP_ERROR_CODE {  
    HB_VIO_ISP_OPEN_DEV_FAIL = 300,  
    HB_VIO_ISP_INIT_FAIL,  
    HB_VIO_ISP_DEINIT_FAIL,  
    HB_VIO_ISP_UPDATE_FAIL,  
    HB_VIO_ISP_STOP_FAIL,  
    HB_VIO_ISP_START_FAIL,  
    HB_VIO_ISP_PARSER_FAIL,  
    HB_VIO_ISP_EPOLL_CREATE_FAIL,  
    HB_VIO_ISP_EPOLL_CTL_FAIL,  
    HB_VIO_ISP_EPOLL_WAIT_FAIL,  
    HB_VIO_ISP_STOP_WORKING,  
    HB_VIO_ISP_BAD_VALUE,  
    HB_VIO_ISP_TIME_OUT,  
    HB_VIO_ISP_INVALID_OPERATION,  
    HB_VIO_ISP_INVALID_CONFIG,  
    HB_VIO_ISP_NULL_POINT,  
    HB_VIO_ISP_BUF_MGR_FAIL  
};  
  
enum HB_VIO_PYM_ERROR_CODE {  
    HB_VIO_PYM_OPEN_DEV_FAIL = 400,  
    HB_VIO_PYM_INIT_FAIL,  
    HB_VIO_PYM_DEINIT_FAIL,  
    HB_VIO_PYM_UPDATE_FAIL,  
    HB_VIO_PYM_STOP_FAIL,  
    HB_VIO_PYM_START_FAIL,  
    HB_VIO_PYM_PARSER_FAIL,  
    HB_VIO_PYM_IS_BUSY,  
    HB_VIO_PYM_EPOLL_CREATE_FAIL,  
    HB_VIO_PYM_EPOLL_CTL_FAIL,  
    HB_VIO_PYM_EPOLL_WAIT_FAIL,  
    HB_VIO_PYM_STOP_WORKING,  
    HB_VIO_PYM_BAD_VALUE,  
    HB_VIO_PYM_TIME_OUT,  
    HB_VIO_PYM_INVALID_OPERATION,  
    HB_VIO_PYM_INVALID_CONFIG,
```

```
HB_VIO_PYM_NULL_POINT,  
HB_VIO_PYM_BUF_MGR_FAIL,  
};  
  
enum HB_VIO_GDC_ERROR_CODE {  
    HB_VIO_GDC_OPEN_DEV_FAIL = 500,  
    HB_VIO_GDC_INIT_FAIL,  
    HB_VIO_GDC_deinit_FAIL,  
    HB_VIO_GDC_UPDATE_FAIL,  
    HB_VIO_GDC_STOP_FAIL,  
    HB_VIO_GDC_START_FAIL,  
    HB_VIO_GDC_PARSER_FAIL,  
    HB_VIO_GDC_EPOLL_CREATE_FAIL,  
    HB_VIO_GDC_EPOLL_CTL_FAIL,  
    HB_VIO_GDC_EPOLL_WAIT_FAIL,  
    HB_VIO_GDC_STOP_WORKING,  
    HB_VIO_GDC_BUF_MGR_FAIL,  
    HB_VIO_GDC_BAD_VALUE,  
    HB_VIO_GDC_NULL_POINT,  
    HB_VIO_GDC_PROCESS_FAIL,  
    HB_VIO_GDC_INVALID_OPERATION,  
    HB_VIO_GDC_GET_STATUS_FAIL  
};  
  
enum HB_VIO_LDC_ERROR_CODE {  
    HB_VIO_LDC_OPEN_DEV_FAIL = 600,  
    HB_VIO_LDC_INIT_FAIL,  
    HB_VIO_LDC_deinit_FAIL,  
    HB_VIO_LDC_UPDATE_FAIL,  
    HB_VIO_LDC_STOP_FAIL,  
    HB_VIO_LDC_START_FAIL,  
    HB_VIO_LDC_PARSER_FAIL,  
    HB_VIO_LDC_STOP_WORKING,  
    HB_VIO_LDC_BAD_VALUE,  
    HB_VIO_LDC_TIME_OUT,  
    HB_VIO_LDC_INVALID_OPERATION,  
    HB_VIO_LDC_INVALID_CONFIG  
};  
  
enum HB_VIO_SIF_ERROR_CODE {  
    HB_VIO_SIF_OPEN_DEV_FAIL = 700,  
    HB_VIO_SIF_INIT_FAIL,  
    HB_VIO_SIF_UPDATE_FAIL,  
    HB_VIO_SIF_STOP_FAIL,
```

```
HB_VIO_SIF_START_FAIL,  
HB_VIO_SIF_PARSER_FAIL,  
HB_VIO_SIF_EPOLL_CREATE_FAIL,  
HB_VIO_SIF_EPOLL_CTL_FAIL,  
HB_VIO_SIF_EPOLL_WAIT_FAIL,  
HB_VIO_SIF_STOP_WORKING,//709  
HB_VIO_SIF_DEINIT_FAIL,  
HB_VIO_SIF_BAD_VALUE,  
HB_VIO_SIF_TIME_OUT,  
HB_VIO_SIF_INVALID_OPERATION,  
HB_VIO_SIF_INVALID_CONFIG,  
HB_VIO_SIF_NULL_POINT,  
HB_VIO_SIF_BUF_MGR_FAIL  
};  
  
enum HB_VIO_IPU_ERROR_CODE {  
    HB_VIO_IPU_INIT_FAIL = 800,  
    HB_VIO_IPU_DEINIT_FAIL,  
    HB_VIO_IPU_START_FAIL,  
    HB_VIO_IPU_STOP_FAIL,  
    HB_VIO_IPU_PARSER_FAIL,  
    HB_VIO_IPU_EPOLL_CREATE_FAIL,//805  
    HB_VIO_IPU_EPOLL_CTL_FAIL,  
    HB_VIO_IPU_EPOLL_WAIT_FAIL,  
    HB_VIO_IPU_STOP_WORKING,  
    HB_VIO_IPU_OPEN_DEV_FAIL,  
    HB_VIO_IPU_UPDATE_FAIL,  
    HB_VIO_IPU_BAD_VALUE,  
    HB_VIO_IPU_TIME_OUT,  
    HB_VIO_IPU_INVALID_OPERATION,  
    HB_VIO_IPU_INVALID_CONFIG,  
    HB_VIO_IPU_NULL_POINT,  
    HB_VIO_IPU_CHN_NOT_ENABLE,  
    HB_VIO_IPU_OPERATE OSD FAIL,  
    HB_VIO_IPU_BUF_MGR_FAIL,  
    HB_VIO_IPU_OSD_OPEN_FILE_FAIL  
};  
  
enum HB_VIO_DWE_ERROR_CODE {  
    HB_VIO_DWE_NULL_POINT = 900,  
    HB_VIO_DWE_PARSER_FAIL,  
    HB_VIO_DWE_BAD_VALUE,  
    HB_VIO_DWE_OPEN_DEV_FAIL,  
    HB_VIO_DWE_INIT_FAIL,  
    HB_VIO_DWE_DEINIT_FAIL,
```

```

    HB_VIO_DWE_START_FAIL,
    HB_VIO_DWE_STOP_FAIL
};
```

4.9 VIO Configuration File

4.9.1 X2/J2 configuration file descriptions

The following example is only compatible with system version 1.2. For the VIO configuration file for system version 1.1, refer to: vio [错误!未找到引用源。](#).

```

hb_vio.json
{
  "sif": {
    "sif_config": {
      "format": 8,           /*sif input format: 0:raw 8:yuyv 9:yvyu 10:uyvy 11:vyuy*/
      "pix_len": 0,          /*Single pixel component length: 0:8 1:10 2:12 3:14 4:16 5:20*/
      "bus_type": 2,         /*sif input interface: 0:DVP 1:BT1120 2:mipi 3:dual channel input*/
      "vsync_inv": 1,        /*Whether vsync needs to inverse depends on the hardware, only dvp*/
      "hsync_inv": 0,        /*Whether hsync needs to inverse depends on the hardware, only dvp*/
      "pclk_in_inv": 0,      /*Whether pclk_in needs to inverse depends on the hardware*/
      "pclk_out_inv": 0,     /*Whether pclk_out needs to inverse depends on the hardware*/
      "drop_frame": 0,       /*No need to focus on it temporarily*/
      "raw_16bit_mode": 0,
      "raw_20bit_mode": 0,
      "yuv_10bit_mode": 1,
      "dualrx_mode": 0,      /*Dual channel uses 0: side by side 1: vc*/
      "mipi2ap_sel": 0,      /*bypass output selection, default: 0*/
      "bt_detect_mode": 0,   /*bt detection mode, 0: Full 16bit sync 1: Low 8bit sync 2: High 8bit sync*/
      "bt_in_exchange": 0,   /*bt input size terminal, whether to exchange*/
      "bt_out_exchange": 0,  /*bt output size terminal, whether to exchange*/
      "width": 1280,         /*sif input size Width*/
      "height": 720,         /*sif input size Height*/
      "bypass_en": 0,        /*Whether to enable bypass*/
    },
    "mot_det_config": {
      /*The detection function is not turned on temporarily*/
      "mot_det_enable": 0,
      "mot_det_top": 0,
      "mot_det_left": 0,
      "mot_det_width": 1280,
      "mot_det_height": 720,
```

```

    "mot_det_step": 16,
    "mot_det_thresh": 128,
    "mot_det_diff_thresh": 8,
    "mot_det_wgt_decay": 196,
    "mot_det_dec_prec": 1
    /*The detection function is not turned on temporarily*/
},
"frame_id_config": {
    "frame_id_enable": 0,           /*Enable sif to print frame id*/
    "frame_id_init_value": 0,       /*Set initial value of frame id*/
    "frame_id_fix_mode": 0         /*Set frame id mode, 0: increasing value 1: fixed value*/
}
},
"ipu": {
    "timeout": 2000,               /*Timeout settings that IPU gets information blocked. Unit: ms*/
    "ipu_config": {
        "w": 1280,                 /*Width of IPU input*/
        "h": 720,                  /*Height of IPU input*/
        "mipi2lines": 0,           /*Whether IPU enables dual channel mipi*/
        "from_isp": 0,             /*Whether IPU data passes through internal ISP*/
        "uv_format": 1,             /*uv format 0:vuvu 1:uvuv*/
        "crop_to_ddr": 1,           /*Whether to enable crop to save to DDR*/
        "scale_to_ddr": 1,          /*Whether to enable crop to save to DDR*/
        "crop_en": 1,               /*Whether to enable crop*/
        "testpattern_en": 0,         /*Whether to enable test mode*/
        "hw_mode": 1                /*Whether to be pym hardware mode*/
    },
    "ipu_frameid_config": {
        "enable": 1,                /*Enable IPU to print frame id*/
        "bus":0,                    /*Interface type 0:dvp/mipi 1:BT*/
        "scale_en": 0,               /*Whether to enable scale to print frame id*/
        "crop_en": 0                 /*Whether to enable crop to print frame id*/
    },
    "crop_config": {
        "st_w": 0,                  /*Start width of crop*/
        "st_h": 0,                  /*Start height of crop*/
        "ed_w": 1280,               /*End width of crop*/
        "ed_h": 720,                /*End height of crop*/
        /*crop 后是 size (ed_w - st_w) * (ed_h - st_h) */
    },
    "scale_config": {
        "st_w": 1280,               /*Input size of scale: Width*/
        "st_h": 720,                /*Input size of scale: Height*/
        "ed_w": 1280,               /*Target size of scale: Width*/
    }
}
}

```

```

"ed_h": 720,           /*Target size of scale: Height*/
"step_x": 0,           /*Biliner coefficient (input(pre_scale_x output)-1)/(output - 1) * 4096*/
"step_y": 0,           /*Biliner coefficient (input(pre_scale_y output)-1)/(output - 1) * 4096*/
"pre_scale_x": 0,      /*Support 0 – 7 layers, at most zoom out 1/128*/
"pre_scale_y": 0,      /*Support 0 - 7 layers, at most zoom out 1/128*/
"bypass_x": 1,         /*Whether to do the bilinear process*/
"bypass_y": 1          /*Whether to do the bilinear process*/

/*Currently, the input is not greater than 2048*2048, so st_w, ed_w, st_h, and ed_h are temporarily configured according to the
actual size. Other parameters are configured according to the default value. */

},
/*
* Scale: Two levels of scaling
* When the width and height of the input size are greater than or equal to twice the target size, pre_scale first.
* Scaled to (1/2)n of the input size, and then perform bilinear processing.
* When the width and height of the input size are less than twice the target size, just perform bilinear processing.
* mid_h = src_h * 1/2pre_scale_x
* if(bypass_x)
*   dst_h = mid_h
* else
*   step_x = 4096 * (mid_h -1)/(dst_h -1)
*/
"pymid_ctrl_config": {
    "src_w": 1280,        /*Input width of pym, consistent with ed_w*/
    "src_h": 720,          /*Input height of pym, consistent with ed_h*/
    "sw_start": 0,          /*Not available temporarily*/
    "image_from_ipu": 1,    /*Whether the data is used for IPU and whether the normal sensor is configured as 1*/
    "pymid_en": 1,          /*Whether to enable pym*/
    "ds_layer_en": 8,       /*The layer number of down scale, 4~23*/
    "ds_uv_bypass": 0,      /*Whether to bypass uv value*/
    "us_layer_en": 0,       /*The layer number of up scale, 0~6*/
    "us_uv_bypass": 0       /*Whether to bypass uv value*/
},
"pymid_ds_config": {
    "roi_l_1": 0,           /*Coordinates corresponding to the left vertex of the base layer in ROI region.*/
    "roi_t_1": 0,           /*Coordinates corresponding to the left vertex of the base layer in ROI region.*/
    "src_w_1": 200,          /*ROI region size, should not exceed the base layer size*/
    "src_h_1": 200,          /*ROI region size, should not exceed the base layer size*/
    "factor_1": 32,
},
/*
* Down Scale: There are two concept needed to know – base layer and ROI layer;
* Base layer: 4k (k = 0--5) layers, and the multiple of scaling per layer is (1/2)k;
* ROI layer: Obtained after scaling selected ROI region by the base layer;
* layer 1, 2, 3 based on layer 0

```

```

* layer 5, 6, 7 based on layer 4
* layer 9, 10, 11 based on layer 8
* layer 13, 14, 15 based on layer 12
* layer 17, 18, 19 based on layer 16
* layer 21, 22, 23 based on layer 20
* roi scale factor 64/(64 + factor). If the factor is 0, it means to disable the zoom of this layer. Factor range: 0~63
* target_width_y = ((src_width - 1) * 64 / (64 + factor) +1 >> 1) << 1
* target_height_y = ((src_height - 1) * 64 / (64 + factor) +1 >> 1) << 1
* target_width_uv = ((src_width/2 - 1) * 64 / (64 + factor) +1 >> 1) << 1
* target_height_uv = ((src_height/2 - 1) * 64 / (64 + factor) +1 >> 1) << 1
* Chip limits:
*   Minimum size of pym zoom output 48 * 32
*   Minimum size of pym zoom output 2048 * 2048
*   Tag width and height are taken below the current value to get an even number. For example, if the size 401 * 401, 400 * 400 will be
got.

*/
"pymid_us_config": {
    "roi_l_0": 0,           /*Coordinates of the upper left corner of the ROI region, shall be even number*/
    "roi_t_0": 0,           /*Coordinates of the upper left corner of the ROI region, shall be even number*/
    "src_w_0": 200,         /*ROI region width, which cannot exceed the raw image width*/
    "src_h_0": 100,         /*ROI region height, which cannot exceed the raw image height*/
    "factor_0": 50,          /*Amplification factor parameter, (64/factor)*/
    "roi_l_1": 0,
    "roi_t_1": 0,
    "src_w_1": 0,
    "src_h_1": 0,
    "factor_1": 40,
    "roi_l_2": 0,
    "roi_t_2": 0,
    "src_w_2": 0,
    "src_h_2": 0,
    "factor_2": 32,
    "roi_l_3": 0,
    "roi_t_3": 0,
    "src_w_3": 0,
    "src_h_3": 0,
    "factor_3": 25,
    "roi_l_4": 0,
    "roi_t_4": 0,
    "src_w_4": 0,
    "src_h_4": 0,
    "factor_4": 20,
    "roi_l_5": 0,
    "roi_t_5": 0,
}

```

```

    "src_w_5": 0,
    "src_h_5": 0,
    "factor_5": 16
},
/*
 * up_scaler image layers are 6 in total. Select 6 regions randomly from the raw image layer for magnification. At present, the factor
magnification size is a fixed value, i.e
 * factor_0:50 1.28x magnification
 * factor_1:40 1.6x magnification
 * factor_2:32 2x magnification
 * factor_3:25 2.56x magnification
 * factor_4:20 3.2x magnification
 * factor_5:16 4x magnification
 * Calculation:
 * Y component:
 *     target_width_y = ((src_w - 1) * (64 / factor) + 1 >> 1) << 1
 *     target_hight_y = ((src_h - 1) * (64 / factor) + 1 >> 1) << 1
 * UV 分量:
 *     target_width_uv = ((src_w/2 - 1) * (64 / factor) + 1 >> 1) << 1
 *     target_hight_uv = ((src_h/2 - 1) * (64 / factor) + 1 >> 1) << 1
*/
"isp": {
    "enable": 0      /*Whether to enable ISP depends on the hardware. No need to focus on it for the time being*/
},
"iar":{
    "enable":0       /*Whether to use iar*/
}
}

```

4.9.1.1 X2/J2 Ver.1.1

```

hb_vio.json
{
"sif": {
    "sif_config": {
        "format": 8,
        sif input format: 0: raw 8: yuyv 9: yyvu 10: uyyv 11: vyuv
        "pix_len": 0,
        Length of single pixel component: 0: 8 1: 10 2: 12 3: 14 4: 16 5: 20
        "bus_type": 2,
        sif input interface: 0: DVP 1: BT1120 2: mipi 3: dual channel input
        "vsync_inv": 1,
        Whether vsync to reverse depends on the hardware, only dvp
        "hsync_inv": 0,
}
}
}

```

```

    Whether hsync to reverse depends on the hardware, only dvp
"pclk_in_inv": 0,
    Whether pclk_in to reverse depends on the hardware
"pclk_out_inv": 0,
    Whether pclk_out to reverse depends on the hardware
"drop_frame": 0,
    No need to focus on it for the time being
"raw_16bit_mode": 0,
"raw_20bit_mode": 0,
"yuv_10bit_mode": 1,
"dualrx_mode": 0,
    dual channel uses 0: side by side 1: vc
"mipi2ap_sel": 0,
    bypass output selection, default: 0
"bt_detect_mode": 0,
    bt detection mode, 0: full 16bit sync 1: low 8bit sync 2: high 8bit sync
"bt_in_exchange": 0,
    bt input size terminal, whether to exchange
"bt_out_exchange": 0,
    bt output size terminal, whether to exchange
"width": 1280,
    sif input size, width
"height": 720,
    sif input size, height
"bypass_en": 0
    Whether to enable bypass
},
"mot_det_config": {
    The detection function is not turned on temporarily.
    "mot_det_enable": 0,
    "mot_det_top": 0,
    "mot_det_left": 0,
    "mot_det_width": 1280,
    "mot_det_height": 720,
    "mot_det_step": 16,
    "mot_det_thresh": 128,
    "mot_det_diff_thresh": 8,
    "mot_det_wgt_decay": 196,
    "mot_det_dec_prec": 1
    The detection function is not turned on temporarily.
},
"frame_id_config": {
    "frame_id_enable": 0,
    Enable sif to print frame id
}

```

```

"frame_id_init_value": 0,
    Set initial value of frame id
"frame_id_fix_mode": 0
    Set frame id mode, 0: increasing value 1: fixed value
}
},
"ipu": {
    "timeout": 2000,
        Timeout settings that IPU gets information blocked. Unit: ms.
    "ipu_config": {
        "w": 1280,
            Width of IPU input
        "h": 720,
            Height of IPU input
        "mipi2lines": 0,
            Whether IPU enables dual channel mipi
        "from_isp": 0,
            Whether IPU data passes through internal ISP
        "uv_format": 1,
            uv format 0: vuuv 1: uvuv
        "crop_to_ddr": 1,
            Whether to enable crop to save to DDR
        "scale_to_ddr": 1,
            Whether to enable crop to save to DDR
        "crop_en": 1,
            Whether to enable crop
        "testpattern_en": 0,
            Whether to enable testpattern test mode
        "hw_mode": 1
            Whether to be pym hardware mode
    },
    "ipu_frameid_config": {
        "enable": 1,
            Enable IPU to print frame id
        "bus": 0,
            Interface type 0: dvp/mipi 1: BT
        "scale_en": 0,
            Whether to enable scale to print frame id
        "crop_en": 0
            Whether to enable crop to print frame id
    },
    "crop_config": {
        "st_w": 0,
            Start width of crop
    }
}
}

```

```

"st_h": 0,
    Start height of crop
"ed_w": 1280,
    End width of crop
"ed_h": 720
    End height of crop
    After cropping, size (ed_w - st_w) * (ed_h - st_h)
},
"scale_config": {
    "st_w": 1280,
        Input size of scale: Width
    "st_h": 720,
        Input size of scale: Height
    "ed_w": 1280,
        Target size of scale: Width
    "ed_h": 720,
        Target size of scale: Height
    "step_x": 0,
        Biliner factor (input(pre_scale_x output)-1)/(output - 1) * 4096
    "step_y": 0,
        Biliner factor (input(pre_scale_y output)-1)/(output - 1) * 4096
    "pre_scale_x": 0,
        Support 0 – 7 layers, at most zoom out 1/128
    "pre_scale_y": 0,
        Support 0 - 7 layers, at most zoom out 1/128
    "bypass_x": 1,
        Whether to do the bilinear process
    "bypass_y": 1
        Whether to do the bilinear process
    Currently, the input is not greater than 2048*2048, so st_w, ed_w, st_h, and ed_h are temporarily configured according to
the actual size. Other parameters are configured according to the default value.
},
/*Scale: Two levels of scaling
When the width and height of the input size are greater than or equal to twice the target size, pre_scale first. Scaled to (1/2)n of the
input size, and then perform bilinear processing.

When the width and height of the input size are less than twice the target size, just perform bilinear processing.

mid_h = src_h * 1/2pre_scale_x
if(bypass_x)
    dst_h = mid_h
else
    step_x = 4096 * (mid_h -1)/(dst_h -1)
*/
"pymid_ctrl_config": {
    "src_w": 1280,

```

```

Width of pym input, consistent with ed_w
"src_h": 720,
Height of pym input, consistent with ed_h
"sw_start": 0,
Not available temporarily
"image_from_ipu": 1,
Whether the data is used for IPU and whether the normal sensor is configured as 1.
"pymid_en": 1,
Whether to enable pym.
"ds_layer_en": 8,
The layer number of down scale, 4 ~ 23
"ds_uv_bypass": 0,
Whether to bypass uv value
"us_layer_en": 0,
The layer number of up scale, 0 ~ 6
"us_uv_bypass": 0
Whether to bypass uv value
},
"pymid_ds_config": {
"roi_l_1": 0,
Coordinates corresponding to the left vertex of the base layer in ROI region.
"roi_t_1": 0,
Coordinates corresponding to the left vertex of the base layer in ROI region.
"src_w_1": 200,
ROI region size, should not exceed the base layer size.
"src_h_1": 200,
ROI region size, should not exceed the base layer size.
"factor_1": 32,
}
/* Down Scale: There are two concept needed to know – base layer and ROI layer:
Base layer: 4*k (k = 0--5) layer, and the multiple of scaling per layer is (1/2)^k;
ROI layer: Obtained after scaling selected ROI region by the base layer;
layer 1, 2, 3 based on layer 0
layer 5, 6, 7 based on layer 4
layer 9, 10, 11 based on layer 8
layer 13, 14, 15 based on layer 12
layer 17, 18, 19 based on layer 16
layer 21, 22, 23 based on layer 20
roi scale factor 64/(64 + factor). If the factor is 0, it means to disable the zoom of this layer. Factor range: 0~63
target_width_y = ((src_width - 1) * 64/(64 + factor) +1 >> 1) << 1
target_height_y = ((src_height - 1) * 64/(64 + factor) +1 >> 1) << 1
target_width_uv = ((src_width/2 - 1) * 64/(64 + factor) +1 >> 1) << 1
target_height_uv = ((src_height/2 - 1) * 64/(64 + factor) +1 >> 1) << 1
Chip limits:
```

Copyright © 2021 Horizon Robotics Co., Ltd. All Rights Reserved. No part of this document may be reproduced or transmitted without prior written permission.

Minimum size of pym zoom output 48 * 32

Minimum size of pym zoom output 2048 * 2048

Tag width and height are taken below the current value to get an even number. For example, if the size 401 * 401, 400 * 400 will be got.

```
* / "pymid_us_config": {
    "roi_l_0": 0,
        Coordinates of the upper left corner of the ROI region, shall be even number.
    "roi_t_0": 0,
        Coordinates of the upper left corner of the ROI region, shall be even number.
    "src_w_0": 200,
        ROI region width, which cannot exceed the raw image width.
    "src_h_0": 100,
        ROI region height, which cannot exceed the raw image height.
    "factor_0": 50,
        Amplification factor parameter (64/factor)
    "roi_l_1": 0,
    "roi_t_1": 0,
    "src_w_1": 0,
    "src_h_1": 0,
    "factor_1": 40,
    "roi_l_2": 0,
    "roi_t_2": 0,
    "src_w_2": 0,
    "src_h_2": 0,
    "factor_2": 32,
    "roi_l_3": 0,
    "roi_t_3": 0,
    "src_w_3": 0,
    "src_h_3": 0,
    "factor_3": 25,
    "roi_l_4": 0,
    "roi_t_4": 0,
    "src_w_4": 0,
    "src_h_4": 0,
    "factor_4": 20,
    "roi_l_5": 0,
    "roi_t_5": 0,
    "src_w_5": 0,
    "src_h_5": 0,
    "factor_5": 16
}
```

/*up_scaler image layers are 6 in total. Select 6 regions randomly from the raw image layer for magnification. At present, the factor magnification size is a fixed value, i.e.

factor_0: 50 1.28x magnification

```

factor_1: 40 1.6x magnification
factor_2: 32 2x magnification
factor_3: 25 2.56x magnification
factor_4: 20 3.2x magnification
factor_5: 16 4x magnification

Calculation:

Y component:
target_width_y = ((src_w - 1) * (64/factor) + 1 >> 1) << 1
target_hight_y = ((src_h - 1) * (64/factor) + 1 >> 1) << 1

UV 分量
target_width_uv = ((src_w/2 - 1) * (64/factor) + 1 >> 1) << 1
target_hight_uv = ((src_h/2 - 1) * (64/factor) + 1 >> 1) << 1

*/
"isp": {
    "enable": 0
        Whether to enable ISP depends on the hardware. No need to focus on it for the time being.
    }
    "iar": {
        "enable": 0
            Whether to use iar
    }
}
}

```

4.9.2 X3/J3 configuration file descriptions

System version: 2.0 and above.

