1. IVE

1.1. Feature

- General filter: 7x7 filter
- Median filter: 3x3 min, max, median filter
- Edge filter: output the edge strength and the direction
- Non-maximal suppression: suppress edges to 0 except the local maximal
- Threshold LUT:
  - 15 taps edge threshold LUT
  - Output: 1/4 bit threshold
- Morphological operation:
  - Dilation / Erosion
  - User-defined structuring element
- Depth processing: for depth related application
- Integral image: Multiple stripe integral
  - Stripe size = 640 pixels
- Maximal width, height: 16383 x 8191 pixels for all features

1.2. Block Diagram

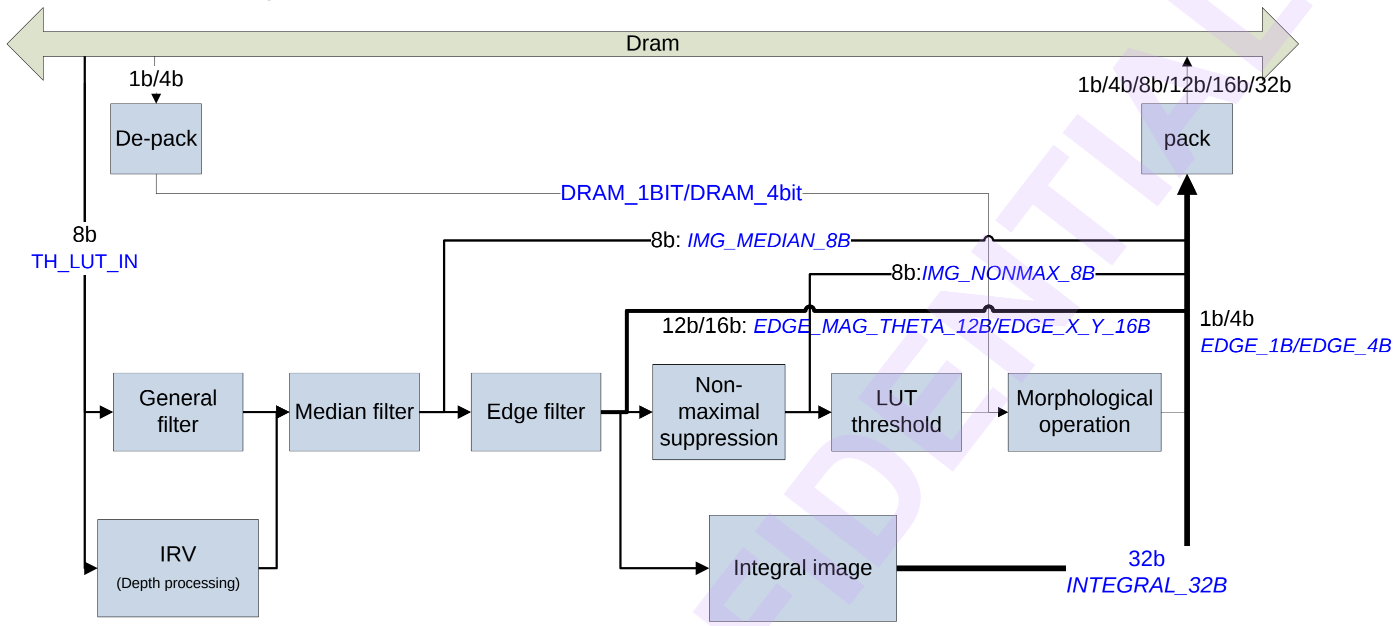


Figure 1-2-1 Function block diagram

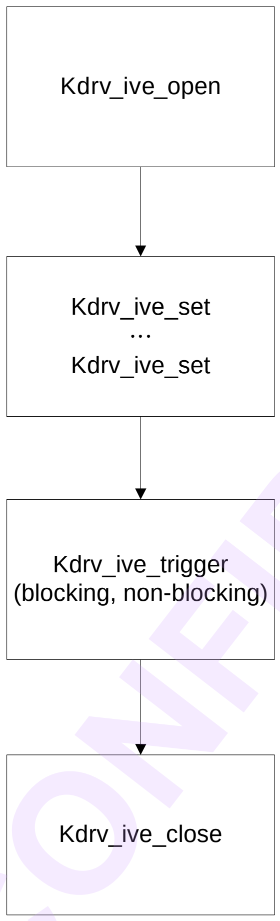


Figure 1-2-2 IVE kdriver control flow.

1.3. Header File

The header file is located in .\code\hdal\vendor\cv\drivers\include\kdrv_ive\kdrv_ive.h

Pleace #include “kdrv_ive.h” in your source code for IVE module.

1.4. Function Description

1.4.1. Function control

- General Description

There are 8 sub-functions in IVE. Some of them are independent, like median filter and edge filter. Others use shared resource and must be enabled separately, like General filter and IRV.

Sub-functions from general filter to morphological filter implement the “Canny Edge Filter”. Other sub-functions are some popular applications for CV.

<u>Input path selection (MorphInSel):</u> There are 3 kinds of input path user can choose. The first one is loading the input data from the function "general filter", and the rest of the 2 are loading the input data from function "morphological filter" directly. Note that when function "morphological filter" is disabled or any function before it is enabled, the input selection must be TH_LUT_IN; when selecting the input as DRAM_4BIT or DRAM_1BIT, all functions except 'morphological filter' should be disabled.

1. ***TH_LUT_IN,***
2. ***DRAM_4BIT,***
3. ***DRAM_1BIT,***

<u>Output path selection (OutDataSel):</u> The output data type selection is done by kdriver according to the function user enabled. All output data types are listed below for your reference.

