地 平 线
Horizon Robotics

X3

Graphics System Reference Manual

V1.0

2021-02

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.

https://www.horizon.ai

Revision History

The revision history 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	2020-12-22	Initialize Document
1.0	2021-02	Release V1.0

Table of Contents

1 Module introduction

1.1 AE

AE algorithm operates by analyzing histogram to calculate new EV value relative to AE target set in calibration. AE target is dynamically controlled by algorithm to ensure correct exposure of low dynamic range scene and high dynamic range scene. For example, in a low dynamic range scene, the AE target will be based on the AE_ LDR_ Target (18% gray target). For high dynamic range scenes, the algorithm will dynamically modify the AE target to ensure that the bright area will not be overexposed. The local tone mapping engine (iridix) will ensure that shadows are displayed and exposed correctly.

1.2 AWB

AWB module is responsible for color stability, because the response of image sensor to neutral tone depends on the illumination of the scene, which is most obvious in neutral tone such as white and gray. AWB algorithm deals with a wide range of lighting conditions and spectra to avoid bad color bias.

1.3 Demosaic

The Demosaic unit is responsible for reconstructing a full color image from the incomplete (spatially undersampled) color samples output by an image sensor overlaid with a Color Filter Array (CFA). Additionally, this module provides advanced control over image sharpening.

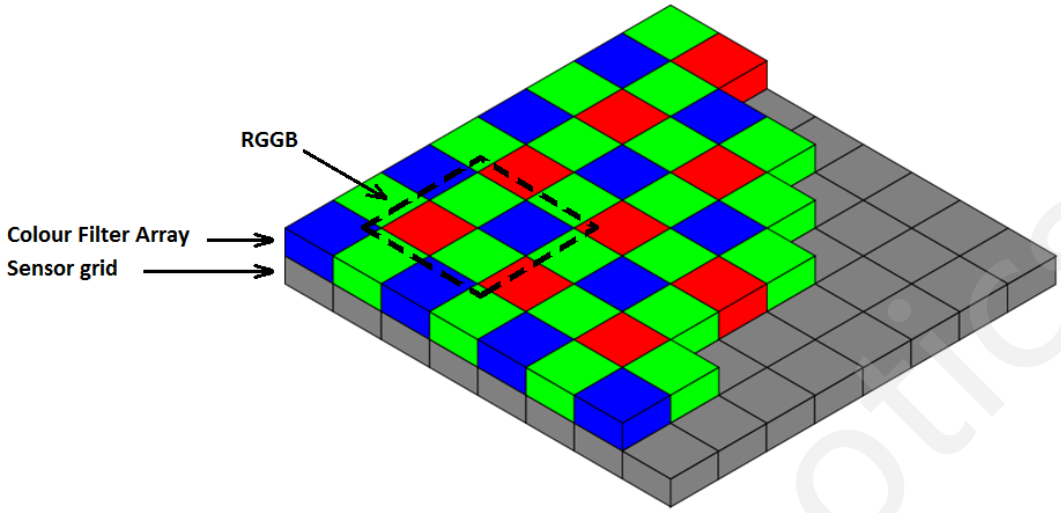
Digital camera sensor elements (pixel sensors) by themselves, can record only the intensity of light falling on them and not differentiate between different colors, hence producing only a grayscale image. To capture color information, filters must be placed over each pixel sensor, that permit only particular colors of light through. The filters used must allow reconstruction of a full color image with Red, Green and Blue (RGB) values for each pixel location.

The most common type of such Color Filter Array is called a "Bayer array," shown in Figure below, so called because the color filters are arranged in a regular pattern with each 2x2 pixel group having an RGGB arrangement.

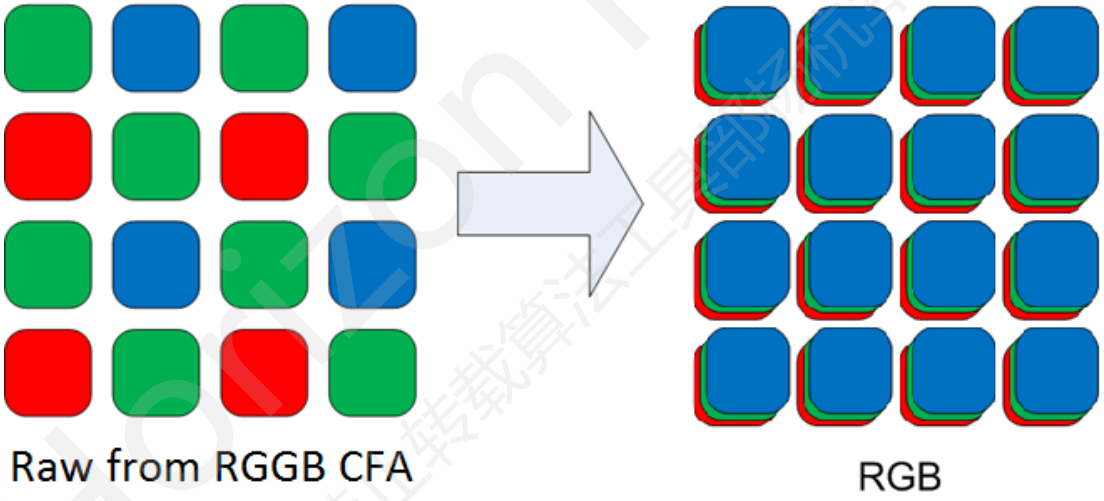
• Half of all the pixels are green (G), while a quarter each are red (R) and blue (B).

• Green cells in the same row as blue are labeled Gb and the green cells in the same row as red are labelled Gr.

• The pattern can start with any of R, Gr, Gb or B.



This arrangement of color filters essentially causes spatial undersampling of color information. The demosaic unit is responsible for reconstructing a full color image (RGB for each pixel) from this incomplete color information as shown below:



The module consists of many filters, which reconstruct chrominance channels based on interpolated luminance channel. It also considers signal-dependent sensor noise (based on earlier determined noise profile) to preserve the sharpness of the edges and smoothness of uniform areas, while interpolating the missing pixel components. Hence, the interpolation of missing pixel components explicitly incorporates the noise characteristics of the sensor. The embedded sharpening also minimizes the amplification of high frequency noise.

1.4 Sharpen

This block control the sharpening in RGB domain after demosaicing.

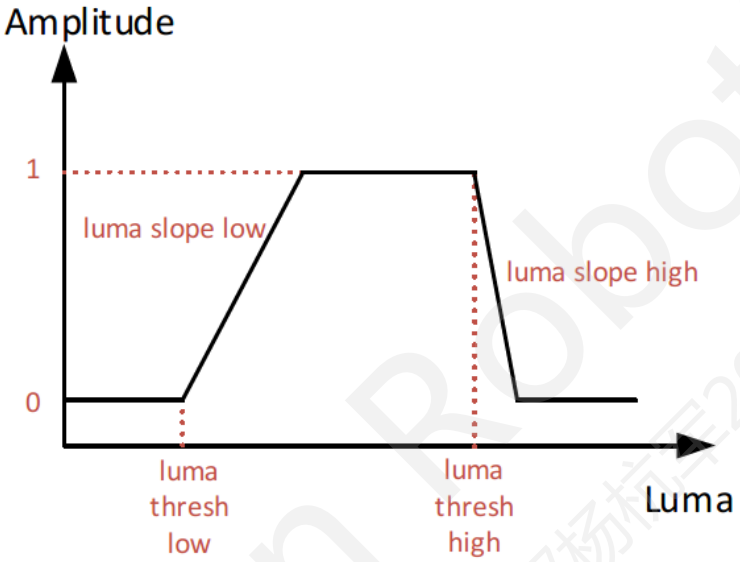
It provides:

Low boosting of very high frequency. It increases dot noise or unnatural overshoot.

Low jagging, Low “worms” artefact.

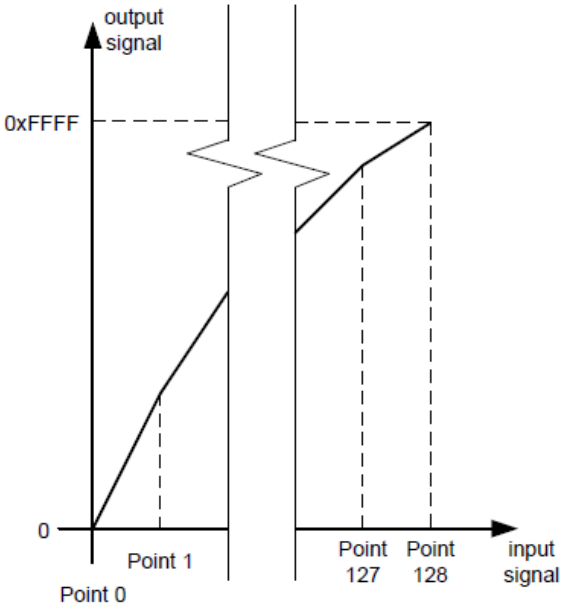
Luminance controls the amplitude of sharpen and is done by decreasing the amplitude of:

- The dark area using the configuration register, luma thresh low and luma slope low.
- The bright area using the configuration register, luma thresh high and luma slope high.



1.5 Gamma

This module encodes the output gamma and is typically set to match BT.709 or sRGB gamma curves.



This module applies a gamma LUT separately for each of the three (R,G,B) color channels. In typical configurations, the LUT has 129 evenly spaced nodes labelled 0…128 and the hardware applies linear interpolation between these nodes.

Each data value is a 16bit unsigned number, so it is expected that Gamma[0]=0 and Gamma[128]=0xFFFF, with the other 127 values defining the Gamma correction curve.

Note:

The adaptive contrast enhancement is performed dynamically by the Iridix module. This LUT should be programmed statically based on the desired output gamma characteristics.

1.6 Iridix

Iridix® uses local tone mapping for dynamic range compression (DRC), attempting to bring back detail from areas of reduced visibility in an HDR scene without affecting the global image. Overall this increases the range of tones available to local regions by increasing gain with respect to scene content.

1.7 CNR

This block reduces the Chroma Noise in the image by using wider Gaussian kernels to target the noise that typically affects the larger areas of the image. During the process the block keeps the intensity content of the image intact and only process the Chroma part of it.

The block converts the RGB image into YUV domain and process the U and V channels individually by further segmenting it based on color content. The Gaussian kernels are then applied to each U and V channels of the corresponding segment and are configured by corresponding offset or slope parameters. To minimize the hardware implementation or reduce the number of lines for large kernels, the vertical Gaussian filtering is replaced by a novel recursive filter. Also, to reduce the area further, the color channels are down-sampled before the processing and then up-sampled at the output stage. The output U and V channels are blend of the processed U and V and original U and V channels respectively which is configured by delta offset or slope parameters. The processed U and V and unprocessed Y are converted back to the RGB domain.

1.8 Sinter

Sinter® is an advanced spatial noise reduction module combining a set of algorithms to suppress sensor noise while preserving edges and image textures.

The use of this module is simplified by using an externally generated sensor Noise Profile LUT.

When this LUT is correctly programmed, the module is controllable through a reduced set of registers. In most cases only the Sinter threshold is modified to adjust noise filter strength. Sinter: Thresh Long and Sinter: Thresh Short registers corresponding to the long and short exposures respectively are used in WDR mode. When the exposure ratio is 1 or when WDR is disabled Sinter: Thresh Long and Sinter: Thresh Short should be set to the same value. The threshold values are determined during a standard calibration procedure using images captured at various ISO values and are modulated with system gain. If the image has been created using frame-switching, these values should be set accordingly based on the exposure ratio.

1.9 Temper

This module is a motion-adaptive temporal noise reduction filter. It recursively averages the current frame with the history of previous frames, with a recursion level set locally by the degree of local motion detected in the current frame. The filter works in the raw domain and requires two frames of external memory. The data bit width is the video bit width + 4 bits. Note that in WDR mode the video bit width will be larger.

The Recursion_Limit can be increased or decreased to adjust the depth of recursion, which in turn sets the effective number of frames over which temporal averaging is performed, depending on the extent of motion in a given region. Increasing this parameter leads to a smaller recursion depth, with smaller noise reduction effect, and minimized motion artefact.

• When Recursion_Limit is set to 0, up to 16 frames can be averaged.

• When Recursion_Limit is set to 0xf, frames are not averaged. This is equivalent to disabling temper.

The Temper thresholds are used to adjust Temper noise filter strength. Thresh_Long and Thresh_Short parameters corresponding to the long and short exposures respectively are used in WDR mode. When the exposure ratio is 1 or when WDR is disabled Thresh_Long and Thresh_Short should be set to the same value. The threshold values are determined during a standard calibration procedure using images captured at various ISO values and are modulated with system gain.

Larger values for Thresh_Short and Thresh_Long increase the strength of the noise reduction.

The performance of this module is governed by an externally-generated sensor Noise Profile LUT.

The reference data is stored in the DDR through DMAs. Thera are 2 DMA reader and 2 DMA writers to manage the reference data storage. 2 sets of DMAs are used only in Temper3 mode. When the temper operates in temper2 mode, only 1 DMA writer and 1 DMA reader is used.

1.10 Mesh Shading

Mesh shading provides a further correction for non-linear shading distortions. This is applied to the image according to 64 x 64 zones, again considering color plane and lighting color temperature.

The ISP provides four pages (R, G, B and IR) each containing an intensity correction table.

Mesh alpha mode	Format of Mesh Table
0	One light source with a maximum of 64x64 nodes (64 rows by 64 columns) for each color plane.
1	Two light sources with a maximum of 32x64 nodes (32 columns by 64 rows) for each color plane and light source.
2	Up to Four light sources with a maximum of 32x32 nodes (32 rows by 32 columns) for each color plane and light source.
3	Two light sources with a maximum of 64x32 nodes (64 columns by 32 rows) for each color plane and light source.