```
{
    "clk": {
        "sif_mclk": 544000000, //SIF mclk, if set null will use default value
        "vpu_mclk": 544000000//vpu mclk, if set null will use default value
    },
    "pipeline0": {
        "sif": {
            "input": {
                "dvp": {
                    "enable": 0,
                    "hsync_inv": 0,
                    "vsync_inv": 1,
                    "width": 1280,
                    "height": 720,
                    "format": 0,
                    "pix_length": 2,
                    "enable_mux_out": 1,
                    "enable_frame_id": 1,

```

```

    "enable_pattern": 0,
    "set_init_frame_id": 100,
    "set_mux_out_index": 0
},
"mipi": {
    "enable": 1, /*Enable mipi input*/
    "ipi_channels": 1, /*The channel number of the used internal mipi
    "mipi_rx_index": 0, //Accessed mipi module number
    "vc_index": 0, //mipi vc num, eg, dol2 mode need two vc
        //and need set "vc_index": 0, "vc_index": 1"
    "width": 1952, /*Mipi access width matching sensor format*/
    "height": 1097, /*Mipi access height matching sensor format*/
    "format": 0, /* 0 raw, 8 yuv422(only support 8 or 10 bit)*/
    "pix_length": 2,
    /* PIX_LEN_8 = 0, PIX_LEN_10 = 1,PIX_LEN_12 = 2,PIX_LEN_14 = 3,PIX_LEN_16 = 4*/
    "enable_mux_out": 1, /* Enable sif internal MUX channel, which needs to be turned on*/
    "enable_pattern": 0,
    /*For use of testing, whether to open the testpattern test data inside sif. No need to enable this function when the
sensor has been accessed*/
    "enable_frame_id": 1,
    "enable_bypass": 0,
    "enable_line_shift": 0, /*Enable sensor line shift dol, which normally does not need to be configured*/
    "enable_id_decoder": 0, /*Enable sensor id decoder dol, which normally does not need to be configured */
    "set_init_frame_id": 1, /*Enable frame id count. Start from 1 by default*/
    "set_line_shift_count": 0, /*Corresponding settings in the sensor line shift mode, which does not normally need to be
configured*/
    "set_bypass_channels": 1,
},
"ddr_to_isp": {
    "ddr_in_enable": 0, /*Enable sending from DDR to ISP for processing, which does not need to be truned on in general
scenarios*/
    "ddr_in_width": 1952, /*Data width from ddr in to isp*/
    "ddr_in_height": 1097, /*Data height from ddr in to isp*/
    "ddr_in_format": 0, /*Currently fixed to raw format, no need to change*/
    "ddr_in_pix_length": 2, /*pix format of raw, same as above*/
    "ddr_in_buf_num": 4, /*The data buf number of ddr in fillback, which normally does not need to be configured*/
    "ddr_raw_feedback_debug": 0
    /*Enable raw fillback. It does not work with normal sensor data and raw dump at the same time.
hb_vio_raw_feedbackavailable*/
}
},
"output": {
    "ddr": {
        "ddr_output_enable": 0, /*sif ddr output enabling*/

```

```

    "stride": 2928, /*Size of the stride that is passed to ddr. Calcluate raw12 as 1952*1.5*/
    "ddr_output_buf_num": 8,/*The number of buf that is passed to memory. In general, 6 or above are
recommended*/
    "ddr_raw_dump_debug": 0
    /*Enable raw dump. hb_vio_raw_dump can be used for dump, which is not used together with raw fillback*/
},
"isp": {
    "isp_enable": 1,
    /*Enable ISP. Generally, raw format needs to be enabled except in YUV sensor scenario*/
    "enable_flyby": 1,
    /*After enabling, the SIF hardware is connected to the ISP online. In the multi-channel scenario, disable it if reusing
from ddr in to isp*/
    "dol_exp_num": 1,/*Exposure mode. 1: general mode, dol 2 or 3: sets the corresponding number*/
    "enable_dgain": 0,/*ISP internal debugging parameters, which can be ignored temporarily*/
    "vc_short_seq":0,/*In the dol mode, sequence marks that receive or send long and short frames. The hardware
needs to make sure the sequence*/
    "vc_medium_seq":0,
    "vc_long_seq":0,
    "set_dgain_short": 0, /*ISP internal debugging parameters, which can be ignored temporarily*/
    "set_dgain_medium": 0, /*ISP internal debugging parameters, which can be ignored temporarily*/
    "set_dgain_long": 0/*ISP internal debugging parameters, which can be ignored temporarily*/
},
"ipu": {
    "enable_flyby": 0/*After enabling, directly connect the SIF to the IPU. Only YUV sensor of single channel online.
Generally, it is disabled.*/
},
"md": {
    "enable": 0,/*Module is disabled temporarily*/
    "path_sel": 0,
    "roi_top": 0,
    "roi_left": 0,
    "roi_width": 0,
    "roi_height": 0,
    "grid_step": 0,
    "grid_tolerance": 0,
    "threshold": 0,
    "weight_decay": 0,
    "precision": 0
}
},
"isp": {
    "sensor_mode": 1,/*NORMAL_MODE=1,DOL2_MODE=2,DOL3_MODE=3,DOL4_MODE=4,PWL_MODE=5(Compression
mode),*/
}

```

```

"bit_width": 12,/*Effective value 8 10 12 14 16 20*/
"isp_raw_bypass": 0, /*Whether to bypass raw processing. Enabled when YUV format passes through*/
"test_pattern_enable" : 0,/*Internal test data enabling*/
"test_pattern_type": 0,
"out_width": 1920,/*ISP output width*/
"out_height": 1080, /*ISP output height*/
"output_dma_enable": 0,/*ISP can be online to the next level and ddr output simultaneously. Whether to write ddr
(enabled when dumping ISP data)*/
"output_format": 0,/*nv12 is fixed currently*/
"output_buf_num": 8,/*The number of buf outputted by ddr. 6 or above are suggested. */
"isp_algo_state": 1,/*Whether to enable isp 3a algorithm */
"calib_mode": 1,/*Whether to enable sensor correction data loading*/
"calib_lname": "/etc/cam/libimx327_linear.so",/*Corresponding to the used calibration library*/
"calib_sname": "/etc/cam/imx327_static.json",/*Corresponding to the used calibration parameters*/
"calib_dname": "/etc/cam/imx327_dynamic.json"/*Corresponding to the used calibration parameters*/
},
"dwe": {
"ldc": {
"ldc_enable": 0,/*1 enable ldc processing, 0 bypass to ipu */
"y_only": 0, /*0 YCC 420 1 only y. No need to be modified*/
"uv_mode": 0, /*0 output nv12 1 output nv21*/
"uv_interpolation": 0, /*0: interpolation; 1: take a point from the upper left*/
"h_blank_cycle": 32, /*The number of empty rows for IPU, 32 at least*/
"image_width": 1919,/* Set the size of the accessed width (-1). At present, if the ISP outputs 1920, set 1919 here */
"image_height": 1079,/* Set the size of the accessed height (-1). At present, if the ISP outputs 1080, set 1079 here */
"y_start_addr": 524288,/*Address used by iram, calculate cache*/
"c_start_addr": 786432, /* Address used by iram, calculate cache */
"line_buf" : 99,/*Cache buf line*/
"algo_xpara_a": 1,/*Parameter tuning is required*/
"algo_xpara_b": 1,/* Parameter tuning is required */
"algo_ypara_a": 1,/* Parameter tuning is required */
"algo_ypara_b": 1,/* Parameter tuning is required */
"center_xoffset": 0,/* Center processing region corrects x offset. 0: Start point of the original image */
"center_yoffset": 0,/* Center processing region corrects y offset. 0: Start point of the original image */
"x_start": 0,/*Input effective region*/
"x_length": 1919,/* Input effective region */
"y_start": 0,/* Input effective region */
"y_length": 1079/* Input effective region */
},
"dis": {
"dis_enable": 0,/*Enable dis.0: bypass*/
"dis_path_sel": 1, /*0 before ldc, 1 after ldc*/
"image_width": 1919, /* Set the size of the accessed width (-1). At present, if the ISP outputs 1920, set 1919 here */
"image_height": 1079, /* Set the size of the accessed height (-1). At present, if the ISP outputs 1080, set 1079 here */
}
}

```

```

    "h_ratio": 65536, /* Zoom out ratio, 16 Decimal eg. 0x20000: zoom 1/2 */
    "v_ratio": 65536, /* Zoom out ratio, 16 Decimal */
    "x_start": 0,/*input crop*/
    "x_end": 1919,/*input crop*/
    "y_start": 0,/*input crop*/
    "y_end": 1079/*input crop*/
}
},
"ipu": {
    "ipu_config": {
        "source_sel": 1,
        /* 0: sif direct connection mode (yuv422), 1: ISP hardware direct connection mode (yuv420), 3: ddr in mode
(yuv420sp) */
        "ds2_to_ddr_en": 0,/* ds2 support online to pym and ddr simultaneously. Whether to pass to ddr*/
        "frame_id": 1,/*Set frame id initial value. Startting with 1 is suggested*/
        "us_frame_id_en": 0,/*Enable frame id count. Enabled by default*/
        "ds_0_frame_id_en": 0,
        "ds_1_frame_id_en": 0,
        "ds_2_frame_id_en": 0,
        "ds_3_frame_id_en": 0,
        "ds_4_frame_id_en": 0,
        "ddr_in_buf_num": 0,/*The number of buf outputted in the ddr in mode*/
        "timeout": 2000/* Timeout for getting data */
    },
    "cfg_size": {
        "source_width": 1920,/* Data width passed to the IPU, 4096 at most*/
        "source_height": 1080, /* Data height passed to the IPU, 4096 at most */
        "source_stride_y": 1920, /*The hardware y that is passed to the IPU writes stride. Same as width*/
        "source_stride_uv": 1920, /* The hardware uv that is passed to the IPU writes stride. Same as width */
        "ipu_us_config": {
            "upscale_roi_en": 0,/* Enable the ROI function. If the channel is outputted to ddr, either one of roi and scale should
be enabled*/
            "us_roi_start_x": 0,/* The X starting point coordinate of ROI. Should be a multiple of 4 and cannot exceed the width
of the input size */
            "us_roi_start_y": 0,/* The y starting point coordinate of ROI. Should be a multiple of 4 and cannot exceed the height
of the input size */
            "us_roi_width": 1920,/* The width of ROI region. Should be a multiple of 4 and cannot exceed the width of the input
size */
            "us_roi_height": 1080,/* The height of ROI region. Should be a multiple of 4 and cannot exceed the height of the
input size */
            "upscale_us_en": 0,/*Enable the scaling function*/
            "us_tag_width": 1920,
            /* Supports zoom in 1.5 times at the maximum in horizontal direction. Width should be a multiple of 4. 32x32 at
the minimum and 4096 at the maximum
        }
    }
}
}

```

```

        Set it equal to roi width if the scale function is disabled */
        "us_tag_height": 1080,
        /* Supports zoom in 1.5 times at the maximum in vertical direction. Height should be an even number. 32x32 at
the minimum and 4096 at the maximum
        Set it equal to roi height if the scale function is disabled */
        "us_buf_num": 8/*The number of buf outputted by channel*/
    },/*
        The downscale outputs 4096x4096 at the maximum
        Zoom out to be 1/8 of the original size at the maximum in horizontal direction (> 1/8)
        Zoom out to be 1/8 of the original size at the maximum in vertical direction (> 1/8)
*/
    "ipu_ds_config": [
        {
            "ds0_roi_en": 1,/* Enable ds roi. If the channel is outputted to ddr, either one of roi and scale should be enabled*/
            "ds0_roi_start_x": 0, /* The X starting point coordinate of ROI. Should be a multiple of 4 and cannot exceed the
width of the input size */
            "ds0_roi_start_y": 0, /* The y starting point coordinate of ROI. Should be a multiple of 4 and cannot exceed the
height of the input size */
            "ds0_roi_width": 800, /* The width of ROI region. Should be a multiple of 4 and cannot exceed the width of the
input size */
            "ds0_roi_height": 480, /* The height of ROI region. Should be a multiple of 4 and cannot exceed the height of the
input size */
            "downscale_ds0_en": 1,/* Enable the scaling function */
            "ds0_tag_width": 800,
            /* Zoom out to be 1/8 of the original size at the maximum in horizontal direction (> 1/8), 32x32 at the minimum
and 4096 at the maximum
            Set it equal to roi width if the scale function is disabled*/
            "ds0_tag_height": 480,
            /* Zoom out to be 1/8 of the original size at the maximum in vertical direction (> 1/8) , 32x32 at the minimum and
4096 at the maximum
            Set it equal to roi height if the scale function is disabled */
            "ds0_buf_num": 16/* The number of buf outputted by channel */
        },
        {
            "ds1_roi_en": 0,
            "ds1_roi_start_x": 0,
            "ds1_roi_start_y": 0,
            "ds1_roi_width": 1280,
            "ds1_roi_height": 720,
            "downscale_ds1_en": 0,
            "ds1_tag_width": 1280,
            "ds1_tag_height": 720,
            "ds1_buf_num": 8
        },
    ],
}

```

```

{
    "ds2_roi_en": 0,
    /* Special note: ds2 channel can be directly connected to the PYM, and can also be outputted to the DDR. When
selecting the PYM as the direct connection mode, either one of roi or scale shall be enabled, otherwise the PYM has no data. Whether
the ds2 ddr outputs or not depends on ds2_to_ddr_en*/
    "ds2_roi_start_x": 0,
    "ds2_roi_start_y": 0,
    "ds2_roi_width": 1280,
    "ds2_roi_height": 720,
    "downscale_ds2_en": 0,
    "ds2_tag_width": 1280,
    "ds2_tag_height": 720,
    "ds2_buf_num": 8
},
{
    "ds3_roi_en": 0,
    "ds3_roi_start_x": 0,
    "ds3_roi_start_y": 0,
    "ds3_roi_width": 1280,
    "ds3_roi_height": 720,
    "downscale_ds3_en": 0,
    "ds3_tag_width": 1280,
    "ds3_tag_height": 720,
    "ds3_buf_num": 8
},
{
    "ds4_roi_en": 0,
    "ds4_roi_start_x": 0,
    "ds4_roi_start_y": 0,
    "ds4_roi_width": 1280,
    "ds4_roi_height": 720,
    "downscale_ds4_en": 0,
    "ds4_tag_width": 1280,
    "ds4_tag_height": 720,
    "ds4_buf_num": 8
}
],
}

"pym": {
    "pym_ctrl_config": {
        "img_src_sel": 0,/* 0: ddr in mode, 1: ipu online mode, receive data from ds2 */
        "frame_id": 0,/*frame id starting settings*/
        "src_w": 1280,
    }
}
}

```

```

/*The size outputted to the pym. If it is outputted from the ds2 direct connection mode, the width should be the same
as ds2 target w
    MAX 4096, MIN 64*/
    "src_h": 720,
    /* The size outputted to the pym. If it is outputted from the ds2 direct connection mode, the width should be the same
as ds2 target h
    MAX 4096, MIN 64*/
    "ds_layer_en": 23, /*The layer number of the downscale, 4 ~ 23*/
    "ds_uv_bypass": 0, /*Whether to bypass uv value*/
    "us_layer_en": 0, /* The layer number of the upscale , 0 ~ 6*/
    "us_uv_bypass": 0, /* Whether to bypass uv value */
    "ddr_in_buf_num" : 6,/*The buf number of the fillback source*/
    "output_buf_num": 8,/* The number of buf outputted by the pym */
    "timeout": 2000/* Timeout to get PYM results */
},
/*
* Down Scale: There are two concept needed to know – base layer and ROI layer:
* Base layer: 4*k (k = 0~5) layers, and the multiple of scaling per layer is (1/2)^k;
* ROI layer: Obtained after scaling selected ROI region by the base layer;
* layer 1, 2, 3 based on layer 0
* layer 5, 6, 7 based on layer 4
* layer 9, 10, 11 based on layer 8
* layer 13, 14, 15 based on layer 12
* layer 17, 18, 19 based on layer 16
* layer 21, 22, 23 based on layer 20
* roi scale factor 64/(64 + factor). If the factor is 0, it means to disable the zoom of this layer. Factor range: 0~63
* target_width_y = ((roi_w - 1) * 64 / (64 + factor) +1 >> 1) << 1
* target_height_y = ((roi_h - 1) * 64 / (64 + factor) +1 >> 1) << 1
* target_width_uv = ((roi_w / 2 - 1) * 64 / (64 + factor) +1 >> 1) << 1
* target_height_uv = (((roi_h / 2 - 1) * 64 / (64 + factor) +1 >> 1) << 1
Chip limits:
    Minimum size of pym zoom output 48 * 32
    Minimum size of pym zoom output 2048 * 2048
    Tag width and height are taken below the current value to get an even number. For example, if the size 401 * 401,
400 * 400 will be got.
*/
"pym_ds_config": {
    "roi_x_1": 0, /* Coordinates of the ROI region corresponding to the left corner of base layer, shall be even number */
    "roi_y_1": 0, /* Coordinates of the ROI region corresponding to the left corner of base layer, shall be even number */
    "roi_w_1": 0, /* ROI region size, which cannot exceed the base layer size. Shall be even number.*/
    "roi_h_1": 0, /* ROI region size, which cannot exceed the base layer size. Shall be even number.*/
    "factor_1": 0,
    "roi_x_2": 0,
    "roi_y_2": 0,
}

```

```
"roi_w_2": 0,  
"roi_h_2": 0,  
"factor_2": 0,  
"roi_x_3": 0,  
"roi_y_3": 0,  
"roi_w_3": 0,  
"roi_h_3": 0,  
"factor_3": 0,  
"roi_x_5": 0,  
"roi_y_5": 0,  
"roi_w_5": 0,  
"roi_h_5": 0,  
"factor_5": 0,  
"roi_x_6": 0,  
"roi_y_6": 0,  
"roi_w_6": 0,  
"roi_h_6": 0,  
"factor_6": 0,  
"roi_x_7": 0,  
"roi_y_7": 0,  
"roi_w_7": 0,  
"roi_h_7": 0,  
"factor_7": 0,  
"roi_x_9": 0,  
"roi_y_9": 0,  
"roi_w_9": 0,  
"roi_h_9": 0,  
"factor_9": 0,  
"roi_x_10": 0,  
"roi_y_10": 0,  
"roi_w_10": 0,  
"roi_h_10": 0,  
"factor_10": 0,  
"roi_x_11": 0,  
"roi_y_11": 0,  
"roi_w_11": 0,  
"roi_h_11": 0,  
"factor_11": 0,  
"roi_x_13": 0,  
"roi_y_13": 0,  
"roi_w_13": 0,  
"roi_h_13": 0,  
"factor_13": 0,  
"roi_x_14": 0,
```

```
"roi_y_14": 0,  
"roi_w_14": 0,  
"roi_h_14": 0,  
"factor_14": 0,  
"roi_x_15": 0,  
"roi_y_15": 0,  
"roi_w_15": 0,  
"roi_h_15": 0,  
"factor_15": 0,  
"roi_x_17": 0,  
"roi_y_17": 0,  
"roi_w_17": 0,  
"roi_h_17": 0,  
"factor_17": 0,  
"roi_x_18": 0,  
"roi_y_18": 0,  
"roi_w_18": 0,  
"roi_h_18": 0,  
"factor_18": 0,  
"roi_x_19": 0,  
"roi_y_19": 0,  
"roi_w_19": 0,  
"roi_h_19": 0,  
"factor_19": 0,  
"roi_x_21": 0,  
"roi_y_21": 0,  
"roi_w_21": 0,  
"roi_h_21": 0,  
"factor_21": 0,  
"roi_x_22": 0,  
"roi_y_22": 0,  
"roi_w_22": 0,  
"roi_h_22": 0,  
"factor_22": 0,  
"roi_x_23": 0,  
"roi_y_23": 0,  
"roi_w_23": 0,  
"roi_h_23": 0,  
"factor_23": 0  
},
```

/* up_scaler image layers are 6 in total. Select 6 regions randomly from the raw image layer for magnification. At present, the factor magnification size is a fixed value, i.e.

* factor_0:50 1.28x magnification

* factor_1:40 1.6x magnification

```

* factor_2:32 2x magnification
* factor_3:25 2.56x magnification
* factor_4:20 3.2x magnification
* factor_5:16 4x magnification

* Calculation:
*     target_width = (((roi_w / 2 - 1) * 64 - 1) / (factor) + 1) * 2
*     target_hight = (((roi_h / 2 - 1) * 64 - 1) / (factor) + 1) * 2
*     MAX 4096 x 4096
*/


"pym_us_config": {


    "roi_x_0": 0, /* Coordinates of the ROI region corresponding to the left corner of base layer, shall be even number.*/
    "roi_y_0": 0, /* Coordinates of the ROI region corresponding to the left corner of base layer, shall be even number.*/
    "roi_w_0": 200, /* ROI region size, which cannot exceed the base layer size, shall be even number*/
    "roi_h_0": 100, /* ROI region size, which cannot exceed the base layer size, shall be even number */
    "factor_0": 50,
    "roi_x_1": 0,
    "roi_y_1": 0,
    "roi_w_1": 0,
    "roi_h_1": 0,
    "factor_1": 40,
    "roi_x_2": 0,
    "roi_y_2": 0,
    "roi_w_2": 0,
    "roi_h_2": 0,
    "factor_2": 32,
    "roi_x_3": 0,
    "roi_y_3": 0,
    "roi_w_3": 0,
    "roi_h_3": 0,
    "factor_3": 25,
    "roi_x_4": 0,
    "roi_y_4": 0,
    "roi_w_4": 0,
    "roi_h_4": 0,
    "factor_4": 20,
    "roi_x_5": 0,
    "roi_y_5": 0,
    "roi_w_5": 0,
    "roi_h_5": 0,
    "factor_5": 16
}
},
/*
*GDC request output width 16 align

```

```
*In 1920x1080 -0 rotate-> Out 1920x1080
*In 1920x1080 -90 rotate->Out 1088x1920
*In 1080x1920 -0 rotate-> Out 1088x1920
*In 1080x1920 -90 rotate->Out 1920x1080
*/
"gdc": {
    "sensor_id": 0, /*Corresponding to the sensor id of the channel, on which The GDC calibration file loading depends.
Support 0,90,180,270 rotation*/
        "input_width": 1280,/*input Width settings of the GDC proccesing*/
        "input_height": 720,/* input Height settings of the GDC proccesing */
        "output_width": 1280,/*output Width settings of the GDC proccesing*/
        "output_height": 720,/* output Height settings of the GDC proccesing */
        "fb_buf_num": 8, /*Buf number settings of the GDC feedback src buf*/
        "buf_num": 8 /*Buf number settings of the GDC output*/
    }
}
}
```

4.10 VIO Accessable Environment Variable

4.10.1 X3/J3 environment variable

environment variable: PYM_UV_ADDR_4K_ALIGN

[Function Description]

- Non-0: UV address of each channel image data outputted by the PYM will be 4K aligned (Y address is aligned by default)
- 0: UV address is close to the Y data, and no alignment reservation is performed

Note: if this parameter is not set, the UV will be aligned by default