1. ***EDGE_1B***, 1 bit edge (after morphological operation)
2. ***EDGE_4B***, 4 bit edge (after morphological operation)
3. ***IMG_MEDIAN_8B***, 8 bit image (after median filter)
4. ***IMG_NONMAX_8B***, 8 bit image (after non-maximal suppression)
5. ***EDGE_MAG_THETA_12B,*** 12 bit (after edge filter, 8 bit edge magnitude + 4 bit theta pack)
6. ***EDGE_X_Y_16B***, 16 bit (after edge filter, 8 bit gradient X + 8 bit gradient Y pack)
7. ***INTEGRAL_32B***, 32 bit integral image

The followings are some limitations in IVE's function control. Please refer to the block diagram in section 1.2.

1. The function 'General filter' and 'Depth processing' cannot be enabled at the same time.
2. When function 'Integral image' is enabled, the output path selection (OutDataSel) should be ***INTEGRAL_32B.*** Moreover, the function 'Non-maximal suppression', 'Threshold LUT', and 'Morphological operation' must be disabled.
3. When the output path selection (OutDataSel) is ***EDGE_1B / EDGE_4B***, the function 'integral image' must be disabled.
4. When the output path selection (OutDataSel) is ***IMG_MEDIAN_8B***, the function 'Edge filter', 'Non-maximal suppression', 'Threshold LUT', 'Morphological operation', and 'integral image' must be disabled.
5. When the output path selection (OutDataSel) is ***IMG_NONMAX_8B***, the function 'Threshold LUT', 'Morphological operation', and 'integral image' must be disabled.
6. When the output path selection (OutDataSel) is ***EDGE_MAG_THETA_12B*** or

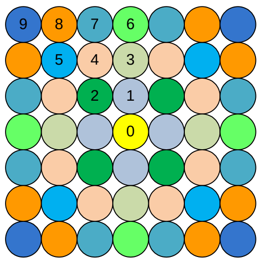
EDGE_X_Y_16B, the function 'Non-maximal suppression', 'Threshold LUT', 'Morphological operation', and 'integral image' must be disabled.

1.4.2. General filter

- General Description

IVE can pass a user-defined 7x7 kernel to filter an image. This filter is symmetric and there are 10 coefficients to be filled, which are 4bit unsigned coefficients. The filter will be normalized automatically.

Kernel:



Example:

$$Out = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 4 & 5 & 4 & 2 & 0 \\ 0 & 4 & 9 & 12 & 9 & 4 & 0 \\ 0 & 5 & 12 & 15 & 12 & 5 & 0 \\ 0 & 4 & 9 & 12 & 9 & 4 & 0 \\ 0 & 2 & 4 & 5 & 4 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \bullet In * \frac{1}{159}$$

- Limitation

1. The function cannot be enabled with function 'Depth Processing' at the same time.

1.4.3. Median filter

- General Description

This function supports a 3x3 filter to filter an image. There are 3 modes, which are median filter, min filter, and max filter.

- Limitation

None

1.4.4. Edge filter

- General Description

This function outputs the magnitude of the edge filter and the tag of the edge direction. There are 2 kinds of filters to be operated, which are ***BI_DIR*** and ***NO_DIR***. The first one is bi-directional 1st order edge filter (ex: Sobel filter) and the other one is non-directional 2nd order edge filter (ex: Laplacian filter).

The coefficients of the filter can be specified by the users and the following is an example.

Bi-directional:

Filter option 1: Sobel filter
(1st order)

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Non-directional:

Filter option 2: Laplacian filter
(2nd order)

$$Lap = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- Limitation
  1. When the mode of edge filter is ***NO_DIR***, the output selection '***EDGE_MAG_THETA_12B***' and '***EDGE_X_Y_16B***' are not allowed.

1.4.5. Non-maximal suppression

- General Description

By using the outputs of function 'edge filter', which are the edge magnitude and the direction angle, this function performs the non-maximal suppression to suppress all the gradient values to 0 except the local maximal.

- Limitation

None

1.4.6. Threshold LUT

- General Description

This function supports a threshold table (LUT) for image thesholding. When the output image is selected to be 4 bit image, the function performs the thresholding by a 15 taps LUT; when the output image is selected to be 1 bit, this function performs the thresholding by a single threshold value.

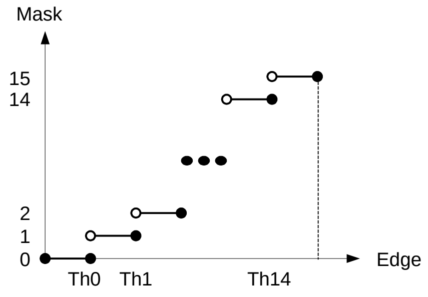


Table should be strictly increasing

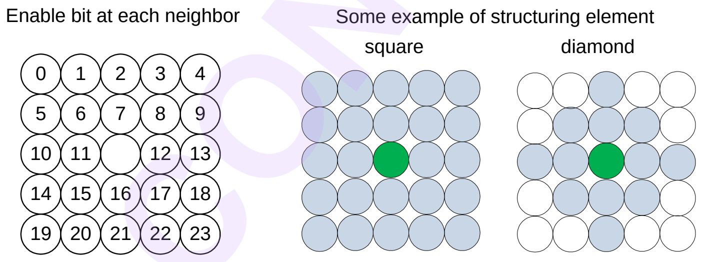
- Limitation

  None

1.4.7. Morphological operation

- General Description

  This function supports the morphological operation including dilation and erosion. The structuring element can be specified by enabling each bit of the 5x5 mask.



The input of this function can be passed from dram or from the previous function 'threshold LUT' (sec.1.4.7).

- Limitation

  None

1.4.8. Depth processing

- General Description

  This function is mainly for depth related application.