Mesh Alpha Mode = 0

LUT Entry	0				1				...	1023			
Page Number	0				0				...	0			
Mesh point	0,3	0,2	0,1	0,0	0,7	0,6	0,5	0,4	...	63,63	63,62	63,61	63,60
Bits	24:31	16:23	8:15	0:7	24:31	16:23	8:15	0:7	...	24:31	16:23	8:15	0:7
Light Source	LS0	LS0	LS0	LS0	LS0	LS0	LS0	LS0	...	LS0	LS0	LS0	LS0

LUT Entry	1024				1025				...	2047			
Page Number	1				1				...	1			
Mesh point	0,3	0,2	0,1	0,0	0,7	0,6	0,5	0,4	...	63,63	63,62	63,61	63,60
Bits	24:31	16:23	8:15	0:7	24:31	16:23	8:15	0:7	...	24:31	16:23	8:15	0:7
Light Source	LS0	LS0	LS0	LS0	LS0	LS0	LS0	LS0	...	LS0	LS0	LS0	LS0

LUT Entry	2048				2049				...	3071			
Page Number	2				2				...	2			
Mesh point	0,3	0,2	0,1	0,0	0,7	0,6	0,5	0,4	...	63,63	63,62	63,61	63,60
Bits	24:31	16:23	8:15	0:7	24:31	16:23	8:15	0:7	...	24:31	16:23	8:15	0:7
Light Source	LS0	LS0	LS0	LS0	LS0	LS0	LS0	LS0	...	LS0	LS0	LS0	LS0

Mesh Alpha Mode = 1

LUT Entry	0				1				...	1023			
Page Number	0				0				...	0			
Mesh point	0,1		0,0		0,3		0,2		...	63,31		63.30	
Bits	24:31	16:23	8:15	0:7	24:31	16:23	8:15	0:7	...	24:31	16:23	8:15	0:7
Light Source	LS1	LS0	LS1	LS0	LS1	LS0	LS1	LS0	...	LS1	LS0	LS1	LS0

LUT Entry	1024				1025				...	2047			
Page Number	1				1				...	1			
Mesh point	0,1		0,0,		0,3		0,2		...	63,31		63.30	
Bits	24:31	16:23	8:15	0:7	24:31	16:23	8:15	0:7	...	24:31	16:23	8:15	0:7
Light Source	LS1	LS0	LS1	LS0	LS1	LS0	LS1	LS0	...	LS1	LS0	LS1	LS0

LUT Entry	2048				2049				...	3071			
Page Number	2				2				...	2			
Mesh point	0,1		0,0,		0,3		0,2		...	63,31		63.30	
Bits	24:31	16:23	8:15	0:7	24:31	16:23	8:15	0:7	...	24:31	16:23	8:15	0:7
Light Source	LS1	LS0	LS1	LS0	LS1	LS0	LS1	LS0	...	LS1	LS0	LS1	LS0

Mesh Alpha Mode = 2

LUT Entry	0				1				...	1023			
Page Number	0				0				...	0			
Mesh point	0,0				0,1				...	31,31			
Bits	24:31	16:23	8:15	0:7	24:31	16:23	8:15	0:7	...	24:31	16:23	8:15	0:7
Light Source	LS3	LS2	LS1	LS0	LS3	LS2	LS1	LS0	...	LS3	LS2	LS1	LS0

LUT Entry	1024				1025				...	2047			
Page Number	1				1				...	1			
Mesh point	0,0				0,1				...	31,31			
Bits	24:31	16:23	8:15	0:7	24:31	16:23	8:15	0:7	...	24:31	16:23	8:15	0:7
Light Source	LS3	LS2	LS1	LS0	LS3	LS2	LS1	LS0	...	LS3	LS2	LS1	LS0

LUT Entry	2048				2049				...	3071			
Page Number	2				2				...	2			
Mesh point	0,0				0,1				...	31,31			
Bits	24:31	16:23	8:15	0:7	24:31	16:23	8:15	0:7	...	24:31	16:23	8:15	0:7
Light Source	LS3	LS2	LS1	LS0	LS3	LS2	LS1	LS0	...	LS3	LS2	LS1	LS0

Mesh Alpha Mode = 3

LUT Entry	0				1				...	1023			
Page Number	0				0				...	0			
Mesh point	0,1		0,0		0,3		0,2		...	31,63		31,62	
Bits	24:31	16:23	8:15	0:7	24:31	16:23	8:15	0:7	...	24:31	16:23	8:15	0:7
Light Source	LS1	LS0	LS1	LS0	LS1	LS0	LS1	LS0	...	LS1	LS0	LS1	LS0

LUT Entry	1024				1025				...	2047			
Page Number	1				1				...	1			
Mesh point	0,1		0,0,		0,3		0,2		...	31,63		31,62	
Bits	24:31	16:23	8:15	0:7	24:31	16:23	8:15	0:7	...	24:31	16:23	8:15	0:7
Light Source	LS1	LS0	LS1	LS0	LS1	LS0	LS1	LS0	...	LS1	LS0	LS1	LS0

LUT Entry	2048				2049				...	3071			
Page Number	2				2				...	2			
Mesh point	0,1		0,0,		0,3		0,2		...	31,63		31,62	
Bits	24:31	16:23	8:15	0:7	24:31	16:23	8:15	0:7	...	24:31	16:23	8:15	0:7
Light Source	LS1	LS0	LS1	LS0	LS1	LS0	LS1	LS0	...	LS1	LS0	LS1	LS0

1.11 Radial Shading

The radial shading coefficients are stored in a 32-bit, 4 x 129-entry LUT. Coefficients are in x.12 format where the lower 12-bits are fractional values. The coefficients are stored starting from the centre to the outer edge for each color plane.

1.12 Color Space Conversion

This module converts the input {R, G, B} pixel values into {Y, U, V} values, and uses standard 3x3 matrix multiplication and vector offset. If the conversion is not active, ISP outputs pixels in RGB format.

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} Coefft11 & Coefft12 & Coefft13 \\ Coefft21 & Coefft22 & Coefft23 \\ Coefft31 & Coefft23 & Coefft33 \end{pmatrix} . \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} Coefft01 \\ Coefft02 \\ Coefft03 \end{pmatrix}$$

If necessary, the parameters can be modified to provide different transformations. Taking BT.709 as an example, the formula is as follows:

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.184 & 0.613 & 0.063 \\ -0.102 & -0.340 & 0.438 \\ 0.438 & -0.398 & -0.039 \end{pmatrix} . \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0.063 \\ 0.5 \\ 0.5 \end{pmatrix}$$

Then the parameters corresponding to RGB2YUV_CONVERSION in calibration are shown in the formula:

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 47 & 157 & 16 \\ 32794 & 32855 & 112 \\ 112 & 32870 & 32778 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 64 \\ 512 \\ 512 \end{pmatrix}$$

其中，若 Coefft11 (Coefft12, Coefft13, Coefft21, Coefft22, Coefft23, Coefft31, Coefft32, Coefft33) 参数为正，clibration 中参数为 Coefft11*256 取整；若该参数为负，则 calibraiton 中参数为(|Coefft11*256|+1<<15)取整。calibration 中参数为 Coefft01(Coefft02, Coefft01)*1024 取整。

Among them, if the parameter of Coefft11 (Coefft12, Coefft13, Coefft21, Coefft22, Coefft23, Coefft31, Coefft32, Coefft33) is positive, the parameter of clibration is Coeffft11 * 256 rounded; if the parameter is negative, the parameter of calibraiton is (| Coeffft11 * 256 | + 1 < < 15) rounded. The parameter in calibration is Coefft01 (Coefft02, Coefft01) * 1024, rounded.

1.13 metering statistics

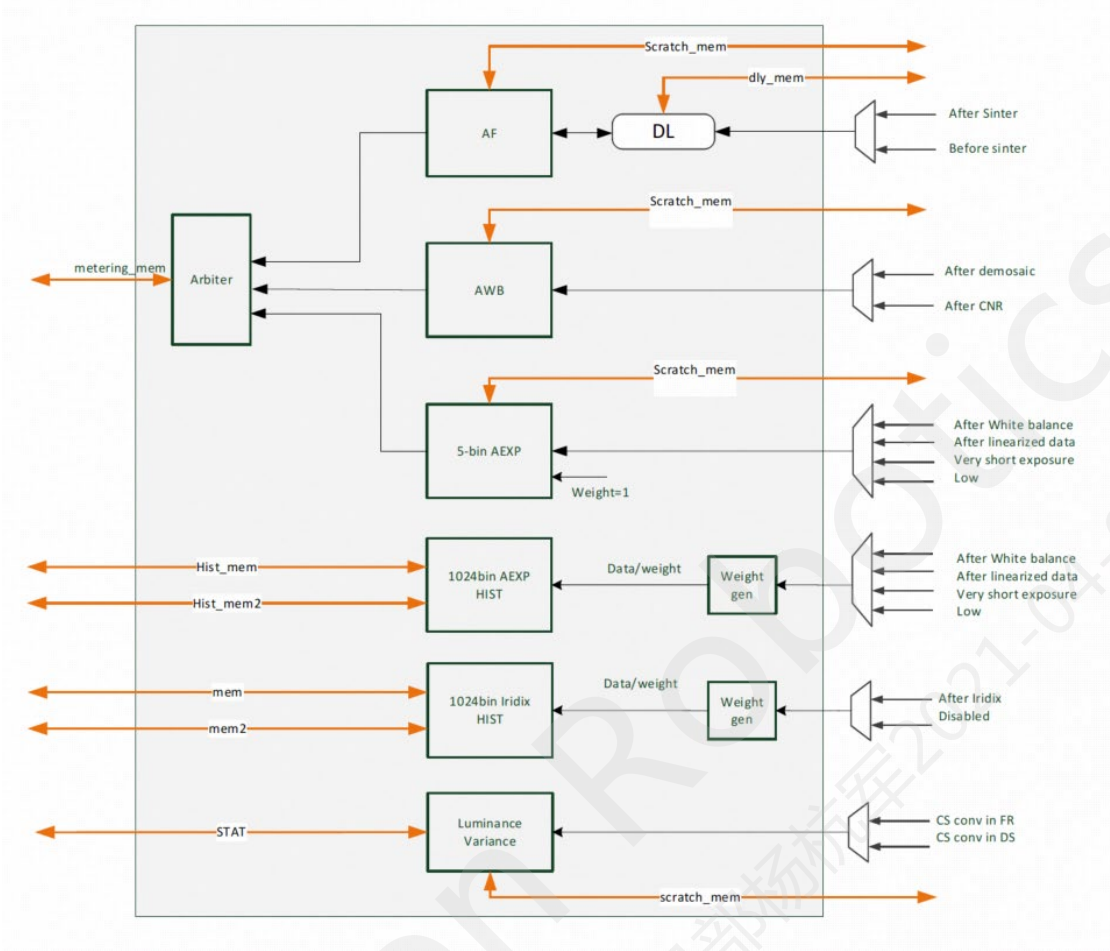
The Image Signal Processor (ISP) comprises of a series of metering blocks that are used in conjunction with high-level 3A algorithms for sensor runtime tuning. These algorithms can include:

- Auto exposure (AE).
- Auto focus (AF).
- Auto white balance (AWB).

These metering blocks are hardware computation accelerators that provide aggregative statistics on a per frame and/or per zone basis to reduce the computational load on the software driver. The ISP contains five types of hardware metering blocks:

1. Zone-wise AF meter.
2. Zone-wise AWB meter
3. Zone-wise AE 5-bin configurable histogram.
4. Global 1024-bin fixed histogram.
5. Zone-wise luminance variance meter.

The ISP provides programmable tag points for various metering modules.



1.13.1 AWB metering

AWB has global and regional statistics.

Global statistics: the average R / g and B / g of the whole image and the number of effective statistical points.

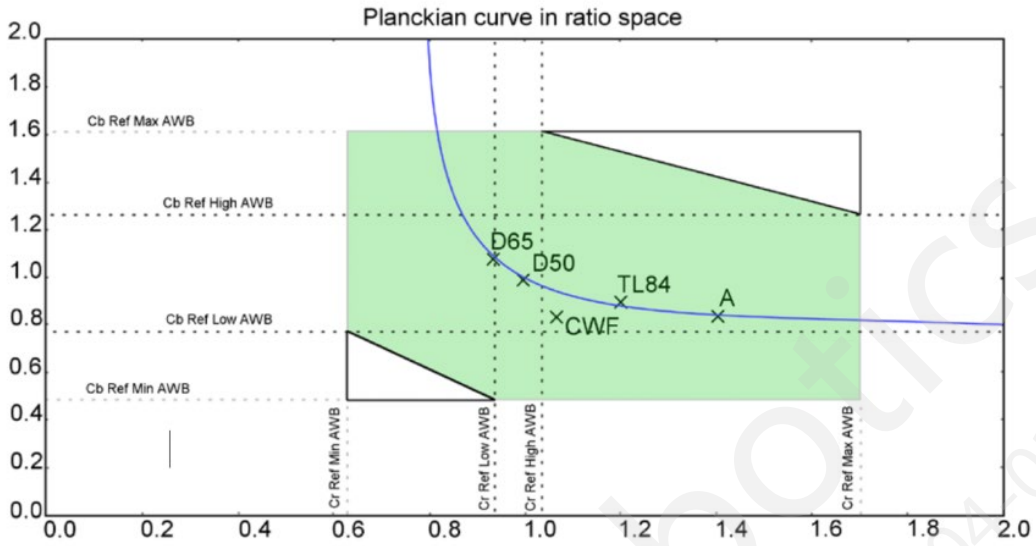
Regional statistics: the maximum support is 33x33 blocks of image. Each block outputs the mean value of R / g and B / g and the number of effective statistical points.

AWB_stats_mode Register can be used to configure the type of mean: R / G, B / G or G / R, G / B.

Through register configuration, valid pixels can be limited:

Cb_Ref_Min/Max，Cr_Ref_Min/Max These four values limit the maximum and minimum ratio of R / G and B / G.

Further, Cb_Ref_Low/High, Cr_Ref_Low/High can be used to limit the smaller R / G, B / G range.



Global statistics: It is stored by three registers AWB RG, AWB BG and sum;

Regional statistics:

Table Index	Zone	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
0	1		Average Green/Blue [27:16]				Average Green/Red [11:0]		
1	1	Number of pixels used for AWB							
2	2		Average Green/Blue [27:16]				Average Green/Red [11:0]		
3	2	Number of pixels used for AWB							
repeated until …									
2mn-2	mn		Average Green/Blue [27:16]				Average Green/Red [11:0]		
2mn-1	mn	Number of pixels used for AWB							

1.13.2 AE metering

Automatic exposure (AE) statistics are collected after application of black level, white balance and ISP gain. There are two types of histograms:

- 5-bin configurable histogram
- Global 1024-bin fixed histogram

1.13.2.1 5-bin configurable histogram

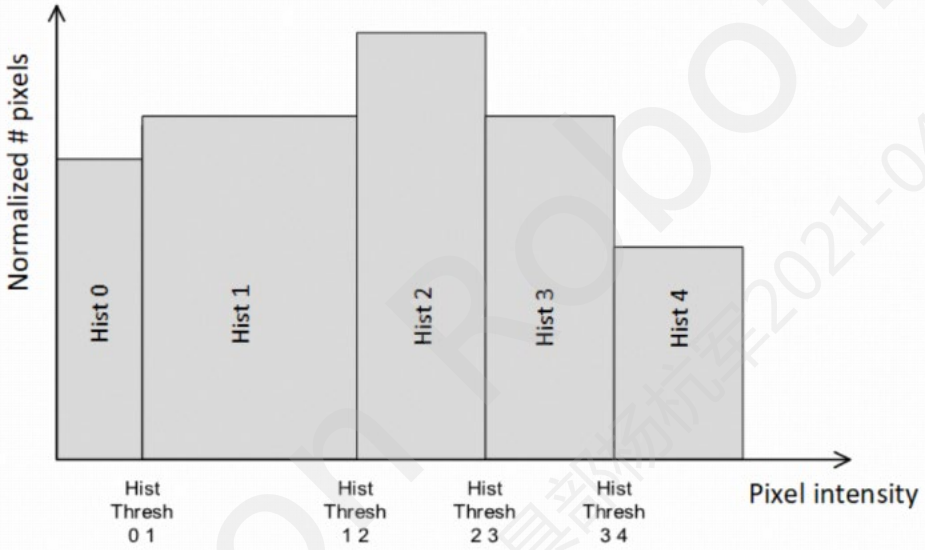
A 5-bin normalized histogram is generated for each zone and for the entire image with adjustable histogram bin boundaries. The parameters Statistics_Hist_Thresh [i] [j] are set to define the threshold intensity values between each bin i and j.