```
export PYM_UV_ADDR_4K_ALIGN=0 //Set environment variable
echo $PYM_UV_ADDR_4K_ALIGN //Display current value
```

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

5 Encoding and Decoding

5.1 MediaCodec Interface Descriptions

Mediacodec module is mainly used for encoding and decoding audio, video, and JPEG images. This module will provide a series of interfaces for users to input the data to be processed and get the processed data. This module supports multi-channel encoding or decoding instances working at the same time, in which the encoding and decoding of videos and JPEG images are completed by hardware, and that of audios are realized by software using FFmpeg interface. Table 5-1 and table 5-2 are the video encoding and decoding specifications and audio encoding and decoding specifications supported by X3/J3, respectively.

Note: The encoding and decoding of AAC should be authorized by license, so users need to obtain authorization before they can enable such codes. And the decoder only supports frame mode.

Table 5-1 Encoding and Decoding Specifications of Videos and Images

Chip Type	Encoding and Decoding Specifications of Videos and Images							
	H264		H265			MJPEG	JPEG	
	Profile	Level	Profile	Level	Tier	ISO/IEC 10918-1	ISO/IEC 10918-1	
X3/J3	Baseline	L5.2	Main	L5.1	High-tier	Baseline sequential	Baseline sequential	
	Constrained Baseline		Main10			Extended sequential	Extended sequential	
	Main							
	High							
	High 10							

Table 5-2 Audio Encoding and Decoding Specifications

Chip Type	Audio Encoding and Decoding Specifications				
	G.711	G.726	ADPCM	FLAC	AAC
X3/J3	A-law	G.726 ADPCM	ADPCM IMA WAV	Flac	AAC LC
	Mu-law				AAC Main
					AAC SSR
					AAC LTP
					AAC LD
					AAC HE
					AAC HEv2
					(The above specifications are not supported by default, and license authorization is required)

In addition, video and image data sources include image input of VIO and YUV data input of users. VIO image input may be the images scaled by IPU in VIO, while users' YUV data input may be obtained by loading from files or transmitting over the network. Audio data sources include audio input of MIC and PCM data input of users. MIC audio input collects digital signals by using Audio Codec, while users' PCM data input may be obtained from files or network transmission.

5.1.1 GOP

H264 and H265 codes support GOP structure settings, and users can select from 8 preset GOP structure or customize GOP structure.

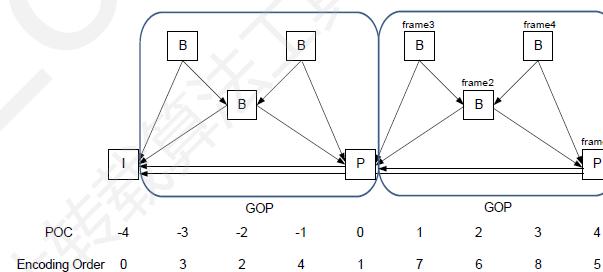
5.1.1.1 GOP structural table

The GOP structural table defines a set of periodic GOP structures that can be used in the whole encoding process. The elements in a single structural table are shown in Table 5-3, in which the reference frame of the image can be specified. If the reference frame specified by other frames behind the IDR frame is the data frame before the IDR frame, the encoder will automatically process it to make it not refer to other frames, so users do not need to care about this situation. When customizing the GOP structure, users need to specify the number of structural tables, which can be defined up to 8. The order of structural tables needs to be arranged in decoding order.

Table 5-3 GOP Structural Table Elements

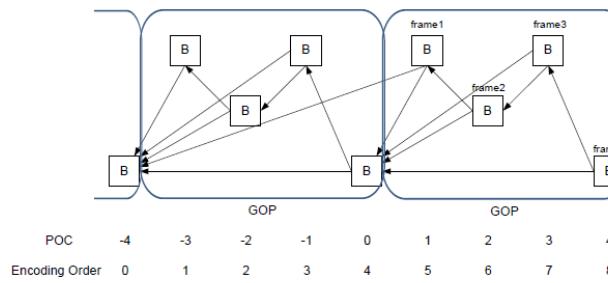
Element	Description
Type	Slice type (I, P or B)
POC	Display order of the frame within a GOP, ranging from 1 to GOP size

Element	Description
Qoffset	A quantization parameter of the picture in the custom GOP
NUM_REF_PIC_L0	Flag to use multi reference picture for P picture It is valid only if PIC_TYPE is P
temporal_id	Temporal layer of the frame. A frame cannot predict from a frame with a higher temporal id(0~6).
1st_ref_POC	The POC of the 1st reference picture of L0
2st_ref_POC	The POC of 1st reference picture of L1 in case that Type is equal to B The POC of 2nd reference picture of L0 in case that Type is equal to P Note that reference_L1 can have the same POC as reference in B slice. But for compression efficiency it is recommended that reference_L1 have a different POC from reference_L0



Frame#	Type	POC	QP	NUM_REF_PIC_L0	temporal_id	1st_ref_POC	2st_ref_POC
Frame1	P	4	1	2	0	0	-4
Frame2	B	2	2	0	1	0	4
Frame3	B	1	3	0	2	0	2
Frame4	B	3	3	0	2	2	4

Table 5-1 Sample Graph 1 of GOP Structural Table



Frame#	Type	POC	QP	NUM_REF_PIC_ID	temporal_id	1st_ref_POC	2st_ref_POC
Frame1	B	1	3	0	0	0	-4
Frame2	B	2	2	0	0	1	0
Frame3	B	3	3	0	0	2	0
Frame4	B	4	1	0	0	3	0

Table 5-2 Sample Graph 2 of GOP Structural Table

5.1.1.2 GOP Preset Structure

Table 5-4 shows the preset 8 GOP structures.

Table 5-4 Gop preset structure

Index	GOP Structure	Low Delay (encoding order and display order are same)	GOP Size	Encoding Order	Minimum Source Frame Buffer	Minimum Decoded Picture Buffer	Intra Period (I Frame Interval) Requirement
1	I	Yes	1	I0-I1-I2...	1	1	
2	P	Yes	1	P0-P1-P2...	1	2	
3	B	Yes	1	B0-B1-B2...	1	3	
4	BP	No	2	B1-P0-B3-P2...	4	3	Multiple of 2
5	BBBBP	No	4	B2-B1-B3-P0...	7	4	Multiple of 4
6	PPPP	Yes	4	P0-P1-P2-P3...	1	2	
7	BBBB	Yes	4	B0-B1-B2-B3...	1	3	

Index	GOP Structure	Low Delay (encoding order and display order are same)	GOP Size	Encoding Order	Minimum Source Frame Buffer	Minimum Decoded Picture Buffer	Intra Period (I Frame Interval) Requirement
8	BBBBBBBB	No	8	B3-B2-B4-B1-B6-B5-B7-B0...	12	5	Multiple of 8

The following describes the 8 preset GOP structures.

- GOP Preset 1
 - I frame only, no cross reference;
 - Low latency

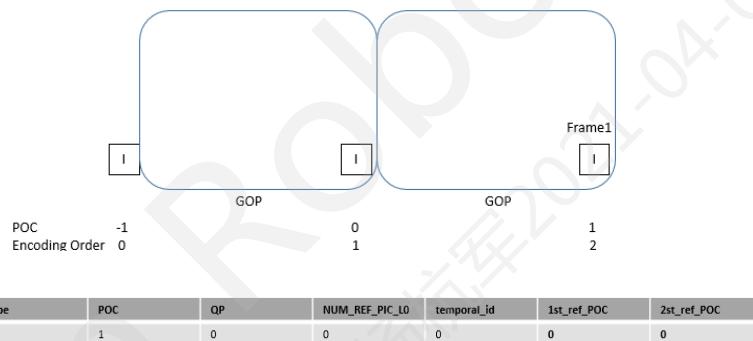


Table 5-3 GOP Preset Structure 1

- GOP Preset 2
 - I frame and P frame only;
 - P frame refers to two forward reference frames;
 - Low latency;

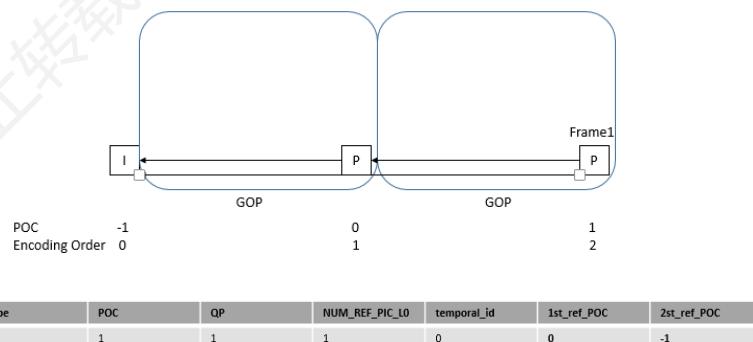


Table 5-4 GOP Preset Structure 2

- GOP Preset 3
 - I frame and B frame only;

- B frame refers to two forward reference frames;
- Low latency;

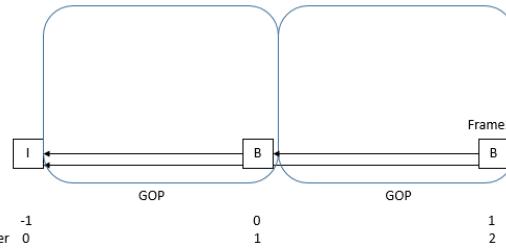


Table 5-5 GOP Preset Structure 3

- GOP Preset 4

- I, P, and B frames;
- P frame refers to two forward reference frames;
- B frame refers to a forward reference frame and a backward reference frame;

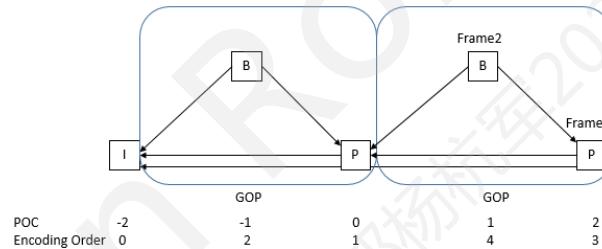
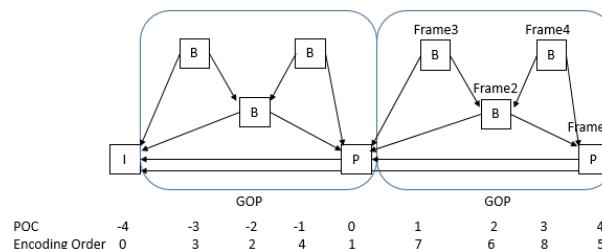


Table 5-6 GOP Preset Structure 4

- GOP Preset 5

- I, P, and B frames;
- P frame refers to two forward reference frames;
- B frame refers to a forward reference frame and a backward reference frame, and the backward reference frame can be P frame or B frame;

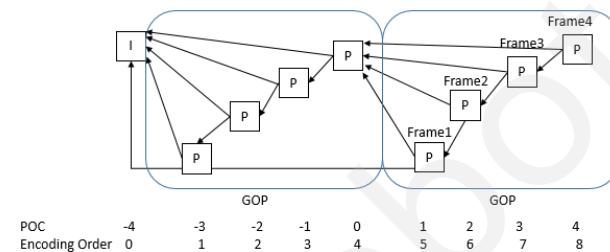


Frame#	Type	POC	QP	NUM_REF_PIC_L0	temporal_id	1st_ref_POC	2st_ref_POC
Frame1	P	4	1	2	0	0	-4
Frame2	B	2	3	0	0	0	4
Frame3	B	1	5	0	0	0	2
Frame4	B	3	5	0	0	2	4

Table 5-7 GOP Preset Structure 5

- GOP Preset 6

- I frame and P frame only;
- P frame refers to two forward reference frames;
- Low latency;

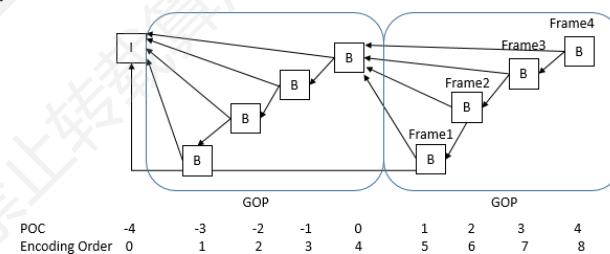


Frame#	Type	POC	QP	NUM_REF_PIC_L0	temporal_id	1st_ref_POC	2st_ref_POC
Frame1	P	1	5	2	0	0	-4
Frame2	P	2	3	2	0	1	0
Frame3	P	3	5	2	0	2	0
Frame4	P	4	1	2	0	3	0

Table 5-8 GOP Preset Structure 6

- GOP Preset 7

- I frame and B frame only;
- B frame refers to two forward reference frames;
- Low latency;



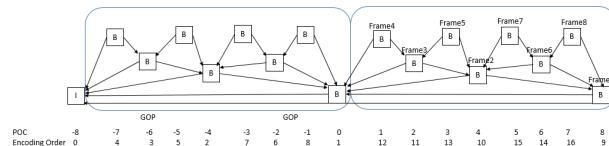
Frame#	Type	POC	QP	NUM_REF_PIC_L0	temporal_id	1st_ref_POC	2st_ref_POC
Frame1	B	1	5	2	0	0	-4
Frame2	B	2	3	2	0	1	0
Frame3	B	3	5	2	0	2	0
Frame4	B	4	1	2	0	3	0

Table 5-9 GOP Preset Structure 7

- GOP Preset 8

- I frame and B frame only;

- B frame refers to a forward reference frame and a backward reference frame;



Frame#	Type	POC	QP	NUM_REF_PIC_L0	temporal_id	1st_ref_POC	2st_ref_POC
Frame1	B	8	1	0	0	0	-8
Frame2	B	4	3	0	0	0	8
Frame3	B	2	5	0	0	0	4
Frame4	B	1	8	0	0	0	2
Frame5	B	3	8	0	0	2	4
Frame6	B	6	5	0	0	4	8
Frame7	B	5	8	0	0	4	6
Frame8	B	7	8	0	0	6	8

Table 5-10 GOP Preset Structure 8

5.1.1.3 Period Relationship between GOP and I Frame

The period relationship between the GOP structure and I frame is shown in the figure below:

GOPSzize (BBBBP) = 4
IntraPeriod = 8
DecodingRefreshType = IDR

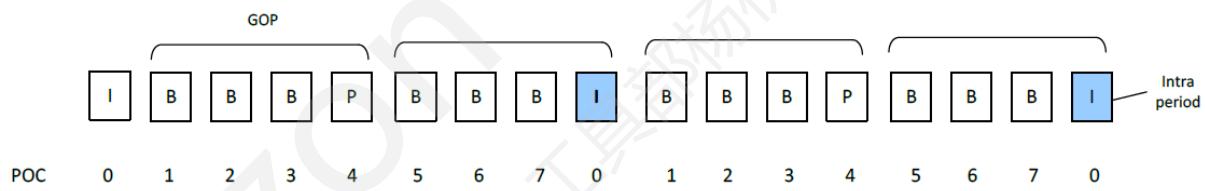


Table 5-11 Period Relationship between GOP preset structure 5 and I Frame

5.1.2 Long-term Reference Frame

You can specify the period of the long-term reference frame and refer to the period of the long-term reference frame, as shown in the following figure:

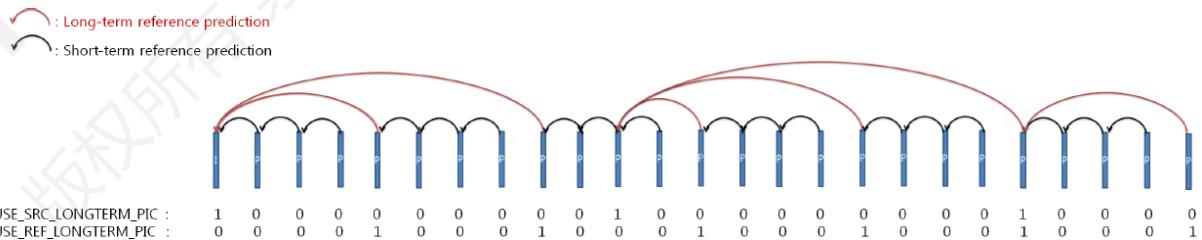


Figure 5-12 Example of Long-term Reference Frame

5.1.3 Intra Refresh

Intra refresh mode improves fault tolerance by periodically inserting intra-frame encoded MB/CTU in non-I frames. It can provide more fixing points for decoder to avoid image damage caused by temporal error. You can specify the number of consecutive rows, columns, or strides of MB/CTU to force the encoder to insert the intra-frame encoding unit, and specify the size of the intra-frame encoding unit. It is up to the encoder to determine which part of the frame should be encoded.

5.1.4 Bitrate Control

MediaCodec supports the bitrate control for H264, H265, and MJPEG protocols. It supports five ways of the bitrate control including the CBR, VBR, AVBR, FixQp, and QpMap of H264/H265 encoding channel and supports for FixQp bitrate control over MJPGE encoding channel. The CBR guarantees the overall coding rate stability; The VBR guarantees the quality stability of the coding image; The AVBR gives consideration to both the bitrate and the image quality, and generates the code stream with the relatively stable bitrate and the image quality; The FixQp is used to fix the QP value of each I frame, P frame, and B frame; The QPMAP is used to specify the QP value for each block in a frame image, wherein the size of H264 block is 16x16, and the size of H265 block is 32x32.

For the CBR and AVBR, the encoder will find the appropriate QP value for each frame to ensure a constant bitrate. The encoder supports three levels of rate control, namely frame level, CTU/MB level, and subCTU/subMB level. The frame level bitrate control mainly generates a QP value for each frame according to the target bitrate, so as to ensure the constant bitrate; the CTU/MB level bitrate control generates a QP value for each block according to the target bitrate of each 64x64 CTU or 16x16 MB, which can get better rate control, but frequent adjustment of QP value will result in unstable image quality; the subCTU/subMB level bitrate control generates a QP value for each 32x32 subCTU or 8x8 subMB, in which the complex block will get a higher QP value, while the static block will get a lower QP value, because compared with the complex area, the human eyes are more sensitive to the static area. The detection of complex and static areas mainly depends on the internal hardware module. This level of rate control is mainly used to improve the subjective image quality and keep the bitrate constant. In this mode, SSIM score is higher, but PSNR score is lower.

5.1.5 ROI

The implementation of ROI coding is similar to QPMAP. You need to set the QP value for each block in the raster scan direction. The following figure shows an example of ROI map of H265. For H264 encoding, the size of each block is 16x16, compared with 32x32 in H265. In the ROI map table, each QP value takes up one byte, and the size is 0~51.

ROI coding can work with CBR and AVBR. When CBR or AVBR is not enabled, the actual QP value of each block region is the value specified in the ROI map. When CBR or AVBR is enabled, the actual value of each block region is obtained by the following formula:

$$QP(i) = MQP(i) + RQP(i) - ROIAvgQP$$

Wherein, MQP is the value in the ROI map, RQP is the value obtained from the bitrate control in the encoder, and ROIAvQP is the average value of QP in the ROI map.

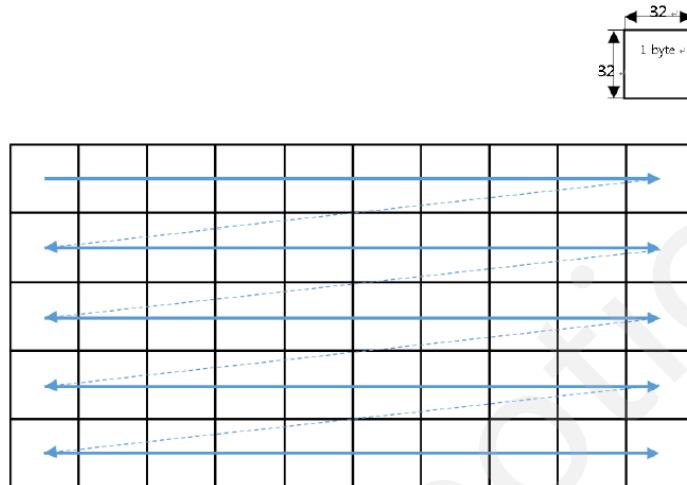


Figure 5-13 H265 ROI map

5.1.6 Smart background encoding

In the video surveillance scene, the background is fixed in many cases. Therefore, it is expected that the encoder will adopt the ignore mode or use less code streams to encode the background region when it detects it. In the actual scene, the background region detection is not easy because of the noise in the camera image. In many cases, the ISP needs to notify the encoder after it detects the background region, which will consume extra bandwidth and system computing resources.

H264 and H265 encodings provide an intelligent background encoding mode integrated in codec, which makes full use of internal hardware modules and on-the-fly processing, without consuming additional bandwidth and system resources. The following figure shows the working mode of background detection. In the intelligent background encoding mode, the internal hardware module will compare each block unit with the block unit of the reference frame to determine whether the block is the background.

For the determination of the background region, you can set the maximum pixel difference (recommended value 8) and the average pixel difference (recommended value 1). You can also adjust the lambda parameters to affect the mode selection in the encoding. When the background region is detected, the corresponding lambda value will be increased for each block in the encoder, so that the encoder is inclined to adopt the ignore mode to encode the block unit. For the lambda control, you can set lambdaQP (recommended value 32) and deltaQP (recommended value 3). The final lambda value is calculated according to the following formula:

$$\text{Lambda} = \text{QP_TO_LAMBDA_TABLE}[\max(\text{lambdaQP}, \text{QP} + \text{deltaQP})]$$

QP_TO_LAMBDA_TABLE is a lambda conversion table, which will also be used for lambda conversion of non-background region.

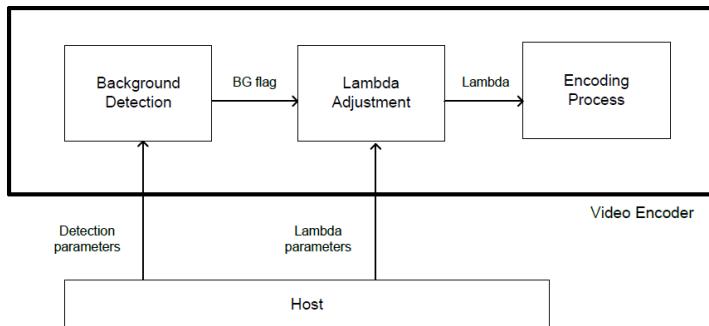


Figure 5-14 Smart background encoding working mode

Note: When the ROI encoding is enabled, the smart background encoding will not work. The I frame interval and bitrate make a big difference with the bandwidth saved by the mode. The larger this interval become, the more the saved bitrate. In addition, you can set the frame with better image quality as a long-term reference frame to improve the quality of background image and save the bitrate.

5.1.7 Frame Skip Settings

You can call `hb_mm_mc_skip_pic` to set the encoding mode of the image input for the next queue operation as the skip mode, which is only valid for non-I frame encoding. In the skip mode, the encoder ignores the input frame internally, and generates the reconstructed frame of this input by using the reconstructed frame of the previous frame, where the input frame is encoded as P frame.

5.1.8 JPEG codec limit

- JPEG/MJPEG encode mode, the input width should be 16 aligned, and the input height should be 8 aligned. If crop is used at the same time, the input X and Y coordinates should be 8 aligned.
- During JPEG / MJPEG encoding with rotating 90 / 270 at the same time, it requires that the width of the input should be 16 aligned, and the height of the input should be 16 aligned. If the crop was set at the same time, it requires that the width of the crop should be 16 aligned, and the height of the crop should be 16 aligned.
- During JPEG / MJPEG decoding with rotate or mirror set at the same time, the output YUV format is required same as the input image format, but when do YUV422 format JPEG / MJPEG decoding with rotated 90 / 270, the output format is required to be yuv440p / yuyv / yvyu / uyvy / vyuy.
- During JPEG / MJPEG decoding, rotate or mirror cannot work with crop at the same time.
- During JPEG / MJPEG decoding, the width and height of the output buffer should be aligned with the input MCU width and height respectively. If crop is enabled, the parameters of crop (including the starting coordinates and width and height) should be aligned with the input MCU width and height respectively; (MCU size for 420: 16x16, 422: 16x8, 440: 8x16, 400: 8x8, 444: 8x8.)
- During JPEG / MJPEG decoding, the output format is packed yuv444, and the input format is required to be yuv444.
- JPEG / MJPEG decoding only supports MC_FEEDING_MODE_FRAME_SIZE mode.
- During JPEG / MJPEG decoding with rotating 90 / 270 at the same time, if the input format is YUV422, the output format is changed to YUV440; if the input format is YUV440, the output format is changed to YUV422.
- During JPEG / MJPEG encoding, the size of bitstream buffer should add additional 4K.

- During JPEG / MJPEG encoding, the width and height is processed according to MCU width and height respectively. If the width and height isn't aligned with MCU size, there will be extra padding data.

5.2 MediaCodec API

5.2.1 hb_mm_mc_get_descriptor

[Function Declaration]

```
const media_codec_descriptor_t *hb_mm_mc_get_descriptor(media_codec_id_t codec_id);
```

[Parameter Description]

- [IN] media_codec_id_t codec_id: Codec type

[Return Value]

- Non-null: Codec descrptions
- NULL: Descriptor corresponding to the codec id cannot be found

[Function Description]

Get the codec information supported by MediaCodec using codec id, which includes codec name, detailed description, MIME type, profile type supported by codec, etc.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
#include "hb_media_codec.h"  
#include "hb_media_error.h"  
  
int main(int argc, char *argv[]){  
    const media_codec_descriptor_t *desc = NULL;  
    desc = hb_mm_mc_get_descriptor(MEDIA_CODEC_ID_H264);  
    return 0;  
}
```

5.2.2 hb_mm_mc_get_default_context

[Function Declaration]

```
hb_s32 hb_mm_mc_get_default_context(media_codec_id_t codec_id, hb_bool encoder,  
media_codec_context_t *context)
```

[Parameter Description]

- [IN] media_codec_id_t codec_id: Codec type
- [IN] hb_bool encoder: Specify whether codec is encoder or decoder
- [OUT] media_codec_context_t *context: Specify the default context of codec type

[Return Value]

- 0: Success
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the default attribute of the specified codec.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
#include "hb_media_codec.h"
#include "hb_media_error.h"

int main(int argc, char *argv[])
{
    int ret = 0;
    media_codec_context_t context;
    memset(&context, 0x00, sizeof(context));
    ret = hb_mm_mc_get_default_context(MEDIA_CODEC_ID_H264, 1, &context);

    return 0;
}
```

5.2.3 hb_mm_mc_initialize

[Function Declaration]

```
hb_s32 hb_mm_mc_initialize(media_codec_context_t *context)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the default context of codec type

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INSUFFICIENT_RES: Insufficient internal memory resources
- HB_MEDIA_ERR_NO_FREE_INSTANCE: No instance available (at most 32 videos, 64 MJPEG/JPEG, and 32 audios)

[Function Description]

Initialize the encoder or decoder. MediaCodec will get into **MEDIA_CODEC_STATE_INITIALIZED** status after the call succeeds.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
#include "hb_media_codec.h"
#include "hb_media_error.h"

static Uint64 osal_gettime(void)
{
    struct timespec tp;

    clock_gettime(CLOCK_MONOTONIC, &tp);

    return ((Uint64)tp.tv_sec*1000 + tp.tv_nsec/1000000);
}

typedef struct MediaCodecTestContext {
    media_codec_context_t *context;
    char *inputFileName;
    char *outputFileName;
} MediaCodecTestContext;

typedef struct AsyncMediaCtx {
    media_codec_context_t *ctx;
    FILE *inFile;
    FILE *outFile;
    int lastStream;
    Uint64 startTime;
    int32_t duration;
} AsyncMediaCtx;

static void on_encoder_input_buffer_available(hb_ptr userdata, media_codec_buffer_t *inputBuffer) {
    AsyncMediaCtx *asyncCtx = (AsyncMediaCtx *)userdata;
```

```

Int noMoreInput = 0;
hb_s32 ret = 0;
Uint64 curTime = 0;

if (!noMoreInput) {
    curTime = osal_gettime();
    if ((curTime - asyncCtx->startTime)/1000 < (uint32_t)asyncCtx->duration) {
        ret = fread(inputBuffer->vframe_buf.vir_ptr[0], 1,
                    inputBuffer->vframe_buf.size, asyncCtx->inFile);
        if (ret <= 0) {
            if(fseek(asyncCtx->inFile, 0, SEEK_SET)) {
                printf("Failed to rewind input file\n");
            } else {
                ret = fread(inputBuffer->vframe_buf.vir_ptr[0], 1,
                            inputBuffer->vframe_buf.size, asyncCtx->inFile);
                if (ret <= 0) {
                    printf("Failed to read input file\n");
                }
            }
        }
    }
    if (!ret) {
        printf("%s There is no more input data!\n", TAG);
        inputBuffer->vframe_buf.frame_end = TRUE;
        noMoreInput = 1;
    }
} else {
    inputBuffer->vframe_buf.frame_end = TRUE;
    inputBuffer->vframe_buf.size = 0;
}
}

static void on_encoder_output_buffer_available(hb_ptr userdata, media_codec_buffer_t *outputBuffer,
media_codec_output_buffer_info_t *extraInfo) {
    AsyncMediaCtx *asyncCtx = (AsyncMediaCtx *)userdata;

    mc_h264_h265_output_stream_info_t info = extraInfo->video_stream_info;

    fwrite(outputBuffer->vstream_buf.vir_ptr, outputBuffer->vstream_buf.size, 1,
asyncCtx->outFile);
    if (outputBuffer->vstream_buf.stream_end) {
        printf("There is no more output data!\n");
        asyncCtx->lastStream = 1;
    }
}

```

```

}

static void on_encoder_media_codec_message(hb_ptr userdata, hb_s32 error) {
    AsyncMediaCtx *asyncCtx = (AsyncMediaCtx *)userdata;
    if (error) {
        asyncCtx->lastStream = 1;
        printf("ERROR happened!\n");
    }
}

static void on_vlc_buffer_message(hb_ptr userdata, hb_s32 * vlc_buf) {
    MediaCodecTestContext *ctx = (MediaCodecTestContext *)userdata;
    printf("%s %s VLC Buffer size = %d; Reset to %d.\n", TAG, __FUNCTION__,
           *vlc_buf, ctx->vlc_buf_size);
    *vlc_buf = ctx->vlc_buf_size;
}

static void do_async_encoding(void *arg) {
    hb_s32 ret = 0;
    FILE *outFile;
    FILE *inFile;
    int step = 0;
    AsyncMediaCtx asyncCtx;
    MediaCodecTestContext *ctx = (MediaCodecTestContext *)arg;
    media_codec_context_t *context = ctx->context;
    char *inputFileName = ctx->inputFileName;
    char *outputFileName = ctx->outputFileName;

    inFile = fopen(inputFileName, "rb");
    if (!inFile) {
        goto ERR;
    }
    outFile = fopen(outputFileName, "wb");
    if (!outFile) {
        goto ERR;
    }

    memset(&asyncCtx, 0x00, sizeof(AsyncMediaCtx));
    asyncCtx.ctx = context;
    asyncCtx.inFile = inFile;
    asyncCtx.outFile = outFile;
    asyncCtx.lastStream = 0;
    asyncCtx.duration = 5;
    asyncCtx.startTime = osal_gettime();
}

```

```
ret = hb_mm_mc_initialize(context);
if (ret) {
    goto ERR;
}

media_codec_callback_t callback;
callback.on_input_buffer_available = on_encoder_input_buffer_available;
callback.on_output_buffer_available = on_encoder_output_buffer_available;
callback.on_media_codec_message = on_encoder_media_codec_message;
ret = hb_mm_mc_set_callback(context, &callback, &asyncCtx);
if (ret) {
    goto ERR;
}

media_codec_callback_t callback2;
callback2.on_vlc_buffer_message = on_vlc_buffer_message;
if (ctx->vlc_buf_size > 0) {
    ret = hb_mm_mc_set_vlc_buffer_listener(context, &callback2, ctx);
    if (ret) {
        goto ERR;
    }
}

ret = hb_mm_mc_configure(context);
if (ret) {
    goto ERR;
}

mc_av_codec_startup_params_t startup_params;
startup_params.video_enc_startup_params.receive_frame_number = 0;
ret = hb_mm_mc_start(context, &startup_params);
if (ret) {
    goto ERR;
}

while(!asyncCtx.lastStream) {
    sleep(1);
}

hb_mm_mc_stop(context);

hb_mm_mc_release(context);
context = NULL;
```

```
ERR:  
  
    if (context && hb_mm_mc_get_state(context) != MEDIA_CODEC_STATE_UNINITIALIZED) {  
        hb_mm_mc_stop(context);  
        hb_mm_mc_release(context);  
    }  
  
    if (inFile)  
        fclose(inFile);  
  
    if (outFile)  
        fclose(outFile);  
}  
  
int main(int argc, char *argv[]){  
    int ret = 0;  
    char outputFileName[MAX_FILE_PATH] = "./tmp.yuv";  
    char inputFileName[MAX_FILE_PATH] = "./output.h264"  
    mc_video_codec_enc_params_t *params = NULL;  
    media_codec_context_t context;  
  
    memset(&context, 0x00, sizeof(media_codec_context_t));  
    context.codec_id = MEDIA_CODEC_ID_H264;  
    context.encoder = 1;  
    params = &context.video_enc_params;  
    params->width = 640;  
    params->height = 480;  
    params->pix_fmt = MC_PIXEL_FORMAT_YUV420P;  
    params->frame_buf_count = 5;  
    params->external_frame_buf = 0;  
    params->bitstream_buf_count = 5;  
    params->rc_params.mode = MC_AV_RC_MODE_H264CBR;  
    ret = hb_mm_mc_get_rate_control_config(&context, &params->rc_params);  
    if (ret) {  
        return -1;  
    }  
    params->rc_params.h264_cbr_params.bit_rate = 5000;  
    params->rc_params.h264_cbr_params.frame_rate = 30;  
    params->rc_params.h264_cbr_params.intra_period = 30;  
    params->gop_params.decoding_refresh_type = 2;  
    params->gop_params.gop_preset_idx = 2;  
    params->rot_degree = MC_CCW_0;  
    params->mir_direction = MC_DIRECTION_NONE;  
    params->frame_cropping_flag = FALSE;
```

```
MediaCodecTestContext ctx;
memset(&ctx, 0x00, sizeof(ctx));
ctx.context = &context;
ctx.inputFileName = inputFileName;
ctx.outputFileName = outputFileName;
do_async_encoding(&ctx);

return 0;
}
```

5.2.4 hb_mm_mc_set_callback

[Function Declaration]

```
hb_s32 hb_mm_mc_set_callback(media_codec_context_t *context, const media_codec_callback_t
*callback, hb_ptr userdata)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const media_codec_callback_t *callback: User callback function
- [IN] hb_ptr userdata: User data pointer, which will be fed as an input parameter when the callback function is called

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed

[Function Description]

Set the callback function pointer. After calling the function, MediaCodec will enter the asynchronous working mode.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.3 hb_mm_mc_initialize;

5.2.5 hb_mm_mc_set_vlc_buffer_listener

[Function Declaration]

```
hb_s32 hb_mm_mc_set_vlc_buffer_listener(media_codec_context_t *context, const  
media_codec_callback_t *callback, hb_ptr userdata)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const media_codec_callback_t *callback: User callback function
- [IN] hb_ptr userdata: User data pointer, which will be fed as an input parameter when the callback function is called

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed

[Function Description]

Set the VLC buffer size callback function pointer. MediaCodec will call the callback if the VLC buffer size is decided according to user setting, and user can modify the buffer size in the callback.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.3 hb_mm_mc_initialize;

5.2.6 hb_mm_mc_configure

[Function Declaration]

```
hb_s32 hb_mm_mc_configure(media_codec_context_t *context)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error

- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INSUFFICIENT_RES: Insufficient internal memory resources
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Configure the encoder or decoder using the input information. MediaCodec will get into **MEDIA_CODEC_STATE_CONFIGURED** status after the call succeeds.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.3 hb_mm_mc_initialize;

5.2.7 hb_mm_mc_start

[Function Declaration]

```
hb_s32 hb_mm_mc_start(media_codec_context_t *context, const mc_av_codec_startup_params_t  
*info)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] mc_av_codec_startup_params_t *info: Specify the start parameters of the encoding and decoding of the audio and video

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INSUFFICIENT_RES: Insufficient internal memory resources
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Start the encoding/decoding process, MediaCodec will create encoding/decoding instances, set a sequence or analyze data flow, register framebuffer, encode header information, etc., and get into **MEDIA_CODEC_STATE_STARTED** status after the call succeeds.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.3 hb_mm_mc_initialize;

5.2.8 hb_mm_mc_stop

[Function Declaration]

```
hb_s32 hb_mm_mc_stop(media_codec_context_t *context)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Stop the encoding / decoding process, exit all sub-threads, and release related resources.

MediaCodec will get back to **MEDIA_CODEC_STATE_INITIALIZED** status after the call succeeds.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.3 hb_mm_mc_initialize;

5.2.9 hb_mm_mc_pause

[Function Declaration]

```
hb_s32 hb_mm_mc_pause(media_codec_context_t *context)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Stop the encoding / decoding process, and pause all sub-threads. MediaCodec will get into **MEDIA_CODEC_STATE_PAUSED** status after the call succeeds.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.14 hb_mm_mc_queue_input_buffer;

5.2.10 hb_mm_mc_flush

[Function Declaration]

```
hb_s32 hb_mm_mc_flush(media_codec_context_t *context)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Refresh the I/O buffer and force the encoder/decoder to refresh the unprocessed I/O buffer. MediaCodec gets into **MEDIA_CODEC_STATE_FLUSHING** status after successful function call. After that, MediaCodec will enter **MEDIA_CODEC_STATE_STARTED** state again.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.14 hb_mm_mc_queue_input_buffer;

5.2.11 hb_mm_mc_release

[Function Declaration]

```
hb_s32 hb_mm_mc_release(media_codec_context_t *context)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Release all resources inside MediaCodec. Before calling this function, users need to call hb_mm_mc_stop to stop encoding and decoding. MediaCodec will enter **MEDIA_CODEC_STATE_UNINITIALIZED** state after the operation is successful.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.3 hb_mm_mc_initialize;

5.2.12 hb_mm_mc_get_state

[Function Declaration]

```
hb_s32 hb_mm_mc_get_state(media_codec_context_t *context, media_codec_state_t *state)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] media_codec_state_t *state: MediaCodec current status

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get MediaCodec current status.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.3 hb_mm_mc_initialize;

5.2.13 hb_mm_mc_get_status

[Function Declaration]

```
hb_s32 hb_mm_mc_get_status(media_codec_context_t *context, mc_inter_status_t *status)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_inter_status_t *status: MediaCodec current internal status

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get MediaCodec current internal status.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.53 hb_mm_mc_get_fd;

5.2.14 hb_mm_mc_queue_input_buffer

[Function Declaration]

```
hb_s32 hb_mm_mc_queue_input_buffer(media_codec_context_t *context, const  
media_codec_buffer_t *buffer, hb_s32 timeout)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] media_codec_buffer_t *buffer: Buffer information inputted
- [IN] hb_s32 timeout: Timeout period

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_INVALID_BUFFER: Invalid buffer
- HB_MEDIA_ERR_WAIT_TIMEOUT: The wait timed out

[Function Description]

Fill the MediaCodec with the buffer that needs to be processed.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
#include "hb_media_codec.h"  
#include "hb_media_error.h"  
  
typedef struct MediaCodecTestContext {  
    media_codec_context_t *context;  
    char *inputFileName;  
    char *outputFileName;  
    int32_t duration; // s  
} MediaCodecTestContext;  
  
Uint64 osal_gettime(void)  
{  
    struct timespec tp;
```

```

clock_gettime(CLOCK_MONOTONIC, &tp);

return ((UInt64)tp.tv_sec*1000 + tp.tv_nsec/1000000);
}

static void do_sync_encoding(void *arg) {
    hb_s32 ret = 0;
    FILE *inFile;
    FILE *outFile;
    int noMoreInput = 0;
    int lastStream = 0;
    Uint64 lastTime = 0;
    Uint64 curTime = 0;
    int needFlush = 1;

    MediaCodecTestContext *ctx = (MediaCodecTestContext *)arg;
    media_codec_context_t *context = ctx->context;
    char *inputFileName = ctx->inputFileName;
    char *outputFileName = ctx->outputFileName;

    inFile = fopen(inputFileName, "rb");
    if (!inFile) {
        goto ERR;
    }
    outFile = fopen(outputFileName, "wb");
    if (!outFile) {
        goto ERR;
    }

    //get current time
    lastTime = osal_gettime();

    ret = hb_mm_mc_initialize(context);
    if (ret) {
        goto ERR;
    }

    ret = hb_mm_mc_configure(context);
    if (ret) {
        goto ERR;
    }

    mc_av_codec_startup_params_t startup_params;
    startup_params.video_enc_startup_params.receive_frame_number = 0;
    ret = hb_mm_mc_start(context, &startup_params);
}

```

```

if (ret) {
    goto ERR;
}

ret = hb_mm_mc_pause(context);
if (ret) {
    goto ERR;
}

do {
    if (!noMoreInput) {
        media_codec_buffer_t inputBuffer;
        memset(&inputBuffer, 0x00, sizeof(media_codec_buffer_t));
        ret = hb_mm_mc_dequeue_input_buffer(context, &inputBuffer, 100);
        if (!ret) {
            curTime = osal_gettime();
            if ((curTime - lastTime)/1000 < (uint32_t)ctx->duration) {
                ret = fread(inputBuffer.vframe_buf.vir_ptr[0], 1,
                            inputBuffer.vframe_buf.size, inFile);
                if (ret <= 0) {
                    if(fseek(inFile, 0, SEEK_SET)) {
                        printf("Failed to rewind input file\n");
                    } else {
                        ret = fread(inputBuffer.vframe_buf.vir_ptr[0], 1,
                                    inputBuffer.vframe_buf.size, inFile);
                        if (ret <= 0) {
                            printf("Failed to read input file\n");
                        }
                    }
                }
            }
        } else {
            printf("Time up(%d)\n",ctx->duration);
            ret = 0;
        }
        if (!ret) {
            printf("There is no more input data!\n");
            inputBuffer.vframe_buf.frame_end = TRUE;
            noMoreInput = 1;
        }
        ret = hb_mm_mc_queue_input_buffer(context, &inputBuffer, 100);
        if (ret) {
            printf("Queue input buffer fail.\n");
            break;
        }
    }
}

```