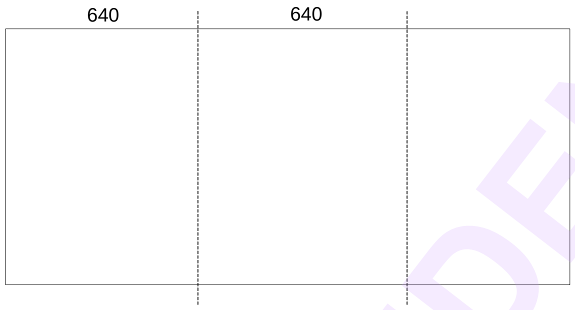
- Limitation

  1. This function cannot enable with 'General filter' at the same time.

1.4.9. Integral image

- General Description

This function performs the multiple stripe integral image, which means the value of the integral image will be reset at the starting x of each stripe. The width of each stripe is 640 pixels.



- Limitation

  1. When function 'Integral image' is enabled, the output path selection (OutDataSel) should be INTEGRAL_32B. Moreover, the function 'Non-maximal suppression', 'Threshold LUT', and 'Morphological operation' must be disabled.

1.4.10. D2D mode

- General Description

The engine starts when triggered and stops after every trigger. The engine reads one image from input address and outputs one image to output address.

- Limitation

None

- Flow

Here is an example of D2D process with general filter enabled. Please call kdrv_ive_open() before any function setting. To set desired function, call kdrv_ive_set with the correct function enum. To trigger engine, please call kdrv_ive_trigger(). After trigger is done, call kdrv_ive_close() to close current process.

```
KDRV_IVE_OPENCFG ive_open_obj ;

KDRV_IVE_IN_IMG_INFO in_img_info = {0} ;

KDRV_IVE_GENERAL_FILTER_PARAM gen_filt = {0} ;
```

```
KDRV_IVE_IMG_IN_DMA_INFO in_addr = {0} ;
KDRV_IVE_IMG_OUT_DMA_INFO out_addr = {0} ;
KDRV_IVE_TRIGGER_PARAM trig_param = {0} ;

/******** ive d2d flow *********/
DBG_IND(« ive d2d flow start\n ») ;
/*open*/
ive_open_obj.ive_clock_sel = 480 ;
kdrv_ive_set(id, KDRV_IVE_PARAM_IPL_OPENCFG, (void *)&ive_open_obj) ;
if(kdrv_ive_open(chip, engine) != 0) {
     DBG_ERR(« set opencfg fail !\r\n ») ;
     goto EOP ;
}

/*set input img parameters*/
in_img_info.width= 2016 ;
in_img_info.height = 1512 ;
in_img_info.lineofst = ((in_w + 3) / 4) * 4 ;
kdrv_ive_set(id, KDRV_IVE_PARAM_IPL_IN_IMG, (void *)&in_img_info) ;
DBG_IND(« KDRV_IVE_PARAM_IPL_IN_IMG\r\n ») ;

/*set input address*/
in_addr.addr = (UINT32)yin_buf_info.va_addr ;
kdrv_ive_set(id, KDRV_IVE_PARAM_IPL_IMG_DMA_IN, (void *)&in_addr) ;
DBG_IND(« KDRV_IVE_PARAM_IPL_IMG_DMA_IN\r\n ») ;

/*set output address*/
out_addr.addr = (UINT32)yout_buf_info.va_addr ;
kdrv_ive_set(id, KDRV_IVE_PARAM_IPL_IMG_DMA_OUT, (void *)&out_addr) ;
DBG_IND(« KDRV_IVE_PARAM_IPL_IMG_DMA_OUT\r\n ») ;

/*set general filter*/
gen_filt.enable = TRUE ;
```

```
for (i = 0 ; i < GEN_COEFF_NUM ; i++) {
     gen_filt.filt_coeff[i] = g_gen_filter_coeff[i] ;
}
gen_filt.filt_norm = 159 ;
kdrv_ive_set(id, KDRV_IVE_PARAM_IQ_GENERAL_FILTER, (void *)&gen_filt) ;
DBG_IND(« KDRV_IVE_PARAM_IQ_GENERAL_FILTER\r\n ») ;

/*set call back funcion*/
kdrv_ive_fmd_flag = 0 ;
kdrv_ive_set(id, KDRV_IVE_PARAM_IPL_ISR_CB, (void *)&KDRV_IVE_ISR_CB) ;
DBG_IND(« KDRV_IVE_PARAM_IPL_ISR_CB\r\n ») ;

/*trig ive start*/
DBG_IND(« kdrv_ive_trigger\r\n ») ;
trig_param.wait_end = TRUE ;//true : blocking, false : non-blocking
trig_param.time_out_ms = 100 ;
kdrv_ive_trigger(id , &trig_param, NULL, NULL) ;
DBG_IND(« trigger end !\r\n ») ;

/*wait end*/
if ( !trig_param.wait_end) {
     while( !kdrv_ive_fmd_flag) {
          DBG_IND(« frame end !\n ») ;
          msleep(10) ;
     }
}

/*close*/
kdrv_ive_close(chip, engine) ;
DBG_IND(« ive d2d flow end\n ») ;
/******** ive d2d flow *********/
```

1.5. Data Structure

1.5.1. KDRV_IVE_OPENCFG

```
typedef struct _KDRV_IVE_OPENCFG {
     UINT32 ive_clock_sel;    //Engine clock selection
} KDRV_IVE_OPENCFG, *pKDRV_IVE_OPENCFG;
```