The registers Statistics_Hist [i] provide the global normalized pixel counts for each bin i. The total sum is normalized to 0xFFFF.

The centre bin of the histogram is not available and can be calculated in firmware by subtracting the values of Hist0, Hist1, Hist3, and Hist4 from 0xFFFF as in the following equation:

$$Hist_2 = 0xFFFF - \sum_i Hist_i$$

where i = 0,1,3, and 4.



An internal table with Histx data gives the normalised values of the histograms for each of the zones as shown in the table below for mxn zones. The order of the zones is in raster order starting from the top left corner of the image. The sum of the histogram data is normalized to 0xFFFF for each zone. Note that as in the global histogram, the Hist2 is not available but can be calculated based on the normalized sum and the values of Hist0,

Hist1, Hist3 and Hist4 for each zone.

Table Index	Zone	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
0	1	Hist 1				Hist 0			
1	1	Hist 4				Hist 3			
2	2	Hist 1				Hist 0			
3	2	Hist 4				Hist 3			
repeated until ...									
2mn-2	mn	Hist 1				Hist 0			
2mn-1	mn	Hist 4				Hist 3			

1.13.2.2 1024-bin histogram

The global histogram is also weighted by zone but is not normalized. It is expected that normalization of the statistics data be performed in the firmware. This is available to the firmware as a consecutive array of 1024x32 bit unsigned integers.

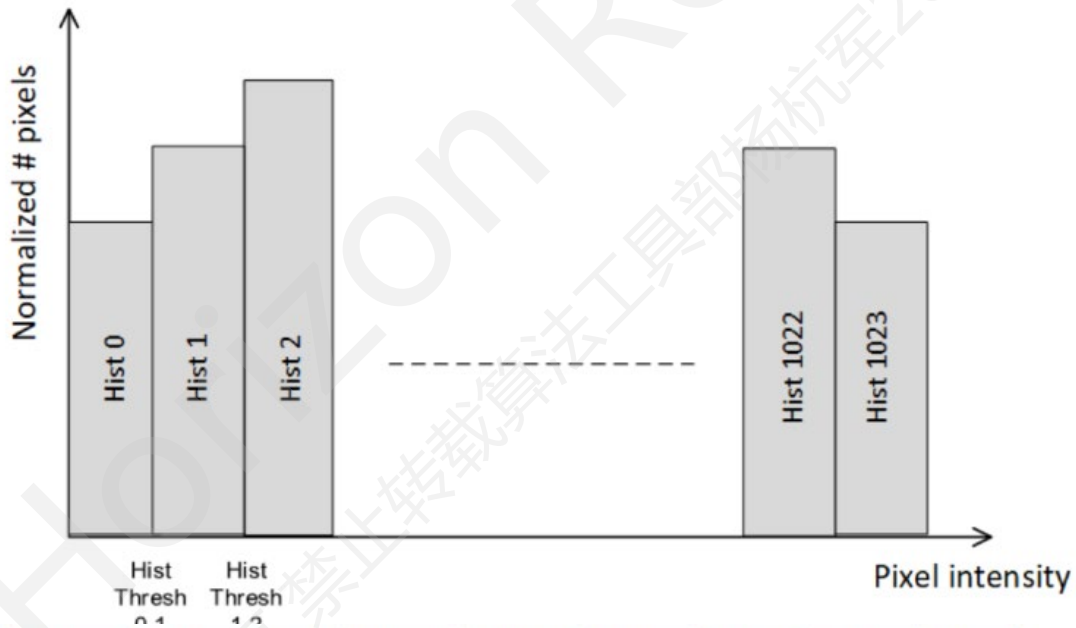


Table Index	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
0	Hist 0							
1	Hist 1							
2	Hist 2							
repeated until...								
1022	Hist 1022							
1023	Hist 1023							

1.13.3 AF metering

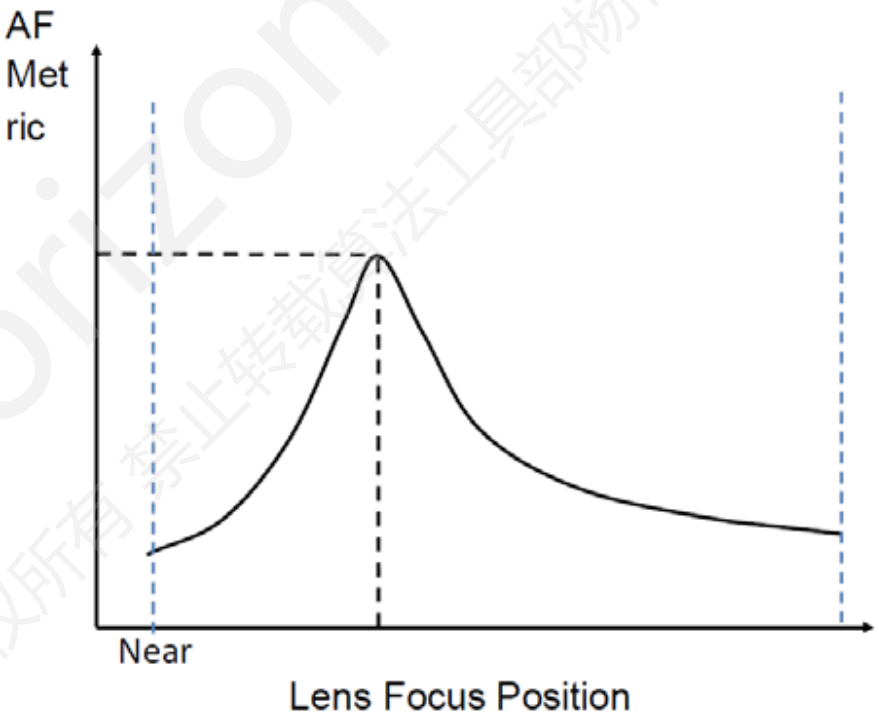
Auto Focus statistics are composed of Region-Of-Interest (ROI) or zonal and normalized Full image multi-direction contrast metrics. This contrast metrics is used by the CPU outside of the ISP core to determine the position of the lens to achieve the best possible focus.

For the AF block, zones are SW configurable. The Zonal Contrast metrics is calculated for each pixel and accumulated over the full zone. For each pixel, contrast is calculated along 4 directions.

Apart from this, angular direction of diagonal lines can also be controlled using a kernel select config parameter as in table1. To improve the response in low light and low pass imaging scenarios, the calculated contrast is of the forth order (quartic contrast sum).

These zonal metrics are not weighted in the hardware, however the software can apply the zone-based weights post computation.

Figure below shows that optimal focus is achieved when the AF contrast metric is at its highest point:



1.13.3.1 AF metrics data computation

The zonal accumulated contrast metrics are stored in floating point format with16b

mantissa and 5b exponent. Apart from contrast metrics we accumulate squared image and quartic image data over the zone.

Each zone accumulates the statistics shown in the following table:

Statistics Name	Data Format	Description
I2	16b Mantissa, 5b Exponent	Zonal squared image pixel sum
I4	16b Mantissa, 6b Exponent	Zonal quartic image pixel sum
E4	16b Mantissa, 5b Exponent	Zonal multi-directional quartic edge sum

AF Zone data stored in internal memory:

Zone 1	Register1 [31:0]
	Register2 [31:0]
Zone 2	Register1 [31:0]
	Register2 [31:0]
….	….
	….
Zone N – 2	Register1 [31:0]
	Register2 [31:0]
Zone N – 1	Register1 [31:0]
	Register2 [31:0]

Global Contrast Statistics: Apart from zonal statistics, AF also accumulates normalized quartic edge sum. Its stored in a 32b register.

1.13.4 Auto Level

1024bin metering data provided by iridix module.

Addr	Mode	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Default
0x034A8	R/W	Data(0)																																--
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x034AC	R/W	Data(1)																																--
... repeated until ...																																		
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x044A4	R/W	Data(1023)																																--

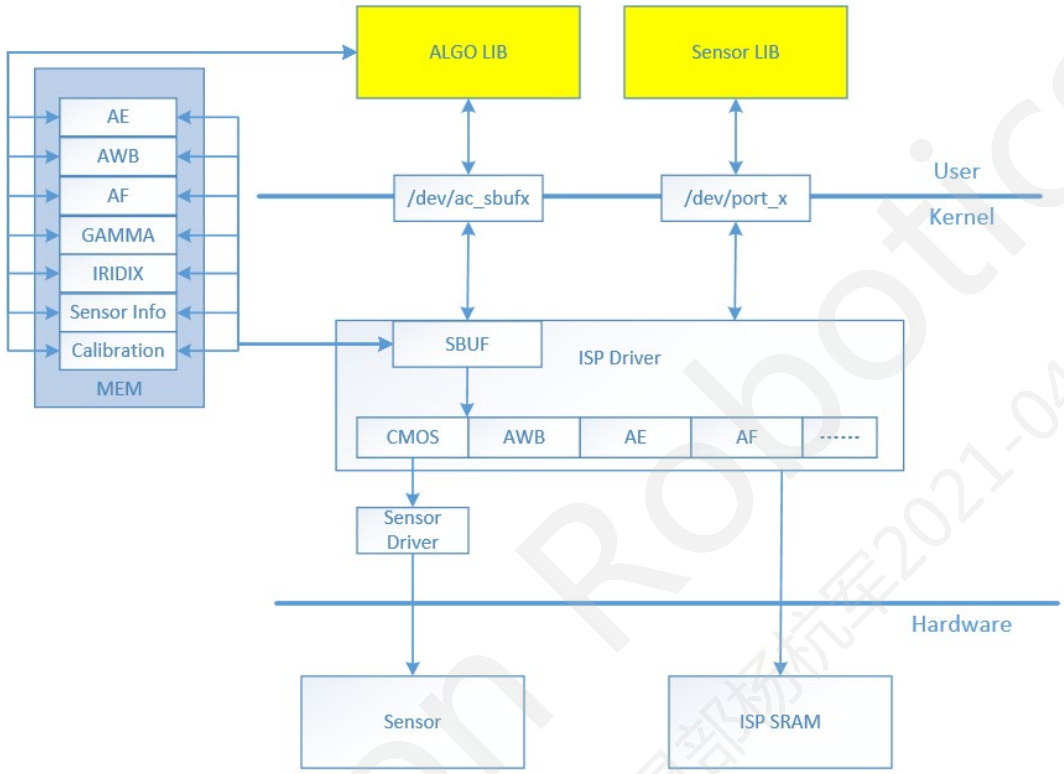
Data [31:0] Please see other documentation for a description of the contents of this array.
This is an array of 1024 32-bit registers. The address for element (i) is: 0x034A8 + 4i.

1.13.5 Mean luminance and variance of mean luminance

The ISP provides a stat of zonal luminance mean and the variance of the zonal luminance mean. The module always uses a fixed zone size of 32x16 (horizontal x vertical) and the minimum frame resolution required for this stat module to give usable result is 512x256. The stats are stored in SRAM of 512 locations. Each location contains 10-bit (LSBs) of mean information and 12 bit (MSBs) of the variance of the luminance mean information for each zone.

1.14 Interactive Data

1.14.1 diagram of algorithm library and ISP firmware



- There are two parts in MEM, one is the data provided to the algorithm library, the other is the value passed in by the algorithm library to the ISP driver. For example, the value of the algorithm configuration received by AWB in the ISP driver will be updated to the ISP register space;
- This is the case of one context (corresponding to one sensor), multiple sensors and multiple data structures like this;
- The yellow color block is the replaceable part. Algorithm library needs several callback interfaces, as shown in the section of "external 3A library registration interface"; in addition, it needs some input and output data, as shown in the section of "data structure-3a library correlation";
- Sensor driver includes the common sensor gain and exposure configuration methods, which do not need to be modified in general;

1.14.2 Interactive data between algorithm library and ISP firmware

Module	ISP Fw -> Algorithm	Algorithm -> ISP Fw
AE	stats_data[ISP_FULL_HISTOGRAM_SIZE]	ae_exposure
	histogram_sum	ae_exposure_ratio
	hist4[33 * 33]	frame_id
AWB	stats_data[MAX_AWB_ZONES]	awb_red_gain
	curr_AWB_ZONES	awb_green_even_gain
		awb_green_odd_gain
		awb_blue_gain
		temperature_detected
		p_high
		light_source_candidate
		awb_warming[3]
		mix_light_contrast
		frame_id
AF	stats_data[AF_ZONES_COUNT_MAX][2]	frame_to_skip
	zones_horiz	af_position
	zones_vert	af_last_sharp
	frame_num	
	skip_cur_frame	
	zoom_step_info	

Gamma	stats_data[ISP_FULL_HISTOGRAM_SIZE]	gamma_gain
	fullhist_sum	gamma_offset
		frame_id
Iridix	N/A（using AE statistics data from AE algorithm）	strength_target
		iridix_dark_enh
		iridix_global_DG
		iridix_contrast
		frame_id

1.14.3 3A Development Instructions

ISP firmware has two parts: user space and kernel space. Kernel space firmware is initialized with system startup, and user space firmware (including default 3A algorithm) is initialized by calling *HB_VIN_StartPipe*. When started, the external 3A algorithm will be preferred during firmware startup. If the external 3A algorithm is not registered, the default 3A algorithm will be started. Each kind of algorithm has two corresponding input parameters: stats, input, and output. ISP firmware will call *proc_func* callback after every frame of statistics is ready, passing in two input parameters, *proc_func* is a practical algorithm implementation. After algorithm calculation, output parameters should be filled in. ISP firmware will apply the output parameters to sensor or ISP hardware.