```

    } else {
        if (ret != (int32_t)HB_MEDIA_ERR_WAIT_TIMEOUT) {
            printf("Dequeue input buffer fail.\n");
            break;
        }
    }

}

if (!lastStream) {
    media_codec_buffer_t outputBuffer;
    media_codec_output_buffer_info_t info;
    memset(&outputBuffer, 0x00, sizeof(media_codec_buffer_t));
    memset(&info, 0x00, sizeof(media_codec_output_buffer_info_t));
    ret = hb_mm_mc_dequeue_output_buffer(context, &outputBuffer, &info, 3000);
    if (!ret && outFile) {
        fwrite(outputBuffer.vstream_buf.vir_ptr, outputBuffer.vstream_buf.size, 1, outFile);

        ret = hb_mm_mc_queue_output_buffer(context, &outputBuffer, 100);
        if (ret) {
            printf("Queue output buffer fail.\n");
            break;
        }
        if (outputBuffer.vstream_buf.stream_end) {
            printf("There is no more output data!\n");
            lastStream = 1;
            break;
        }
    } else {
        if (ret != (int32_t)HB_MEDIA_ERR_WAIT_TIMEOUT) {
            printf("Dequeue output buffer fail.\n");
            break;
        }
    }
}

if (needFlush) {
    ret = hb_mm_mc_flush(context);
    needFlush = 0;
    if (ret) {
        break;
    }
}
}while(TRUE);

hb_mm_mc_stop(context);

```

```
hb_mm_mc_release(context);
context = NULL;

ERR:
if (context && hb_mm_mc_get_state(context) != MEDIA_CODEC_STATE_UNINITIALIZED) {
    hb_mm_mc_stop(context);
    hb_mm_mc_release(context);
}

if (inFile)
    fclose(inFile);

if (outFile)
    fclose(outFile);
}

int main(int argc, char *argv[])
{
    hb_s32 ret = 0;
    char outputFileName[MAX_FILE_PATH] = "./tmp.yuv";
    char inputFileName[MAX_FILE_PATH] = "./output.stream";
    mc_video_codec_enc_params_t *params;
    media_codec_context_t context;

    memset(&context, 0x00, sizeof(media_codec_context_t));
    context.codec_id = MEDIA_CODEC_ID_H264;
    context.encoder = TRUE;
    params = &context.video_enc_params;
    params->width = 640;
    params->height = 480;
    params->pix_fmt = MC_PIXEL_FORMAT_YUV420P;
    params->frame_buf_count = 5;
    params->external_frame_buf = FALSE;
    params->bitstream_buf_count = 5;
    params->rc_params.mode = MC_AV_RC_MODE_H264CBR;
    ret = hb_mm_mc_get_rate_control_config(&context, &params->rc_params);
    if (ret) {
        return -1;
    }
    params->rc_params.h264_cbr_params.bit_rate = 5000;
    params->rc_params.h264_cbr_params.frame_rate = 30;
    params->rc_params.h264_cbr_params.intra_period = 30;
    params->gop_params.decoding_refresh_type = 2;
```

```
params->gop_params.gop_preset_idx = 2;  
params->rot_degree = MC_CCW_0;  
params->mir_direction = MC_DIRECTION_NONE;  
params->frame_cropping_flag = FALSE;  
  
MediaCodecTestContext ctx;  
memset(&ctx, 0x00, sizeof(ctx));  
ctx.context = &context;  
ctx.inputFileName = inputFileName;  
ctx.outputFileName = outputFileName;  
ctx.duration = 5;  
do_sync_encoding(&ctx);  
}
```

5.2.15 hb_mm_mc_dequeue_input_buffer

[Function Declaration]

```
hb_s32 hb_mm_mc_dequeue_input_buffer(media_codec_context_t *context, media_codec_buffer_t  
*buffer, hb_s32 timeout)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] hb_s32 timeout: Timeout period
- [OUT] media_codec_buffer_t *buffer: Buffer information inputted

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_INVALID_BUFFER: Invalid buffer
- HB_MEDIA_ERR_WAIT_TIMEOUT: The wait timed out

[Function Description]

Get the inputted buffer.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.14 hb_mm_mc_queue_input_buffer;

5.2.16 hb_mm_mc_queue_output_buffer

[Function Declaration]

```
hb_s32 hb_mm_mc_queue_output_buffer(media_codec_context_t *context, media_codec_buffer_t  
*buffer, hb_s32 timeout)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] media_codec_buffer_t *buffer: Buffer information outputted
- [IN] hb_s32 timeout: Timeout period

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_INVALID_BUFFER: Invalid buffer
- HB_MEDIA_ERR_WAIT_TIMEOUT: The wait timed out

[Function Description]

Return the processed output buffer to MediaCodec.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.14 hb_mm_mc_queue_input_buffer;

5.2.17 hb_mm_mc_dequeue_output_buffer

[Function Declaration]

```
hb_s32 hb_mm_mc_dequeue_output_buffer(media_codec_context_t *context, media_codec_buffer_t  
*buffer, media_codec_output_buffer_info_t *info, hb_s32 timeout)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] hb_s32 timeout: Timeout period

- [OUT] media_codec_buffer_t *buffer: Buffer information outputted
- [OUT] media_codec_output_buffer_info_t *info: Information of outputted data flow

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_INVALID_BUFFER: Invalid buffer
- HB_MEDIA_ERR_WAIT_TIMEOUT: The wait timed out

[Function Description]

Return the processed output buffer to MediaCode.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.14 hb_mm_mc_queue_input_buffer;

5.2.18 hb_mm_mc_get_longterm_ref_mode

[Function Declaration]

```
hb_s32 hb_mm_mc_get_longterm_ref_mode(media_codec_context_t *context,  
mc_video_longterm_ref_mode_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_video_longterm_ref_mode_t *params: Long-term reference frame mode parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get parameters of long-term reference frame mode, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
#include "hb_media_codec.h"
#include "hb_media_error.h"

typedef enum ENC_CONFIG_MESSAGE {
    ENC_CONFIG_NONE = (0 << 0),
    ENC_CONFIG_LONGTERM_REF = (1 << 0),
    ENC_CONFIG_INTRA_REFRESH = (1 << 1),
    ENC_CONFIG_RATE_CONTROL = (1 << 2),
    ENC_CONFIG_DEBLK_FILTER = (1 << 3),
    ENC_CONFIG_SAO = (1 << 4),
    ENC_CONFIG_ENTROPY = (1 << 5),
    ENC_CONFIG_VUI_TIMING = (1 << 6),
    ENC_CONFIG_SLICE = (1 << 7),
    ENC_CONFIG_REQUEST_IDR = (1 << 8),
    ENC_CONFIG_SKIP_PIC = (1 << 9),
    ENC_CONFIG_SMART_BG = (1 << 10),
    ENC_CONFIG_MONOCHROMA = (1 << 11),
    ENC_CONFIG_PRED_UNIT = (1 << 12),
    ENC_CONFIG_TRANSFORM = (1 << 13),
    ENC_CONFIG_ROI = (1 << 14),
    ENC_CONFIG_MODE_DECISION = (1 << 15),
    ENC_CONFIG_USER_DATA = (1 << 16),
    ENC_CONFIG_MJPEG = (1 << 17),
    ENC_CONFIG_JPEG = (1 << 18),
    ENC_CONFIG_CAMERA = (1 << 19),
    ENC_CONFIG_INSERT_USERDATA = (1 << 20),
    ENC_CONFIG_VUI = (1 << 21),
    ENC_CONFIG_3DNR = (1 << 22),
    ENC_CONFIG_REQUEST_IDR_HEADER = (1 << 23),
    ENC_CONFIG_ENABLE_IDR = (1 << 24),
    ENC_CONFIG_TOTAL = (1 << 25),
} ENC_CONFIG_MESSAGE;

typedef struct MediaCodecTestContext {
    media_codec_context_t *context;
    char *inputFileName;
    char *outputFileName;
    int32_t duration; // s
```

```

ENC_CONFIG_MESSAGE message;
mc_video_longterm_ref_mode_t ref_mode;
mc_rate_control_params_t rc_params;
mc_video_intra_refresh_params_t intra_refr;
mc_video_deblk_filter_params_t deblk_filter;
mc_h265_sao_params_t sao;
mc_h264_entropy_params_t entropy;
mc_video_vui_params_t vui;
mc_video_vui_timing_params_t vui_timing;
mc_video_slice_params_t slice;
mc_video_3dnr_enc_params_t noise_reduction;
mc_video_smart_bg_enc_params_t smart_bg;
mc_video_pred_unit_params_t pred_unit;
mc_video_transform_params_t transform;
mc_video_roi_params_t roi;
mc_video_mode_decision_params_t mode_decision;

} MediaCodecTestContext;

Uint64 osal_gettime(void)
{
    struct timespec tp;

    clock_gettime(CLOCK_MONOTONIC, &tp);

    return ((Uint64)tp.tv_sec*1000 + tp.tv_nsec/1000000);
}

uint8_t uuid[] =
    "dc45e9bd-e6d948b7-962cd820-d923eeef+HorizonAI";

static void set_message(MediaCodecTestContext *ctx) {
    int ret = 0;
    media_codec_context_t *context = ctx->context;

    mc_video_longterm_ref_mode_t *ref_mode = &ctx->ref_mode;
    hb_mm_mc_get_longterm_ref_mode(context, ref_mode);
    ref_mode->use_longterm = TRUE;
    ref_mode->longterm_pic_using_period = 20;
    ref_mode->longterm_pic_period = 30;
    //ctx->message = ENC_CONFIG_LONGTERM_REF;
    if (ctx->message & ENC_CONFIG_LONGTERM_REF) {
        ret = hb_mm_mc_set_longterm_ref_mode(context, &ctx->ref_mode);
    }
    if (ctx->message & ENC_CONFIG_INTRA_REFRESH) {
        hb_mm_mc_get_intra_refresh_config(context, &ctx->intra_refr)
        ret = hb_mm_mc_set_intra_refresh_config(context, &ctx->intra_refr);
    }
}

```

```

}

if (ctx->message & ENC_CONFIG_SAO) {
    hb_mm_mc_get_sao_config(context, &ctx->sao);
    ret = hb_mm_mc_set_sao_config(context, &ctx->sao);
}

if (ctx->message & ENC_CONFIG_ENTROPY) {
    hb_mm_mc_get_entropy_config(context, &ctx->entropy);
    ret = hb_mm_mc_set_entropy_config(context, &ctx->entropy);
}

if (ctx->message & ENC_CONFIG_VUI) {
    hb_mm_mc_get_vui_config(context, &ctx->vui);
    ret = hb_mm_mc_set_vui_config(context, &ctx->vui);
}

if (ctx->message & ENC_CONFIG_VUI_TIMING) {
    hb_mm_mc_get_vui_timing_config(context, &ctx->vui_timing);
    ret = hb_mm_mc_set_vui_timing_config(context, &ctx->vui_timing);
}

mc_rate_control_params_t *rc_params = &ctx->rc_params;
rc_params->mode = context->video_enc_params.rc_params.mode;
hb_mm_mc_get_rate_control_config(context, rc_params);
switch (rc_params->mode) {
case MC_AV_RC_MODE_H264CBR:
    rc_params->h264_cbr_params.bit_rate = 5000;
    rc_params->h264_cbr_params.intra_period = 60;
    break;
case MC_AV_RC_MODE_H264VBR:
    rc_params->h264_vbr_params.intra_qp = 20;
    rc_params->h264_vbr_params.intra_period = 30;
    break;
case MC_AV_RC_MODE_H264AVBR:
    rc_params->h264_avbr_params.intra_period = 15;
    rc_params->h264_avbr_params.intra_qp = 25;
    rc_params->h264_avbr_params.bit_rate = 2000;
    rc_params->h264_avbr_params.vbv_buffer_size = 3000;
    rc_params->h264_avbr_params.min_qp_I = 15;
    rc_params->h264_avbr_params.max_qp_I = 50;
    rc_params->h264_avbr_params.min_qp_P = 15;
    rc_params->h264_avbr_params.max_qp_P = 45;
    rc_params->h264_avbr_params.min_qp_B = 15;
    rc_params->h264_avbr_params.max_qp_B = 48;
    rc_params->h264_avbr_params.hvs_qp_enable = 0;
}

```

```
rc_params->h264_avbr_params.hvs_qp_scale = 2;
rc_params->h264_avbr_params.max_delta_qp = 5;
rc_params->h264_avbr_params.qp_map_enable = 0;
break;

case MC_AV_RC_MODE_H264FIXQP:
    rc_params->h264_fixqp_params.force_qp_I = 23;
    rc_params->h264_fixqp_params.force_qp_P = 23;
    rc_params->h264_fixqp_params.force_qp_B = 23;
    rc_params->h264_fixqp_params.intra_period = 23;
    break;

case MC_AV_RC_MODE_H264QPMAP:
    break;

case MC_AV_RC_MODE_H265CBR:
    rc_params->h265_cbr_params.bit_rate = 5000;
    rc_params->h265_cbr_params.intra_period = 60;
    break;

case MC_AV_RC_MODE_H265VBR:
    rc_params->h265_vbr_params.intra_qp = 20;
    rc_params->h265_vbr_params.intra_period = 30;
    break;

case MC_AV_RC_MODE_H265AVBR:
    rc_params->h265_avbr_params.intra_period = 15;
    rc_params->h265_avbr_params.intra_qp = 25;
    rc_params->h265_avbr_params.bit_rate = 2000;
    rc_params->h265_avbr_params.vbv_buffer_size = 3000;
    rc_params->h265_avbr_params.min_qp_I = 15;
    rc_params->h265_avbr_params.max_qp_I = 50;
    rc_params->h265_avbr_params.min_qp_P = 15;
    rc_params->h265_avbr_params.max_qp_P = 45;
    rc_params->h265_avbr_params.min_qp_B = 15;
    rc_params->h265_avbr_params.max_qp_B = 48;
    rc_params->h265_avbr_params.hvs_qp_enable = 0;
    rc_params->h265_avbr_params.hvs_qp_scale = 2;
    rc_params->h265_avbr_params.max_delta_qp = 5;
    rc_params->h265_avbr_params.qp_map_enable = 0;
    break;

case MC_AV_RC_MODE_H265FIXQP:
    rc_params->h265_fixqp_params.force_qp_I = 23;
    rc_params->h265_fixqp_params.force_qp_P = 23;
    rc_params->h265_fixqp_params.force_qp_B = 23;
    rc_params->h265_fixqp_params.intra_period = 23;
    break;

case MC_AV_RC_MODE_H265QPMAP:
    break;
```

```

default:
    break;
}

//ctx->message = ENC_CONFIG_RATE_CONTROL;
if (ctx->message & ENC_CONFIG_RATE_CONTROL) {
    ret = hb_mm_mc_set_rate_control_config(context, &ctx->rc_params);
}

mc_video_deblk_filter_params_t *deblk_filter = &ctx->deblk_filter;
hb_mm_mc_get_deblk_filter_config(context, deblk_filter);
if (context->codec_id == MEDIA_CODEC_ID_H264) {
    deblk_filter->h264_deblk.disable_deblocking_filter_idc = 2;
    deblk_filter->h264_deblk.slice_alpha_c0_offset_div2 = 6;
    deblk_filter->h264_deblk.slice_beta_offset_div2 = 6;
} else {
    deblk_filter->h265_deblk.slice_deblocking_filter_disabled_flag = 1;
    deblk_filter->h265_deblk.slice_beta_offset_div2 = 6;
    deblk_filter->h265_deblk.slice_tc_offset_div2 = 6;
    deblk_filter->h265_deblk.slice_loop_filter_across_slices_enabled_flag = 1;
}
//ctx->message = ENC_CONFIG_DEBLK_FILTER;
if (ctx->message & ENC_CONFIG_DEBLK_FILTER) {
    ret = hb_mm_mc_set_deblk_filter_config(context, &ctx->deblk_filter);
}

if (context->codec_id == MEDIA_CODEC_ID_H264) {
    mc_h264_entropy_params_t *entropy = &ctx->entropy;
    hb_mm_mc_get_entropy_config(context, entropy);
    entropy->entropy_coding_mode = 0;
    ctx->message = ENC_CONFIG_ENTROPY;
    if (ctx->message & ENC_CONFIG_ENTROPY) {
        ret = hb_mm_mc_set_entropy_config(context, &ctx->entropy);
    }
}

//ctx->message = ENC_CONFIG_SKIP_PIC;
if (ctx->message & ENC_CONFIG_SKIP_PIC) {
    ret = hb_mm_mc_skip_pic(context, 0), (int32_t)0;
}

//ctx->message = ENC_CONFIG_REQUEST_IDR;
if (ctx->message & ENC_CONFIG_REQUEST_IDR) {
    ret = hb_mm_mc_request_idr_frame(context);
}

```

```
mc_video_slice_params_t *slice = &ctx->slice;
hb_mm_mc_get_slice_config(context, slice);
if (context->codec_id == MEDIA_CODEC_ID_H264) {
    slice->h264_slice.h264_slice_mode = 0;
    slice->h264_slice.h264_slice_arg = 60;
} else {
    slice->h265_slice.h265_dependent_slice_mode = 0;
    slice->h265_slice.h265_dependent_slice_arg = 80;
    slice->h265_slice.h265_independent_slice_mode = 1;
    slice->h265_slice.h265_independent_slice_arg = 100;
}
//ctx->message = ENC_CONFIG_SLICE;
if (ctx->message & ENC_CONFIG_SLICE) {
    ret = hb_mm_mc_set_slice_config(context, &ctx->slice);
}

mc_video_smart_bg_enc_params_t *smart_bg = &ctx->smart_bg;
hb_mm_mc_get_smart_bg_enc_config(context, smart_bg);
smart_bg->bg_detect_enable = 0;
smart_bg->bg_threshold_diff = 8;
smart_bg->bg_threshold_mean_diff = 1;
smart_bg->bg_lambda_qp = 32;
smart_bg->bg_delta_qp = 3;
smart_bg->s2fme_disable = 0;
//ctx->message = ENC_CONFIG_SMART_BG;
if (ctx->message & ENC_CONFIG_SMART_BG) {
    ret = hb_mm_mc_set_smart_bg_enc_config(context, &ctx->smart_bg);
}

mc_video_pred_unit_params_t *pred_unit = &ctx->pred_unit;
hb_mm_mc_get_pred_unit_config(context, pred_unit);
if (context->codec_id == MEDIA_CODEC_ID_H264) {
    pred_unit->h264_intra_pred.constrained_intra_pred_flag = 1;
} else {
    pred_unit->h265_pred_unit.intra_nxn_enable = 1;
    pred_unit->h265_pred_unit.constrained_intra_pred_flag = 1;
    pred_unit->h265_pred_unit.strong_intra_smoothing_enabled_flag = 0;
    pred_unit->h265_pred_unit.max_num_merge = 2;
}
//ctx->message = ENC_CONFIG_PRED_UNIT;
if (ctx->message & ENC_CONFIG_PRED_UNIT) {
    ret = hb_mm_mc_set_pred_unit_config(context, &ctx->pred_unit);
}
```

```
mc_video_transform_params_t *transform = &ctx->transform;
hb_mm_mc_get_transform_config(context, transform);
if (context->codec_id == MEDIA_CODEC_ID_H264) {
    transform->h264_transform.transform_8x8_enable = 1;
    transform->h264_transform.chroma_cb_qp_offset = 4;
    transform->h264_transform.chroma_cr_qp_offset = 3;
    transform->h264_transform.user_scaling_list_enable = 0;
} else {
    transform->h265_transform.chroma_cb_qp_offset = 6;
    transform->h265_transform.chroma_cr_qp_offset = 5;
    transform->h265_transform.user_scaling_list_enable = 0;
}
//ctx->message = ENC_CONFIG_TRANSFORM;
if (ctx->message & ENC_CONFIG_TRANSFORM) {
    ret = hb_mm_mc_set_transform_config(context, &ctx->transform);
}

mc_video_roi_params_t *roi = &ctx->roi;
hb_mm_mc_get_roi_config(context, roi);
roi->roi_enable = 0;
//ctx->message = ENC_CONFIG_ROI;
if (ctx->message & ENC_CONFIG_ROI) {
    ret = hb_mm_mc_set_roi_config(context, &ctx->roi);
}

mc_video_mode_decision_params_t *mode_decision = &ctx->mode_decision;
hb_mm_mc_get_mode_decision_config(context, mode_decision);
mode_decision->mode_decision_enable = FALSE;
mode_decision->pu04_delta_rate = 76;
mode_decision->pu08_delta_rate = 80;
mode_decision->pu16_delta_rate = 86;
mode_decision->pu32_delta_rate = 87;
mode_decision->pu04_intra_planar_delta_rate = 0;
mode_decision->pu04_intra_dc_delta_rate = 0;
mode_decision->pu04_intra_angle_delta_rate = 0;
mode_decision->pu08_intra_planar_delta_rate = 0;
mode_decision->pu08_intra_dc_delta_rate = 0;
mode_decision->pu08_intra_angle_delta_rate = 0;
mode_decision->pu16_intra_planar_delta_rate = 0;
mode_decision->pu16_intra_dc_delta_rate = 0;
mode_decision->pu16_intra_angle_delta_rate = 0;
mode_decision->pu32_intra_planar_delta_rate = 0;
mode_decision->pu32_intra_dc_delta_rate = 0;
```

```

mode_decision->pu32_intra_angle_delta_rate = 0;
mode_decision->cu08_intra_delta_rate = 0;
mode_decision->cu08_inter_delta_rate = 0;
mode_decision->cu08_merge_delta_rate = 0;
mode_decision->cu16_intra_delta_rate = 0;
mode_decision->cu16_inter_delta_rate = 0;
mode_decision->cu16_merge_delta_rate = 0;
mode_decision->cu32_intra_delta_rate = 0;
mode_decision->cu32_inter_delta_rate = 0;
mode_decision->cu32_merge_delta_rate = 0;
//ctx->message = ENC_CONFIG_MODE_DECISION;
if (ctx->message & ENC_CONFIG_MODE_DECISION) {
    ret = hb_mm_mc_set_mode_decision_config(context, &ctx->mode_decision);
}
if (ctx->message & ENC_CONFIG_INSERT_USERDATA) {
    hb_u32 length = sizeof(uuid)/sizeof(uuid[0]);
    ret = hb_mm_mc_insert_user_data(context, uuid, length);
}
if (ctx->message & ENC_CONFIG_3DNR) {
    hb_mm_mc_get_3dnr_enc_config(context, &ctx->noise_reduction);
    ret = hb_mm_mc_set_3dnr_enc_config(context, &ctx->noise_reduction);
}
if (ctx->message & ENC_CONFIG_ENABLE_IDR) {
    // disable idr frame first
    if (ctx->enable_idr_num) {
        ret = hb_mm_mc_enable_idr_frame(context, 0);
    }
}
if (ctx->message & ENC_CONFIG_REQUEST_IDR_HEADER) {
    ret = hb_mm_mc_request_idr_header(context, ctx->force_idr_header);
}
}

static void do_sync_encoding(void *arg) {
hb_s32 ret = 0;
FILE *inFile;
FILE *outFile;
int noMoreInput = 0;
int lastStream = 0;
Uint64 lastTime = 0;
Uint64 curTime = 0;
int needFlush = 1;
MediaCodecTestContext *ctx = (MediaCodecTestContext *)arg;

```

```
media_codec_context_t *context = ctx->context;
char *inputFileName = ctx->inputFileName;
char *outputFileName = ctx->outputFileName;

inFile = fopen(inputFileName, "rb");
if (!inFile) {
    goto ERR;
}
outFile = fopen(outputFileName, "wb");
if (!outFile) {
    goto ERR;
}

//get current time
lastTime = osal_gettime();

ret = hb_mm_mc_initialize(context);
if (ret) {
    goto ERR;
}

ret = hb_mm_mc_configure(context);
if (ret) {
    goto ERR;
}

mc_av_codec_startup_params_t startup_params;
startup_params.video_enc_startup_params.receive_frame_number = 0;
ret = hb_mm_mc_start(context, &startup_params);
if (ret) {
    goto ERR;
}

ret = hb_mm_mc_pause(context);
if (ret) {
    goto ERR;
}

do {
    set_message(ctx);
    if (!noMoreInput) {
        media_codec_buffer_t inputBuffer;
        memset(&inputBuffer, 0x00, sizeof(media_codec_buffer_t));
        ret = hb_mm_mc_dequeue_input_buffer(context, &inputBuffer, 100);
    }
}
```

```

if (!ret) {
    curTime = osal_gettime();
    if ((curTime - lastTime)/1000 < (uint32_t)ctx->duration) {
        ret = fread(inputBuffer.vframe_buf.vir_ptr[0], 1,
                    inputBuffer.vframe_buf.size, inFile);
        if (ret <= 0) {
            if(fseek(inFile, 0, SEEK_SET)) {
                printf("Failed to rewind input file\n");
            } else {
                ret = fread(inputBuffer.vframe_buf.vir_ptr[0], 1,
                            inputBuffer.vframe_buf.size, inFile);
                if (ret <= 0) {
                    printf("Failed to read input file\n");
                }
            }
        }
    } else {
        printf("Time up(%d)\n",ctx->duration);
        ret = 0;
    }
    if (!ret) {
        printf("There is no more input data!\n");
        inputBuffer.vframe_buf.frame_end = TRUE;
        noMoreInput = 1;
    }
    ret = hb_mm_mc_queue_input_buffer(context, &inputBuffer, 100);
    if (ret) {
        printf("Queue input buffer fail.\n");
        break;
    }
} else {
    if (ret != (int32_t)HB_MEDIA_ERR_WAIT_TIMEOUT) {
        printf("Dequeue input buffer fail.\n");
        break;
    }
}
}

if (!lastStream) {
    media_codec_buffer_t outputBuffer;
    media_codec_output_buffer_info_t info;
    memset(&outputBuffer, 0x00, sizeof(media_codec_buffer_t));
    memset(&info, 0x00, sizeof(media_codec_output_buffer_info_t));
    ret = hb_mm_mc_dequeue_output_buffer(context, &outputBuffer, &info, 3000);
}

```

```

if (!ret && outFile) {
    fwrite(outputBuffer.vstream_buf.vir_ptr, outputBuffer.vstream_buf.size, 1, outFile);

    ret = hb_mm_mc_queue_output_buffer(context, &outputBuffer, 100);
    if (ret) {
        printf("Queue output buffer fail.\n");
        break;
    }
    if (outputBuffer.vstream_buf.stream_end) {
        printf("There is no more output data!\n");
        lastStream = 1;
        break;
    }
} else {
    if (ret != (int32_t)HB_MEDIA_ERR_WAIT_TIMEOUT) {
        printf("Dequeue output buffer fail.\n");
        break;
    }
}
}

if (needFlush) {
    ret = hb_mm_mc_flush(context);
    needFlush = 0;
    if (ret) {
        break;
    }
}
}

}while(TRUE);

hb_mm_mc_stop(context);

hb_mm_mc_release(context);
context = NULL;

ERR:
if (context && hb_mm_mc_get_state(context) != MEDIA_CODEC_STATE_UNINITIALIZED) {
    hb_mm_mc_stop(context);
    hb_mm_mc_release(context);
}

if (inFile)
    fclose(inFile);

if (outFile)

```

```
    fclose(outFile);
}

int main(int argc, char *argv[])
{
    hb_s32 ret = 0;
    char outputFileName[MAX_FILE_PATH] = "./tmp.yuv";
    char inputFileName[MAX_FILE_PATH] = "./output.stream";
    mc_video_codec_enc_params_t *params;
    media_codec_context_t context;

    memset(&context, 0x00, sizeof(media_codec_context_t));
    context.codec_id = MEDIA_CODEC_ID_H264;
    context.encoder = TRUE;
    params = &context.video_enc_params;
    params->width = 640;
    params->height = 480;
    params->pix_fmt = MC_PIXEL_FORMAT_YUV420P;
    params->frame_buf_count = 5;
    params->external_frame_buf = FALSE;
    params->bitstream_buf_count = 5;
    params->rc_params.mode = MC_AV_RC_MODE_H264CBR;
    ret = hb_mm_mc_get_rate_control_config(&context, &params->rc_params);
    if (ret) {
        return -1;
    }
    params->rc_params.h264_cbr_params.bit_rate = 5000;
    params->rc_params.h264_cbr_params.frame_rate = 30;
    params->rc_params.h264_cbr_params.intra_period = 30;
    params->gop_params.decoding_refresh_type = 2;
    params->gop_params.gop_preset_idx = 2;
    params->rot_degree = MC_CCW_0;
    params->mir_direction = MC_DIRECTION_NONE;
    params->frame_cropping_flag = FALSE;

    MediaCodecTestContext ctx;
    memset(&ctx, 0x00, sizeof(ctx));
    ctx.context = &context;
    ctx.inputFileName = inputFileName;
    ctx.outputFileName = outputFileName;
    ctx.duration = 5;
    do_sync_encoding(&ctx);
}
```