1.5.2. KDRV_IVE_PARAM_ID

```
typedef enum _KDRV_IVE_PARAM_ID {
     KDRV_IVE_PARAM_IPL_OPENCFG,             // [Set]  , use KDRV_IVE_OPENCFG, set clock info
     KDRV_IVE_PARAM_IPL_IN_IMG,              // [Set/Get], use KDRV_IVE_IN_IMG_INFO structure, set
input image size info, line offset
     KDRV_IVE_PARAM_IPL_IMG_DMA_IN,          // [Set/Get], use KDRV_IVE_IMG_IN_DMA_INFO structure, set
the dram input address
     KDRV_IVE_PARAM_IPL_IMG_DMA_OUT,         // [Set/Get], use KDRV_IVE_IMG_OUT_DMA_INFO structure,
set the dram output address
     KDRV_IVE_PARAM_IPL_ISR_CB,              // [Set/Get], use KDRV_IPP_ISRCB structure, set ive
external isr cb
     KDRV_IVE_PARAM_IQ_GENERAL_FILTER,  // [Set/Get], use KDRV_IVE_GENERAL_FILTER_PARAM structure, set
general filter parameters
     KDRV_IVE_PARAM_IQ_MEDIAN_FILTER,   // [Set/Get], use KDRV_IVE_MEDIAN_FILTER_PARAM structure, set
median filter parameters
     KDRV_IVE_PARAM_IQ_EDGE_FILTER,          // [Set/Get], use KDRV_IVE_EDGE_FILTER_PARAM structure,
set edge filter parameters
     KDRV_IVE_PARAM_IQ_NON_MAX_SUP,          // [Set/Get], use KDRV_IVE_NON_MAX_SUP_PARAM structure,
set non-maximal supression parameters
     KDRV_IVE_PARAM_IQ_THRES_LUT,       // [Set/Get], use KDRV_IVE_THRES_LUT_PARAM structure, set edge
threshold LUT parameters
     KDRV_IVE_PARAM_IQ_MORPH_FILTER,         // [Set/Get], use KDRV_IVE_MORPH_FILTER_PARAM structure,
set morphological filter parameters
     KDRV_IVE_PARAM_IQ_INTEGRAL_IMG,         // [Set/Get], use KDRV_IVE_INTEGRAL_IMG_PARAM structure,
set integral image parameters
     KDRV_IVE_PARAM_IQ_ITER_REGION_VOTE,     // [Set/Get], use KDRV_IVE_ITER_REG_VOTE_PARAM
sturcture, set iterative region voting parameters
```

```
     KDRV_IVE_PARAM_MAX,

     KDRV_IVE_PARAM_REV = 0x80000000,

     ENUM_DUMMY4WORD(KDRV_IVE_PARAM_ID)

} KDRV_IVE_PARAM_ID;
```

1.5.3. KDRV_IVE_IMG_IN_DMA_INFO, KDRV_IVE_IMG_OUT_DMA_INFO

```
typedef struct _KDRV_IVE_IMG_DMA_INFO {

     UINT32 addr;   //DMA address of Y channel

} KDRV_IVE_IMG_IN_DMA_INFO, KDRV_IVE_IMG_OUT_DMA_INFO;
```

1.5.4. KDRV_IVE_IN_IMG_INFO

```
typedef struct _KDRV_IVE_IN_IMG_INFO {

     UINT32 width;       //image width

     UINT32 height;      //image height

     UINT32 lineofst;    //DRAM line offset of input channel

} KDRV_IVE_IN_IMG_INFO;
```

1.5.5. KDRV_IVE_GENERAL_FILTER_PARAM

```
typedef struct _KDRV_IVE_GENERAL_FILTER_PARAM {

     BOOL enable;

     UINT32 filt_coeff[IVE_GEN_FILT_NUM];

     UINT32 filt_norm;

} KDRV_IVE_GENERAL_FILTER_PARAM;
```

1.5.6. KDRV_IVE_MEDIAN_FILTER_PARAM

```
typedef enum {

     KDRV_IVE_MEDIAN = 0,     //find median value in the filter

     KDRV_IVE_MAX = 1,   //find the maximum in the filter

     KDRV_IVE_MIN = 2,   //find the minimum in the filter

     KDRV_IVE_MEDIAN_W_INVAL = 3,

     ENUM_DUMMY4WORD(KDRV_IVE_MEDN_MODE)

} KDRV_IVE_MEDN_MODE;
```

```
typedef struct _KDRV_IVE_MEDIAN_FILTER_PARAM {
     BOOL enable;
     KDRV_IVE_MEDN_MODE mode;
} KDRV_IVE_MEDIAN_FILTER_PARAM;
```

1.5.7. KDRV_IVE_EDGE_FILTER_PARAM

```
typedef enum {
     KDRV_IVE_BI_DIR = 0,     //use 2 edge filters of X & Y direction to calculate X & Y edge response
separately
     KDRV_IVE_NO_DIR = 1,     //use 1 edge filter without direction to calculate edge response
     ENUM_DUMMY4WORD(KDRV_IVE_EDGE_MODE)
} KDRV_IVE_EDGE_MODE;

typedef struct _KDRV_IVE_EDGE_FILTER_PARAM {
     BOOL enable;
     KDRV_IVE_EDGE_MODE mode;
     UINT32 edge_coeff1[IVE_EDGE_COEFF_NUM];
     UINT32 edge_coeff2[IVE_EDGE_COEFF_NUM];
     UINT32 edge_shift_bit;
} KDRV_IVE_EDGE_FILTER_PARAM;
```

1.5.8. KDRV_IVE_MORPH_FILTER_PARAM

```
typedef enum {
     KDRV_IVE_DILATE = 0,     //do dilation
     KDRV_IVE_ERODE = 1, //do erosion
     ENUM_DUMMY4WORD(KDRV_IVE_MORPH_OP)
} KDRV_IVE_MORPH_OP;

typedef enum {
     KDRV_IVE_TH_LUT_IN = 0, //4 bit input from threshold LUT
     KDRV_IVE_DRAM_4BIT = 1, //4 bit input from dram directly
     KDRV_IVE_DRAM_1BIT = 2, //1 bit input from dram directly
     ENUM_DUMMY4WORD(KDRV_IVE_MORPH_IN_SEL)
} KDRV_IVE_MORPH_IN_SEL;
```

```
typedef struct _KDRV_IVE_MORPH_FILTER_PARAM {
     BOOL enable;
     KDRV_IVE_MORPH_IN_SEL in_sel;
     KDRV_IVE_MORPH_OP operation;
     BOOL neighbor[IVE_MORPH_NEIGH_NUM];
} KDRV_IVE_MORPH_FILTER_PARAM;
```