1.14.3.1 AE Algorithm Register

AE register callback function to ISP Firmware:

From	Users need to implement	To
AELIB	init_func	ISP Firmware
	proc_func	
	deinit_func	

Interface description:

Callback function	Description
init_func	Algorithm initialization function
proc_func	Practical algorithm implementation, such as

	target and exposure calculation. ISP Firmware calls this function when the AE statistics of each frame are ready. incoming parameters and parameters to be outgoing please refer to AE structure description
deinit_func	Algorithm deinitialization function

1.14.3.2 AWB Algorithm Register

AWB register callback function to ISP Firmware:

From	Users need to implement	To
AWBLIB	init_func	ISP Firmware
	proc_func	
	deinit_func	

Interface description:

Callback function	Description
init_func	Algorithm initialization function
proc_func	Practical algorithm implementation. ISP Firmware calls this function when the AWB statistics of each frame are ready. incoming parameters and parameters to be outgoing please refer to AWB structure description
deinit_func	Algorithm deinitialization function

1.14.3.3 AF Algorithm Register

AF register callback function to ISP Firmware:

From	Users need to implement	To
AFLIB	init_func	ISP Firmware
	proc_func	
	deinit_func	

Interface description:

Callback function	Description
init_func	Algorithm initialization function
proc_func	Practical algorithm implementation. ISP Firmware calls this function when the AF statistics of each frame are ready. incoming parameters and parameters to be outgoing please refer to AF structure description
deinit_func	Algorithm deinitialization function

1.14.3.4 Example of Algorithm Register

For example of AWB:

```
ISP_AWB_FUNC_S stAwbFunc = {
        .init_func = awb_init_func,
        .proc_func = awb_proc_func,
        .deinit_func = awb_deinit_func,
};
HB_ISP_AWBLibRegCallback(0, "libawb.so", &stAwbFunc);


void *awb_init_func(uint32_t ctx_id)
{
        pr_info("ctx id is %d", ctx_id);
        return NULL;
}

int32_t awb_proc_func(void *awb_ctx, awb_stats_data_t *stats, awb_input_data_t *input, awb_output_data_t
*output)
{
    awb_acamera_core_obj_t *p_awb_core_obj = (awb_acamera_core_obj_t *)awb_ctx;
    awb_acamera_input_t *p_acamera_input = (awb_acamera_input_t *)input->acamera_input;
    awb_calibration_data_t *p_cali_data = &( p_acamera_input->cali_data );
    awb_acamera_output_t *p_acamera_output = (awb_acamera_output_t *)output->acamera_output;

    //Specific algorithm implementation, input, statistical data for calculation
```

```
    awb_calc_avg_weighted_gr_gb_mesh( p_awb_core_obj, stats, input );
    awb_detect_light_source( p_awb_core_obj );
    awb_calculate_warming_effect( p_awb_core_obj, p_cali_data );

    //Assign calculation results to output parameters
    p_acamera_output->rg_coef = p_awb_core_obj->rg_coef;
    p_acamera_output->bg_coef = p_awb_core_obj->bg_coef;
    p_acamera_output->temperature_detected = p_awb_core_obj->temperature_detected;
    p_acamera_output->p_high = p_awb_core_obj->p_high;
    p_acamera_output->light_source_candidate = p_awb_core_obj->light_source_candidate;
    memcpy( p_acamera_output->awb_warming, p_awb_core_obj->awb_warming,
sizeof( p_acamera_output->awb_warming ) );
    p_acamera_output->awb_converged = p_awb_core_obj->awb_converged;
      return 0;
}

int32_t awb_deinit_func(void *awb_ctx)
{
      pr_info("done");
      return 0;
}
```

2 Software Interface

2.1 Control Interface

int HB_ISP_SetFWState(uint8_t pipeId, const ISP_FW_STATE_E enState);

int HB_ISP_GetFWState(uint8_t pipeId, ISP_FW_STATE_E *penState);

int HB_ISP_SetRegister(uint8_t pipeId, uint32_t u32Addr, uint32_t u32Value);

int HB_ISP_GetRegister(uint8_t pipeId, uint32_t u32Addr, uint32_t *pu32Value);

int HB_ISP_SetModuleControl(uint8_t pipeId, const ISP_MODULE_CTRL_U *punModCtrl);

int HB_ISP_GetModuleControl(uint8_t pipeId, ISP_MODULE_CTRL_U *punModCtrl);

int HB_ISP_SwitchScence(uint8_t pipeId, const char *cname);

int HB_ISP_StartI2CBus(uint8_t pipeId);

void HB_ISP_StopI2CBus(uint8_t pipeId);

int HB_ISP_SendI2CData(ISP_I2C_DATA_S data);

2.1.1 HB_ISP_SetFWState/HB_ISP_GetFWState

[Function Declaration]

int HB_ISP_SetFWState(uint8_t pipeId, const ISP_FW_STATE_E enState);

int HB_ISP_GetFWState(uint8_t pipeId, ISP_FW_STATE_E *penState);

[Function Description]

Set or get state of ISP Firmware.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input
penState	ISP Firmware State	Input

[Return Value]

Return value	Description

0	pass
Non 0	failed

2.1.2 HB_ISP_SetRegister/HB_ISP_GetRegister

[Function Declaration]

int HB_ISP_SetRegister(uint8_t pipeId, uint32_t u32Addr, uint32_t u32Value);

int HB_ISP_GetRegister(uint8_t pipeId, uint32_t u32Addr, uint32_t *pu32Value);

[Function Description]

Set or get ISP registers.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input
u32Addr	ISP register address	Input
u32Value	Value to set	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.1.3 HB_ISP_SetModuleControl/ HB_ISP_GetModuleControl

[Function Declaration]

int HB_ISP_SetModuleControl(uint8_t pipeId, const ISP_MODULE_CTRL_U *punModCtrl);

int HB_ISP_GetModuleControl(uint8_t pipeId, ISP_MODULE_CTRL_U *punModCtrl);

[Function Description]

Set / get the bypass status of each module in ISP.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
punModCtrl	bypass control inner modules of ISP	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.1.4 HB_ISP_SwitchScence

[Function Declaration]

int HB_ISP_SwitchScence(uint8_t pipeId, const char *cname);

[Function Description]

Set calibration library.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
cname	Path of Calibration library	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.1.5 HB_ISP_StartI2CBus/HB_ISP_StopI2CBus

[Function Declaration]

int HB_ISP_StartI2CBus(uint8_t pipeId);

void HB_ISP_StopI2CBus(uint8_t pipeId);

[Function Description]

Start/stop thread of I2C-write.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

[attention]

HB_ISP_GetSetInit should be called before these two interface.

2.1.6 HB_ISP_SendI2CData

[Function Declaration]

int HB_ISP_SendI2CData(ISP_I2C_DATA_S data);

[Function Description]

Write data to sensor via I2C.

[Parameter Description]

Parameter	Description	Input/Output
Data	Data that will be send	Input

[Return Value]

Return Value	Description
0	pass
Non 0	failed

[attention]

Before using, call HB_ISP_StartI2CBus interface to start writing I2C thread.

2.2 External 3A Library Register Interface

int HB_ISP_AELibRegCallback(uint8_t pipeId, char *name, ISP_AE_FUNC_S *pstAeFunc);

int HB_ISP_AWBLibRegCallback(uint8_t pipeId, char *name, ISP_AWB_FUNC_S *pstAWBFunc);

int HB_ISP_AFLibRegCallback(uint8_t pipeId, char *name, ISP_AF_FUNC_S *pstAFFunc);

int HB_ISP_AELibUnRegCallback(uint8_t pipeId);

int HB_ISP_AWBLibUnRegCallback(uint8_t pipeId);

int HB_ISP_AFLibUnRegCallback(uint8_t pipeId);

Note:

1. If you use the default 3A algorithm, you do not need to pay attention to the interface in this chapter.
2. Without zoom and aperture control algorithm, users can implement it by themselves, and give out output parameters to modify ISP firmware for adaptation.

2.2.1 HB_ISP_AELibRegCallback

[Function Declaration]

int HB_ISP_AELibRegCallback(uint8_t pipeId, char *name, ISP_AE_FUNC_S *pstAeFunc);

[Function Description]

AE library register.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input
name	Library name, length 20 bytes	Input
pstAeFunc	AE algorithm callback function pointer	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Note]

HB_VIN_StartPipe will start the algorithm library. You need to register the algorithm before calling the HB_VIN_StartPipe function.

[Reference code]

Refer to: Example of Algorithm Register

2.2.2 HB_ISP_AWBLibRegCallback

[Function Declaration]

int HB_ISP_AWBLibRegCallback(uint8_t pipeId, char *name, ISP_AWB_FUNC_S *pstAWBFunc);

[Function Description]

AWB library register.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input
name	Library name, length 20 bytes	Input
pstAWBFunc	AWB algorithm callback function pointer	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Note]

HB_VIN_StartPipe will start the algorithm library. You need to register the algorithm before calling the HB_VIN_StartPipe function.

[Reference code]

Refer to: Example of Algorithm Register

2.2.3 HB_ISP_AFLibRegCallback

[Function Declaration]

int HB_ISP_AFLibRegCallback(uint8_t pipeId, char *name, ISP_AF_FUNC_S *pstAFFunc);

[Function Description]

AF library register.

[Parameter Description]

Parameter	Description	Input/Output

pipeId	Pipeline index number	Input
name	Library name, length 20 bytes	Input
pstAFFunc	AF algorithm callback function pointer	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Reference code]

Refer to: Example of Algorithm Register

2.2.4 HB_ISP_AELibUnRegCallback

[Function Declaration]

int HB_ISP_AELibUnRegCallback(uint8_t pipeId);

[Function Description]

AE library unregister.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input

[Return Value]

Return Value	Description
0	pass

non 0	failed

2.2.5 HB_ISP_AWBLibUnRegCallback

[Function Declaration]

int HB_ISP_AWBLibUnRegCallback(uint8_t pipeId);

[Function Description]

AWB library unregister.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.2.6 HB_ISP_AFLibUnRegCallback

[Function Declaration]

int HB_ISP_AFLibUnRegCallback(uint8_t pipeId);

[Function Description]

AF library unregister.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3 App Tuning Interface

int HB_ISP_GetSetInit(void);

int HB_ISP_GetSetExit(void);

int HB_ISP_SetAeAttr(uint8_t pipeId, const ISP_AE_ATTR_S *pstAeAttr);

int HB_ISP_GetAeAttr(uint8_t pipeId, ISP_AE_ATTR_S *pstAeAttr);

int HB_ISP_SetAfAttr(uint8_t pipeId, ISP_AF_ATTR_S *pstAfAttr);

int HB_ISP_GetAfAttr(uint8_t pipeId, ISP_AF_ATTR_S *pstAfAttr);

int HB_ISP_SetAwbAttr(uint8_t pipeId, const ISP_AWB_ATTR_S *pstAwbAttr);

int HB_ISP_GetAwbAttr(uint8_t pipeId, ISP_AWB_ATTR_S *pstAwbAttr);

int HB_ISP_SetBlackLevelAttr(uint8_t pipeId, const ISP_BLACK_LEVEL_ATTR_S *pstBlackLevelAttr);

int HB_ISP_GetBlackLevelAttr(uint8_t pipeId, ISP_BLACK_LEVEL_ATTR_S *pstBlackLevelAttr);

int HB_ISP_SetDemosaicAttr(uint8_t pipeId, const ISP_DEMOSAIC_ATTR_S *pstDemosaicAttr);

int HB_ISP_GetDemosaicAttr(uint8_t pipeId, ISP_DEMOSAIC_ATTR_S *pstDemosaicAttr);

int HB_ISP_SetSharpenAttr(uint8_t pipeId, const ISP_SHARPEN_ATTR_S *pstSharpenAttr);

int HB_ISP_GetSharpenAttr(uint8_t pipeId, ISP_SHARPEN_ATTR_S *pstSharpenAttr);

int HB_ISP_SetGammaAttr(uint8_t pipeId, const ISP_GAMMA_ATTR_S *pstGammaAttr);

int HB_ISP_GetGammaAttr(uint8_t pipeId, ISP_GAMMA_ATTR_S *pstGammaAttr);

int HB_ISP_SetIridixAttr(uint8_t pipeId, const ISP_IRIDIX_ATTR_S *pstIridixAttr);

int HB_ISP_GetIridixAttr(uint8_t pipeId, ISP_IRIDIX_ATTR_S *pstIridixAttr);

int HB_ISP_SetIridixStrengthLevel(uint8_t pipeId, uint16_t level);

int HB_ISP_GetIridixStrengthLevel(uint8_t pipeId, uint16_t *level);

int HB_ISP_SetCnrAttr(uint8_t pipeId, const ISP_CNR_ATTR_S *pstCnrAttr);

int HB_ISP_GetCnrAttr(uint8_t pipeId, ISP_CNR_ATTR_S *pstCnrAttr);

int HB_ISP_SetSinterAttr(uint8_t pipeId, const ISP_SINTER_ATTR_S *pstSinterAttr);

int HB_ISP_GetSinterAttr(uint8_t pipeId, ISP_SINTER_ATTR_S *pstSinterAttr);

int HB_ISP_SetTemperAttr(uint8_t pipeId, const ISP_TEMPER_ATTR_S *pstTemperAttr);

int HB_ISP_GetTemperAttr(uint8_t pipeId, ISP_TEMPER_ATTR_S *pstTemperAttr);

int HB_ISP_SetMeshShadingAttr(uint8_t pipeId, const MESH_SHADING_ATTR_S *pstMeshShadingAttr);

int HB_ISP_GetMeshShadingAttr(uint8_t pipeId, MESH_SHADING_ATTR_S *pstMeshShadingAttr);

int HB_ISP_SetMeshShadingLUT(uint8_t pipeId, const MESH_SHADING_LUT_S *pstMeshShadingLUT);

int HB_ISP_GetMeshShadingLUT(uint8_t pipeId, MESH_SHADING_LUT_S *pstMeshShadingLUT);

int HB_ISP_SetRadialShadingAttr(uint8_t pipeId, const RADIAL_SHADING_ATTR_S *pstRadialShadingAttr);

int HB_ISP_GetRadialShadingAttr(uint8_t pipeId, RADIAL_SHADING_ATTR_S *pstRadialShadingAttr);

int HB_ISP_SetRadialShadingLUT(uint8_t pipeId, const RADIAL_SHADING_LUT_S *pstRadialShadingLUT);

int HB_ISP_GetRadialShadingLUT(uint8_t pipeId, RADIAL_SHADING_LUT_S *pstRadialShadingLUT);

int HB_ISP_SetCSCAttr(uint8_t pipeId, const ISP_CSC_ATTR_S *pstCSCAttr);