5.2.19 hb_mm_mc_set_longterm_ref_mode

[Function Declaration]

```
hb_s32 hb_mm_mc_set_longterm_ref_mode(media_codec_context_t *context, const  
mc_video_longterm_ref_mode_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_video_longterm_ref_mode_t *params: Long-term reference frame mode parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the parameters of long-term reference frame mode, which are dynamic parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.20 hb_mm_mc_get_intra_refresh_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_intra_refresh_config(media_codec_context_t *context,  
mc_video_intra_refresh_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_video_intra_refresh_params_t *params: Intra-frame refresh parameters

[Return Value]

- 0: Success

- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the intra-frame refresh parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.21 hb_mm_mc_set_intra_refresh_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_intra_refresh_config(media_codec_context_t *context, const  
mc_video_intra_refresh_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_video_intra_refresh_params_t *params: Intra-frame refresh parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the parameters of intra-frame refresh mode, which are static parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.22 hb_mm_mc_get_rate_control_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_rate_control_config(media_codec_context_t *context,  
mc_rate_control_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_rate_control_params_t *params: Rate control parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the rate control parameters, which are dynamic parameters, applicable to H264/H265/MJPEG.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.23 hb_mm_mc_set_rate_control_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_rate_control_config(media_codec_context_t *context, const  
mc_rate_control_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_rate_control_params_t *params: Rate control parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed

- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the rate control parameters, which are dynamic parameters, applicable to H264/H265/MJPEG.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.24 hb_mm_mc_get_deblk_filter_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_deblk_filter_config(media_codec_context_t *context,  
mc_video_deblk_filter_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_video_deblk_filter_params_t * params: Deblocking filter parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the deblocking filter parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.25 hb_mm_mc_set_deblk_filter_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_deblk_filter_config(media_codec_context_t *context, const  
mc_video_deblk_filter_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_video_deblk_filter_params_t *params: Deblocking filter parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the deblocking filter parameters, which are dynamic parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.26 hb_mm_mc_get_sao_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_sao_config(media_codec_context_t *context, mc_h265_sao_params_t  
*params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_h265_sao_params_t *params: SAO parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get SAO parameters, applicable to H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.27 hb_mm_mc_set_sao_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_sao_config(media_codec_context_t *context, const mc_h265_sao_params_t  
*params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_h265_sao_params_t *params: SAO parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the SAO parameters, which are static parameters, applicable to H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.28 hb_mm_mc_get_entropy_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_entropy_config(media_codec_context_t *context,  
mc_h264_entropy_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_h264_entropy_params_t *params: Entropy encoding parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the entropy encoding parameters, applicable to H264.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.29 hb_mm_mc_set_entropy_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_entropy_config(media_codec_context_t *context, const  
mc_h264_entropy_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_h264_entropy_params_t *params: Entropy encoding parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the entropy encoding parameters, which are dynamic parameters, applicable to H264.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.30 hb_mm_mc_get_vui_timing_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_vui_timing_config(media_codec_context_t *context,  
mc_video_vui_timing_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_video_vui_timing_params_t *params: VUI Timing parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the VUI Timing parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.31 hb_mm_mc_set_vui_timing_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_vui_timing_config(media_codec_context_t *context, const  
mc_video_vui_timing_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_video_vui_timing_params_t *params: VUI Timing parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the VUI Timing parameters, which are static parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.32 hb_mm_mc_get_slice_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_slice_config(media_codec_context_t *context, mc_video_slice_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_video_slice_params_t *params: Slice encoding parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the slice encoding parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.33 hb_mm_mc_set_slice_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_slice_config(media_codec_context_t *context, const mc_video_slice_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_video_slice_params_t *params: Slice encoding parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the slice encoding parameters, which are dynamic parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.34 hb_mm_mc_insert_user_data

[Function Declaration]

```
hb_s32 hb_mm_mc_insert_user_data(media_codec_context_t * context, hb_u8 *data, hb_u32 length)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] hb_u8 *data: User data

- [IN] hb_u32 length: Length of user data

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Insert the user data into the encoding stream, which is a dynamic parameter, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.35 hb_mm_mc_request_idr_frame

[Function Declaration]

```
hb_s32 hb_mm_mc_request_idr_frame(media_codec_context_t *context)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Request the IDR frame, where the interface can be set dynamically, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.36 hb_mm_mc_skip_pic

[Function Declaration]

```
hb_s32 hb_mm_mc_skip_pic(media_codec_context_t *context, hb_s32 src_idx)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] hb_s32 src_idx: Source buffer index value

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Enable the skip mode coding of the specified image, where the interface can be set dynamically, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.37 hb_mm_mc_get_smart_bg_enc_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_smart_bg_enc_config(media_codec_context_t *context,  
mc_video_smart_bg_enc_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_video_smart_bg_enc_params_t *params: Intelligent background coding mode parameters

[Return Value]

- 0: Success

- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the parameters of intelligent background coding mode, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.38 hb_mm_mc_set_smart_bg_enc_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_smart_bg_enc_config(media_codec_context_t *context, const  
mc_video_smart_bg_enc_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] const mc_video_smart_bg_enc_params_t *params: Intelligent background coding mode parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the parameters of intelligent background coding mode, which are dynamic parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.39 hb_mm_mc_get_pred_unit_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_pred_unit_config(media_codec_context_t *context,  
mc_video_pred_unit_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_video_pred_unit_params_t *params: Prediction unit parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the prediction unit parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.40 hb_mm_mc_set_pred_unit_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_pred_unit_config(media_codec_context_t *context, const  
mc_video_pred_unit_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_video_smart_bg_enc_params_t *params: Prediction unit parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error

- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the prediction unit parameters, which are dynamic parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.41 hb_mm_mc_get_transform_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_transform_config(media_codec_context_t *context,  
mc_video_transform_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_video_transform_params_t *params: Transform parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the Transform parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.42 hb_mm_mc_set_transform_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_transform_config(media_codec_context_t *context, const  
mc_video_transform_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_video_transform_params_t *params: Transform parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the Transform parameters, which are dynamic parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.43 hb_mm_mc_get_roi_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_roi_config(media_codec_context_t *context, mc_video_roi_params_t  
*params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_video_roi_params_t *params: ROI coding parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error

- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the ROI coding parameter, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.44 hb_mm_mc_set_roi_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_roi_config(media_codec_context_t *context, const mc_video_roi_params_t  
*params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_video_roi_params_t *params: ROI coding parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the ROI coding parameter, which are dynamic parameters, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.45 hb_mm_mc_get_mode_decision_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_mode_decision_config(media_codec_context_t *context,  
mc_video_mode_decision_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_video_mode_decision_params_t *params: Mode decision parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the mode decision parameters, applicable to H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.46 hb_mm_mc_set_mode_decision_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_mode_decision_config(media_codec_context_t *context, const  
mc_video_mode_decision_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_video_mode_decision_params_t *params: Mode decision parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed

- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the mode decision parameters, which are dynamic parameters, applicable to H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.47 hb_mm_mc_get_user_data

[Function Declaration]

```
hb_s32 hb_mm_mc_get_user_data(media_codec_context_t *context, mc_user_data_buffer_t  
*params, hb_s32 timeout)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_user_data_buffer_t *params: User data
- [IN] timeout: Timeout period

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the user data from the decoding stream, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.48 hb_mm_mc_release_user_data

[Function Declaration]

```
hb_s32 hb_mm_mc_release_user_data(media_codec_context_t * context, const  
mc_user_data_buffer_t * params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_user_data_buffer_t * params: User data

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Release the user data from the decoding stream, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.49 hb_mm_mc_get_mjpeg_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_mjpeg_config(media_codec_context_t *context, mc_mjpeg_enc_params_t  
*params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_mjpeg_enc_params_t *params: MJPE encoding parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error

- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the MJPEG encoding parameters, applicable to MJPEG.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.50 hb_mm_mc_set_mjpeg_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_mjpeg_config(media_codec_context_t *context, const  
mc_mjpeg_enc_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_mjpeg_enc_params_t *params: MJPEG encoding parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the MJPEG encoding parameters, which are dynamic parameters, applicable to MJPEG.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.51 hb_mm_mc_get_jpeg_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_jpeg_config(media_codec_context_t *context, mc_jpeg_enc_params_t  
*params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_jpeg_enc_params_t *params: JPEG encoding parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the JPEG encoding parameters, applicable to JPEG.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.52 hb_mm_mc_set_jpeg_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_jpeg_config(media_codec_context_t *context, const mc_jpeg_enc_params_t  
*params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_jpeg_enc_params_t *params: JPEG encoding parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed

- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the JPEG encoding parameters, which are dynamic parameters, applicable to JPEG.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.53 hb_mm_mc_get_fd

[Function Declaration]

```
hb_s32 hb_mm_mc_get_fd(media_codec_context_t * context, hb_s32 *fd)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] hb_s32 *fd: Device node fd

[Return Value]

- 0: Success
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the device node fd, which can be used for the selection and listening to the encoding and decoding results.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
#include "hb_media_codec.h"
#include "hb_media_error.h"

typedef struct MediaCodecTestContext {
    media_codec_context_t *context;
```

```

char *inputFileName;
char *outputFileName;
int abnormal;
int32_t duration; // s
} MediaCodecTestContext;

Uint64 osal_gettime(void)
{
    struct timespec tp;

    clock_gettime(CLOCK_MONOTONIC, &tp);

    return ((Uint64)tp.tv_sec*1000 + tp.tv_nsec/1000000);
}

static void do_poll_encoding_select(void *arg) {
    hb_s32 ret = 0;
    int pollFd = -1;
    FILE *outFile;
    int lastStream = 0;
    fd_set readFds;
    MediaCodecTestContext *ctx = (MediaCodecTestContext *)arg;
    media_codec_context_t *context = ctx->context;
    char *outputFileName = ctx->outputFileName;

    outFile = fopen(outputFileName, "wb");
    if (!outFile) {
        goto ERR;
    }

    ret = hb_mm_mc_get_fd(context, &pollFd);
    if (ret) {
        goto ERR;
    }

    do {
        FD_ZERO(&readFds);
        FD_SET(pollFd, &readFds);
        ret = select(pollFd+1, &readFds, NULL, NULL, NULL);
        if (ret < 0) {
            printf("Failed to select fd = %d.(err %s)\n", pollFd, strerror(errno));
            ctx->abnormal = TRUE;
            break;
        } else if (ret == 0) {

```

```

printf("Time out to select fd = %d.\n", pollFd);
ctx->abnormal = TRUE;
break;
} else {
    if (FD_ISSET(pollFd, &readFds)) {
        media_codec_buffer_t outputBuffer;
        media_codec_output_buffer_info_t info;
        memset(&outputBuffer, 0x00, sizeof(media_codec_buffer_t));
        memset(&info, 0x00, sizeof(media_codec_output_buffer_info_t));
        ret = hb_mm_mc_dequeue_output_buffer(context, &outputBuffer, &info, 100);
        if (!ret && outFile) {
            fwrite(outputBuffer.vstream_buf.vir_ptr, outputBuffer.vstream_buf.size, 1, outFile);
            ret = hb_mm_mc_queue_output_buffer(context, &outputBuffer, 100);

            if (outputBuffer.vstream_buf.stream_end) {
                printf("%There is no more output data!\n");
                lastStream = 1;
                break;
            }
            if (ret) {
                ctx->abnormal = TRUE;
                break;
            }
        } else {
            if (ret != (int32_t)HB_MEDIA_ERR_WAIT_TIMEOUT) {
                printf("Dequeue output buffer fail.\n");
                break;
            }
        }
    }
}
} while (!lastStream && !ctx->abnormal);

ERR:
if (pollFd) {
    hb_mm_mc_close_fd(context, pollFd)
}
if (outFile)
    fclose(outFile);
}

static void do_poll_encoding(void *arg) {
    pthread_t thread_id;
    void* retVal;
}

```

```

hb_s32 ret = 0;
FILE *inFile;
int noMoreInput = 0;
MediaCodecTestContext *ctx = (MediaCodecTestContext *)arg;
media_codec_context_t *context = ctx->context;

char *inputFileName = ctx->inputFileName;
char *outputFileName = ctx->outputFileName;

inFile = fopen(inputFileName, "rb");
if (!inFile) {
    goto ERR;
}

ret = hb_mm_mc_initialize(context);
if (ret) {
    goto ERR;
}

ret = hb_mm_mc_configure(context);
if (ret) {
    goto ERR;
}

mc_av_codec_startup_params_t startup_params;
startup_params.video_enc_startup_params.receive_frame_number = 0;
ret = hb_mm_mc_start(context, &startup_params);
if (ret) {
    goto ERR;
}

pthread_create(&thread_id, NULL, (void* (*)(void*))do_poll_encoding_select, ctx);

do {
    media_codec_buffer_t inputBuffer;
    memset(&inputBuffer, 0x00, sizeof(media_codec_buffer_t));
    ret = hb_mm_mc_dequeue_input_buffer(context, &inputBuffer, 100);
    if (!ret) {
        ret = fread(inputBuffer.vframe_buf.vir_ptr[0], 1,
                    inputBuffer.vframe_buf.size, inFile);
        if (!ret) {
            printf("There is no more input data!\n");
            inputBuffer.vframe_buf.frame_end = TRUE;
            noMoreInput = 1;
        }
    }
}

```

```

        }

        ret = hb_mm_mc_queue_input_buffer(context, &inputBuffer, 100);
        if (ret) {
            printf("Queue input buffer fail.\n");
            break;
        }
    } else {
        if (ret != (int32_t)HB_MEDIA_ERR_WAIT_TIMEOUT) {
            printf("Dequeue input buffer fail.\n");
            break;
        }
    }
}

}while(!noMoreInput && !ctx->abnormal);

pthread_join(thread_id, &retVal);

hb_mm_mc_stop(context);

hb_mm_mc_release(context);
context = NULL;

ERR:
if (context && hb_mm_mc_get_state(context) != MEDIA_CODEC_STATE_UNINITIALIZED) {
    hb_mm_mc_stop(context);
    hb_mm_mc_release(context);
}

if (inFile)
    fclose(inFile);
}

int main(int argc, char *argv[])
{
    hb_s32 ret = 0;
    char outputFileName[MAX_FILE_PATH] = "./tmp.yuv";
    char inputFileName[MAX_FILE_PATH] = "./output.stream";
    mc_video_codec_enc_params_t *params;
    media_codec_context_t context;

    memset(&context, 0x00, sizeof(media_codec_context_t));
    context.codec_id = MEDIA_CODEC_ID_H264;
    context.encoder = TRUE;
    params = &context.video_enc_params;
    params->width = 640;
}

```

```

params->height = 480;
params->pix_fmt = MC_PIXEL_FORMAT_YUV420P;
params->frame_buf_count = 5;
params->external_frame_buf = FALSE;
params->bitstream_buf_count = 5;
params->rc_params.mode = MC_AV_RC_MODE_H264CBR;
ret = hb_mm_mc_get_rate_control_config(&context, &params->rc_params);
if (ret) {
    return -1;
}
params->rc_params.h264_cbr_params.bit_rate = 5000;
params->rc_params.h264_cbr_params.frame_rate = 30;
params->rc_params.h264_cbr_params.intra_period = 30;
params->gop_params.decoding_refresh_type = 2;
params->gop_params.gop_preset_idx = 2;
params->rot_degree = MC_CCW_0;
params->mir_direction = MC_DIRECTION_NONE;
params->frame_cropping_flag = FALSE;

MediaCodecTestContext ctx;
memset(&ctx, 0x00, sizeof(ctx));
ctx.context = &context;
ctx.inputFileName = inputFileName;
ctx.outputFileName = outputFileName;
ctx.duration = 5;
do_poll_encoding(&ctx);
}

```

5.2.54 hb_mm_mc_close_fd

[Function Declaration]

```
hb_s32 hb_mm_mc_close_fd(media_codec_context_t * context, hb_s32 fd)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] hb_s32 fd: Device node fd

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Close the device node.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.53 hb_mm_mc_get_fd;

5.2.55 hb_mm_mc_set_camera

[Function Declaration]

```
hb_s32 hb_mm_mc_set_camera(media_codec_context_t *context, hb_s32 pipeline, hb_s32 channel_port_id)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] hb_s32 pipeline: pipeline
- [IN] hb_s32 channel_port_id: Channel port number

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the camera information of the VIO, which is a static parameter, applicable to H264/H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.56 hb_mm_mc_get_vui_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_vui_config(media_codec_context_t *context, mc_video_vui_params_t  
*params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] mc_video_vui_params_t *params: VUI parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the VUI parameters.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.57 hb_mm_mc_set_vui_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_vui_config(media_codec_context_t *context, const mc_video_vui_params_t  
*params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_video_vui_params_t *params: VUI parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed

- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the VUI parameters, which is a static parameter.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.58 hb_mm_mc_get_3dnr_enc_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_3dnr_enc_config(media_codec_context_t *context,  
mc_video_3dnr_enc_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_video_3dnr_enc_params_t *params: Noise suppression parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the noise suppression parameters, applicable to H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.59 hb_mm_mc_set_3dnr_enc_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_3dnr_enc_config(media_codec_context_t *context, const  
mc_video_3dnr_enc_params_t *params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] const mc_video_3dnr_enc_params_t *params: Noise suppression parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set the noise suppression parameters, which is a dynamic parameter, applicable to H265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.60 hb_mm_mc_request_idr_header

[Function Declaration]

```
hb_s32 hb_mm_mc_request_idr_header(media_codec_context_t *context, hb_u32 force_header)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] hb_u32 force_header:
 - 0 : No forced header(VPS/SPS/PPS)
 - 1 : Forced header before IDR frame
 - 2 : Forced header before I frame for H264 or forced header before CRA and IDR frame for H265

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Request frame header IDR frame header information, applicable to H264 / h265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.61 hb_mm_mc_enable_idr_frame

[Function Declaration]

```
hb_s32 hb_mm_mc_enable_idr_frame(media_codec_context_t *context, hb_bool enable)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] hb_bool enable: 0: disable 1: enable

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Enable IDR frame, applicable to H264 / h265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.2.18 hb_mm_mc_get_longterm_ref_mode;

5.2.62 hb_mm_mc_register_audio_encoder

[Function Declaration]

```
hb_s32 hb_mm_mc_register_audio_encoder(hb_s32 *handle, mc_audio_encode_param_t *encoder)
```

[Parameter Description]

- [IN] mc_audio_encode_param_t *encoder: audio handle
- [OUT] hb_s32 *handle: encode handle

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Register audio encoder.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.63 hb_mm_mc_unregister_audio_encoder

[Function Declaration]

```
hb_s32 hb_mm_mc_unregister_audio_encoder(hb_s32 handle)
```

[Parameter Description]

- [IN] hb_s32 handle: encode handle

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Unregister audio encoder.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.64 hb_mm_mc_register_audio_decoder

[Function Declaration]

```
hb_s32 hb_mm_mc_register_audio_decoder(hb_s32 *handle, mc_audio_decode_param_t *decoder)
```

[Parameter Description]

- [OUT] hb_s32 *handle: decode handle
- [IN] mc_audio_decode_param_t *decoder: aduio decode handle

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Register audio decoder.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.65 hb_mm_mc_unregister_audio_decoder

[Function Declaration]

```
hb_s32 hb_mm_mc_unregister_audio_decoder(hb_s32 handle)
```

[Parameter Description]

- [IN] hb_s32 handle: decode handle

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Unregister audio decoder.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.66 hb_mm_mc_get_explicit_header_config

[Function Declaration]

```
hb_s32 hb_mm_mc_get_explicit_header_config (media_codec_context_t *context, hb_s32 *status)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] hb_s32 *status: enable/disable idr with header

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get the information about whether IDR frame including header, 0: independent IDR and header, 1: IDR with header, applicable to H264 / h265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.67 hb_mm_mc_set_explicit_header_config

[Function Declaration]

```
hb_s32 hb_mm_mc_set_explicit_header_config (media_codec_context_t *context, hb_s32 status)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] hb_s32 status: enable/disable idr with header

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Enable/disable whether IDR frame including header, 0: independent IDR and header, 1: IDR with header, applicable to H264 / h265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.68 hb_mm_mc_get_roi_avg_qp

[Function Declaration]

```
hb_s32 hb_mm_mc_get_roi_avg_qp(media_codec_context_t * context, hb_u32 * params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [OUT] hb_u32 *params: ROI average qp

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Get roi average qp, 0: decided by QPMAP average qp, applicable to H264 / h265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.2.69 hb_mm_mc_set_roi_avg_qp

[Function Declaration]

```
hb_s32 hb_mm_mc_set_roi_avg_qp(media_codec_context_t * context, hb_u32 params)
```

[Parameter Description]

- [IN] media_codec_context_t *context: Specify the context of codec type
- [IN] hb_u32 params: ROI average qp

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

[Function Description]

Set roi average qp, 0: decided by QPMAP average qp, applicable to H264 / h265.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.3 Descriptions of MediaCodec Main Parameters

[Description]

Define the internal working state of MediaCodec.

[Definition]

```
1.  typedef enum _media_codec_state {  
2.      MEDIA_CODEC_STATE_NONE = -1,
```

```
3.     MEDIA_CODEC_STATE_UNINITIALIZED,
4.     MEDIA_CODEC_STATE_INITIALIZED,
5.     MEDIA_CODEC_STATE_CONFIGURED,
6.     MEDIA_CODEC_STATE_STARTED,
7.     MEDIA_CODEC_STATE_PAUSED,
8.     MEDIA_CODEC_STATE_FLUSHING,
9.     MEDIA_CODEC_STATE_ERROR,
10.    MEDIA_CODEC_STATE_TOTAL,
11. } media_codec_state_t;
```

[Description]

Define the codec ID supported by MediaCodec.

[Definition]

```
1. typedef enum _media_codec_id {
2.     MEDIA_CODEC_ID_NONE = -1,
3.
4.     /* Video Codecs */
5.     MEDIA_CODEC_ID_H264,
6.     MEDIA_CODEC_ID_H265,
7.     MEDIA_CODEC_ID_MJPEG,
8.     MEDIA_CODEC_ID_JPEG,
9.
10.    /* Audio Codecs */
11.    MEDIA_CODEC_ID_FLAC,
12.    MEDIA_CODEC_ID_PCM_MULAW,
13.    MEDIA_CODEC_ID_PCM_ALAW,
14.    MEDIA_CODEC_ID_ADPCM_G726,
15.    MEDIA_CODEC_ID_ADPCM,
16.    MEDIA_CODEC_ID_AAC,
17.    MEDIA_CODEC_ID_MP3,
18.    MEDIA_CODEC_ID_MP2,
19.    MEDIA_CODEC_ID_TAK,
20.    MEDIA_CODEC_ID_AC3,
21.    MEDIA_CODEC_ID_WMA,
22.    MEDIA_CODEC_ID_AMR,
23.    MEDIA_CODEC_ID_APE,
24.    MEDIA_CODEC_ID_G729,
25.    MEDIA_CODEC_ID_G723,
26.    MEDIA_CODEC_ID_G722,
27.    MEDIA_CODEC_ID_IAC,
28.    MEDIA_CODEC_ID_RALF,
29.    MEDIA_CODEC_ID_QDMC,
30.    MEDIA_CODEC_ID_DTS,
31.    MEDIA_CODEC_ID_GSM,
```

```
32.     MEDIA_CODEC_ID_TTA,
33.     MEDIA_CODEC_ID_QCELP,
34.     MEDIA_CODEC_ID_MLP,
35.     MEDIA_CODEC_ID_ATRAC1,
36.     MEDIA_CODEC_ID_IMC,
37.     MEDIA_CODEC_ID_EAC,
38.     MEDIA_CODEC_ID_MP1,
39.     MEDIA_CODEC_ID_SIPR,
40.     MEDIA_CODEC_ID_OPUS,
41.     MEDIA_CODEC_ID_CELT,
42.     MEDIA_CODEC_ID_MOV_TEXT,
43.     MEDIA_CODEC_ID_TOTAL,
44. } media_codec_id_t;
```

[Description]

Define the rate control mode of video. At present, it only supports the rate control of H264, H265, and MJPEG encoding channels.

[Definition]

```
1.  typedef enum _mc_video_rate_control_mode {
2.      MC_AV_RC_MODE_NONE = -1,
3.      MC_AV_RC_MODE_H264CBR,
4.      MC_AV_RC_MODE_H264VBR,
5.      MC_AV_RC_MODE_H264AVBR,
6.      MC_AV_RC_MODE_H264FIXQP,
7.      MC_AV_RC_MODE_H264QPMAP,
8.      MC_AV_RC_MODE_H265CBR,
9.      MC_AV_RC_MODE_H265VBR,
10.     MC_AV_RC_MODE_H265AVBR,
11.     MC_AV_RC_MODE_H265FIXQP,
12.     MC_AV_RC_MODE_H265QPMAP,
13.     MC_AV_RC_MODE_MJPEGFIXQP,
14.     MC_AV_RC_MODE_TOTAL,
15. } mc_video_rate_control_mode_t;
```

[Description]

Define the adjustable parameter sets under the CBR control mode of H264.

[Definition]

```
1.  typedef struct _mc_h264_cbr_params {
2.      hb_u32 intra_period;
3.      hb_u32 intra_qp;
4.      hb_u32 bit_rate;
5.      hb_u32 frame_rate;
6.      hb_u32 initial_rc_qp;
7.      hb_s32 vbv_buffer_size;
8.      hb_u32 mb_level_rc_enalbe;
```

```
9.     hb_u32 min_qp_I;
10.    hb_u32 max_qp_I;
11.    hb_u32 min_qp_P;
12.    hb_u32 max_qp_P;
13.    hb_u32 min_qp_B;
14.    hb_u32 max_qp_B;
15.    hb_u32 hvs_qp_enable;
16.    hb_s32 hvs_qp_scale;
17.    hb_u32 max_delta_qp;
18.    hb_bool qp_map_enable;
19. } mc_h264_cbr_params_t;
```

[Description]

Define the adjustable parameter sets under the VBR control mode of H264.

[Definition]

```
1.  typedef struct _mc_h264_vbr_params {
2.      hb_u32 intra_period;
3.      hb_u32 intra_qp;
4.      hb_u32 frame_rate;
5.      hb_bool qp_map_enable;
6. } mc_h264_vbr_params_t;
```

[Description]

Define the adjustable parameter sets under the AVBR control mode of H264.

[Definition]

```
1.  typedef struct mc_h264_avbr_params {
2.      hb_u32 intra_period;
3.      hb_u32 intra_qp;
4.      hb_u32 bit_rate;
5.      hb_u32 frame_rate;
6.      hb_u32 initial_rc_qp;
7.      hb_s32 vbv_buffer_size;
8.      hb_u32 mb_level_rc_enalbe;
9.      hb_u32 min_qp_I;
10.     hb_u32 max_qp_I;
11.     hb_u32 min_qp_P;
12.     hb_u32 max_qp_P;
13.     hb_u32 min_qp_B;
14.     hb_u32 max_qp_B;
15.     hb_u32 hvs_qp_enable;
16.     hb_s32 hvs_qp_scale;
17.     hb_u32 max_delta_qp;
18.     hb_bool qp_map_enable;
19. } mc_h264_avbr_params_t;
```

[Description]

Define the adjustable parameter sets under the FixQP control mode of H264.

[Definition]

```
1.     typedef struct _mc_h264_fix_qp_params {  
2.         hb_u32 intra_period;  
3.         hb_u32 frame_rate;  
4.         hb_u32 force_qp_I;  
5.         hb_u32 force_qp_P;  
6.         hb_u32 force_qp_B;  
7.     } mc_h264_fix_qp_params_t;
```

[Description]

Define the adjustable parameter sets under the QPMAP control mode of H264.

[Definition]

```
1.     typedef struct _mc_h264_qp_map_params {  
2.         hb_u32 intra_period;  
3.         hb_u32 frame_rate;  
4.         hb_byte qp_map_array;  
5.         hb_u32 qp_map_array_count;  
6.     } mc_h264_qp_map_params_t;
```

[Description]

Define the adjustable parameter sets under the CBR control mode of H265.

[Definition]

```
1.     typedef struct _mc_h265_cbr_params {  
2.         hb_u32 intra_period;  
3.         hb_u32 intra_qp;  
4.         hb_u32 bit_rate;  
5.         hb_u32 frame_rate;  
6.         hb_u32 initial_rc_qp;  
7.         hb_s32 vbv_buffer_size;  
8.         hb_u32 ctu_level_rc_enalbe;  
9.         hb_u32 min_qp_I;  
10.        hb_u32 max_qp_I;  
11.        hb_u32 min_qp_P;  
12.        hb_u32 max_qp_P;  
13.        hb_u32 min_qp_B;  
14.        hb_u32 max_qp_B;  
15.        hb_u32 hvs_qp_enable;  
16.        hb_s32 hvs_qp_scale;  
17.        hb_u32 max_delta_qp;  
18.        hb_bool qp_map_enable;  
19.    } mc_h265_cbr_params_t;
```

[Description]

Define adjustable parameter sets under VBR control mode of H265.

[Definition]

```
1.  typedef struct _mc_h265_vbr_params {  
2.      hb_u32 intra_period;  
3.      hb_u32 intra_qp;  
4.      hb_u32 frame_rate;  
5.      hb_bool qp_map_enable;  
6.  } mc_h265_vbr_params_t;
```

[Description]

Define the adjustable parameter sets under the AVBR control mode of H265.

[Definition]

```
1.  typedef struct _mc_h265_avbr_params {  
2.      hb_u32 intra_period;  
3.      hb_u32 intra_qp;  
4.      hb_u32 bit_rate;  
5.      hb_u32 frame_rate;  
6.      hb_u32 initial_rc_qp;  
7.      hb_s32 vbv_buffer_size;  
8.      hb_u32 ctu_level_rc_enalbe;  
9.      hb_u32 min_qp_I;  
10.     hb_u32 max_qp_I;  
11.     hb_u32 min_qp_P;  
12.     hb_u32 max_qp_P;  
13.     hb_u32 min_qp_B;  
14.     hb_u32 max_qp_B;  
15.     hb_u32 hvs_qp_enable;  
16.     hb_s32 hvs_qp_scale;  
17.     hb_u32 max_delta_qp;  
18.     hb_bool qp_map_enable;  
19.  } mc_h265_avbr_params_t;
```

[Description]

Define the adjustable parameter sets under the FixQP control mode of H265.

[Definition]

```
1.  typedef struct _mc_h265_fix_qp_params {  
2.      hb_u32 intra_period;  
3.      hb_u32 frame_rate;  
4.      hb_u32 force_qp_I;  
5.      hb_u32 force_qp_P;  
6.      hb_u32 force_qp_B;  
7.  } mc_h265_fix_qp_params_t;
```

[Description]

Define the adjustable parameter sets under the QPMAP control mode of H265.

[Definition]

```
1.     typedef struct _mc_h265_qp_map_params {  
2.         hb_u32 intra_period;  
3.         hb_u32 frame_rate;  
4.         hb_byte qp_map_array;  
5.         hb_u32 qp_map_array_count;  
6.     } mc_h265_qp_map_params_t;
```

[Description]

Define the adjustable parameter sets under the FixQP control mode of MJPEG.

[Definition]

```
1.     typedef struct _mc_mjpeg_fix_qp_params {  
2.         hb_u32 frame_rate;  
3.         hb_u32 quality_factor;  
4.     } mc_mjpeg_fix_qp_params_t;
```

[Description]

Define the data structure of the custom GOP structure table.

[Definition]

```
1.     typedef struct _mc_video_custom_gop_pic_params {  
2.         hb_u32 pic_type;  
3.         hb_s32 poc_offset;  
4.         hb_u32 pic_qp;  
5.         hb_s32 num_ref_picL0;  
6.         hb_s32 ref_pocL0;  
7.         hb_s32 ref_pocL1;  
8.         hb_u32 temporal_id;  
9.     } mc_video_custom_gop_pic_params_t;
```

[Description]

Define MediaCodec internal status information.

[Definition]