1.5.9. KDRV_IVE_NON_MAX_SUP_PARAM

```
typedef struct _KDRV_IVE_NON_MAX_SUP_PARAM {
     BOOL enable;
     UINT32 mag_thres;
} KDRV_IVE_NON_MAX_SUP_PARAM;
```

1.5.10. KDRV_IVE_THRES_LUT_PARAM

```
typedef struct _KDRV_IVE_THRES_LUT_PARAM {
     BOOL enable;
     UINT32 thres_lut[IVE_EDGE_TH_LUT_TAP];
} KDRV_IVE_THRES_LUT_PARAM;;
```

1.5.11. KDRV_IVE_INTEGRAL_IMG_PARAM

```
typedef struct _KDRV_IVE_INTEGRAL_IMG_PARAM {
     BOOL enable;
} KDRV_IVE_INTEGRAL_IMG_PARAM;
```

1.5.12. KDRV_IVE_ITER_REG_VOTE_PARAM

```
typedef enum {
     KDRV_IVE_HIST_7X7 = 0,
     KDRV_IVE_HIST_5X5 = 1,
     ENUM_DUMMY4WORD(KDRV_IVE_HIST_MODE)
} KDRV_IVE_HIST_MODE;

typedef enum {
     KDRV_IVE_INVAL_0 = 0,
     KDRV_IVE_INVAL_63 = 1,
     ENUM_DUMMY4WORD(KDRV_IVE_IRV_INVAL_VAL)
} KDRV_IVE_IRV_INVAL_VAL;

typedef struct _KDRV_IVE_ITER_REG_VOTE_PARAM {
     BOOL enable;
     KDRV_IVE_HIST_MODE hist_mode;
     KDRV_IVE_IRV_INVAL_VAL invalid_value;
     UINT32 sum_ratio_thres;
     UINT32 freq_ratio_thres;
     UINT32 median_invalid_thres;
} KDRV_IVE_ITER_REG_VOTE_PARAM;
```

1.6. Ive user space interface

1.6.1. Data structure

.1.6.1.1. VENDOR_IVE_TRIGGER_PARAM

```
typedef struct _VENDOR_IVE_TRIGGER_PARAM {
     BOOL    wait_end;
     UINT32    time_out_ms;   ///< force time out when wait_end == TRUE, set 0 for disable time out
} VENDOR_IVE_TRIGGER_PARAM;
```

.1.6.1.2. VENDOR_IVE_OPENCFG

```
typedef struct _VENDOR_IVE_OPENCFG {
     UINT32 ive_clock_sel;    //Engine clock selection
} VENDOR_IVE_OPENCFG;
```

.1.6.1.3. VENDOR_IVE_FUNC

```
typedef enum {
   VENDOR_IVE_INPUT_INFO       = 0,   //input info
     VENDOR_IVE_INPUT_ADDR       = 1,   //input addr
     VENDOR_IVE_OUTPUT_ADDR      = 2,   //output addr
     VENDOR_IVE_GENERAL_FILTER   = 3,   //general filter
```

```
    VENDOR_IVE_MEDIAN_FILTER    = 4,    //median filter
    VENDOR_IVE_EDGE_FILTER      = 5,    //edge filter
    VENDOR_IVE_NON_MAX_SUP      = 6,    //non-maximum suppression
    VENDOR_IVE_THRES_LUT        = 7,    //edge filter
    VENDOR_IVE_MORPH_FILTER     = 8,    //morphology operation
    VENDOR_IVE_INTEGRAL_IMG     = 9,    //integral image
    VENDOR_IVE_ITER_REGION_VOTE = 10,   //iterative region voting
    ENUM_DUMMY4WORD(VENDOR_IVE_FUNC)
} VENDOR_IVE_FUNC;
```

.1.6.1.4. VENDOR_IVE_IN_IMG_INFO

```
typedef struct _VENDOR_IVE_IN_IMG_INFO {
    UINT32 width;       //image width
    UINT32 height;      //image height
    UINT32 lineofst;    //DRAM line offset of input channel
} VENDOR_IVE_IN_IMG_INFO;
```

.1.6.1.5. VENDOR_IVE_IMG_IN_DMA_INFO, VENDOR_IVE_IMG_OUT_DMA_INFO

```
typedef struct _VENDOR_IVE_IMG_DMA_INFO {
    UINT32 addr;     //DMA address of Y channel
} VENDOR_IVE_IMG_IN_DMA_INFO, VENDOR_IVE_IMG_OUT_DMA_INFO;
```

.1.6.1.6. VENDOR_IVE_GENERAL_FILTER_PARAM

```
typedef struct _VENDOR_IVE_GENERAL_FILTER_PARAM {
    BOOL enable;
    UINT32 filt_coeff[VENDOR_IVE_GEN_FILT_NUM];
    UINT32 filt_norm;
} VENDOR_IVE_GENERAL_FILTER_PARAM;
```