int HB_ISP_GetCSCAttr(uint8_t pipeId, ISP_CSC_ATTR_S *pstCSCAttr);

int HB_ISP_SetSceneModesAttr(uint8_t pipeId, const ISP_SCENE_MODES_ATTR_S *pstSceneModesAttr);

int HB_ISP_GetSceneModesAttr(uint8_t pipeId, ISP_SCENE_MODES_ATTR_S *pstSceneModesAttr);

int HB_ISP_GetAwbZoneInfo(uint8_t pipeId, ISP_ZONE_ATTR_S *awbZoneInfo);

int HB_ISP_SetAwbZoneInfo(uint8_t pipeId, ISP_ZONE_ATTR_S awbZoneInfo);

int HB_ISP_GetAfZoneInfo(uint8_t pipeId, ISP_ZONE_ATTR_S *afZoneInfo);

int HB_ISP_SetAfZoneInfo(uint8_t pipeId, ISP_ZONE_ATTR_S afZoneInfo);

int HB_ISP_GetAe5binZoneInfo(uint8_t pipeId, ISP_ZONE_ATTR_S *ae5binZoneInfo);

int HB_ISP_SetAe5binZoneInfo(uint8_t pipeId, ISP_ZONE_ATTR_S ae5binZoneInfo);

int HB_ISP_GetAfKernelInfo(uint8_t pipeId, uint32_t *af_kernel);

int HB_ISP_SetAfKernelInfo(uint8_t pipeId, uint32_t af_kernel);

int HB_ISP_GetAeFullHist(uint8_t pipeId, uint32_t *pu32AeFullHist);

int HB_ISP_GetAwbZoneHist(uint8_t pipeId, ISP_STATISTICS_AWB_ZONE_ATTR_S *pstAwbZonesAttr);

int HB_ISP_GetAe5binZoneHist(uint8_t pipeId, ISP_STATISTICS_AE_5BIN_ZONE_ATTR_S

*pst32Ae5bin);

int HB_ISP_GetAfZoneHist(uint8_t pipeId, af_stats_data_t *pstAfZonesAttr);

int HB_ISP_GetVDTTimeOut(uint8_t pipeId, uint8_t vdt_type, uint64_t timeout);

int HB_ISP_GetLumaZoneHist(uint8_t pipeId, ISP_STATISTICS_LUMVAR_ZONE_ATTR_S *pst32Luma);

int HB_ISP_GetAwbStatAreaAttr(uint8_t pipeId, ISP_AWB_STAT_AREA_ATTR_S *pstAwbStatAreaAttr);

int HB_ISP_SetAwbStatAreaAttr(uint8_t pipeId, ISP_AWB_STAT_AREA_ATTR_S *pstAwbStatAreaAttr);

int HB_ISP_SetAeParam(uint8_t pipeId, const ISP_AE_PARAM_S *pstAeParam);

int HB_ISP_GetAeParam(uint8_t pipeId, const ISP_AE_PARAM_S *pstAeParam);

int HB_ISP_SetAeRoiInfo(uint8_t pipeId, ISP_AE_ROI_ATTR_S aeRoiInfo);

int HB_ISP_GetAeRoiInfo(uint8_t pipeId, ISP_AE_ROI_ATTR_S *aeRoiInfo);

2.3.1 HB_ISP_GetSetInit/HB_ISP_GetSetExit

[Function Declaration]

int HB_ISP_GetSetInit(void);

int HB_ISP_GetSetExit(void);

[Function Description]

Init function before calling Set/Get interface.

[Parameter Description]

Non

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Note]

Just call once in the same process. Before calling all get / set class interfaces, first call Hb_ ISP_ Getsetinit for initialization.

2.3.2 HB_ISP_SetAeAttr/HB_ISP_GetAeAttr

[Function Declaration]

int HB_ISP_SetAeAttr(uint8_t pipeId, const ISP_AE_ATTR_S *pstAeAttr);

int HB_ISP_GetAeAttr(uint8_t pipeId, ISP_AE_ATTR_S *pstAeAttr);

[Function Description]

Set AE releated properties.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input
pstAeAttr	Pointer to AE parameter	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Note]

Set is only for manual mode, and get is transferred to different modes to get the corresponding mode value.

2.3.3 HB_ISP_SetAfAttr/HB_ISP_GetAfAttr

[Function Declaration]

int HB_ISP_SetAfAttr(uint8_t pipeId, ISP_AF_ATTR_S *pstAfAttr);

int HB_ISP_GetAfAttr(uint8_t pipeId, ISP_AF_ATTR_S *pstAfAttr);

[Function Description]

Set AF-Zoom property.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input

pstAfAttr	Pointer to AF parameter	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.4 HB_ISP_SetAwbAttr/HB_ISP_GetAwbAttr

[Function Declaration]

int HB_ISP_SetAwbAttr(uint8_t pipeId, const ISP_AWB_ATTR_S *pstAwbAttr);

int HB_ISP_GetAwbAttr(uint8_t pipeId, ISP_AWB_ATTR_S *pstAwbAttr);

[Function Description]

Set AWB properties.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input
pstAwbAttr	Pointer to AWB parameter	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Note]

Set is only for manual mode, and get is transferred to different modes to get the corresponding mode value.

2.3.5 HB_ISP_SetBlackLevelAttr/HB_ISP_GetBlackLevelAttr

[Function Declaration]

int HB_ISP_SetBlackLevelAttr(uint8_t pipeId, const ISP_BLACK_LEVEL_ATTR_S *pstBlackLevelAttr);

int HB_ISP_GetBlackLevelAttr(uint8_t pipeId, ISP_BLACK_LEVEL_ATTR_S *pstBlackLevelAttr);

[Function Description]

Set properties of Black Level.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pstBlackLevelAttr	A Pointer point to blacklevel attr	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Attention]

Set only for manual mode, pass different mode to get interface will get corresponding value.

In auto mode,black_level value is calibraiton parameter BLACK_LEVEL_B/BLACK_LEVEL_GB/BLACK_LEVEL_GR/BLACK_LEVEL_R, interpolation according to the current exposure gain.

In Manual mode, user can set black_level value.

[Reference Code]

2.3.6 HB_ISP_SetDemosaicAttr/HB_ISP_GetDemosaicAttr

[Function Declaration]

int HB_ISP_SetDemosaicAttr(uint8_t pipeId, const ISP_DEMOSAIC_ATTR_S *pstDemosaicAttr);

int HB_ISP_GetDemosaicAttr(uint8_t pipeId, ISP_DEMOSAIC_ATTR_S *pstDemosaicAttr);

[Function Description]

Set the properties of the demosaic module.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input
pstDemosaicAttr	Pointer to the desmosaics parameter	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.7 HB_ISP_SetSharpenAttr/HB_ISP_GetSharpenAttr

[Function Declaration]

int HB_ISP_SetSharpenAttr(uint8_t pipeId, const ISP_SHARPEN_ATTR_S *pstSharpenAttr);

int HB_ISP_GetSharpenAttr(uint8_t pipeId, ISP_SHARPEN_ATTR_S *pstSharpenAttr);

[Function Description]

Set sharpen properties.

[Parameter Description]

Parameter	Description	Input/Output

pipeId	Pipeline index number	Input
pstSharpenAttr	Pointer to sharpen parameter	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.8 HB_ISP_SetGammaAttr/HB_ISP_GetGammaAttr

[Function Declaration]

int HB_ISP_SetGammaAttr(uint8_t pipeId, const ISP_GAMMA_ATTR_S *pstGammaAttr);

int HB_ISP_GetGammaAttr(uint8_t pipeId, ISP_GAMMA_ATTR_S *pstGammaAttr);

[Function Description]

Set gamma properties.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input
pstGammaAttr	Pointer to gamma parameter	Input

[Return Value]

Return Value	Description
0	pass

non 0	failed

2.3.9 HB_ISP_SetIridixAttr/HB_ISP_GetIridixAttr

[Function Declaration]

int HB_ISP_SetIridixAttr(uint8_t pipeId, const ISP_IRIDIX_ATTR_S *pstIridixAttr);

int HB_ISP_GetIridixAttr(uint8_t pipeId, ISP_IRIDIX_ATTR_S *pstIridixAttr);

[Function Description]

Setting iridix module properties.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipeline index number	Input
pstIridixAttr	Pointer to iridix parameter	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.10 HB_ISP_SetIridixStrengthLevel/HB_ISP_GetIridixStrengthLevel

[Function Declaration]

int HB_ISP_SetIridixStrengthLevel(uint8_t pipeId, uint16_t level);

int HB_ISP_GetIridixStrengthLevel(uint8_t pipeId, uint16_t *level);

[Function Description]

Set Iridix strength level.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
level	Strength level, range [0, 255]	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.11 HB_ISP_SetCnrAttr/HB_ISP_GetCnrAttr

[Function Declaration]

int HB_ISP_SetCnrAttr(uint8_t pipeId, const ISP_CNR_ATTR_S *pstCnrAttr);

int HB_ISP_GetCnrAttr(uint8_t pipeId, ISP_CNR_ATTR_S *pstCnrAttr);

[Function Description]

Set chroma noise reduction module properties.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pstCnrAttr	Pointer to chroma noise reduction	Input

	parameter	

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.12 HB_ISP_SetSinterAttr/HB_ISP_GetSinterAttr

[Function Declaration]

int HB_ISP_SetSinterAttr(uint8_t pipeId, const ISP_SINTER_ATTR_S *pstSinterAttr);

int HB_ISP_GetSinterAttr(uint8_t pipeId, ISP_SINTER_ATTR_S *pstSinterAttr);

[Function Description]

Set attributes of spatial noise reduction module.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pstSinterAttr	Pointer to spatial noise reduction parameter	Input

[Return Value]

Return Value	Description
0	pass

non 0	failed

2.3.13 HB_ISP_SetTemperAttr/HB_ISP_GetTemperAttr

[Function Declaration]

int HB_ISP_SetTemperAttr(uint8_t pipeId, const ISP_TEMPER_ATTR_S *pstTemperAttr);

int HB_ISP_GetTemperAttr(uint8_t pipeId, ISP_TEMPER_ATTR_S *pstTemperAttr);

[Function Description]

Set temporal noise reduction module properties.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pstTemperAttr	Pointer to temper noise reduction parameter	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.14 HB_ISP_SetMeshShadingAttr/HB_ISP_GetMeshShadingAttr

[Function Declaration]

int HB_ISP_SetMeshShadingAttr(uint8_t pipeId, const MESH_SHADING_ATTR_S *pstMeshShadingAttr);

int HB_ISP_GetMeshShadingAttr(uint8_t pipeId, MESH_SHADING_ATTR_S *pstMeshShadingAttr);

[Function Description]

Set properties of Mesh Shading.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pstMeshShadingAttr	A pointer point to MeshShading parameter	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.15 HB_ISP_SetMeshShadingLUT/HB_ISP_GetMeshShadingLUT

[Function Declaration]

int HB_ISP_SetMeshShadingLUT(uint8_t pipeId, const MESH_SHADING_LUT_S *pstMeshShadingLUT);

int HB_ISP_GetMeshShadingLUT(uint8_t pipeId, MESH_SHADING_LUT_S *pstMeshShadingLUT);

[Function Description]

Set properties of Mesh Shading LUT.

[Parameter Description]

Parameter	Description	Input/Output

pipeId	Pipelineindex number	Input
pstMeshShadingLUT	A pointer point to MeshShading LUT	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.16 HB_ISP_SetRadialShadingAttr/HB_ISP_GetRadialShadingAttr

[Function Declaration]

int HB_ISP_SetRadialShadingAttr(uint8_t pipeId, const RADIAL_SHADING_ATTR_S *pstRadialShadingAttr);

int HB_ISP_GetRadialShadingAttr(uint8_t pipeId, RADIAL_SHADING_ATTR_S *pstRadialShadingAttr);

[Function Description]

Set properties of Radial Shading.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pstRadialShadingAttr	A pointer point to Radial Shading	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.17 HB_ISP_SetRadialShadingLUT/HB_ISP_GetRadialShadingLUT

[Function Declaration]

int HB_ISP_SetRadialShadingLUT(uint8_t pipeId, const RADIAL_SHADING_LUT_S *pstRadialShadingLUT);

int HB_ISP_GetRadialShadingLUT(uint8_t pipeId, RADIAL_SHADING_LUT_S *pstRadialShadingLUT);

[Function Description]

Set properties of Radial Shading LUT.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pstRadialShadingLUT	A pointer point to Radial Shading LUT	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Attention]

[Reference Code]

2.3.18 HB_ISP_SetCSCAttr/HB_ISP_GetCSCAttr

[Function Declaration]

int HB_ISP_SetCSCAttr(uint8_t pipeId, const ISP_CSC_ATTR_S *pstCSCAttr);

int HB_ISP_GetCSCAttr(uint8_t pipeId, ISP_CSC_ATTR_S *pstCSCAttr);

[Function Description]

Set properties of CSC.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pstCSCAttr	A pointer point to CSC parameter	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Attention]

[Reference Code]

2.3.19 HB_ISP_SetSceneModesAttr/HB_ISP_GetSceneModesAttr

[Function Declaration]

int HB_ISP_SetSceneModesAttr(uint8_t pipeId, const ISP_SCENE_MODES_ATTR_S *pstSceneModesAttr);

int HB_ISP_GetSceneModesAttr(uint8_t pipeId, ISP_SCENE_MODES_ATTR_S *pstSceneModesAttr);

[Function Description]

Set scene mode.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pstSceneModesAttr	Pointer to the scene mode parameter	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.20 HB_ISP_SetAwbZoneInfo/HB_ISP_GetAwbZoneInfo

[Function Declaration]

int HB_ISP_GetAwbZoneInfo(uint8_t pipeId, ISP_ZONE_ATTR_S *awbZoneInfo);

int HB_ISP_SetAwbZoneInfo(uint8_t pipeId, ISP_ZONE_ATTR_S awbZoneInfo);

[Function Description]

Get/set AWB zones information.