```
1.     typedef struct _mc_inter_status {  
2.         hb_u32 cur_input_buf_cnt;  
3.         hb_u64 cur_input_buf_size;  
4.         hb_u32 cur_output_buf_cnt;  
5.         hb_u64 cur_output_buf_size;  
6.         hb_u32 left_recv_frame;  
7.         hb_u32 left_enc_frame;  
8.         hb_u32 total_input_buf_cnt;  
9.         hb_u32 total_output_buf_cnt;  
10.        hb_s32 pipeline;
```

```
11.     hb_s32 channel_port_id;  
12. } mc_inter_status_t;
```

[Description]

Define the context of MediaCodec.

[Definition]

```
1.  typedef struct _media_codec_context {  
2.      media_codec_id_t codec_id;  
3.      hb_bool encoder;  
4.      hb_s32 instance_index;  
5.      union {  
6.          mc_video_codec_enc_params_t video_enc_params;  
7.          mc_video_codec_dec_params_t video_dec_params;  
8.          mc_audio_codec_enc_params_t audio_enc_params;  
9.          mc_audio_codec_dec_params_t audio_dec_params;  
10.     };  
11. } media_codec_context_t;
```

[Description]

Define the encoding parameters of video encoder that includes H264, H265, MJPEG, and JPEG.

[Definition]

```
1.  typedef struct _mc_video_codec_enc_params {  
2.      hb_s32 width, height;  
3.      mc_pixel_format_t pix_fmt;  
4.      hb_u32 frame_buf_count;  
5.      hb_bool external_frame_buf;  
6.      hb_u32 bitstream_buf_count;  
7.      mc_rate_control_params_t rc_params;  
8.      mc_video_gop_params_t gop_params;  
9.      mc_rotate_degree_t rot_degree;  
10.     mc_mirror_direction_t mir_direction;  
11.     hb_u32 frame_cropping_flag;  
12.     mc_av_codec_rect_t crop_rect;  
13.     hb_bool enable_user_pts;  
14.     union {  
15.         mc_h264_enc_config_t h264_enc_config;  
16.         mc_h265_enc_config_t h265_enc_config;  
17.         mc_mjpeg_enc_config_t mjpeg_enc_config;  
18.         mc_jpeg_enc_config_t jpeg_enc_config;  
19.     };  
20. } mc_video_codec_enc_params_t;
```

[Description]

Define the decoding parameters of video decoder that includes H264, H265, MJPEG, and JPEG.

[Definition]

```
1.  typedef struct _mc_video_codec_dec_params {  
2.      mc_video_stream_feeding_mode_t feed_mode;  
3.      mc_pixel_format_t pix_fmt;  
4.      hb_u32 bitstream_buf_size;  
5.      hb_u32 bitstream_buf_count;  
6.      hb_bool external_bitstream_buf;  
7.      hb_u32 frame_buf_count;  
8.      union {  
9.          mc_h264_dec_config_t h264_dec_config;  
10.         mc_h265_dec_config_t h265_dec_config;  
11.         mc_mjpeg_dec_config_t mjpeg_dec_config;  
12.         mc_jpeg_dec_config_t jpeg_dec_config;  
13.     };  
14. } mc_video_codec_dec_params_t;
```

[Description]

Define the encoding parameters of audio codec.

[Definition]

```
1.  typedef struct _mc_audio_codec_enc_params {  
2.      hb_u32 bit_rate;  
3.      hb_s32 frame_size;  
4.      hb_s32 frame_buf_count;  
5.      hb_s32 packet_count;  
6.      mc_audio_sample_format_t sample_fmt;  
7.      mc_audio_sample_rate_t sample_rate;  
8.      mc_audio_channel_layout_t channel_layout;  
9.      hb_s32 channels;  
10.     hb_ptr enc_config;  
11. } mc_audio_codec_enc_params_t;
```

[Description]

Define the decoding parameters of audio codec.

[Definition]

```
1.  typedef struct _mc_audio_codec_dec_params {  
2.      mc_video_stream_feeding_mode_t feed_mode;  
3.      hb_s32 packet_size;  
4.      hb_s32 packet_count;  
5.      hb_s32 frame_buf_count;  
6.      hb_ptr dec_config;  
7. } mc_audio_codec_dec_params_t;
```

5.4 MediaMuxer Interface Descriptions

The MediaMuxer module is mainly used to encapsulate the coded video stream and audio stream into the MP4 container. This module will provide a series of interfaces to support audio and video track adding and data stream filling. Users can add audio or video tracks to the MediaMuxer, and then fill the audio or video streams, so that Mediamuxer can mix the original streams into MP4 files. The following table shows the formats supported by MediaMuxer:

Table 5-5 MediaMuxer Specifications

Module	Audio Steam		Video Steam		Output Format
	Format	Number of Tracks	Format	Number of Tracks	
MediaMuxer	FLAC	1	H264/H265	1	MP4

5.5 MediaMuxer API

5.5.1 hb_mm_mx_get_default_context

[Function Declaration]

```
hb_s32 hb_mm_mx_get_default_context(media_muxer_context_t *context)
```

[Parameter Description]

- [OUT] media_codec_context_t *context: Default muxer context

[Return Value]

- 0: Success
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameters

[Function Description]

Get the default context parameter value.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

#include <libavformat/avformat.h>
```

```
#include <libavutil/timestamp.h>
#ifndef __cplusplus
}
#endif /* __cplusplus */

#include "hb_media_codec.h"
#include "hb_media_error.h"
#include "hb_media_muxer.h"

int main(int argc, char *argv[])
{
    int ret = 0;
    char inputAudioFileName[256] = "./tmp.h264";
    char inputVideoFileName[256] = "./input.aac";
    char outputFileName[256] = "./output.mp4";
    int doAudioExit = 0, doVideoExit = 0;
    media_muxer_context_t muxerCtx;
    memset(&muxerCtx, 0x00, sizeof(media_muxer_context_t));
    ret = hb_mm_mx_get_default_context(&muxerCtx);
    if (ret) {
        return -1;
    }
    muxerCtx.output_file_name = outputFileName;
    media_muxer_state_t state = MEDIA_MUXER_STATE_NONE;
    hb_mm_mx_get_state(&muxerCtx, &state);
    mx_stream_params_t videoStream;
    memset(&videoStream, 0x00, sizeof(mx_stream_params_t));
    videoStream.codec_id = MEDIA_CODEC_ID_H264;
    videoStream.video_params.bit_rate = 5000;
    videoStream.video_params.frame_rate = 25;
    videoStream.video_params.pix_fmt = MC_PIXEL_FORMAT_YUV420P;
    videoStream.video_params.width = 640;
    videoStream.video_params.height = 480;
    videoStream.video_params.intra_period = 30;
    videoStream.numerator = 1;
    videoStream.denominator = videoStream.video_params.frame_rate;

    mx_stream_params_t audioStream;
    memset(&audioStream, 0x00, sizeof(mx_stream_params_t));
    audioStream.codec_id = MEDIA_CODEC_ID_AAC;
    audioStream.audio_params.bit_rate = 135000;
    audioStream.audio_params.sample_fmt = MC_AV_SAMPLE_FMT_FLTP;
    audioStream.audio_params.sample_rate = MC_AV_SAMPLE_RATE_48000;
    audioStream.audio_params.channel_layout = MC_AV_CHANNEL_LAYOUT_MONO;
    audioStream.audio_params.channels = 1;
```

```
mx_stream_t mxStream;
memset(&mxStream, 0x00, sizeof(mx_stream_t));

AVFormatContext *iVFmtCtx = NULL;
ret = avformat_open_input(&iVFmtCtx, inputVideoFileName, 0, 0);
if (ret < 0) {
    goto ERR;
}
ret = avformat_find_stream_info(iVFmtCtx, 0);
if (ret < 0) {
    goto ERR;
}

AVFormatContext *iAFmtCtx = NULL;
ret = avformat_open_input(&iAFmtCtx, inputAudioFileName, 0, 0);
if (ret < 0) {
    goto ERR;
}
ret = avformat_find_stream_info(iAFmtCtx, 0);
if (ret < 0) {
    goto ERR;
}
if (iAFmtCtx->nb_streams != (unsigned int)1) {
    goto ERR;
}
audioStream.numerator = iAFmtCtx->streams[0]->time_base.num;
audioStream.denominator = iAFmtCtx->streams[0]->time_base.den;

ret = hb_mm_mx_initialize(&muxerCtx);
if (ret < 0) {
    goto ERR;
}
ret = hb_mm_mx_add_stream(&muxerCtx, &videoStream);
if (ret < 0) {
    goto ERR;
}
ret = hb_mm_mx_add_stream(&muxerCtx, &audioStream);
if (ret < 0) {
    goto ERR;
}
ret = hb_mm_mx_start(&muxerCtx);
if (ret < 0) {
    goto ERR;
}
```

}

```

AVPacket pkt = {0};

int64_t v_next_pts = 0, a_next_pts = 0;

while (!doAudioExit || !doVideoExit) {
    if ((!doVideoExit) && (doAudioExit) ||
        hb_mm_mx_compare_ts(&muxerCtx, v_next_pts,
        a_next_pts, COMPARE PTS_VIDEO_AUDIO))) {
        if (av_read_frame(iVFmtCtx, &pkt) >= 0) {
            static int count = 0;
            mxStream.is_audio = 0;
            mxStream.vir_ptr = (unsigned char*)pkt.data;
            mxStream.phy_ptr = 0;
            mxStream.size = pkt.size;
            mxStream.pts = count++; //outputBuffer.vstream_buf.pts;
            mxStream.is_key_frame = 0;
            ret = hb_mm_mx_write_stream(&muxerCtx, &mxStream);
            if (ret < 0) {
                goto ERR;
            }
            v_next_pts += videoStream.denominator/videoStream.numerator/videoStream.video_params.frame_rate;
            av_packet_unref(&pkt);
        } else {
            doVideoExit = TRUE;
        }
    } else {
        if (av_read_frame(iAFmtCtx, &pkt) >= 0) {
            mxStream.is_audio = 1;
            mxStream.vir_ptr = (unsigned char*)pkt.data;
            mxStream.phy_ptr = 0;
            mxStream.size = pkt.size;
            mxStream.pts = pkt.pts;
            mxStream.is_key_frame = 0;
            ret = hb_mm_mx_write_stream(&muxerCtx, &mxStream);
            if (ret < 0) {
                goto ERR;
            }
            a_next_pts += audioStream.denominator/audioStream.numerator/audioStream.audio_params.sample_rate;
            av_packet_unref(&pkt);
        } else {
            doAudioExit = TRUE;
        }
    }
}

```

```
ERR:  
    if (muxerCtx)  
        hb_mm_mx_stop(&muxerCtx);  
  
    if (iVFmtCtx)  
        avformat_close_input(&iVFmtCtx);  
  
    if (iAFmtCtx)  
        avformat_close_input(&iAFmtCtx);  
  
    return -1;  
}
```

5.5.2 hb_mm_mx_initialize

[Function Declaration]

```
hb_s32 hb_mm_mx_initialize(media_muxer_context_t *context)
```

[Parameter Description]

- [IN] media_muxer_context_t *context: muxer context

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INSUFFICIENT_RES: Insufficient internal memory resources
- HB_MEDIA_ERR_NO_FREE_INSTANCE: Unable to allocate more instances (32 at most)
- HB_MEDIA_ERR_FILE_OPERATION_FAILURE: Unable to open or create a file

[Function Description]

Initialize the internal state of MediaMuxer. MediaMuxer will get into **MEDIA_MUXER_STATE_INITIALIZED** status after successfully calling.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.5.1 hb_mm_mx_get_default_context;

5.5.3 hb_mm_mx_add_stream

[Function Declaration]

```
hb_s32 hb_mm_mx_add_stream(media_muxer_context_t *context, const mx_stream_params_t  
*params)
```

[Parameter Description]

- [IN] media_muxer_context_t *context: muxer context
- [IN] const mx_stream_params_t *params: Track information

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INSUFFICIENT_RES: Insufficient internal memory resources
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Add the audio and video tracks.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.5.1 hb_mm_mx_get_default_context;

5.5.4 hb_mm_mx_start

[Function Declaration]

```
hb_s32 hb_mm_mx_start(media_muxer_context_t *context)
```

[Parameter Description]

- [IN] media_muxer_context_t *context: muxer context

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter

- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INSUFFICIENT_RES: Insufficient internal memory resources
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Start the internal process of MediaMuxer. Get into **MEDIA_MUXER_STATE_STARTED** status after successfully calling.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.5.1 hb_mm_mx_get_default_context;

5.5.5 hb_mm_mx_stop

[Function Declaration]

```
hb_s32 hb_mm_mx_stop(media_muxer_context_t *context)
```

[Parameter Description]

- [IN] media_muxer_context_t *context: muxer context

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Stop the internal process of MediaMuxer. Get into **MEDIA_MUXER_STATE_UNINITIALIZED** status after successfully calling.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.5.1 hb_mm_mx_get_default_context;

5.5.6 hb_mm_mx_write_stream

[Function Declaration]

```
hb_s32 hb_mm_mx_write_stream(media_muxer_context_t *context, const mx_stream_t *buffer)
```

[Parameter Description]

- [IN] media_muxer_context_t *context: muxer context
- [IN] const mx_stream_t *buffer: Audio or video data stream

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance
- HB_MEDIA_ERR_INVALID_BUFFER: Invalid buffer
- HB_MEDIA_ERR_FILE_OPERATION_FAILURE: File operation failed

[Function Description]

Write audio or video data.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.5.1 hb_mm_mx_get_default_context;

5.5.7 hb_mm_mx_get_state

[Function Declaration]

```
hb_s32 hb_mm_mx_get_state (media_muxer_context_t *context, media_muxer_state_t *state)
```

[Parameter Description]

- [IN] media_muxer_context_t *context: muxer context
- [IN] const mx_stream_t *buffer: Audio or video data stream.

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Get current MediaMuxer status.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.5.1 hb_mm_mx_get_default_context;

5.6 Descriptions of MediaMuxer Main Parameters

[Description]

Define the internal working state of MediaMuxer.

[Definition]

```
1.     typedef enum _media_muxer_state {  
2.         MEDIA_MUXER_STATE_NONE = -1,  
3.         MEDIA_MUXER_STATE_UNINITIALIZED,  
4.         MEDIA_MUXER_STATE_INITIALIZED,  
5.         MEDIA_MUXER_STATE_STARTED,  
6.         MEDIA_MUXER_STATE_ERROR,  
7.         MEDIA_MUXER_STATE_TOTAL  
8.     } media_muxer_state_t;
```

[Description]

Define the input parameters for the audio stream.

[Definition]

```
1.     typedef struct _mx_audio_stream_input_params {  
2.         hb_s64 bit_rate;  
3.         mc_audio_sample_format_t sample_fmt;  
4.         mc_audio_sample_rate_t sample_rate;  
5.         mc_audio_channel_layout_t channel_layout;  
6.         hb_s32 channels;  
7.     } mx_audio_stream_input_params_t;
```

[Description]

Define the input parameters for the video stream.

[Definition]

```
1.     typedef struct _mx_video_stream_input_params {  
2.         hb_u32 bit_rate;  
3.         hb_u32 frame_rate;  
4.         mc_pixel_format_t pix_fmt;  
5.         hb_s32 width, height;
```

```

6.      hb_s32 intra_period;
7. } mx_video_stream_input_params_t;
```

[Description]

Define the parameter information of the input stream.

[Definition]

```

1. typedef struct _mx_stream_params {
2.     media_codec_id_t codec_id;
3.     hb_s32 numerator;
4.     hb_s32 denominator;
5.     union {
6.         mx_audio_stream_input_params_t audio_params;
7.         mx_video_stream_input_params_t video_params;
8.         mx_subtitle_stream_input_params_t subtitle_params;
9.     };
10. } mx_stream_params_t;
```

[Description]

Define the buffer information of the input stream.

[Definition]

```

1. typedef struct _mx_stream {
2.     hb_s32 is_audio;
3.     hb_u8 *vir_ptr;
4.     hb_u64 phy_ptr;
5.     hb_u32 size;
6.     hb_u64 pts;
7.     hb_bool is_key_frame;
8. } mx_stream_t;
```

[Description]

Define Mediamuxer context information.

[Definition]

```

1. typedef struct _media_muxer_context {
2.     hb_string output_file_name,
3.     mx_output_format_t output_format;
4.     hb_s32 instance_index;
5. } media_muxer_context_t;
```

5.7 MediaRecorder Interface Descriptions

The MediaRecorder module will also provide a series of interfaces for audio and video recording. Users can set the input sources and encoding specifications of audio and video, and specify the output formats and files. MediaRecorder will encode the input source by using MediaCodec, and then mix and encapsulate

the encoding stream using MediaMuxer to complete the whole recording process. The following table shows the supported formats for MediaRecorder:

Table 5-6 MediaRecorder Specifications

Module	Audio		Video		Output Format
	Encoding Specification	Input Source	Encoding Specification	Input Source	
MediaRecorder	FLAC	MIC	H264/H265	Camera	MP4

5.8 MediaRecorder API

5.8.1 hb_mm_mr_get_default_context

[Function Declaration]

```
hb_s32 hb_mm_mr_get_default_context(media_recorder_context_t *context)
```

[Parameter Description]

- [OUT] media_recorder_context_t *context: Default recorder context

[Return Value]

- 0: Success
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameters

[Function Description]

Gets the default context parameter value.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

```
#include "hb_media_error.h"
#include "hb_media_recorder.h"
#include "include/common.h"
#include "hb_vio_interface.h"
#include "hb_camera_interface.h"

#define TAG "[RecorderTest]"
typedef struct MediaRecorderTestContext {
    media_recorder_context_t *recorderCtx;
    media_codec_context_t *encCtx;
```

```

media_codec_context_t *decCtx;
const char *video_output_prefix;
int error;
int terminate;
int duration;
uint32_t fps;
int pipeline;
int channel_id;
int testCamera;
int testCameraRotate;
int testCodec;
char *vioCfgFileName;
char *camCFGFileName;
MRCameraBuffer *camBuf;
pthread_t video_encoder_thread;
pthread_mutex_t cam_buf_lock;
int encoder_thread_finished;
pthread_t video_decoder_thread;
int decoder_thread_finished;
pthread_t video_display_thread;
int display_thread_finished;
FILE *fp;
FILE *encoderfp;
} MediaRecorderTestContext;

static int stopRecorder(MediaRecorderTestContext *mr_ctx)
{
    if (!mr_ctx || !mr_ctx->recorderCtx) {
        printf("%s Invalid parameters.\n", TAG);
        return -1;
    }

    hb_mm_mr_stop(mr_ctx->recorderCtx);

    hb_mm_mr_release(mr_ctx->recorderCtx);
    return 0;
}

static void media_recorder_event_listener(hb_s32 event_type,
                                         hb_s32 event, hb_s32 message, void *user) {
    MediaRecorderTestContext *ctx = (MediaRecorderTestContext *)user;
    if (!ctx || !ctx->recorderCtx) {
        printf("%s Invalid user data.\n", TAG);
        return;
    }
}

```

```

}

printf("%s Received event %d(type=%d) and message %d.\n", TAG, event,
       event_type, message);

if (event == MR_ERROR_GENERAL
    || event == MR_ERROR_AUDIO_CODEC
    || event == MR_ERROR_VIDEO_CODEC
    || event == MR_ERROR_MEDIA_MUXER) {
    ctx->error = 1;
    ctx->terminate = 1;
    printf("%s Try to stop the process due to error.\n", TAG);
} else if (event == MR_INFO_MAX_DURATION_REACHED
    || event == MR_INFO_MAX_FILESIZE_REACHED) {
    ctx->terminate = 1;
    printf("%s Try to stop the process.\n", TAG);
}
}

int startRecorder(MediaRecorderTestContext *mr_ctx) {
    char outputFileName[MAX_FILE_PATH];
    char outputSuffix[MAX_FILE_PATH] = "smoke_h265.mp4";
    int mr_ret = 0;
    snprintf(outputFileName, MAX_FILE_PATH, "%s%s",
             mr_ctx->video_output_prefix, outputSuffix);

    // Setup recorder context
    mr_video_encoder_params_t video_enc;
    media_recorder_context_t *recorderCtx = NULL;
    media_recorder_state_t state = MEDIA_RECORDER_STATE_NONE;

    if (!mr_ctx || !mr_ctx->recorderCtx) {
        printf("%s Invalid parameters.\n", TAG);
        return -1;
    }
    recorderCtx = mr_ctx->recorderCtx;
    memset(&video_enc, 0x00, sizeof(mr_video_encoder_params_t));

    recorderCtx->output_file_name = outputFileName;
    recorderCtx->output_format = MEDIA_MUXER_OUTPUT_FORMAT_MP4;
    recorderCtx->max_file_duration = mr_ctx->duration;
    recorderCtx->max_file_size = 0;

    mr_ret = hb_mm_mr_initialize(recorderCtx);
    if (mr_ret) {
        printf("%s Failed to initialize media recorder.\n", TAG);
    }
}

```

```
        return -1;
    }

    mr_ret = hb_mm_mr_set_listener(recorderCtx,
        media_recorder_event_listener, mr_ctx);
    if (mr_ret) {
        hb_mm_mr_release(recorderCtx);
        printf("%s Failed to set media recorder listener.\n", TAG);
        return -1;
    }

    mr_ret = hb_mm_mr_set_camera(recorderCtx, mr_ctx->pipeline,
        mr_ctx->channel_id);
    if (mr_ret) {
        hb_mm_mr_release(recorderCtx);
        printf("%s Failed to set camera.\n", TAG);
        return -1;
    }

    mr_ret = hb_mm_mr_get_mr_video_source(recorderCtx, &video_enc);
    if (mr_ret) {
        hb_mm_mr_release(recorderCtx);
        printf("%s Failed to get video source.\n", TAG);
        return -1;
    }

    switch(mr_ctx->testCameraRotate) {
    case 0:
        video_enc.rot_degree = MC_CCW_0;
        break;
    case 90:
        video_enc.rot_degree = MC_CCW_90;
        break;
    case 180:
        video_enc.rot_degree = MC_CCW_180;
        break;
    case 270:
        video_enc.rot_degree = MC_CCW_270;
        break;
    default:
        video_enc.rot_degree = MC_CCW_0;
        break;
    }

    video_enc.frame_rate = mr_ctx->fps;
```

```

video_enc.bit_rate = 5000; //kbps
video_enc.codec_id = mr_ctx->testCodec
    ? MEDIA_CODEC_ID_H264 : MEDIA_CODEC_ID_H265;
video_enc.intra_period = 30;
video_enc.mir_direction = MC_DIRECTION_NONE;
video_enc.mr_video_source = MR_VIDEO_SOURCE_CAMERA;
mr_ret = hb_mm_mr_set_mr_video_source(recorderCtx, &video_enc);
if (mr_ret) {
    hb_mm_mr_release(recorderCtx);
    printf("%s Failed to set media recorder video source.\n", TAG);
    return -1;
}

mr_ret = hb_mm_mr_configure(recorderCtx);
if (mr_ret) {
    hb_mm_mr_release(recorderCtx);
    printf("%s Failed to configure media recorder.\n", TAG);
    return -1;
}

mr_ret = hb_mm_mr_prepare(recorderCtx);
if (mr_ret) {
    hb_mm_mr_release(recorderCtx);
    printf("%s Failed to prepare media recorder.\n", TAG);
    return -1;
}

mr_ret = hb_mm_mr_start(recorderCtx);
if (mr_ret) {
    hb_mm_mr_release(recorderCtx);
    printf("%s Failed to start media recorder.\n", TAG);
    return -1;
}

mr_ret = hb_mm_mr_get_state(&recorderCtx, &state);
if (mr_ret || state != MEDIA_RECORDER_STATE_STARTED) {
    hb_mm_mr_release(recorderCtx);
    printf("%s Failed to get media recorder state.\n", TAG);
    return -1;
}

printf("%s Success to start media recorder.\n", TAG);
return 0;
}

```

```
static int startVIO(MediaRecorderTestContext *mr_ctx) {  
    int ret = 0;  
    uint32_t fps = 0;  
    int camIndex = 1;  
    int pipelineNum = 1;  
    int needCamera = 0;  
  
    if (!mr_ctx) {  
        return -1;  
    }  
    needCamera = mr_ctx->testCamera;  
    ret = hb_vio_init(mr_ctx->vioCfgFileName);  
    if (ret < 0) {  
        printf("%s Failed to init vio.\n", TAG);  
        return -1;  
    }  
  
    if (needCamera) {  
        ret = hb_cam_init(camIndex, mr_ctx->camCFGFileName);  
        if (ret < 0) {  
            printf("%s Failed to init camera.\n", TAG);  
            hb_vio_deinit();  
            return -1;  
        }  
    }  
  
    if (pipelineNum > 0) {  
        for (int i = 0; i < pipelineNum; i++) {  
            ret = hb_vio_start_pipeline(i);  
            if (ret < 0) {  
                printf("%s Failed to start vio pipeline.\n", TAG);  
                if (needCamera) {  
                    hb_cam_deinit(camIndex);  
                }  
                hb_vio_deinit();  
                return -1;  
            }  
        }  
    }  
} else {  
    printf("%s No test to do!\n", TAG);  
    if (needCamera) {  
        hb_cam_deinit(camIndex);  
    }  
}
```

```

    hb_vio_deinit();

    return -1;
}

if (needCamera) {
    ret = hb_cam_start(0);

    if (ret < 0) {
        printf("%s Failed to start camera.\n", TAG);
        hb_cam_stop(0);
        hb_cam_deinit(camIndex);
        hb_vio_deinit();
        return -1;
    }
}

if (needCamera) {
    ret = hb_cam_get_fps(0, &fps);

    if (ret < 0) {
        printf("%s Failed to get camera fps.\n", TAG);
        hb_cam_stop(0);
        hb_cam_deinit(camIndex);
        hb_vio_deinit();
        return -1;
    }

    mr_ctx->fps = fps;
} else {
    mr_ctx->fps = 30;
}

printf("%s Camera fps is %u\n", TAG, mr_ctx->fps);

return 0;
}

static int stopVIO(MediaRecorderTestContext *mr_ctx) {
    int camIndex = 1;
    int pipelineNum = 1;
    int needCamera = 0;

    if (!mr_ctx) {
        return -1;
    }

    needCamera = mr_ctx->testCamera;

    if (needCamera) {

```

```

        hb_cam_stop(0);
    }

    sleep(1);

    if (pipelineNum > 0) {
        for (int i = 0; i < pipelineNum; i++) {
            hb_vio_stop_pipeline(i);
        }
    }

    if (needCamera) {
        hb_cam_deinit(camIndex);
    }

    hb_vio_deinit();

    return 0;
}

int main(int argc, char *argv[])
{
    int ret = 0;

    char vioCfgFileName[MAX_FILE_PATH] = "camera_vio_venc_mx_emmc.json";
    char camCFGFileName[MAX_FILE_PATH] = "/etc/cam/hb_x3dev.json";
    media_recorder_context_t mRecorderCtx;
    MediaRecorderTestContext mMRTTestCtx;
    char outputFile[MAX_FILE_PATH] = "./output.mp4";

    memset(&mMRTTestCtx, 0x00, sizeof(mMRTTestCtx));
    memset(&mRecorderCtx, 0x00, sizeof(mRecorderCtx));
    ret = hb_mm_mr_get_default_context(&mRecorderCtx);
    if (ret) {
        return -1;
    }

    mMRTTestCtx.recorderCtx = &mRecorderCtx;
    mMRTTestCtx.terminate = 0;
    mMRTTestCtx.error = 0;
    mMRTTestCtx.video_output_prefix = outputFile;
    mMRTTestCtx.duration = 5*1000;
    mMRTTestCtx.vioCfgFileName = vioCfgFileName;
    mMRTTestCtx.camCFGFileName = camCFGFileName;
    mMRTTestCtx.pipeline = 0;
    mMRTTestCtx.channel_id = HB_VIO_IPU_US_DATA;
    mMRTTestCtx.testCamera = 1;
    mMRTTestCtx.testCameraRotate = 0;
    mMRTTestCtx.testCodec = 0;

    ret = startVIO(&mMRTTestCtx);
}

```

```

if (ret) {
    return -1;
}

ret = startRecorder(&mMRTestCtx);
if (ret < 0) {
    printf("%s Failed to start recorder.\n", TAG);
    stopVIO(&mMRTestCtx);
    mMRTestCtx.terminate = 1;
    mMRTestCtx.error = 1;
    return -1;
}

while (mMRTestCtx.terminate != 1) {
    usleep(10000);
}

ret = stopRecorder(&mMRTestCtx);
if (ret < 0) {
    printf("%s Failed to stop recorder.\n", TAG);
    mMRTestCtx.error = 1;
}

ret = stopVIO(&mMRTestCtx)
if (ret < 0) {
    printf("%s Failed to stop vio.\n", TAG);
    mMRTestCtx.error = 1;
}

if (mMRTestCtx.error == 1) {
    printf("%s Failed to do recorder\n", TAG);
    return -1;
}

return 0;
}

```

5.8.2 hb_mm_mr_initialize

[Function Declaration]

```
hb_s32 hb_mm_mr_initialize(media_recorder_context_t *context)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: muxer context

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INSUFFICIENT_RES: Insufficient internal memory resources
- HB_MEDIA_ERR_NO_FREE_INSTANCE: Unable to allocate more instances (32 at most)

[Function Description]

Initialize the internal state of MediaRecorder. After the call succeeds, MediaRecorder will get into **MEDIA_RECORDER_STATE_INITIALIZED** status.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.8.1 hb_mm_mr_get_default_context;

5.8.3 hb_mm_mr_set_listener

[Function Declaration]

```
hb_s32 hb_mm_mr_set_listener(media_recorder_context_t *context, media_recorder_listener  
listener, void *userdata)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context
- [IN] media_recorder_listener listener: Listen function
- [IN] void *userdata: User data pointer, which is passed in as an input parameter when the callback function is called

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Set the listen function.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.8.1 hb_mm_mr_get_default_context;

5.8.4 hb_mm_mr_get_mr_video_source

[Function Declaration]

```
hb_s32 hb_mm_mr_get_mr_video_source(media_recorder_context_t *context,  
mr_video_encoder_params_t *params)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context
- [OUT] mr_video_encoder_params_t *params: Video encoding related parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Get the encoding parameters of the video source.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.8.1 hb_mm_mr_get_default_context;

5.8.5 hb_mm_mr_set_mr_video_source

[Function Declaration]

```
hb_s32 hb_mm_mr_set_mr_video_source(media_recorder_context_t *context, const  
mr_video_encoder_params_t *params)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context
- [IN] mr_video_encoder_params_t *params: Video encoding related parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Set the encoding parameters of the video source.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.8.1 hb_mm_mr_get_default_context;