.1.6.1.7. VENDOR_IVE_MEDIAN_FILTER_PARAM

```
typedef enum {
    VENDOR_IVE_MEDIAN = 0,    //find median value in the filter
    VENDOR_IVE_MAX = 1,      //find the maximum in the filter
```

```
     VENDOR_IVE_MIN = 2,      //find the minimum in the filter
     VENDOR_IVE_MEDIAN_W_INVAL = 3,
     ENUM_DUMMY4WORD(VENDOR_IVE_MEDN_MODE)
} VENDOR_IVE_MEDN_MODE;

/**
   IVE VENDOR median filter param and Enable/Disable
*/
typedef struct _VENDOR_IVE_MEDIAN_FILTER_PARAM {
     BOOL enable;
     VENDOR_IVE_MEDN_MODE mode;
} VENDOR_IVE_MEDIAN_FILTER_PARAM;
```

.1.6.1.8. VENDOR_IVE_EDGE_FILTER_PARAM

```
typedef enum {
     VENDOR_IVE_BI_DIR = 0,    //use 2 edge filters of X & Y direction to calculate X & Y edge response
separately
     VENDOR_IVE_NO_DIR = 1,    //use 1 edge filter without direction to calculate edge response
     ENUM_DUMMY4WORD(VENDOR_IVE_EDGE_MODE)
} VENDOR_IVE_EDGE_MODE;

/**
   IVE VENDOR edge filter param and Enable/Disable
*/
typedef struct _VENDOR_IVE_EDGE_FILTER_PARAM {
     BOOL enable;
     VENDOR_IVE_EDGE_MODE mode;
     UINT32 edge_coeff1[VENDOR_IVE_EDGE_COEFF_NUM];
     UINT32 edge_coeff2[VENDOR_IVE_EDGE_COEFF_NUM];
     UINT32 edge_shift_bit;
} VENDOR_IVE_EDGE_FILTER_PARAM;
```

.1.6.1.9. VENDOR_IVE_NON_MAX_SUP_PARAM

```
typedef struct _VENDOR_IVE_NON_MAX_SUP_PARAM {
     BOOL enable;
     UINT32 mag_thres;
} VENDOR_IVE_NON_MAX_SUP_PARAM;
```

.1.6.1.10. VENDOR_IVE_THRES_LUT_PARAM

```
typedef struct _VENDOR_IVE_THRES_LUT_PARAM {
     BOOL enable;
     UINT32 thres_lut[VENDOR_IVE_EDGE_TH_LUT_TAP];
} VENDOR_IVE_THRES_LUT_PARAM;
```

.1.6.1.11. VENDOR_IVE_MORPH_FILTER_PARAM

```
typedef enum {
     VENDOR_IVE_DILATE = 0,    //do dilation
     VENDOR_IVE_ERODE = 1,     //do erosion
     ENUM_DUMMY4WORD(VENDOR_IVE_MORPH_OP)
} VENDOR_IVE_MORPH_OP;

typedef enum {
     VENDOR_IVE_TH_LUT_IN = 0, //4 bit input from threshold LUT
     VENDOR_IVE_DRAM_4BIT = 1, //4 bit input from dram directly
     VENDOR_IVE_DRAM_1BIT = 2, //1 bit input from dram directly
     ENUM_DUMMY4WORD(VENDOR_IVE_MORPH_IN_SEL)
} VENDOR_IVE_MORPH_IN_SEL;

/**
   IVE VENDOR morphological filter param and Enable/Disable
*/
typedef struct _VENDOR_IVE_MORPH_FILTER_PARAM {
     BOOL enable;
   VENDOR_IVE_MORPH_IN_SEL in_sel;
     VENDOR_IVE_MORPH_OP operation;
     BOOL neighbor[VENDOR_IVE_MORPH_NEIGH_NUM];
} VENDOR_IVE_MORPH_FILTER_PARAM;
```

.1.6.1.12. VENDOR_IVE_INTEGRAL_IMG_PARAM

```
typedef struct _VENDOR_IVE_INTEGRAL_IMG_PARAM {
     BOOL enable;
} VENDOR_IVE_INTEGRAL_IMG_PARAM;
```

.1.6.1.13. VENDOR_IVE_ITER_REG_VOTE_PARAM

```
typedef enum {
     VENDOR_IVE_HIST_7X7 = 0,
     VENDOR_IVE_HIST_5X5 = 1,
     ENUM_DUMMY4WORD(VENDOR_IVE_HIST_MODE)
} VENDOR_IVE_HIST_MODE;

typedef enum {
     VENDOR_IVE_INVAL_0 = 0,
     VENDOR_IVE_INVAL_63 = 1,
     ENUM_DUMMY4WORD(VENDOR_IVE_IRV_INVAL_VAL)
} VENDOR_IVE_IRV_INVAL_VAL;

/**
   IVE VENDOR iterative region voting param and Enable/Disable
*/
typedef struct _VENDOR_IVE_ITER_REG_VOTE_PARAM {
     BOOL enable;
     VENDOR_IVE_HIST_MODE hist_mode;
     VENDOR_IVE_IRV_INVAL_VAL invalid_value;
     UINT32 sum_ratio_thres;
     UINT32 freq_ratio_thres;
     UINT32 median_invalid_thres;
} VENDOR_IVE_ITER_REG_VOTE_PARAM;
```

1.6.2. User space API

.1.6.2.1. extern INT32 **vendor_ive_init**(void);

intial IVE vendor.

.1.6.2.2. extern INT32 **vendor_ive_uninit**(void);

Close IVE vendor.