[Parameter Description]

Parameter	Description	Input/Output

pipeId	Pipelineindex number	Input
awbZoneInfo	A pointer point to AWB zones info	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.21 HB_ISP_SetAfZoneInfo/HB_ISP_GetAfZoneInfo

[Function Declaration]

int HB_ISP_GetAfZoneInfo(uint8_t pipeId, ISP_ZONE_ATTR_S *afZoneInfo);

int HB_ISP_SetAfZoneInfo(uint8_t pipeId, ISP_ZONE_ATTR_S afZoneInfo);

[Function Description]

Get/set AF zones information.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
afZoneInfo	A pointer point to AF zones info	Input

[Return Value]

Return Value	Description
0	pass

non 0	failed

2.3.22 HB_ISP_SetAe5binZoneInfo/HB_ISP_GetAe5binZoneInfo

[Function Declaration]

int HB_ISP_GetAe5binZoneInfo(uint8_t pipeId, ISP_ZONE_ATTR_S *ae5binZoneInfo);

int HB_ISP_SetAe5binZoneInfo(uint8_t pipeId, ISP_ZONE_ATTR_S ae5binZoneInfo);

[Function Description]

Get/set AE 5bin zones information.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
ae5binZoneInfo	A pointer point to AE 5bin zones info	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Attention]

[Reference Code]

2.3.23 HB_ISP_SetAfKernelInfo/HB_ISP_GetAfKernelInfo

[Function Declaration]

int HB_ISP_GetAfKernelInfo(uint8_t pipeId, uint32_t *af_kernel);

int HB_ISP_SetAfKernelInfo(uint8_t pipeId, uint32_t af_kernel);

[Function Description]

Get/set AF KERNEL information.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
af_kernel	A pointer point to af_kernel	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Attention]

This interface is mainly used by applications, such as getting or changing the current AF statistical data kernel.

[Reference Code]

2.3.24 HB_ISP_SetAeParam/HB_ISP_GetAeParam

[Function Declaration]

int HB_ISP_SetAeParam(uint8_t pipeId, const ISP_AE_PARAM_S *pstAeParam);

int HB_ISP_GetAeParam(uint8_t pipeId, ISP_AE_PARAM_S *pstAeParam);

[Function Description]

Set/get AE parameters, such as line and total_gain.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input

pstAeParam	A pointer point to ae parameter	Input

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.25 HB_ISP_SetAeRoiInfo/HB_ISP_GetAeRoiInfo

[Function Declaration]

int HB_ISP_SetAeRoiInfo(uint8_t pipeId, ISP_AE_ROI_ATTR_S aeRoiInfo);

int HB_ISP_GetAeRoiInfo(uint8_t pipeId, ISP_AE_ROI_ATTR_S *aeRoiInfo);

[Function Description]

Set ae ROI weight.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
aeRoiInfo	Roi info	Output

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Attention]

The ROI region setting is independent from the ROI weight setting in 3a-ae, and only one of the functions can be used at the same time.

2.3.26 HB_ISP_SetAwbStatAreaAttr/HB_ISP_GetAwbStatAreaAttr

[Function Declaration]

int HB_ISP_GetAwbStatAreaAttr(uint8_t pipeId, ISP_AWB_STAT_AREA_ATTR_S *pstAwbStatAreaAttr);

int HB_ISP_SetAwbStatAreaAttr(uint8_t pipeId, ISP_AWB_STAT_AREA_ATTR_S *pstAwbStatAreaAttr);

[Function Description]

Set awb metering data range that will be counted.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pstAwbStatAreaAttr	Parameters of metering data range	Output

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.3.27 HB_ISP_GetAeFullHist

[Function Declaration]

int HB_ISP_GetAeFullHist(uint8_t pipeId, uint32_t *pu32AeFullHist);

[Function Description]

Get AE full histogram.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pu32AeFullHist	Pointer to AE histogram	Output

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Note]

This interface is mainly used for applications. For example, it is unnecessary for 3A algorithm library to use this interface to make some policy judgments based on statistical information.

[Reference Code]

```
#define HB_ISP_FULL_HISTOGRAM_SIZE 1024
int i;
uint32_t ae[HB_ISP_FULL_HISTOGRAM_SIZE];
memset(ae, 0, sizeof(ae));
HB_ISP_ AppGetAeFullHist(0, ae);
printf("\n--AE--\n");
for (i = 0; i < HB_ISP_FULL_HISTOGRAM_SIZE; i++) {
    printf("%-8d       ", ae[i]);
    if ((i + 1) % 8 == 0)
        printf("\n");
}
```

2.3.28 HB_ISP_GetAwbZoneHist

[Function Declaration]

int HB_ISP_GetAwbZoneHist(uint8_t pipeId, ISP_STATISTICS_AWB_ZONE_ATTR_S *pstAwbZonesAttr);

[Function Description]

Get AWB metering data.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pstAwbZonesAttr	Point to AWB metering data	Output

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Attention]

This interface is mainly used by applications. For example, it is not necessary for 3A algorithm library to make some policy judgments based on statistical information.

[Reference Code]

2.3.29 HB_ISP_GetAe5binZoneHist

[Function Declaration]

int HB_ISP_GetAe5binZoneHist(uint8_t pipeId, ISP_STATISTICS_AE_5BIN_ZONE_ATTR_S *pst32Ae5bin);

[Function Description]

Get AE 5bin metering data.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pst32Ae5bin	Point to AE 5binmetering data	Output

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Attention]

This interface is mainly used by applications. For example, it is not necessary for 3A algorithm library to make some policy judgments based on statistical information.

[Reference Code]

```
ISP_STATISTICS_AE_5BIN_ZONE_ATTR_S ae_5bin[HB_ISP_MAX_AE_5BIN_ZONES];
ISP_ZONE_ATTR_S ae5binZoneInfo;
memset(ae_5bin, 0, sizeof(ae_5bin));
HB_ISP_GetAe5binZoneHist(ctx_idx, ae_5bin);
```

2.3.30 HB_ISP_GetAfZoneHist

[Function Declaration]

int HB_ISP_GetAfZoneHist(uint8_t pipeId, af_stats_data_t *pstAfZonesAttr);

[Function Description]

Get AF metering data。

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pstAfZonesAttr	point to AF metering data	Output

[Return Value]

Return Value	Description
0	pass
non 0	failed

[Attention]

This interface is mainly used by applications. For example, it is not necessary for 3A algorithm library to make some policy judgments based on statistical information.

[Reference Code]

```
uint32_t af_data[HB_ISP_AF_ZONES_COUNT_MAX * 2];
af_stats_data_t af;
ISP_ZONE_ATTR_S afZoneInfo;
memset(af_data, 0, sizeof(af_data));
af.zones_stats = (uint32_t *)&af_data;
HB_ISP_GetAfZoneHist(ctx_idx, &af);
```

2.3.31 HB_ISP_GetVDTTimeOut

[Function Declaration]

int HB_ISP_GetVDTTimeOut(uint8_t pipeId, uint8_t vdt_type, uint64_t timeout);

[Function Description]

get ISP FRAME_START or FRAME_END information.

[Parameter Description]

Parameter	Description	Input/Output

pipeId	Pipelineindex number	Input
vdt_type	get ISP frame_start or frame_end	Input
timeout	Timeout value	Input

[Return Value]

Return Value	Description
0	FS/FE sync
non 0	timeout

[Attention]

The interface is mainly used for applications, such as timing synchronization.

2.3.32 HB_ISP_GetLumaZoneHist

[Function Declaration]

int HB_ISP_GetLumaZoneHist(uint8_t pipeId, ISP_STATISTICS_LUMVAR_ZONE_ATTR_S *pst32Luma);

[Function Description]

get LUMVAR Statistical information mean and variance.

[Parameter Description]

Parameter	Description	Input/Output
pipeId	Pipelineindex number	Input
pst32Luma	Point to lumvar metering data	Output

[Return Value]

Return Value	Description
0	pass
non 0	failed

2.4 Data Structure

2.4.1 Control

2.4.1.1 HB_ISP_FW_STATE_E

[Structure Definition]

```
typedef enum HB_ISP_FW_STATE_E {
        ISP_FW_STATE_RUN = 0,
        ISP_FW_STATE_FREEZE,
} ISP_FW_STATE_E;
```

[Function Description]

ISP run state control.

[Field Description]

Field	Description
ISP_FW_STATE_RUN	Firmware run normal
ISP_FW_STATE_FREEZE	Firmware freeze

2.4.1.2 HB_ ISP_MODULE_CTRL_U

[Structure Definition]

```
typedef union HB_ISP_MODULE_CTRL_U {
        uint64_t u64Value;
        struct {
                uint64_t bitBypassVideoTestGen      : 1 ;
                uint64_t bitBypassInputFormatter    : 1 ;
                uint64_t bitBypassDecompander       : 1 ;
                uint64_t bitBypassSensorOffsetWDR   : 1 ;
```

```
        uint64_t bitBypassGainWDR                   : 1 ;
        uint64_t bitBypassFrameStitch               : 1 ;
        uint64_t bitBypassDigitalGain               : 1 ;
        uint64_t bitBypassFrontendSensorOffset      : 1 ;
        uint64_t bitBypassFeSqrt                    : 1 ;
        uint64_t bitBypassRAWFrontend               : 1 ;
        uint64_t bitBypassDefectPixel               : 1 ;
        uint64_t bitBypassSinter                    : 1 ;
        uint64_t bitBypassTemper                    : 1 ;
        uint64_t bitBypassCaCorrection              : 1 ;
        uint64_t bitBypassSquareBackend             : 1 ;
        uint64_t bitBypassSensorOffsetPreShading    : 1 ;
        uint64_t bitBypassRadialShading             : 1 ;
        uint64_t bitBypassMeshShading               : 1 ;
        uint64_t bitBypassWhiteBalance              : 1 ;
        uint64_t bitBypassIridixGain                : 1 ;
        uint64_t bitBypassIridix                    : 1 ;
        uint64_t bitBypassMirror                    : 1 ;
        uint64_t bitBypassDemosaicRGB               : 1 ;
        uint64_t bitBypassPfCorrection              : 1 ;
        uint64_t bitBypassCCM                       : 1 ;
        uint64_t bitBypassCNR                       : 1 ;
        uint64_t bitBypass3Dlut                     : 1 ;
        uint64_t bitBypassNonequGamma               : 1 ;
        uint64_t bitBypassFrCrop                    : 1 ;
        uint64_t bitBypassFrGammaRGB                : 1 ;
        uint64_t bitBypassFrSharpen                 : 1 ;
        uint64_t bitBypassFrCsConv                  : 1 ;
        uint64_t bitBypassRAW                       : 1 ;
        uint64_t bitRsv                             : 31 ;
    };
} ISP_MODULE_CTRL_U;
```

[Function Description]

Bit of ISP Inner module bypass control.

2.4.1.3 HB_ISP_I2C_DATA_S

[Structure Definition]

```
typedef struct HB_ISP_I2C_DATA_S {
        uint8_t u8DevId;
        uint8_t u8IntPos;
        uint8_t u8Update;
        uint8_t u8DelayFrameNum;
        uint32_t u32RegAddr;
        uint32_t u32AddrByteNum;
        uint32_t u32Data;
        uint32_t u32DataByteNum;
} ISP_I2C_DATA_S;
```

[Function Description]

[Field Description]

Field	Description
u8DevId	pipeline id
u8IntPos	0 frame_start 1 frame_done
u8Update	0 update_disable 1 update_enable
u8DelayFrameNum	delay frame [0,5]
u32RegAddr	sensor reg addr eg. 0x3058
u32AddrByteNum	sensor reg num[0,4] eg.2
u32Data	Data to write
u32DataByteNum	sensor reg num[0,4]

2.4.2 3A Library

2.4.2.1 HB_ISP_AE_FUNC_S

[Structure Definition]

```
typedef struct HB_ISP_AE_FUNC_S {
    void *( *init_func )( uint32_t ctx_id );
int32_t ( *proc_func )( void *ae_ctx, ae_stats_data_t *stats,
        ae_input_data_t *input, ae_output_data_t *output );
    int32_t ( *deinit_func )( void *ae_ctx );
} ISP_AE_FUNC_S;
```

[Function Description]

Definition of AE library callback function structure.

[Field Description]

Field	Description
init_func	Description: AE algorithm init Parameter: 1. ctx_id input parameter，pipeline index number Return: A pointer point to one pipeline
proc_func	Description: AE algorithm implementation Parameter: 1. ae_ctx Input, return value of init_func 2. stats Input, it is AE metering data 3. input Input, parameters from ISP

	Firmware 4. output Output, result of AE algorithm, will be set to ISP Firmware 5. Return 0 pass, 1 failed
deinit_func	Description： AE deinit Parameter: 1. ae_ctx Input, return value of init_func

ae_stats_data_t

[Structure Definition]

```
typedef struct _ae_stats_data_ {
    uint32_t *fullhist;
    uint32_t fullhist_size;
    uint32_t fullhist_sum;
    uint16_t *zone_hist;
    uint32_t zone_hist_size;
} ae_stats_data_t;
```

[Function Description]

AE metering structure definition. See AE metering section for detailed description of AE statistics.

[Field Description]

Field	Description
fullhist	Pointer of full histogram
fullhist_size	Size of full histogram
fullhist_sum	Number of points participating in statistics
zone_hist	Zonal histogram pointer
zone_hist_size	Size of zonal histogram

ae_input_data_t

[Structure Definition]

typedef struct _ae_input_data_ {

    void *custom_input;

    void *acamera_input;

} ae_input_data_t;

[Function Description]

Input parameter structure definition of AE algorighm.

[Field Description]

Field	Description
custom_input	reserved
acamera_input	Parameter AE algorithm really use, type is ae_acamera_input_t

ae_acamera_input_t

[Structure Definition]

typedef struct _ae_acamera_input_ {

    uint32_t ctx_id;

    ae_balanced_param_t *ae_ctrl;

    ae_misc_info_t misc_info;

    ae_calibration_data_t cali_data;

} ae_acamera_input_t;

[Function Description]

Definition the input parameter that will pass to AE algorighm.

Below structure included:

typedef struct _ae_balanced_param_t {

    uint32_t pi_coeff;

    uint32_t target_point;

    uint32_t tail_weight;

    uint32_t long_clip;

    uint32_t er_avg_coeff;

```
    uint32_t hi_target_prc;
    uint32_t hi_target_p;
    uint32_t enable_iridix_gdg;
    uint32_t AE_tol;
} ae_balanced_param_t;

typedef struct _ae_misc_info_ {
    int32_t sensor_exp_number;
    int32_t total_gain;
    int32_t max_exposure_log2;
    uint32_t global_max_exposure_ratio;
    uint32_t iridix_contrast;
    uint32_t global_exposure;
    uint8_t global_ae_compensation;
    uint8_t global_manual_exposure;
    uint8_t global_manual_exposure_ratio;
    uint8_t global_exposure_ratio;
} ae_misc_info_t;

typedef struct _ae_calibration_data_ {
    uint8_t *ae_corr_lut;
    uint32_t ae_corr_lut_len;
    uint32_t *ae_exp_corr_lut;
    uint32_t ae_exp_corr_lut_len;
    modulation_entry_t *ae_hdr_target;
    uint32_t ae_hdr_target_len;
    modulation_entry_t *ae_exp_ratio_adjustment;
    uint32_t ae_exp_ratio_adjustment_len;
} ae_calibration_data_t;
typedef struct _ae_5bin_info_ {
        uint32_t zones_size;
        uint16_t zones_v;
```

```
    uint16_t zones_h;
    uint16_t threshold0_1;
    uint16_t threshold1_2;
    uint16_t threshold3_4;
    uint16_t threshold4_5;
    uint16_t normal_bin0;
    uint16_t normal_bin1;
    uint16_t normal_bin3;
    uint16_t normal_bin4;
} ae_5bin_info_t;
typedef struct _modulation_entry_t {
    uint16_t x, y;
} modulation_entry_t;
```

ae_output_data_t

[Structure Definition]

```
typedef struct _ae_output_data_ {
    void *custom_output;
    void *acamera_output;
} ae_output_data_t;
```

[Function Description]

Definition the output parameter that AE algorighm output.