5.8.6 hb_mm_mr_get_mr_audio_source

[Function Declaration]

```
hb_s32 hb_mm_mr_get_mr_audio_source(media_recorder_context_t *context,  
mr_audio_encoder_params_t *params)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context
- [OUT] mr_audio_encoder_params_t *params: Audio encoding related parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Get the encoding parameters of the audio source.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.8.7 hb_mm_mr_set_mr_audio_source

[Function Declaration]

```
hb_s32 hb_mm_mr_set_mr_audio_source(media_recorder_context_t *context, const  
mr_audio_encoder_params_t *params)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context
- [IN] const mr_audio_encoder_params_t: Audio encoding related parameters

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Set the encoding parameters of the audio source.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

5.8.8 hb_mm_mr_set_camera

[Function Declaration]

```
hb_s32 hb_mm_mr_set_camera(media_recorder_context_t *context, hb_s32 pipeline, hb_s32  
channel_port_id)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context
- [IN] hb_s32 pipeline: pipeline
- [IN] hb_s32 channel_port_id: Channel port number

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Set the encoding parameters of camera input source.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.8.1 hb_mm_mr_get_default_context;

5.8.9 hb_mm_mr_set_camera_pym

[Function Declaration]

```
hb_s32 hb_mm_mr_set_camera_pym(media_recorder_context_t *context,  
hb_s32 pipeline, hb_s32 channel_port_id, hb_s32 layer_idx)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context
- [IN] hb_s32 pipeline: pipeline
- [IN] hb_s32 channel_port_id: Channel port number
- [IN] hb_s32 layer_idx: pyramid index

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Set the encoding parameters of camera pyramid input source.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.8.1 hb_mm_mr_get_default_context;

5.8.10 hb_mm_mr_configure

[Function Declaration]

```
hb_s32 hb_mm_mr_configure(media_recorder_context_t *context)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Configure the media recorder parameters. After the call succeeds, MediaRecorder will get into **MEDIA_RECORDER_STATE_CONFIGURED** status.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.8.1 hb_mm_mr_get_default_context;

5.8.11 hb_mm_mr_prepare

[Function Declaration]

```
hb_s32 hb_mm_mr_prepare(media_recorder_context_t *context)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INSUFFICIENT_RES: Insufficient memory resources
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Prepare the internal process of media recorder. After the call succeeds, MediaRecorder will get into **MEDIA_RECORDER_STATE_PREPARED** status.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.8.1 hb_mm_mr_get_default_context;

5.8.12 hb_mm_mr_start

[Function Declaration]

```
hb_s32 hb_mm_mr_start(media_recorder_context_t *context)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_PARAMS: Invalid parameter
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INSUFFICIENT_RES: Insufficient memory resources
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Start the internal process of media recorder. After the call succeeds, MediaRecorder will get into **MEDIA_RECORDER_STATE_STARTED** status.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.8.1 hb_mm_mr_get_default_context;

5.8.13 hb_mm_mr_stop

[Function Declaration]

```
hb_s32 hb_mm_mr_stop(media_recorder_context_t *context)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Stop the recording process, exit all sub threads and release related resources. After the call succeeds, MeidaRecorder returns to **MEDIA_RECORDER_STATE_INITIALIZED** status.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.8.1 hb_mm_mr_get_default_context;

5.8.14 hb_mm_mr_release

[Function Declaration]

```
hb_s32 hb_mm_mr_release(media_recorder_context_t *context)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_OPERATION_NOT_ALLOWED: Operation is not allowed

- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Release the media recorder resource. After the call succeeds, MeidaRecorder returns to **MEDIA_RECORDER_STATE_RELEASED** status.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.8.1 hb_mm_mr_get_default_context;

5.8.15 hb_mm_mr_get_state

[Function Declaration]

```
hb_s32 hb_mm_mr_get_state(media_recorder_context_t *context, media_recorder_state_t *state)
```

[Parameter Description]

- [IN] media_recorder_context_t *context: recorder context
- [OUT] media_recorder_state_t *state: MediaRecorder current status

[Return Value]

- 0: Success
- HB_MEDIA_ERR_UNKNOWN: Unknown error
- HB_MEDIA_ERR_INVALID_INSTANCE: Invalid instance

[Function Description]

Get the MediaRecorder current status.

[Compatibility]

System version: 2.0 and above.

Hardware: X3/J3

[Sample Code]

Refer to 5.8.1 hb_mm_mr_get_default_context;

5.9 Descriptions of MediaRecorder Main Parameters

[Description]

Define the internal working state of media recorder.

[Definition]

```
1.     typedef enum _media_recorder_state {  
2.         MEDIA_RECORDER_STATE_NONE = -1,  
3.         MEDIA_RECORDER_STATE_UNINITIALIZED,  
4.         MEDIA_RECORDER_STATE_INITIALIZED,  
5.         MEDIA_RECORDER_STATE_CONFIGURED,  
6.         MEDIA_RECORDER_STATE_PREPARED,  
7.         MEDIA_RECORDER_STATE_STARTED,  
8.         MEDIA_RECORDER_STATE_ERROR,  
9.         MEDIA_RECORDER_STATE_TOTAL  
10.    } media_recorder_state_t;
```

[Description]

Define the listen function of media recorder.

[Definition]

```
1.     typedef void (*media_recorder_listener)(hb_s32 event_type, hb_s32 event, hb_s32 message, void *user);
```

[Description]

Define the video encoding parameters.

[Definition]

```
1.     typedef struct _mr_video_encoder_params {  
2.         mr_video_source_t mr_video_source,  
3.         media_codec_id_t id;  
4.         mc_rotate_degree_t rot_degree;  
5.         mc_mirror_direction_t mir_direction;  
6.         hb_u32 frame_rate;  
7.         hb_u32 bit_rate;  
8.         hb_s32 intra_period;  
9.     } mr_video_encoder_params_t
```

[Description]

Define the audio encoding parameters.

[Definition]

```
1.     typedef struct _mr_audio_encoder_params {  
2.         mr_audio_source_t mr_audio_source,  
3.         media_codec_id_t id;  
4.         hb_s32 bit_rate;  
5.         mc_audio_sample_format_t sample_fmt;  
6.         mc_audio_sample_rate_t sample_rate;
```

```
7.     mc_audio_channel_layout_t channel_layout;
8.     hb_s32 channels;
9. } mr_audio_encoder_params_t;
```

[Description]

Define the context information of media recorder.

[Definition]

```
1. typedef struct _media_recorder_context {
2.     hb_string output_file_name,
3.     mx_output_format_t output_format;
4.     hb_s32 max_file_duration;
5.     hb_s32 max_file_size;
6.     hb_s32 instance_index;
7. } media_recorder_context_t;
```

5.10 Media Return Code

Error Code	Macro Definition	Description
0xF0000001	HB_MEDIA_ERR_UNKNOWN	Unknown error
0xF0000002	HB_MEDIA_ERR_CODEC_NOT_FOUND	No corresponding codec found
0xF0000003	HB_MEDIA_ERR_CODEC_OPEN_FAIL	Unable to open codec device
0xF0000004	HB_MEDIA_ERR_CODEC_RESPONSE_TIMEOUT	Codec response timeout
0xF0000005	HB_MEDIA_ERR_CODEC_INIT_FAIL	Codec initialization failed
0xF0000006	HB_MEDIA_ERR_OPERATION_NOT_ALLOWED	Operation not permitted
0xF0000007	HB_MEDIA_ERR_INSUFFICIENT_RES	Insufficient internal memory resources
0xF0000008	HB_MEDIA_ERR_NO_FREE_INSTANCE	No instance available (32 VPUs at most, 64 JPUs at most, 32 audios at most)
0xF0000009	HB_MEDIA_ERR_INVALID_PARAMS	Invalid parameter
0xF000000A	HB_MEDIA_ERR_INVALID_INSTANCE	Invalid instance
0xF000000B	HB_MEDIA_ERR_INVALID_BUFFER	Invalid buffer
0xF000000C	HB_MEDIA_ERR_INVALID_COMMAND	Invalid command
0xF000000D	HB_MEDIA_ERR_WAIT_TIMEOUT	The wait timed out
0xF000000E	HB_MEDIA_ERR_FILE_OPERATION_FAILURE	File operation failed
0xF000000F	HB_MEDIA_ERR_PARAMS_SET_FAILURE	Parameter setting failed
0xF0000010	HB_MEDIA_ERR_PARAMS_GET_FAILURE	Parameter acquisition failed
0xF0000011	HB_MEDIA_ERR_CODING_FAILED	Encoding and decoding failed
0xF0000012	HB_MEDIA_ERR_OUTPUT_BUF_FULL	Output buffer full

6 Display Output System

6.1 Display Subsystem Description

DISP is fully called Display and IAR is Intelligent Analysis Result (the old names for X2), which is no longer used by X3/J3. Figure 6.1 shows the block diagram of the display subsystem of X3/J3.

The DISP display subsystem reads video or image data from four-channel FBUF (frame buffer), superimposes them on the background and HW cursor using Overlay & Alpha-Blending and Key-color, selectively amplifies them using Up-Scale or adjusts colors including brightness, contrast, saturation, hue, gamma, and dithering as required, and then chooses one of four output modes from LCD_IF for output.

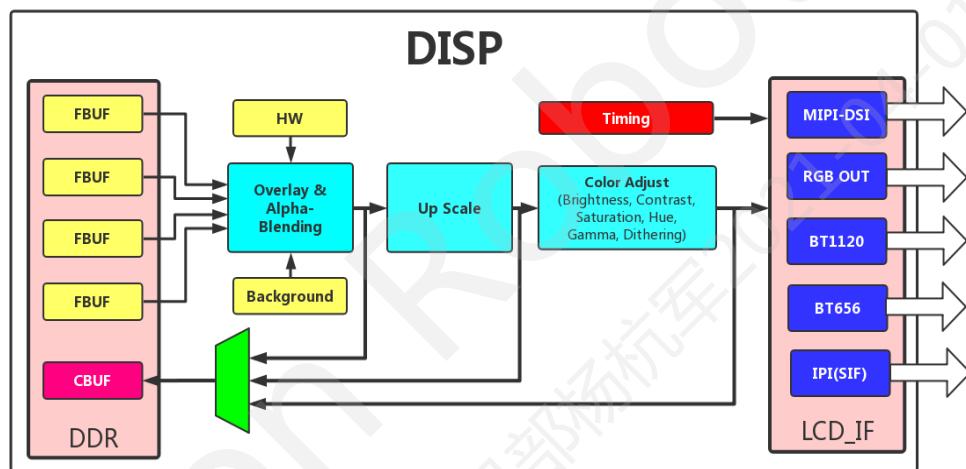


Figure 6.1 Display system overview

The X3/J3 DISP module mainly supports the following functions:

Use FBUF #1 and FBUF #2 as video layer channels and FBUF #3 and FBUF #4 as layer channels, all of which support the Crop function.

FBUF #1 and FBUF #2 only support YUV420sp(nv12), mainly used to display videos from sensors or ISPs

FBUF #3 and FBUF #4 support 8-bpp (CLUT and palette), RGB565, Unpacked RGB888, Packed RGB888, ARGB, and RGBA, which are mainly used to display UI OSD. The 8-bpp format has no endian problem. RGB565, Unpacked RGB888, and Packed RGB888 only support little-endian format. ARGB and RGBA supports little-endian and big-endian format. The mainstream image format is big-endian.

4-channel FBUF can be overlaid with the background layer and HW cursor using Overlay & Alpha-blending and key-color. avalue and overlay priority can be configured.

The output supports Up-Scale

The output supports adjusting colors (brightness, contrast, saturation, hue, gamma, and dithering)

The image data of the output nodes including Overlay & Alpha-blending, Up-Scale, and Color-Adjust can be saved back to CBUF (Capture Buffer) for verification or debugging (Only little-endian format is supported).

LCD_IF interface supports RGB panel, MIPI-DSI panel, BT1120, and MIPI-CSI-TX output. Only one of the output methods can be selected.

RGB Panel and MIPI-DSI panel support RGB888 format only. BT1120 and MIPI-CSI-TX support YUV422 format only.

Maximum pixel rate: 163MHz, Maximum output resolution: 1920x1080 or 1080x1920.

The data stream of DISP module is shown in Figure 6.2:

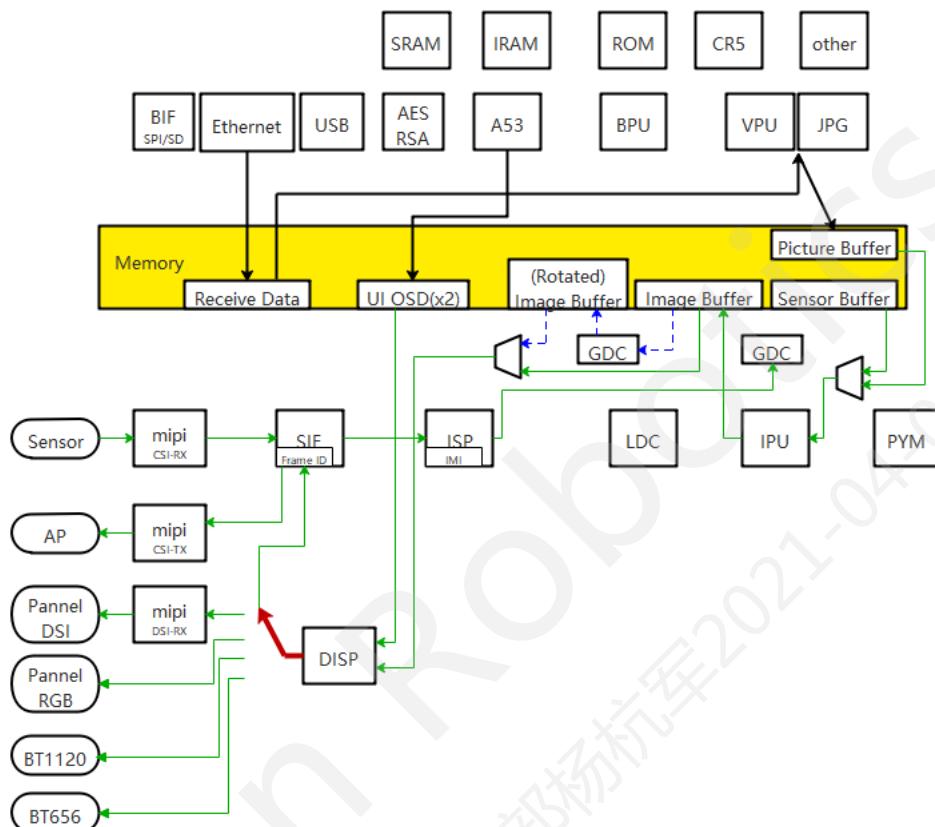


Figure 6.2 Display Subsystem data stream diagram

6.2 DISP API Definition

6.2.1 hb_disp_init_cfg

[Function Declaration]

```
hb_disp_init_cfg(const char *config_file)
```

[Parameter Description]

- [IN]const char *config_file: Displays subsystem configuration file

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Initialize the DISP module. Configure each layer and output channel according to the configuration file.

[Compatibility]

Hardware: X3/J3

6.2.2 hb_disp_init

[Function Declaration]

`hb_disp_init(void)`

[Parameter Description]

None

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Initialize the DISP module. Configure each layer and output channel according to the configuration file.

[Compatibility]

Hardware: X2/J2

[Sample Code]

```
#include "hb_vio_interface.h"
#include "x2_camera.h"
#include "iar_interface.h"

int main(int argc, char *argv[])
{
    int ret = 0;
    int condition_time = 0;
    int need_time_condition = 0;
    hb_vio_buffer_t buf = {0};
    char *cam_cfg_file = "/etc/cam/hb_x3dev.json";
    char *iar_cfg_file = "/etc/iar/iar_x3_lcd.json";
    ret = hb_vio_init("/cfg/imx327_raw_12bit_1952x1097_online_Pipeline.json");
    if (ret < 0) {
        printf("vio init fail\n");
        return -1;
    }
    ret = hb_cam_init(0, cam_cfg_file);
    if (ret < 0) {
        printf("cam init fail\n");
        hb_vio_deinit();
        return -1;
    }
    ret = hb_disp_init(iar_cfg_file);
    if (ret < 0) {
        printf("display init fail\n");
        hb_cam_deinit(0);
        hb_vio_deinit();
        return -1;
    }
}
```

```
    }
```



```
ret = hb_vio_start_pipeline(0);
if (ret < 0) {
    printf("vio start fail, do cam&vio&display_deinit.\n");
    hb_cam_deinit(0);
    hb_vio_deinit();
    hb_disp_deinit();
    return -1;
}

ret = hb_cam_start(0);
if (ret < 0) {
    printf("cam start fail, do cam&vio&display_deinit.\n");
    hb_vio_stop_pipeline(0);
    hb_cam_deinit(0);
    hb_vio_deinit();
    hb_disp_deinit();
}

ret = hb_disp_start();
if (ret < 0) {
    printf("display start fail, do cam&vio&display_deinit.\n");
    goto err;
}

hb_disp_stop();
hb_cam_stop(0);
hb_vio_stop_pipeline(0);
hb_cam_deinit(0);
hb_vio_deinit();
hb_disp_deinit();
return 0;

err:
hb_cam_stop(0);
hb_vio_stop_pipeline(0);
hb_cam_deinit(0);
hb_vio_deinit();
hb_disp_deinit();
return -1;
}
```

6.2.3 hb_disp_start

[Function Declaration]

hb_disp_start()

[Parameter Description]

None

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Start the DISP module.

[Compatibility]

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 6.2.2hb_disp_init

6.2.4 hb_disp_stop

[Function Declaration]

hb_disp_stop()

[Parameter Description]

None

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Stop the DISP module.

[Compatibility]

Hardware: X2/J2; X3/J3

[Sample Code]Refer to 6.2.2hb_disp_init

6.2.5 hb_disp_close

[Function Declaration]

hb_disp_close()

[Parameter Description]

None

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Release various resources requested in hb_disp_init, and close the device file.

[Compatibility]

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 6.2.2hb_disp_init

6.2.6 hb_disp_layer_on

[Function Declaration]

```
hb_disp_layer_on(unsigned int layer_number)
```

[Parameter Description]

- [IN]unsigned int layer_number: Layer number (0, 1, 2, 3) to enable

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Enable one of the four layers (0 and 1 for video layer, 2 and 3 for image layer).

[Compatibility]

Hardware: X2/J2; X3/J3

[Sample Code]

```
#include "hb_vio_interface.h"  
  
#include "x2_camera.h"  
  
#include "iar_interface.h"  
  
#define TEST_YUVIMAGE1_PATH "./400_240yuv8.yuv"  
#define TEST_YUVIMAGE2_PATH "./800_480yuv8.yuv"  
  
  
uint32_t get_file( const char *path, char **buff )  
{  
    FILE *file = NULL;  
    struct stat statbuf;  
  
    file = fopen(path, "r");  
    if ( NULL == file )  
        return -1;  
    stat(path, &statbuf);  
    if ( 0 == statbuf.st_size ) {  
        fclose(file);  
        return -1;  
    }  
    *buff = (char *)malloc(statbuf.st_size);  
    if ( NULL == *buff ) {  
        fclose(file);  
        return 0;  
    }  
    fread(*buff, statbuf.st_size, 1, file);  
    fclose(file);  
    return statbuf.st_size;  
}
```

```

int main(int argc, char *argv[])
{
    int ret;
    char* framebuffer[2];
    int framesize[2];

    ret = hb_disp_init(iar_cfg_file);
    if (ret < 0) {
        printf("display init fail\n");
        return -1;
    }
    ret = hb_disp_start();
    if (ret < 0) {
        printf("display start fail, do display deinit.\n");
        goto err;
    }
    ret = hb_disp_layer_on(0);
    if (ret < 0) {
        printf("display layer on 0 fail, do display deinit.\n");
        goto err;
    }
    ret = hb_disp_layer_off(2);
    if (ret < 0) {
        printf("display layer off 2 fail, do display deinit.\n");
        goto err;
    }

    framesize[1] = get_file( TEST_YUVIMAGE1_PATH, &framebuffer[1]);
    if ( !framesize[1] || NULL == framebuffer[1] ) {
        fprintf(stderr, "open fail %s error %d, %s/n", TEST_YUVIMAGE1_PATH,
                errno, strerror(errno));
        goto err;
    }
    framesize[0] = get_file( TEST_YUVIMAGE2_PATH, &framebuffer[0]);
    if ( !framesize[0] || NULL == framebuffer[0] ) {
        fprintf(stderr, "open fail %s error %d, %s/n", TEST_YUVIMAGE2_PATH,
                errno, strerror(errno));
        goto err;
    }
    ret = hb_check_video_bufaddr_valid(framesize[1], 0);
    if (ret) {
        printf ("app: unmatch framesize with layer config!\n");
        goto err;
    }
}

```

```

ret = hb_check_video_bufaddr_valid(framesize[0], 1);
if (ret) {
    printf ("app: unmatch framesize with layer config!\n");
    goto err;
}

ret = hb_set_video_bufaddr(framebuf[1], NULL, framebuf[0], NULL);
if (ret) {
    printf("hb set video bufaddr error!!!!\n");
    goto err;
}

hb_disp_stop();
hb_disp_deinit();

return 0;
err:
hb_disp_deinit();
return -1;
}

```

6.2.7 hb_disp_layer_off

[Function Declaration]

`hb_disp_layer_off(unsigned int layer_number)`

[Parameter Description]

- [IN]unsigned int layer_number: Layer number (0, 1, 2, 3) to disable

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Disable one of the four layers (0 and 1 for video layer, 2 and 3 for image layer).

[Compatibility]

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 6.2.6hb_disp_layer_on

6.2.8 hb_disp_set_video_channel

[Function Declaration]

`hb_disp_set_video_channel(unsigned int channel_number, unsigned int layer_number)`

[Parameter Description]

- [IN]unsigned int channel_number: Camera channel to display
- [IN]unsigned int layer_number: Video layer serial number

[Return Value]

- Success: Returns HB_OK 0

- Failure: -1

[Function Description]

When multi-channel cameras access the system, configure the serial number of the camera channel to be displayed and which video layer the camera channel will display (0 and 1 are the video layer).

[Compatibility]

Hardware: X2/J2;

6.2.9 hb_disp_set_video_display_ddr_layer

[Function Declaration]

```
hb_disp_set_video_display_ddr_layer(unsigned int ddr_layer_number, unsigned int layer_number)
```

[Parameter Description]

- [IN]unsigned int ddr_layer_number: Memory serial number to display
- [IN]unsigned int layer_number: Video layer serial number

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Change the memory address displayed in a video layer (for example, the image in the memory after cropping, image in the memory after scaling, or image in one layer of PYM can be displayed)

[Compatibility]

Hardware: X2/J2;

6.2.10 hb_disp_set_vio_channel

[Function Declaration]

```
hb_disp_set_vio_channel(char disp_layer, char pipeline, char channel_no);
```

[Parameter Description]

- [IN]char disp_layer: The iar video layer serial number to display (0, 1)
- [IN]char pipeline: The VIO pipeline to display (0, 1, 2, 3)
- [IN]char channel_no: The VIO channel to display

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Change the memory address displayed in a video layer (such as one of 6 IPU channels, a layer of pyramid down-scale, a layer of pyramid up-scale, and the output of a resolution after GDC rotation)

[Compatibility]

Hardware: X3/J3;

6.2.11 hb_disp_set_lcd_backlight

[Function Declaration]

```
hb_disp_set_lcd_backlight(unsigned int backlight_level)
```

[Parameter Description]

- [IN] unsigned int backlight_level: Backlight brightness level, range [0,10]

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Set the display backlight brightness.

[Compatibility]

Hardware: X2/J2; X3/J3

6.2.12 hb_disp_get_output_cfg

[Function Declaration]

```
hb_disp_get_output_cfg(output_cfg_t *cfg)
```

[Parameter Description]

- [IN] output_cfg_t *cfg: Structure pointer for output configuration

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Get the output configuration of the current display subsystem.

[Compatibility]

Hardware: X2/J2; X3/J3

[Sample Code]

```
#include "hb_vio_interface.h"
#include "x2_camera.h"
#include "iar_interface.h"

#define TEST_YUVIMAGE1_PATH "./400_240yuv8.yuv"
#define TEST_YUVIMAGE2_PATH "./800_480yuv8.yuv"

uint32_t get_file( const char *path, char **buff )
{
    FILE *file = NULL;
    struct stat statbuf;

    file = fopen(path, "r");
    if ( NULL == file )
        return -1;
```

```
stat(path, &statbuf);
if ( 0 == statbuf.st_size ) {
    fclose(file);
    return -1;
}
*buff = (char *)malloc(statbuf.st_size);
if ( NULL == *buff ) {
    fclose(file);
    return 0;
}
fread(*buff, statbuf.st_size, 1, file);
fclose(file);
return statbuf.st_size;
}

int main(int argc, char *argv[])
{
    int ret;
    char* framebuf[2];
    int framesize[2];
    channel_base_cfg_t chn0_cfg;
    channel_base_cfg_t chn1_cfg;
    output_cfg_t out_cfg;
    upscaling_cfg_t upscale_cfg;
    ppcon1_cfg_t ppcon1_cfg;
    ppcon2_cfg_t ppcon2_cfg;

    chn0_cfg.channel = 0;
    chn0_cfg.enable = 1;
    chn0_cfg.pri = 2;
    chn0_cfg.width = 400;
    chn0_cfg.height = 240;
    chn0_cfg.buf_width = 400;
    chn0_cfg.buf_height = 240;
    chn0_cfg.xposition = 50;
    chn0_cfg.yposition = 50;
    chn0_cfg.format = FORMAT_YUV420SP_UV;
    chn0_cfg.alpha = 255;
    chn0_cfg.keycolor = 0;
    chn0_cfg.alpha_sel = 0;
    chn0_cfg.ov_mode = 0;
    chn0_cfg.alpha_en = 1;
    chn0_cfg.crop_width = 400;
    chn0_cfg.crop_height = 240;
    chn1_cfg.channel = 0;
```

```
chn1_cfg.enable = 1;
chn1_cfg.pri = 3;
chn1_cfg.width = 800;
chn1_cfg.height = 480;
chn1_cfg.buf_width = 800;
chn1_cfg.buf_height = 480;
chn1_cfg.xposition = 0;
chn1_cfg.yposition = 0;
chn1_cfg.format = FORMAT_YUV420SP_UV;
chn1_cfg.alpha = 255;
chn1_cfg.keycolor = 0;
chn1_cfg.alpha_sel = 0;
chn1_cfg.ov_mode = 0;
chn1_cfg.alpha_en = 1;
chn1_cfg.crop_width = 800;
chn1_cfg.crop_height = 480;
ppcon1_cfg.dithering_flag = 0;
ppcon1_cfg.dithering_en = 0;
ppcon1_cfg.gamma_en = 0;
ppcon1_cfg.hue_en = 0;
ppcon1_cfg.sat_en = 0;
ppcon1_cfg.con_en = 0;
ppcon1_cfg.bright_en = 1;
ppcon1_cfg.theta_sign = 0;
ppcon1_cfg.contrast = 0;
ppcon2_cfg.theta_abs = 0;
ppcon2_cfg.saturation = 0;
ppcon2_cfg.off_contrast = 0;
ppcon2_cfg.off_bright = -20;
output_cfg.bgcolor = 16744328;
output_cfg.out_sel = OUTPUT_RGB;
output_cfg.width = 800;
output_cfg.height = 480;
output_cfg.big_endian = 0;
output_cfg.display_addr_type = DISPLAY_CHANNEL1;
output_cfg.display_cam_no = PIPELINE0;
output_cfg.ppcon1 = ppcon1_cfg;
output_cfg.ppcon2 = ppcon2_cfg;
output_cfg.rotate = 0;
output_cfg.user_control_disp = 0;

ret = hb_disp_set_channel_cfg(0, &chn0_cfg);
if (ret < 0) {
    printf("channel 0 init fail\n");
```

```

        return -1;
    }

    ret = hb_disp_set_channel_cfg(1, &chn1_cfg);
    if (ret < 0) {
        printf("channel 1 init fail\n");
        return -1;
    }

    ret = hb_disp_set_output_cfg(&output_cfg);
    if (ret < 0) {
        printf("output channel init fail\n");
        return -1;
    }

    ret = hb_disp_start();
    if (ret < 0) {
        printf("display start fail, do display deinit.\n");
        goto err;
    }

    ret = hb_disp_layer_on(0);
    if (ret < 0) {
        printf("display layer on 0 fail, do display deinit.\n");
        goto err;
    }

    ret = hb_disp_layer_off(2);
    if (ret < 0) {
        printf("display layer off 2 fail, do display deinit.\n");
        goto err;
    }

    framesize[1] = get_file( TEST_YUVIMAGE1_PATH, &framebuf[1]);
    if ( !framesize[1] || NULL == framebuf[1] ) {
        fprintf(stderr, "open fail %s error %d, %s/n", TEST_YUVIMAGE1_PATH,
                errno, strerror(errno));
    }

    framesize[0] = get_file( TEST_YUVIMAGE2_PATH, &framebuf[0]);
    if ( !framesize[0] || NULL == framebuf[0] ) {
        fprintf(stderr, "open fail %s error %d, %s/n", TEST_YUVIMAGE2_PATH,
                errno, strerror(errno));
    }

    ret = hb_set_video_bufaddr(framebuf[1], NULL, framebuf[0], NULL);
    if (ret) {
        printf("hb set video bufaddr error!!!!\n");
        goto err;
    }
}

```

```

ret = hb_disp_get_channel_cfg(0, &chn0_cfg);
if (ret < 0) {
    printf("Get channel 0 configuration fail\n");
    goto err;
}

ret = hb_disp_get_channel_cfg(1, &chn1_cfg);
if (ret < 0) {
    printf("Get channel 1 configuration fail\n");
    goto err;
}

ret = hb_disp_get_output_cfg(&output_cfg);
if (ret < 0) {
    printf("Get output configuration fail\n");
    goto err;
}

ret = hb_disp_get_upscaling_cfg(&upscale_cfg);
if (ret < 0) {
    printf("Get upscaling configuration fail\n");
    goto err;
}

hb_disp_stop();
return 0;
err:
hb_disp_stop();
return -1;
}

```

6.2.13 hb_disp_set_output_cfg

[Function Declaration]

`hb_disp_set_output_cfg(output_cfg_t *cfg)`

[Parameter Description]

- [IN] `output_cfg_t *cfg`: The output structure pointer to be configured

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Configure the output settings of the display subsystem

[Compatibility]

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 6.2.12hb_disp_get_output_cfg

6.2.14 hb_disp_get_upscaling_cfg

[Function Declaration]

```
hb_disp_get_upscaling_cfg(upscaling_cfg_t *cfg)
```

[Parameter Description]

- [IN] upscaling_cfg_t *cfg: The structure pointer of the amplification configuration of the output channel to be obtained

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Get the amplification configuration of the output channel.

[Compatibility]

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 6.2.12hb_disp_get_output_cfg

6.2.15 hb_disp_set_upscaling_cfg

[Function Declaration]