.1.6.2.3. extern INT32 **vendor_ive_set_param**(VENDOR_IVE_FUNC param_id, void *p_param);

Set IVE parameters. The following chart shows the param_id and corresponding p_param structure:

param_id	p_param
VENDOR_IVE_INPUT_INFO	VENDOR_IVE_IN_IMG_INFO
VENDOR_IVE_INPUT_ADDR	VENDOR_IVE_IMG_IN_DMA_INFO
VENDOR_IVE_OUTPUT_ADDR	VENDOR_IVE_IMG_OUT_DMA_INFO
VENDOR_IVE_GENERAL_FILTER	VENDOR_IVE_GENERAL_FILTER_PARAM
VENDOR_IVE_MEDIAN_FILTER	VENDOR_IVE_MEDIAN_FILTER_PARAM
VENDOR_IVE_EDGE_FILTER	VENDOR_IVE_EDGE_FILTER_PARAM
VENDOR_IVE_NON_MAX_SUP	VENDOR_IVE_NON_MAX_SUP_PARAM
VENDOR_IVE_THRES_LUT	VENDOR_IVE_THRES_LUT_PARAM
VENDOR_IVE_MORPH_FILTER	VENDOR_IVE_MORPH_FILTER_PARAM
VENDOR_IVE_INTEGRAL_IMG	VENDOR_IVE_INTEGRAL_IMG_PARAM
VENDOR_IVE_ITER_REGION_VOTE	VENDOR_IVE_ITER_REG_VOTE_PARAM

.1.6.2.4. extern INT32 **vendor_ive_get_param**(VENDOR_IVE_FUNC param_id, void *p_param);

Get IVE parameters. The input id and corresponding structures are shown on the previous chart.

.1.6.2.5. extern INT32 **vendor_ive_trigger**(VENDOR_IVE_TRIGGER_PARAM *p_param);

Start IVE engine.