[Field Description]

Field	Description
custom_output	reserved
acamera_output	Output parameter of AE algorighm, type is ae_acamera_output_t

ae_acamera_output_t

[Structure Definition]

```
typedef struct _ae_acamera_output_ {
    int32_t exposure_log2;
    uint32_t exposure_ratio;
```

```
    uint8_t ae_converged;
    uint16_t sensor_ctrl_enable;
    ae_out_info_t ae_out_info;
    ae_1024bin_weight_t ae_1024bin_weight;
} ae_acamera_output_t;
```

[Function Description]

Definition the output parameter that AE algorighm output.

[Field Description]

Field	Description
exposure_log2	exposure target
exposure_ratio	Exposure ratio
ae_converged	AE algorithm state, running or done
sensor_ctrl_enable	Sensor control enable
ae_out_info	AE related control information
ae_1024bin_weight	Global statistical information weight

2.4.2.2 HB_ISP_AWB_FUNC_S

[Structure Definition]

```
typedef struct HB_ISP_AWB_FUNC_S {
    void *( *init_func )( uint32_t ctx_id );
int32_t ( *proc_func )( void *awb_ctx, awb_stats_data_t *stats,
    awb_input_data_t *input, awb_output_data_t *output );
    int32_t ( *deinit_func )( void *awb_ctx );
} ISP_AWB_FUNC_S;
```

[Function Description]

Definition of AWB library callback function structure.

[Field Description]

Field	Description
init_func	Description： AWB algorithm init

	Parameter: 1. ctx_id Input, pipeline index Return: A pointer point to one pipeline
proc_func	Description： AWB algorighm implementation Parameter: 1. awb_ctx Input, return value of init_func 2. stats Input, point to AWB metering data 3. input Input, parameter from ISP Firmware 4. output Output, result of AWB algorighm, will be set to isp firmware 5. Return: 0:pass, not 0:failed
deinit_func	Description： AWB algorithm deinit Parameter: 1. awb_ctx Input, return value of init_func 2. Return: 0:pass, not 0:failed

awb_stats_data_t

[Structure Definition]

typedef struct _awb_stats_data_ {

```
    awb_zone_t *awb_zones;
    uint32_t zones_size;
} awb_stats_data_t;
```

[Function Description]

AWB metering structure definition. See AWB metering section for detailed description of AWB.

[Field Description]

Field	Description
awb_zones	Point to zone-wise metering data
zones_size	Zone size

awb_zone_t

[Structure Definition]

```
typedef struct _awb_zone_t {
    uint16_t rg;
    uint16_t bg;
    uint32_t sum;
} awb_zone_t;
```

[Function Description]

Definition of AWB zonal metering data structure.

[Field Description]

Field	Description
rg	Zone weighted average of R/G or G/R
bg	Zone weighted average of B/G or G/B
sum	Total number of pixels participating in the statistics

awb_input_data_t

[Structure Definition]

```
typedef struct _awb_input_data_ {
    void *custom_input;
    void *acamera_input;
} awb_input_data_t;
```

[Function Description]

Definition the input parameter that will pass to AWB algorighm.

[Field Description]

Field	Description
custom_input	reserved
acamera_input	AWB algorithm really use, type is awb_acamera_input_t

awb_acamera_input_t

[Structure Definition]

```
typedef struct _awb_acamera_input_ {
    awb_misc_info_t misc_info;
    awb_calibration_data_t cali_data;
} awb_acamera_input_t;
```

[Function Description]

Definition the input parameter that will pass to AWB algorighm.

Below structure included:

```
typedef struct _awb_misc_info_ {
    uint16_t log2_gain;
    int cur_exposure_log2;
    uint32_t iridix_contrast;
    uint8_t global_manual_awb;
    uint16_t global_awb_red_gain;
    uint16_t global_awb_blue_gain;
} awb_misc_info_t;

typedef unsigned short ( *calibration_light_src_t )[2];
typedef struct _awb_calibration_data_ {
```

```
calibration_light_src_t cali_light_src;
uint32_t cali_light_src_len;

uint32_t *cali_evtolux_ev_lut;
uint32_t cali_evtolux_ev_lut_len;

uint32_t *cali_evtolux_lux_lut;
uint32_t cali_evtolux_lux_lut_len;

uint8_t *cali_awb_avg_coef;
uint32_t cali_awb_avg_coef_len;

uint16_t *cali_rg_pos;
uint32_t cali_rg_pos_len;

uint16_t *cali_bg_pos;
uint32_t cali_bg_pos_len;

uint16_t *cali_color_temp;
uint32_t cali_color_temp_len;

uint16_t *cali_ct_rg_pos_calc;
uint32_t cali_ct_rg_pos_calc_len;

uint16_t *cali_ct_bg_pos_calc;
uint32_t cali_ct_bg_pos_calc_len;

modulation_entry_t *cali_awb_bg_max_gain;
uint32_t cali_awb_bg_max_gain_len;

uint16_t *cali_mesh_ls_weight;
uint16_t *cali_mesh_rgbg_weight;
```

```
    uint8_t *cali_evtolux_probability_enable;
    uint32_t *cali_awb_mix_light_param;
    uint16_t *cali_ct65pos;
    uint16_t *cali_ct40pos;
    uint16_t *cali_ct30pos;
    uint16_t *cali_sky_lux_th;
    uint16_t *cali_wb_strength;
    uint16_t *cali_mesh_color_temperature;
    uint16_t *cali_awb_warming_ls_a;
    uint16_t *cali_awb_warming_ls_d75;
    uint16_t *cali_awb_warming_ls_d50;
    uint16_t *cali_awb_colour_preference;
} awb_calibration_data_t;
```

awb_output_data_t

[Structure Definition]

```
typedef struct _awb_output_data_ {
    void *custom_output;
    void *acamera_output;
} awb_output_data_t;
```

[Function Description]

Definition the output parameter that AWB algorighm output.

[Field Description]

Field	Description
custom_output	reserved
acamera_output	Output parameter of AWB algorithm, type is awb_acamera_output_t

awb_acamera_output_t

[Structure Definition]

```
typedef struct _awb_acamera_output_ {
    uint16_t rg_coef;
```

```
    uint16_t bg_coef;
    int32_t temperature_detected;
    uint8_t p_high;
    uint8_t light_source_candidate;
    int32_t awb_warming[3];
    uint8_t awb_converged;
} awb_acamera_output_t;
```

[Function Description]

Definition the output parameter that AWB algorighm output.

[Field Description]

Field	Description
rg_coef	R Gain
bg_coef	B Gain
temperature_detected	Temperature value
p_high	High color temperature threshold for CCM control
light_source_candidate	Used for CCM switch
awb_warming[3]	CCM WB coeff of R, G, B
awb_converged	AWB algorithm state, running or done

2.4.2.3 HB_ISP_AF_FUNC_S

[Structure Definition]

```
typedef struct HB_ISP_AF_FUNC_S {
    void *( *init_func )( uint32_t ctx_id );
int32_t ( *proc_func )( void *af_ctx, af_stats_data_t *stats,
        af_input_data_t *input, af_output_data_t *output );
    int32_t ( *deinit_func )( void *af_ctx );
} ISP_AF_FUNC_S;
```

[Function Description]

Definition of AF library callback function structure.

[Field Description]

Field	Description
init_func	Description： AF algorithm init Parameter: 1. ctx_id Input, pipeline index Return: Point to one pipeline
proc_func	Description： AF algorithm implementation Parameter: 1. af_ctx Input, return value of init_func 2. stats Input, point to metering data of AF 3. input Input, parameter from ISP Firmware 4. output Output, AF algorithm output, will be set to ISP Firmware 5. Return: 0:pass, not 0:failed
deinit_func	Description： AF algorithm deinit Parameter: 1. af_ctx Input, return value of init_func 2. Return: 0:pass, not 0:failed

af_stats_data_t

[Structure Definition]

typedef struct _af_stats_data_ {

uint32_t *zones_stats;

uint32_t zones_size;

} af_stats_data_t;

[Function Description]

AFmetering structure definition. See AF metering section for detailed description of AF.

[Field Description]

Field	Description
zones_stats	Point to AF metering data
zones_size	Zone size

af_input_data_t

[Structure Definition]

typedef struct _af_input_data_ {

void *custom_input;

void *acamera_input;

} af_input_data_t;

[Function Description]

Definition the input parameter that will pass to AF algorighm.

[Field Description]

Field	Description
custom_input	reserved
acamera_input	AF algorighm really use, type is af_acamera_input_t

af_acamera_input_t

[Structure Definition]

typedef struct _af_acamera_input_ {

af_info_t af_info;

```
    af_misc_info_t misc_info;
    af_calibration_data_t cali_data;
} af_acamera_input_t;
```

[Function Description]

Definition the input parameter that will pass to AF algorighm.

Below structure included:

```
typedef struct _af_lms_param_t {
    uint32_t pos_min_down;
    uint32_t pos_min;
    uint32_t pos_min_up;
    uint32_t pos_inf_down;
    uint32_t pos_inf;
    uint32_t pos_inf_up;
    uint32_t pos_macro_down;
    uint32_t pos_macro;
    uint32_t pos_macro_up;
    uint32_t pos_max_down;
    uint32_t pos_max;
    uint32_t pos_max_up;
    uint32_t fast_search_positions;
    uint32_t skip_frames_init;
    uint32_t skip_frames_move;
    uint32_t dynamic_range_th;
    uint32_t spot_tolerance;
    uint32_t exit_th;
    uint32_t caf_trigger_th;
    uint32_t caf_stable_th;
    uint32_t print_debug;
} af_lms_param_t;

typedef struct _af_info_ {
    uint8_t af_mode;
```

```
    uint8_t refocus_required;
    uint8_t zones_horiz;
    uint8_t zones_vert;
    uint32_t roi;
    uint32_t af_pos_manual;
    uint32_t zoom_step_info;
} af_info_t;

typedef struct _af_misc_info_ {
    int16_t accel_angle;
    uint16_t lens_min_step;
} af_misc_info_t;

typedef struct _af_calibration_data_ {
    af_lms_param_t *af_param;
    uint16_t *af_zone_whgh_h;
    uint32_t af_zone_whgh_h_len;
    uint16_t *af_zone_whgh_v;
    uint32_t af_zone_whgh_v_len;
} af_calibration_data_t;
```

af_output_data_t

[Structure Definition]

```
typedef struct _af_output_data_ {
    void *custom_output;
    void *acamera_output;
} af_output_data_t;
```

[Function Description]

Definition the output parameter that AF algorighm output.

[Field Description]

Field	Description
custom_output	reserved
acamera_output	AF algorighm output parameter,

	type is af_acamera_output_t

af_acamera_output_t

[Structure Definition]

```
typedef struct _af_acamera_output_ {
    uint16_t af_lens_pos;
    int32_t af_sharp_val;
    af_state_t state;
} af_acamera_output_t;
```

[Function Description]

Definition the output parameter that AWB algorighm output.

[Field Description]

Field	Description
af_lens_pos	Lens position
af_sharp_val	Sharpen value
state	typedef enum af_state { AF_STATE_INACTIVE, AF_STATE_SCAN, AF_STATE_FOCUSED, AF_STATE_UNFOCUSED } af_state_t;

2.4.3 App Tuning

2.4.3.1 HB_ISP_OP_TYPE_E

[Structure Definition]

```
typedef enum HB_ISP_OP_TYPE_E {
   OP_TYPE_AUTO = 0,
   OP_TYPE_MANUAL,
} ISP_OP_TYPE_E;
```

[Field Description]

Field	Description

OP_TYPE_AUTO	Auto mode
OP_TYPE_MANUAL	Manual mode

2.4.3.2 HB_ISP_AE_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_AE_ATTR_S {
    uint32_t u32Exposure;
    uint32_t u32ExposureRatio;
    uint32_t u32IntegrationTime;
    uint32_t u32SensorAnalogGain;
    uint32_t u32SensorDigitalGain;
    uint32_t u32IspDigitalGain;
    uint32_t u32MaxExposureRatio;
    uint32_t u32MaxIntegrationTime;
    uint32_t u32MaxSensorAnalogGain;
    uint32_t u32MaxSensorDigitalGain;
    uint32_t u32MaxIspDigitalGain;
    ISP_OP_TYPE_E enOpType;
} ISP_AE_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
u32Exposure	Control the exposure time parameter
u32ExposureRatio	Control the exposure ratio parameter
u32IntegrationTime	Control the integration time parameter
u32SensorAnalogGain	Control the sensor analog gain parameter Values: [0-255]
u32SensorDigitalGain	Control the sensor digital gain parameter Values: [0-255]
u32IspDigitalGain	Control the isp digital gain parameter Values: [0-255]

u32MaxExposureRatio	Control the max exposure ratio parameter
u32MaxIntegrationTime	Control the max integration time parameter
u32MaxSensorAnalogGain	Control the max sensor analog gain parameter Values: [0-255]
u32MaxSensorDigitalGain	Control the max sensor digital gain parameter Values: [0-255]
u32MaxIspDigitalGain	Control the max isp digital gain parameter Values: [0-255]
enOpType	See the definition of HB_ISP_OP_TYPE_E structure for the value. Because there is no auto parameter, for the set interface, setting auto will change the mode to auto, and setting manual will change the mode to manual and set the manual parameter; For the get interface, set auto or manual to get the parameter values in different modes;

2.4.3.3 HB_ISP_AF_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_AF_ATTR_S {
        uint32_t u32ZoomPos;
        ISP_OP_TYPE_E enOpType;
} ISP_AF_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
uint32_t u32ZoomPos	Control the zoom parameter Values: [10-80]
enOpType	Reserved, no use