```
hb_disp_set_upscaling_cfg(upscaling_cfg_t *cfg)
```

[Parameter Description]

- [IN] upscaling_cfg_t *cfg: The structure pointer of the amplification configuration of the output channel to be set

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Configure the amplification function of the output channel.

[Compatibility]

Hardware: X2/J2; X3/J3

6.2.16 hb_disp_get_channel_cfg

[Function Declaration]

```
hb_disp_get_channel_cfg(uint32_t chn, channel_cfg_t *cfg)
```

[Parameter Description]

- [IN] channel_cfg_t *cfg: Channel configuration structure pointer to be obtained
- [IN] chn: The display layers (0, 1, 2, 3)

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Get the configuration to display a layer.

[Compatibility]

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 6.2.12hb_disp_get_output_cfg

6.2.17 hb_disp_set_channel_cfg

[Function Declaration]

```
hb_disp_set_channel_cfg(uint32_t chn, channel_cfg_t *cfg)
```

[Parameter Description]

- [IN] channel_cfg_t *cfg: The channel configuration structure pointer to be configured
- [IN] chn: Serial number of the display layers (0, 1, 2, 3)

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Configure to display a layer

[Compatibility]

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 6.2.12hb_disp_get_output_cfg

6.2.18 hb_set_layer_cfg

[Function Declaration]

```
hb_set_layer_cfg(uint32_t layer_no, uint32_t width, uint32_t height, uint32_t x_pos, uint32_t y_pos)
```

[Parameter Description]

- [IN] uint32_t layer_no: Serial number of the display layers (0, 1, 2, 3)
- [IN] uint32_t width: Width of the input image
- [IN] uint32_t height: Height of the input image
- [IN] uint32_t x_pos: The display image is horizontally offset relative to the upper left corner of the panel
- [IN] uint32_t y_pos: The display image is vertically offset relative to the upper left corner of the panel

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Configure the basic settings of a layer, including the width and height of the input image and the display position

[Compatibility]

Hardware: X2/J2; X3/J3

6.2.19 hb_set_video_bufaddr

[Function Declaration]

```
hb_set_video_bufaddr(void *addr0_y, void *addr0_c, void *addr1_y, void *addr1_c)
```

[Parameter Description]

- [IN] void *addr0_y: The y address of the image to be displayed by the video layer 0
- [IN] void *addr0_c: The c address of the image to be displayed by the video layer 0 (If the images are continuous, the address can be NULL)
- [IN] void *addr1_y: The y address of the image to be displayed by the video layer 1
- [IN] void *addr1_c: The c address of the image to be displayed by the video layer 1 (If the images are continuous, the address can be NULL)

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Set the address of the video or image to be displayed

[Compatibility]

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 6.2.6hb_disp_layer_on

6.2.20 hb_check_video_bufaddr_valid

[Function Declaration]

```
hb_check_video_bufaddr_valid(size_t graphic_size, uint32_t disp_layer_no)
```

[Parameter Description]

- [IN] size_t graphic_size: Image size that users read in
- [IN] uint32_t disp_layer_no: Layers to be displayed (0, 1)

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1. Indicates that the image size to be displayed as users need is smaller than the size of layer channel buffer. Call hb_set_video_bufaddr may cause the system to crash.
- Failure: 1. Indicates that the image size to be displayed as users need is larger than the size of layer channel buffer. Call hb_set_video_bufaddr may display an exception but will not crash the system

[Function Description]

Check whether the image to be displayed as users need matches the configuration of the current layer channel. If users are not sure whether it matches, call this function for checking before calling hb_set_video_bufaddr, and then perform the operations according to the return value. When the display image is completely matched with the configuration of the graphic layer channel, it returns 0 and the image can be displayed correctly.

[Compatibility]

Hardware: X2/J2; X3/J3

[Sample Code] Refer to 6.2.6hb_disp_layer_on

6.2.21 hb_disp_set_timing

[Function Declaration]

```
hb_disp_set_timing(struct disp_timing *user_timing)
```

[Parameter Description]

- [IN] struct disp_timing *user_timing: Timing structure pointer to be set

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Set the timing parameters of the display module.

[Compatibility]

Hardware: X2/J2; X3/J3

6.2.22 hb_disp_out_upscale

[Function Declaration]

```
hb_disp_out_upscale(uint32_t src_w, uint32_t src_h, uint32_t tag_w, uint32_t tag_h)
```

[Parameter Description]

- [IN] uint32_t src_w: Original image width
- [IN] uint32_t src_h: Original image height
- [IN] uint32_t tag_w: Amplified image width
- [IN] uint32_t tag_h: Amplified image height

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Set the amplification function of the output channel.

[Compatibility]

Hardware: X2/J2; X3/J3

6.2.23 hb_disp_get_gamma_cfg

[Function Declaration]

```
hb_disp_get_gamma_cfg()
```

[Parameter Description]

- [IN] None

[Return Value]

- Return the current gamma value

[Function Description]

Get the gamma value of the current settings.

[Compatibility]

Hardware: X2/J2; X3/J3

6.2.24 hb_disp_set_gamma_cfg

[Function Declaration]

```
hb_disp_set_gamma_cfg(int gamma)
```

[Parameter Description]

- [IN] int gamma: Gamma value to be configured

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Configure the gamma adjustment function of disp module output effects.

[Compatibility]

Hardware: X2/J2; X3/J3

6.2.25 hb_disp_wb_start

[Function Declaration]

```
hb_disp_wb_start(int srcsel, int format)
```

[Parameter Description]

- [IN] int srcsel: write-back source
- [IN] int format: write-back image format

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Start write-back function..

[Compatibility]

Hardware: X3/J3

[Sample Code] Refer to 6.2.30hb_disp_wb_setcfg

6.2.26 hb_get_disp_done

[Function Declaration]

```
hb_get_disp_done()
```

[Parameter Description]

- [IN] uint32_t src_w: None

[Return Value]

- Success: Returns 0 (not finished); returns 1 (finished)

- Failure: -1

[Function Description]

Set the amplification function of the output channel.

[Compatibility]

Hardware: X2/J2

6.2.27 hb_disp_wb_stop

[Function Declaration]

```
hb_disp_wb_stop(void)
```

[Parameter Description]

- [IN] void

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Stop write-back function..

[Compatibility]

Hardware: X3/J3

[Sample Code] Refer to 6.2.30hb_disp_wb_setcfg

6.2.28 hb_disp_get_screen_frame

[Function Declaration]

```
hb_disp_get_screen_frame(hb_vio_buffer_t *iar_buffer)
```

[Parameter Description]

- [IN] hb_vio_buffer_t *iar_buffer: image frame data format, this struct definition is described in 4.7.2X3/J3 main parameters

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Get one display frame(write-back image)

[Compatibility]

Hardware: X3/J3

[Sample Code] Refer to 6.2.30hb_disp_wb_setcfg

6.2.29 hb_disp_release_screen_frame

[Function Declaration]

```
hb_disp_release_screen_frame(hb_vio_buffer_t *iar_buffer)
```

[Parameter Description]

- [IN] hb_vio_buffer_t *iar_buffer: image frame data format

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Release one display frame(write-back image)

[Compatibility]

Hardware: X3/J3

[Sample Code] Refer to 6.2.30hb_disp_wb_setcfg

6.2.30 hb_disp_wb_setcfg

[Function Declaration]

hb_disp_wb_setcfg(int sel, int format)

[Parameter Description]

- [IN] int sel: write-back source
- [IN] int format: write-back image format

[Return Value]

- Success: Returns HB_OK 0
- Failure: -1

[Function Description]

Set write-back function attributes

[Compatibility]

Hardware: X3/J3

[Sample Code]

```
#include "hb_vio_interface.h"
#include "iar_interface.h"

int sample_wb(int wb_src, int wb_format) {
    int ret = 0;
    hb_vio_buffer_t *iar_wb_buf;

    iar_wb_buf = malloc(sizeof(hb_vio_buffer_t));
    memset(iar_wb_buf, 0, sizeof(hb_vio_buffer_t));
    ret = hb_disp_wb_setcfg(wb_src, wb_format);
    if (ret) {
        printf("error set write back config!!\n");
        goto err;
    }
    ret = hb_disp_wb_start(wb_src, wb_format);
    if (ret) {
        printf("error start write back!!\n");
        goto err;
    }
    ret = hb_disp_get_screen_frame(iar_wb_buf);
    if (ret) {

```

```

    printf("error get screen frame!!\n");
    goto err;
}
ret = hb_disp_release_screen_frame(iar_wb_buf)
if (ret) {
    printf("error release screen frame!!");
    goto err;
}
ret = hb_disp_wb_stop();
if (ret) {
    printf("error stop write back!!\n");
    goto err;
}
err:
free(iar_wb_buf);
iar_wb_buf = NULL;
if (ret)
    return -1;
else
    return 0;
}

```

6.3 DISP Parameter Descriptions

Users configure the display timing structure definition:

```

struct disp_timing {
    uint32_t hbp; /* horizon back porch */
    uint32_t hfp; /* horizon front porch */
    uint32_t hs; /* horizon sync width */
    uint32_t vbp; /* vertical back porch */
    uint32_t vfp; /* vertical front porch */
    uint32_t vs; /* vertical sync width */
    uint32_t vfp_cnt; /* vsync front porch besides DPI_VPF */
};

```

Descriptions of layer channel configuration structure:

```

typedef struct _channel_base_cfg_t {
    uint32_t channel; /* Layer serial number: 0, 1, 2, 3 */
    uint32_t enable; /* Whether to enable the layer, 0: disable, 1: enable */
    uint32_t pri; /* Layer priority configuration, 0, 1, 2, 3(0 highest priority, 3 lowest priority) */
    uint32_t width; /* Input image width */
    uint32_t height; /* Input image width */
};

```

```

        uint32_t      buf_width;          /* Image buffer width, configured as input image width */
        uint32_t      buf_height;         /* Image buffer height, configured as input image height */
        uint32_t      xposition;         /* The horizontal distance between the upper left corner of the image display and the
                                         upper left corner of the panel */
        uint32_t      yposition;         /* The vertical distance between the upper left corner of the image display and the upper
                                         left corner of the panel */
        uint32_t      format;            /* Input image format */
        uint32_t      alpha;              /* The alpha value (transparency) of the layer, 0: full transparency, 255: full opacity
                                         */
        uint32_t      keycolor;           /* Keycolor value of the layer */
        uint32_t      alpha_sel;
        uint32_t      ov_mode;            /* Overlay mode of the layer */
        uint32_t      alpha_en;           /* Whether to enable the alpha blending function, 0: disable, 1: enable */
        uint32_t      crop_width;          /* Width of the image cropping */
        uint32_t      crop_height;         /* Height of the image cropping */
    } channel_base_cfg_t;
}

```

Descriptions of the output effect structure 1:

```

typedef struct _ppcon1_cfg_t {
    uint32_t      dithering_flag;     /* dithering flag, 0: RGB666, 1: RGB565 */
    uint32_t      dithering_en;        /* Whether to enable the dithering function, 0: disable, 1: enable */
    uint32_t      gamma_en;            /* Whether to enable the gamma correction function, 0: disable, 1: enable */
    uint32_t      hue_en;              /* Whether to enable the hue adjustment function, 0: disable, 1: enable */
    uint32_t      sat_en;              /* Whether to enable the saturation adjustment function, 0: disable, 1: enable */
    uint32_t      con_en;              /* Whether to enable the contrast adjustment function, 0: disable, 1: enable */
    uint32_t      bright_en;           /* Whether to enable the brightness adjustment function, 0: disable, 1: enable */
    uint32_t      theta_sign;          /* theta sign */
    uint32_t      contrast;            /* Contrast value */
} ppcon1_cfg_t;

```

Descriptions of output effect structure 2:

```

typedef struct _ppcon2_cfg_t {
    uint32_t      theta_abs;           /* theta absolute value */
    uint32_t      saturation;          /* Saturation value */
    uint32_t      off_contrast;         /* Contrast value offset */
    uint32_t      off_bright;           /* Brightness value */
} ppcon2_cfg_t;

```

Output refresh structure descriptions:

```

typedef struct _refresh_cfg_t {
    uint32_t      dbi_refresh_mode;
    /* 0: block refresh mode, 1:line refresh mode */
    uint32_t      panel_color_type;
    /* 0: RGB type; 1: YUV422 type; 2: YUV444 type */
    uint32_t      interlace_sel;
    /* 0: non-interlace; 1: interlace */
    uint32_t      odd_polarity;
}

```

```

/* 0: low for odd field; 1: high for odd field */

uint32_t      pixel_rate;

/* one pixel per (M + 1) cycles of DE_PCLK, M = 0, 1, 2, 3 */

uint32_t      ycbcr_out;           /* when DE output YUV444 or YUV422, maybe conversion from YUV to YCbCr is
needed. 0: no need conversion; 1: conversion needed */

uint32_t      uv_sequence;        /* when output YUV422, this bit besides UV sequence. 0: YU->YV->YU->YV; 1:
YV->YU->YV->YU */

uint32_t      itu_r656_en;         /* ccir656 output enable. 0: disable; 1: enable */

uint32_t      auto_dbi_refresh_cnt;

/* refresh period = Tde_pixel_clk * (AUTO_DBI_REFRESH_CNT) */

uint32_t      auto_dbi_refresh_en;

/* 0: manual refresh mode, assert DBI_START, then DE refresh one frame and stops; 1: auto refresh mode, assert
AUTO_DBI_REFRESH_EN(no need to assert DBI_START), then DE refreshes panel automatically one frame by one frame just like "DPI
refresh mode". When SW wants to stop DE refreshing, just deassert AUTO_DBI_REFRESH_EN. The frequency of auto refresh in this
mode is depended by AUTO_DBI_REFRESH_CNT. */

} refresh_cfg_t;

```

Output configuration structure descriptions:

```

typedef struct _output_cfg_t {

    uint32_t      bgcolor;   /* Background layer color */

    uint32_t      out_sel;    /* Output mode, see output mode enumeration descriptions */

    uint32_t      width;      /* Display device width */

    uint32_t      height;     /* Display device height */

    uint32_t      big_endian; /* End configuration of image format size of the graphic layer */

    uint32_t      display_addr_type; /* The displayed vio channel, refer to the output channel type enumeration
descriptions */

    uint32_t      display_cam_no; /* The displayed camera channel */

    ppcon1_cfg_t  ppcon1;     /* Output effect configuration structure 1 */

    ppcon2_cfg_t  ppcon2;     /* Output effect configuration structure 2 */

    refresh_cfg_t refresh_cfg; /* Output refresh configuration structure */

    uint32_t      panel_type; /* Display device type, 0: HDMI, 1: display screen */

    uint32_t      rotate;     /* Display whether to rotate, 0: not rotate, 1 rotates 90 ° clockwise */

    uint32_t      user_control_disp;

    /* Display whether the video is controlled by users, 0: controlled by the underlying driver, 1: controlled by users */
} output_cfg_t;

```

Output amplification configuration structure descriptions:

```

typedef struct _upscaling_cfg_t {

    uint32_t      enable;      /* Whether to enable the amplification function, 0: disabled, 1: enabled */

    uint32_t      src_width;   /* Source image width */

    uint32_t      src_height;  /* Source image height */

    uint32_t      tgt_width;   /* Image width after magnification */

    uint32_t      tgt_height;  /* Image height after magnification */

    uint32_t      step_x;

    /* X-direction magnification scale, 4096 * (src_width - 1) / (tgt_width - 1) */

    uint32_t      step_y;

```

```

/* y-direction magnification scale, 4096 * (src_height - 1) / (tat_height - 1) */

uint32_t      pos_x;          /* the starting x coordinate of up-scaling image */
uint32_t      pos_y;          /* the starting y coordinate of up-scaling image */

} upscaling_cfg_t;

descriptions of video layer image format enumeration:

enum format_yuv_e {

    FORMAT_YUV422_UYVY      = 0,   /* UYVY Interleaved YUV422 */
    FORMAT_YUV422_VYUY      = 1,   /* VYUY Interleaved YUV422 */
    FORMAT_YUV422_YVYU      = 2,   /* YUYV Interleaved YUV422 */
    FORMAT_YUV422_YUYV      = 3,   /* YYUV Interleaved YUV422 */
    FORMAT_YUV422SP_UV      = 4,   /* UV Semi-planar YUV422 */
    FORMAT_YUV422SP_VU      = 5,   /* VU Semi-planar YUV422 */
    FORMAT_YUV420SP_UV      = 6,   /* UV Semi-planar YUV420 */
    FORMAT_YUV420SP_VU      = 7,   /* VU Semi-planar YUV420 */
    FORMAT_YUV422P_UV       = 8,   /* Planar YUV422 (YU YV) */
    FORMAT_YUV422P_VU       = 9,   /* Planar YUV422 (YV YU) */
    FORMAT_YUV420P_UV       = 10,  /* Planar YUV420 (YU YV) */
    FORMAT_YUV420P_VU       = 11,  /* Planar YUV420 (YV YU) */

};

```

Graphic layer format enumeration Descriptions:

```

enum format_rgb_e {

    FORMAT_8BPP           = 0,   /* 8-bit per pixel */
    FORMAT_RGB565          = 1,
    FORMAT_RGB888          = 2,   /* unpacked RGB888 */
    FORMAT_RGB888P         = 3,   /* packed RGB888 */
    FORMAT_ARGB8888        = 4,   /* ARGB8888 */
    FORMAT_RGBA8888        = 5,   /* RGBA8888 */

};

```

Output mode enumeration Descriptions:

```

enum output_mode_e {

    OUTPUT_MIPI_DSI     = 0,   /* mipi dsi output */
    OUTPUT_BT1120        = 1,   /* bt1120 output */
    OUTPUT_RGB            = 2,   /* rgb output */
    OUTPUT_BT656          = 3,   /* bt656 output */
    OUTPUT_IPI            = 4,   /* ipi output */

};

```

Display type enumeration descriptions (X2/J2):

```

enum DISPLAY_ADDR_TYPE {

    BASE, //0    /* IPU original image */
    CROP, //1    /* Images cropped by IPU */
    SCALE, //2    /* Images scaled by IPU */
    DS0, //3 /* Input channel 0 pym down scale, Layer 0 */
    DS1, //4 /* Input channel 0 pym down scale, Layer 1 */
    DS2, //5 /* Input channel 0 pym down scale, Layer 2 */

};

```



```
DS3, //6 /* Input channel 0 pym down scale, Layer 3 */  
DS4, //7 /* Input channel 0 pym down scale, Layer 4 */  
DS5, //8 /* Input channel 0 pym down scale, Layer 5 */  
DS6, //9 /* Input channel 0 pym down scale, Layer 6 */  
DS7, //10 /*Input channel 0 pym down scale, Layer 7 */  
DS8, //11 /*Input channel 0 pym down scale, Layer 8 */  
DS9, //12 /*Input channel 0 pym down scale, Layer 9 */  
DS10, //13 /*Input channel 0 pym down scale, Layer 10 */  
DS11, //14 /*Input channel 0 pym down scale, Layer 11 */  
DS12, //15 /*Input channel 0 pym down scale, Layer 12 */  
DS13, //16 /*Input channel 0 pym down scale, Layer 13 */  
DS14, //17 /*Input channel 0 pym down scale, Layer 14 */  
DS15, //18 /*Input channel 0 pym down scale, Layer 15 */  
DS16, //19 /*Input channel 0 pym down scale, Layer 16 */  
DS17, //20 /*Input channel 0 pym down scale, Layer 17 */  
DS18, //21 /*Input channel 0 pym down scale, Layer 18 */  
DS19, //22 /*Input channel 0 pym down scale, Layer 19 */  
DS20, //23 /*Input channel 0 pym down scale, Layer 20 */  
DS21, //24 /*Input channel 0 pym down scale, Layer 21 */  
DS22, //25 /*Input channel 0 pym down scale, Layer 22 */  
DS23, //26 /*Input channel 0 pym down scale, Layer 23 */  
US0, //27 /*Input channel 0 pym up scale, Layer 0 */  
US1, //28 /*Input channel 0 pym up scale, Layer 1 */  
US2, //29 /*Input channel 0 pym up scale, Layer 2 */  
US3, //30 /*Input channel 0 pym up scale, Layer 3 */  
US4, //31 /*Input channel 0 pym up scale, Layer 4 */  
US5, //32 /*Input channel 0 pym up scale, Layer 5 */  
DS_2_0, //33 /*Input channel 1 pym down scale, Layer 0 */  
DS_2_1, //34 /*Input channel 1 pym down scale, Layer 1 */  
DS_2_2, //35 /*Input channel 1 pym down scale, Layer 2 */  
DS_2_3, //36 /*Input channel 1 pym down scale, Layer 3 */  
DS_2_4, //37 /*Input channel 1 pym down scale, Layer 4 */  
DS_2_5, //38 /*Input channel 1 pym down scale, Layer 5 */  
DS_2_6, //39 /*Input channel 1 pym down scale, Layer 6 */  
DS_2_7, //40 /*Input channel 1 pym down scale, Layer 7 */  
DS_2_8, //41 /*Input channel 1 pym down scale, Layer 8 */  
DS_2_9, //42 /*Input channel 1 pym down scale, Layer 9 */  
DS_2_10, //43 /*Input channel 1 pym down scale, Layer 10 */  
DS_2_11, //44 /*Input channel 1 pym down scale, Layer 11 */  
DS_2_12, //45 /*Input channel 1 pym down scale, Layer 12 */  
DS_2_13, //46 /*Input channel 1 pym down scale, Layer 13 */  
DS_2_14, //47 /*Input channel 1 pym down scale, Layer 14 */  
DS_2_15, //48 /*Input channel 1 pym down scale, Layer 15 */  
DS_2_16, //49 /*Input channel 1 pym down scale, Layer 16 */
```

```

DS_2_17, //50 /*Input channel 1 pym down scale, Layer 17 */
DS_2_18, //51 /*Input channel 1 pym down scale, Layer 18 */
DS_2_19, //52 /*Input channel 1 pym down scale, Layer 19 */
DS_2_20, //53 /*Input channel 1 pym down scale, Layer 20 */
DS_2_21, //54 /*Input channel 1 pym down scale, Layer 21 */
DS_2_22, //55 /*Input channel 1 pym down scale, Layer 22 */
DS_2_23, //56 /*Input channel 1 pym down scale, Layer 23 */
US_2_0, //57 /*Input channel 1 pym up scale, Layer 0 */
US_2_1, //58 /*Input channel 1 pym up scale, Layer 1 */
US_2_2, //59 /*Input channel 1 pym up scale, Layer 2 */
US_2_3, //60 /*Input channel 1 pym up scale, Layer 3 */
US_2_4, //61 /*Input channel 1 pym up scale, Layer 4 */
US_2_5, //62 /*Input channel 1 pym up scale, Layer 5 */
}


```

Display type enumeration descriptions (X3/J3):

```

enum XJ3_DISPLAY_TYPE {
    DISPLAY_CHANNEL0 = 1,          /* ipu channel 0 */
    DISPLAY_CHANNEL1 = 2,          /* ipu channel 1 */
    DISPLAY_CHANNEL2 = 3,          /* ipu channel 2 */
    DISPLAY_CHANNEL3 = 4,          /* ipu channel 3 */
    DISPLAY_CHANNEL4 = 5,          /* ipu channel 4 */
    DISPLAY_CHANNEL5 = 6,          /* ipu channel 5 */
    DS0 = 7, /* pym down scale, Layer 0 */
    DS1 = 8, /* pym down scale, Layer 1 */
    DS2 = 9, /* pym down scale, Layer 2 */
    DS3 = 10, /* pym down scale, Layer 3 */
    DS4 = 11, /* pym down scale, Layer 4 */
    DS5 = 12, /* pym down scale, Layer 5 */
    DS6 = 13, /* pym down scale, Layer 6 */
    DS7 = 14, /* pym down scale, Layer 7 */
    DS8 = 15, /* pym down scale, Layer 8 */
    DS9 = 16, /* pym down scale, Layer 9 */
    DS10 = 17, /* pym down scale, Layer 10 */
    DS11 = 18, /* pym down scale, Layer 11 */
    DS12 = 19, /* pym down scale, Layer 12 */
    DS13 = 20, /* pym down scale, Layer 13 */
    DS14 = 21, /* pym down scale, Layer 14 */
    DS15 = 22, /* pym down scale, Layer 15 */
    DS16 = 23, /* pym down scale, Layer 16 */
    DS17 = 24, /* pym down scale, Layer 17 */
    DS18 = 25, /* pym down scale, Layer 18 */
    DS19 = 26, /* pym down scale, Layer 19 */
    DS20 = 27, /* pym down scale, Layer 20 */
    DS21 = 28, /* pym down scale, Layer 21 */
}

```

```

DS22 = 29,      /* pym down scale, Layer 22 */
DS23 = 30,      /* pym down scale, Layer 23 */
US0 = 31,       /* pym up scale, Layer 0 */
US1 = 32,       /* pym up scale, Layer 1 */
US2 = 33,       /* pym up scale, Layer 2 */
US3 = 34,       /* pym up scale, Layer 3 */
US4 = 35,       /* pym up scale, Layer 4 */
US5 = 36,       /* pym up scale, Layer 5 */
GDC0 = 37,      /* 1280*720 image that is rotated from 720*1280 image using GDC */
GDC1 = 38,      /* 1920*1080 image that is rotated from 1080*1920 image using GDC */
PIPELINE0 = 39, /* camera channel 0 */
PIPELINE1 = 40, /* camera channel 0 */
PIPELINE2 = 41, /* camera channel 0 */
PIPELINE3 = 42, /* camera channel 0 */
}

```

6.4 DISP Configuration File Descriptions

The following example is only compatible with system version 1.2. For VIO configuration file descriptions in system version 1.1, please refer to the following:

```

"channel0_config":      /* Video layer 0 (layer 0, channel 0) configuration */
{
    "ch_en": 1,          /* Channel enable configuration, 0: disable, 1: enable*/
    "pri": 2,            /* Channel priority, 0~3*/
    "src_width": 800,   /* Input image width */
    "src_height": 480,  /* Input image height */
    "dis_width": 800,   /* Output image width */
    "dis_height": 480,  /* Output image height */
    "format": 6,         /* Input image format */
    "xpos": 0,           /* The horizontal distance between the upper left corner of the image display and the
upper left corner of the panel */
    "ypos": 0,           /* The vertical distance between the upper left corner of the image display and the upper
left corner of the panel */
    "alpha_sel": 0,
    "overmode": 0,        /* Layer overlay mode */
    "alpha_en": 1,         /* Whether to enable the alpha blending function, 0: disable, 1: enable*/
    "alpha": 255,          /*alpha value (0~255), 0: Full transparent, 255: full opacity */
    "key-color": 0,        /*key color value*/
    "crop_width": 800,    /*Cropping image width*/
    "crop_height": 480     /*Cropping image height*/
},
"channel1_config":      /* Video layer 1 (layer1, channel 1) configuration */
{
}

```

```
"ch_en": 0,  
"pri": 3,  
"src_width": 0,  
"src_height": 0,  
"dis_width": 0,  
"dis_height": 0,  
"format": 6,  
"xpos": 0,  
"ypos": 0,  
"alpha": 128,  
"alpha_sel": 0,  
"overmode": 0,  
"alpha_en": 1,  
"key-color": 0,  
"crop_width": 800,  
"crop_height": 480  
},  
"channel2_config": /* Graphic layer 0 (layer 2, channel 2) configuration */  
{  
    "ch_en": 1,  
    "pri": 0,  
    "src_width": 800,  
    "src_height": 480,  
    "dis_width": 800,  
    "dis_height": 480,  
    "format": 4,  
    "xpos": 0,  
    "ypos": 0,  
    "alpha": 128,  
    "alpha_sel": 0,  
    "overmode": 0,  
    "alpha_en": 1,  
    "key-color": 0,  
    "crop_width": 800,  
    "crop_height": 480  
},  
"channel3_config": /* Graphic layer 1 (layer 3, channel 3) configuration */  
{  
    "ch_en": 0,  
    "pri": 1,  
    "src_width": 0,  
    "src_height": 0,  
    "dis_width": 0,  
    "dis_height": 0,
```

```

        "format": 4,
        "xpos": 0,
        "ypos": 0,
        "alpha": 128,
        "alpha_sel": 0,
        "overmode": 0,
        "alpha_en": 1,
        "key-color": 0,
        "crop_width": 800,
        "crop_height": 480
    },
    "upscale_config": /* Amplification function configuration */
    {
        "scale_en": 0, /*Whether to enable the amplification function, 0: disable, 1: enable*/
        "src_size_width": 800, /* Width of source image */
        "src_size_height": 480, /* Height of source image */
        "tgt_size_width": 800, /* Width of target image */
        "tgt_size_height": 480 /* Height of target image */
    },
    "outputconfig": /* Output configuration */
    {
        "output_mode": 2, /* Output mode */
        "panel_width": 800, /*panel width*/
        "panel_height": 480, /*panel height*/
        "display_addr_type": 8, /* VIO layer to be displayed */
        "display_cam_no": 0, /* Camera to be displayed */
        "backlight_level": 10, /* Backlight brightness */
        "panel_type": 1, /* Display device type. None for X3 */
        "bg_color": 16744328, /* Background layer color */
        "contrast_en": 0, /* Whether to enable the contrast adjustment function */
        "contrast_val": 0, /* Contrast value */
        "bright_en": 0, /* Whether to enable the brightness adjustment */
        "bright_val": 0, /* Brightness value */
        "off_contrast": 0, /* Contrast offset value */
        "sat_en": 0, /* Whether to enable the saturation adjustment function */
        "sat_val": 0, /* Saturation value */
        "hue_en": 0, /*Whether to enable the hue adjustment function*/
        "theta_sign": 0, /*theta sign*/
        "theta_abs": 0, /*theta absolute value*/
        "dithering_en": 0, /*Whether to enable the dithering adjustment function*/
        "dithering_flag": 0, /*dithering value*/
        "gamma_en": 0, /*Whether to enable the gamma adjustment function */
        "gamma_val": 0, /*gamma value*/
    }
}

```

```
        "user_refresh": 0,      /* Display whether the video stream is controlled by users, 1: controlled by users*/
        "rotate": 0,           /* Display whether to rotate, 1: rotation */
        "platform": 1,         /* Platform, 0: X2/I2, 1: X3/I3
        "big_endian": 0       /* End configuration of image format size of the graphic layer, 0: small end, 1: large
    end */
}
```

Horizon Robotics
版权所有 禁止转载 算法工具部 杨帆 2021-04-01 10:02:46

7 Appendix

Horizon Robotics
版权所有 禁止转载 算法工具部 杨杭军 2021-04-01 10:02:46