1.6.3. User space sample

```
// init ive module
   ret = vendor_ive_init();
   if (ret != HD_OK) {
        printf("err:vendor_ive_init error %d\r\n",ret);
        goto exit;
   }

   // set input & output info
   ive_input_addr.addr = g_input_mem.addr;
   ive_output_addr.addr = g_output_mem.addr;
   ive_input_info.width = 16;
   ive_input_info.height = 16;
   ive_input_info.lineofst = 16;
   ret = vendor_ive_set_param(VENDOR_IVE_INPUT_INFO, &ive_input_info);
   if (ret != HD_OK) {
        printf("err: ive set input info error %d\n", ret);
        goto exit;
   }
   ret = vendor_ive_set_param(VENDOR_IVE_INPUT_ADDR, &ive_input_addr);
   if (ret != HD_OK) {
        printf("err: ive set input addr error %d\n", ret);
        goto exit;
   }
   ret = vendor_ive_set_param(VENDOR_IVE_OUTPUT_ADDR, &ive_output_addr);
   if (ret != HD_OK) {
        printf("err: ive set output addr error %d\n", ret);
        goto exit;
   }

   // set func enable
```

```
ive_general_filter_param.enable = 0;
ive_median_filter_param.enable  = 0;
ive_edge_filter_param.enable    = 0;
ive_non_max_sup_param.enable    = 0;
ive_thres_lut_param.enable      = 0;
ive_morph_filter_param.enable   = 0;
ive_integral_img_param.enable   = 0;
ive_iter_reg_vote_param.enable  = 0;
ive_trigger_param.time_out_ms   = 0;
ive_trigger_param.wait_end      = 1;




printf("usage:\n");
printf("  enter q: exit\n");
printf("  enter r: run engine\n");
printf("  enter c: clear all enable function\n");
printf("  enter s: check enable status\n");
printf("  enter 1: enable general filter\n");
printf("  enter 2: enable median filter\n");
printf("  enter 3: enable edge filter\n");
printf("  enter 4: enable non-maximum suppresion\n");
printf("  enter 5: enable thres lut\n");
printf("  enter 6: enable morph filter\n");
printf("  enter 7: enable integral img\n");
printf("  enter 8: enable IRV\n");

while (1) {
     key = getchar();
     if (key == 'q') {
          //enc_exit = 1;
          break;
     }
```

```
        else if (key == 'c') {
             ive_general_filter_param.enable = 0;
             ive_median_filter_param.enable  = 0;
             ive_edge_filter_param.enable    = 0;
             ive_non_max_sup_param.enable    = 0;
             ive_thres_lut_param.enable      = 0;
             ive_morph_filter_param.enable   = 0;
             ive_integral_img_param.enable   = 0;
             ive_iter_reg_vote_param.enable  = 0;
             printf("clear all enable func!\n");
        }
        else if (key == 's') {
             printf("current enable functions:\n");
             printf("1. general filter: %d\n", ive_general_filter_param.enable);
             printf("2.  median filter: %d\n", ive_median_filter_param.enable);
             printf("3.    edge filter: %d\n", ive_edge_filter_param.enable);
             printf("4.            NMS: %d\n", ive_non_max_sup_param.enable);
             printf("5.      thres lut: %d\n", ive_thres_lut_param.enable);
             printf("6.   morph filter: %d\n", ive_morph_filter_param.enable);
             printf("7. integral image: %d\n", ive_integral_img_param.enable);
             printf("8.            IRV: %d\n", ive_iter_reg_vote_param.enable);
        }
        else if (key == '1') {
             ive_general_filter_param.enable = 1;
             ive_iter_reg_vote_param.enable = 0;
             printf("enable general filter\n");
        }
        else if (key == '2') {
             ive_median_filter_param.enable = 1;
             printf("enable median filter\n");
        }
        else if (key == '3') {
             ive_edge_filter_param.enable = 1;
```

```
            printf("enable edge filter\n");
        }
        else if (key == '4') {
            ive_non_max_sup_param.enable = 1;
            printf("enable non-maximum suppresion\n");
        }
        else if (key == '5') {
            ive_thres_lut_param.enable = 1;
            printf("enable thres lut\n");
        }
        else if (key == '6') {
            ive_morph_filter_param.enable = 1;
            printf("enable morph filter\n");
        }
        else if (key == '7') {
            ive_integral_img_param.enable = 1;
            printf("enable integral image\n");
        }
        else if (key == '8') {
            ive_iter_reg_vote_param.enable = 1;
            ive_general_filter_param.enable = 0;
            printf("enable IRV\n");
        }
        else if (key == 'r') {
            // set general filter param
            if (ive_general_filter_param.enable) {

                for (i=0; i<VENDOR_IVE_GEN_FILT_NUM; i++)

                    ive_general_filter_param.filt_coeff[i] = general_filter_coef[i];
                ive_general_filter_param.filt_norm = 77;
            }
            ret = vendor_ive_set_param(VENDOR_IVE_GENERAL_FILTER, &ive_general_filter_param);
            if (ret != HD_OK) {
                 printf("err: ive set general filter param error %d\n", ret);
                 goto exit;
            }



            // set median filter param
            if (ive_median_filter_param.enable)
                 ive_median_filter_param.mode = VENDOR_IVE_MEDIAN;
            ret = vendor_ive_set_param(VENDOR_IVE_MEDIAN_FILTER, &ive_median_filter_param);
            if (ret != HD_OK) {
                 printf("err: ive set median filter param error %d\n", ret);
                 goto exit;
            }

            // set edge filter param
            if (ive_edge_filter_param.enable) {
                 ive_edge_filter_param.mode = VENDOR_IVE_NO_DIR;
                 for (i=0;i<VENDOR_IVE_EDGE_COEFF_NUM;i++){

                      ive_edge_filter_param.edge_coeff1[i] = edge_filter_coef[0][i];
                      ive_edge_filter_param.edge_coeff2[i] = edge_filter_coef[1][i];
                 }
                 ive_edge_filter_param.edge_shift_bit = 2;
            }
            ret = vendor_ive_set_param(VENDOR_IVE_EDGE_FILTER, &ive_edge_filter_param);
            if (ret != HD_OK) {
                 printf("err: ive set edge filter param error %d\n", ret);
                 goto exit;
            }

            // set non max sup param
            if (ive_non_max_sup_param.enable)
            ive_non_max_sup_param.mag_thres = 20;
        ret = vendor_ive_set_param(VENDOR_IVE_NON_MAX_SUP, &ive_non_max_sup_param);
        if (ret != HD_OK) {
            printf("err: ive set non max sup param error %d\n", ret);
            goto exit;
        }

        // set thres lut param
        if (ive_thres_lut_param.enable) {
            for (i=0; i<VENDOR_IVE_EDGE_TH_LUT_TAP; i++)
                ive_thres_lut_param.thres_lut[i] = lut_thres_coef[i];
        }
        ret = vendor_ive_set_param(VENDOR_IVE_THRES_LUT, &ive_thres_lut_param);
        if (ret != HD_OK) {
            printf("err: ive set thres lut param error %d\n", ret);
            goto exit;
        }

        // set morph filter param
        if (ive_morph_filter_param.enable) {
            ive_morph_filter_param.in_sel = VENDOR_IVE_TH_LUT_IN;
            ive_morph_filter_param.operation = VENDOR_IVE_DILATE;
            for (i=0; i<VENDOR_IVE_MORPH_NEIGH_NUM; i++)
                ive_morph_filter_param.neighbor[i] = morph_coef[i];
        }
        ret = vendor_ive_set_param(VENDOR_IVE_MORPH_FILTER, &ive_morph_filter_param);
        if (ret != HD_OK) {
            printf("err: ive set morph filter param error %d\n", ret);
            goto exit;
        }

        // set integral img param
        ret = vendor_ive_set_param(VENDOR_IVE_INTEGRAL_IMG, &ive_integral_img_param);
```

```
          if (ret != HD_OK) {
               printf("err: ive set integral img param error %d\n", ret);
               goto exit;
          }



          // set iter reg vote param
          if (ive_iter_reg_vote_param.enable) {
               ive_iter_reg_vote_param.hist_mode = VENDOR_IVE_HIST_7X7;
               ive_iter_reg_vote_param.invalid_value = VENDOR_IVE_INVAL_0;
               ive_iter_reg_vote_param.sum_ratio_thres = 32;
               ive_iter_reg_vote_param.freq_ratio_thres = 10;
               ive_iter_reg_vote_param.median_invalid_thres = 4;
          }
          ret = vendor_ive_set_param(VENDOR_IVE_ITER_REGION_VOTE, &ive_iter_reg_vote_param);
          if (ret != HD_OK) {
               printf("err: ive set iter reg vote param error %d\n", ret);
               goto exit;
          }

          // trigger engine
          ret = vendor_ive_trigger(&ive_trigger_param);
          if (ret != HD_OK) {
               printf("err: ive trigger engine error %d\n", ret);
               goto exit;
          }
          if (!ive_save_result())
               printf("err: ive save result fail\n");
          printf("[ive sample] write result done!\n");
          continue;
     }
     else {
          printf("usage:\n");
```

```
            printf("  enter q: exit\n");
            printf("  enter c: clear all enable function\n");
            printf("  enter 1: enable general filter\n");
            printf("  enter 2: enable median filter\n");
            printf("  enter 3: enable edge filter\n");
            printf("  enter 4: enable non-maximum suppresion\n");
            printf("  enter 5: enable thres lut\n");
            printf("  enter 6: enable morph filter\n");
            printf("  enter 7: enable integral img\n");
            printf("  enter 8: enable IRV\n");
            printf("  enter r: run engine\n");
            continue;
        }
    }
exit:
    vendor_ive_uninit();
    mem_exit();
    ret = hd_common_uninit();
     if(ret != HD_OK) {
        printf("uninit fail=%d\n", ret);
        goto exit;
     }
```

2. Revision history

Revision	Date	Author	Changes
0.1	2017/5/02	Shuchun Lin	First draft version.
0.2	2018/11/15	K W Wang	Modify content for kdriver
0.3	2018/11/19	K W Wang	Modify content for kdriver
0.4	2018/11/21	Vincent CC Huang	Add content for user space interface