2.4.3.4 HB_ISP_AWB_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_AWB_ATTR_S {
    uint32_t u32RGain;
    uint32_t u32BGain;
    ISP_OP_TYPE_E enOpType;
} ISP_AWB_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
u32RGain	Control the awb_red_gain parameter Values: [0-4096] format: unsigned 4.8 bit fixed-point
u32BGain	Control the awb_blue_gain parameter Values: [0-4096] format: unsigned 4.8 bit fixed-point
enOpType	See the definition of HB_ISP_OP_TYPE_E structure for the value. Because there is no auto parameter, for the set interface, setting auto will change the mode to auto, and setting manual will change the mode to manual and set the manual parameter; For the get interface, set auto or manual to get the parameter values in different modes;

2.4.3.5 HB_ISP_BLACK_LEVEL_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_BLACK_LEVEL_ATTR_S {
    uint32_t u32OffsetR;
    uint32_t u32OffsetGr;
```

```
    uint32_t u32OffsetGb;
    uint32_t u32OffsetB;
    ISP_OP_TYPE_E enOpType;
} ISP_BLACK_LEVEL_ATTR_S;
```

[Function Description]

black_level parameter definition.

[Field Description]

Field	Description
u32OffsetR	Black offset subtraction for each channel in linear domain: Channel 00 (R). Values: [0-1048575]
u32OffsetGr	Black offset subtraction for each channel in linear domain: Channel 01 (Gr). Values: [0-1048575]
u32OffsetGb	Black offset subtraction for each channel in linear domain: Channel 10 (Gb). Values: [0-1048575]
u32OffsetB	Black offset subtraction for each channel in linear domain: Channel 11 (B). Values: [0-1048575]
enOpType	Reference to ISP_OP_TYPE_E. Because there is no auto parameter, for set interface, setting auto will change mode to auto, setting manual will change mode to manual and set manual parameter; For the get interface, set auto or manual to get the parameter values in different modes;

2.4.3.6 HB_ISP_DEMOSAIC_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_DEMOSAIC_ATTR_S {
    uint32_t u32FcSlope;
    uint32_t u32FcAliasSlope;
    uint32_t u32FcAliasThresh;
    ISP_CTRL_PARAM_ATTR_S VhParam;
    ISP_CTRL_PARAM_ATTR_S AaParam;
    ISP_CTRL_PARAM_ATTR_S VaParam;
    ISP_CTRL_PARAM_ATTR_S UuParam;
    ISP_CTRL_PARAM_ATTR_S UuShParam;
    ISP_CTRL_PARAM_ATTR_S SharpLumaLowD;
    ISP_CTRL_PARAMA_ATTR_S SharpLumaHighD;
    ISP_CTRL_PARAM_ATTR_S SharpLumaLowUD;
    ISP_CTRL_PARAMA_ATTR_S SharpLumaHighUD;
    uint32_t u32MinDstrength;
    uint32_t u32MinUDstrength;
    uint32_t u32MaxDstrength;
    uint32_t u32MaxUDstrength;
    uint16_t u16NpOffset[ISP_AUTO_ISO_STRENGTH_NUM][2];
} ISP_DEMOSAIC_ATTR_S;

typedef struct HB_ISP_CTRL_PARAM_ATTR_S {
    uint32_t u32Offset;
    uint32_t u32Thresh;
    uint32_t u32Slope;
} ISP_CTRL_PARAM_ATTR_S;

typedef struct HB_ISP_CTRL_PARAMA_ATTR_S {
    uint32_t u32Thresh;
    uint32_t u32Slope;
} ISP_CTRL_PARAMA_ATTR_S;
```

[Function Description]

[Field Description]

#define ISP_AUTO_ISO_STRENGTH_NUM 16

Field	Description
u32FcSlope	Slope (strength) of false color correction
u32FcAliasSlope	Slope (strength) of false colour correction after blending with saturation value in 2.6 unsigned format
u32FcAliasThresh	Threshold of false colour correction after blending with saturation valuet in in 0.8 unsigned format
VhParam	**Slope** of vertical/horizontal blending threshold in 4.4 logarithmic format. High values will tend to favor one direction over the other (depending on VH Thresh) while lower values will give smoother blending. **Threshold** for the range of vertical/horizontal blending The threshold defines the difference of vertical and horizontal gradients at which the vertical gradient will start to be taken into account in the blending (if VH Offset is set to 0). Setting the offset not null (or the slope low) will include proportion of the vertical gradient in the blending before even the gradient difference reaches the threshold (see VH Offset for more details). **Offset** for vertical/horizontal blending threshold
AaParam	**Slope** of angular (45/135) blending threshold in 4.4 format. High values will tend to favor one direction over the other (depending

	on AA Thresh) while lower values will give smoother blending. **Threshold** for the range of angular (45/135) blending. The threshold defines the difference of 45 and 135 gradients at which the 45 gradient will start to be taken into account in the blending (if AA Offset is set to 0). Setting the offset not null (or the slope low) will include proportion of the 45 gradient in the blending before even the gradient difierence reaches the threshold (see AA Offset for more details). **Offset** for angular (A45/A135) blending threshold. This register has great impact on how AA Thresh is used. Setting this register to a value offset tells the blending process to weight the 45 and 135 gradients, at the threshold, with respectively offset/16 and 255 - (offset/16). If AA Thresh not equals to 0, these same blending weights apply from -AA Thresh to +AA Thresh.
VaParam	**Slope** [7:0] Slope of VH-AA blending threshold in 4.4 log format. High values will tend to favor one direction over the other (depending on VA Thresh) while lower values will give smoother blending. **Threshold** for the range of VH-AA blending.

	The threshold defines the difference of VH and AA gradients at which the VH gradient will start to be taken into account in the blending (if VA Offset is set to 0). Setting the offset not null (or the slope low) will include proportion of the VH gradient in the blending before even the gradient difference reaches the threshold (see VA Offiset for more details). **Offset** for VH-AA blending threshold. This register has great impact on how VA Thresh is used. Setting this register to a value offset tells the blending process to weight the VH and AA gradients, at the threshold, with respectively offset/16 and 255 - (offset/16). If VA Thresh not equals to 0, these same blending weights apply from -VA Thresh to +VA Thresh.
UuParam	**Slope** [7:0] Slope of undefined blending threshold in 4.4 logarithmic format **Threshold** for the range of undefined blending **Offset** for undefined blending threshold
UuShParam	**Threshold** for the range of undefined blending **Offset** for undefined blending threshold

	Slope of undefined blending threshold in 4.4 logarithmic format
SharpLumaLowD	**thresh** Intensity values above this value will be sharpen **Slope** Linear threshold slope corresponding to luma_thresh_low_d **offset** Linear threshold offset corresponding to luma_thresh_low_d
SharpLumaHighD	**thresh** Intensity values below this value will be sharpen **Slope** Linear threshold slope corresponding to luma_thresh_high_d
SharpLumaLowUD	**thresh** Intensity values above this value will be sharpen **offset** Linear threshold offset corresponding to luma_thresh_low_ud **Slope** Linear threshold slope corresponding to luma_thresh_low_ud
SharpLumaHighUD	**thresh** Intensity values below this value will be sharpen **Slope** Linear threshold slope corresponding to luma_thresh_high_ud
u32MinDstrength	Min threshold for the directional L_L in signed 2's complement s.12 format
u32MinUDstrength	Min threshold for the un-directional L_Lu in signed 2's complement s.12 format
u32MaxDstrength	Max threshold for the directional L_L in signed 2's complement s1+0.12 format
u32MaxUDstrength	Max threshold for the undirectional L_Lu in signed 2's complement s1+0.12 format

u16NpOffset	Noise profile offset in logarithmic 4.4 format

2.4.3.7 HB_ISP_SHARPEN_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_SHARPEN_ATTR_S {
    uint16_t u16SharpFR[ISP_AUTO_ISO_STRENGTH_NUM][2];
    uint16_t u16SharpAltD[ISP_AUTO_ISO_STRENGTH_NUM][2];
    uint16_t u16SharpAltDU[ISP_AUTO_ISO_STRENGTH_NUM][2];
    uint16_t u16SharpAltUD[ISP_AUTO_ISO_STRENGTH_NUM][2];
} ISP_SHARPEN_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
u16SharpFR	Controls strength of sharpening effect. u5.4
u16SharpAltD	Sharpen strength for L_Ld in unsigned 4.4 format
u16SharpAltDU	Sharpen strength for L_Ldu in unsigned 4.4 format
u16SharpAltUD	Sharpen strength for L_Lu in unsigned 4.4 format

2.4.3.8 HB_ISP_GAMMA_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_GAMMA_ATTR_S {
        uint16_t au16Gamma[129];
} ISP_GAMMA_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
au16Gamma	Gamma LUT

2.4.3.9 ISP_IRIDIX_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_IRIDIX_ATTR_S {
    ISP_OP_TYPE_E enOpType;
    ISP_IRIDIX_AUTO_ATTR_S stAuto;
    ISP_IRIDIX_MANUAL_ATTR_S stManual;
    uint32_t u32BlackLevel;
    uint32_t u32WhiteLevel;
    uint32_t u32Svariance;
    uint32_t u32Bright_pr;
    uint32_t u32Contrast;
    uint32_t u32IridixOn;
    uint32_t u32FilterMux;
    uint32_t u32VarianceSpace;
    uint32_t u32VarianceIntensity;
    uint32_t u32SlopeMax;
    uint32_t u32SlopeMin;
    uint32_t u32FwdPerceptCtrl;
    uint32_t u32RevPerceptCtrl;
    uint32_t u32FwdAlpha;
    uint32_t u32RevAlpha;
    uint32_t u32GtmSelect;
} ISP_IRIDIX_ATTR_S;

typedef struct HB_ISP_IRIDIX_AUTO_ATTR_S {
    uint8_t u8AvgCoef;
    uint32_t au32EvLimNoStr[2];
    uint32_t u32EvLimFullStr;
    uint32_t au32StrengthDkEnhControl[15];
} ISP_IRIDIX_AUTO_ATTR_S;

typedef struct HB_ISP_IRIDIX_MANUAL_ATTR_S {
```

```
    uint32_t u32RoiHorStart;
    uint32_t u32RoiHorEnd;
    uint32_t u32RoiVerStart;
    uint32_t u32RoiVerEnd;
    uint32_t u32StrengthInRoi;
    uint32_t u32StrengthOutRoi;
     uint32_t u32DarkEnh;
} ISP_IRIDIX_MANUAL_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
enOpType	Reference to ISP_OP_TYPE_E.
u8AvgCoef	The average coefficient value for Iridix
au32EvLimNoStr	The Expose Value maximum value, in terms of EV_log2 without Strength
u32EvLimFullStr	The Expose Value maximum value, in terms of EV_log2 with Strength
au32StrengthDkEnhControl	strength and dark enhancememt control: [0] - dark_prc [1] - bright_prc [2] - min_dk: minimum dark enhancement [3] - max_dk: maximum dark enhancement [4] - pD_cut_min: minimum intensity cut for dark regions in which dk_enh will be applied [5] - pD_cut_max: maximum intensity cut for dark regions in which dk_enh will be applied [6] - dark contrast min [7] - dark contrast max [8] - min_str: iridix strength in

	percentage [9] - max_str: iridix strength in percentage: 50 = 1x gain. 100 = 2x gain [10] - dark_prc_gain_target: target in histogram (percentage) for dark_prc after iridix is applied [11] - contrast_min: clip factor of strength for LDR scenes. [12] - contrast_max: clip factor of strength for HDR scenes. [13] - max iridix gain [14] - print debug
u32RoiHorStart	Horizontal starting point of ROI Values: [0-65535]
u32RoiHorEnd	Horizontal ending point of ROI Values: [0-65535]
u32RoiVerStart	Vertical starting point of ROI Values: [0-65535]
u32RoiVerEnd	Vertical ending point of ROI Values: [0-65535]
u32StrengthInRoi	Manual Strength value for inside of ROI Values: [0-65535]
u32StrengthOutRoi	Manual Strength value for outside of ROI Values: [0-65535]
u32DarkEnh	Manual Dark Enhance value to control Iridix core Values: [0-65535]
u32BlackLevel	Iridix black level. Values below this will not be affected by Iridix.
u32WhiteLevel	Iridix white level. Values above this will not be affected by Iridix.
u32Svariance	Iridix8 transform sensitivity to different areas of image

u32Bright_pr	Manual Bright_Preserve value to control Iridix core
u32Contrast	Iridix8 contrast control parameter
u32IridixOn	Iridix enable: 0=off 1=on
u32FilterMux	Selects between Iridix8 and Iridix7, 1=Iridix8 and 0=Iridix7
u32VarianceSpace	Sets the degree of spatial sensitivity of the algorithm(Irdx7F)
u32VarianceIntensity	Sets the degree of luminance sensitivity of the algorithm(Irdx7F)
u32SlopeMax u32SlopeMin	Restricts the maximum/minimum slope (gain) which can be generated by the adaptive algorithm
u32FwdPerceptCtrl u32RevPerceptCtrl	Iridix gamma processing select: 0=pass through 1=gamma_dl 2=sqrt 3=gamma_lut.
u32FwdAlpha u32RevAlpha	alpha for gamma_dl
u32GtmSelect	Global Tone map select : 0 : Local TM 1: Full Global TM

2.4.3.10 HB_ISP_CNR_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_CNR_ATTR_S {
    uint16_t u16UvDelta12Slope[ISP_AUTO_ISO_STRENGTH_NUM][2];
} ISP_CNR_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
u16UvDelta12Slope	Strength of color noise reduction

2.4.3.11 HB_ISP_SINTER_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_SINTER_ATTR_S {
        uint16_t aau16Strength[ISP_AUTO_ISO_STRENGTH_NUM][2];
        uint16_t aau16Strength1[ISP_AUTO_ISO_STRENGTH_NUM][2];
        uint16_t aau16Thresh1[ISP_AUTO_ISO_STRENGTH_NUM][2];
        uint16_t aau16Thresh4[ISP_AUTO_ISO_STRENGTH_NUM][2];
} ISP_SINTER_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
aau16Strength	Noise reduction effect for high spatial frequencies
aau16Strength1	Noise reduction effect for low spatial frequencies
aau16Thresh1	Noise threshold for high horizontal/vertical spatial frequencies
aau16Thresh4	Noise threshold for low horizontal/vertical spatial frequencies

2.4.3.12 HB_ISP_TEMPER_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_TEMPER_ATTR_S {
        uint16_t aau16strength[ISP_AUTO_ISO_STRENGTH_NUM][2];
        uint32_t u32RecursionLimit;
        uint32_t u32LutEnable;
} ISP_TEMPER_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
aau16strength	Noise reduction effect for temporal frequencies

u32RecursionLimit	Controls length of filter history. Low values result in longer history and stronger temporal filtering Range: [0, 16]
u32LutEnable	Choose from LUT or exp_mask Range: [0, 3]

2.4.3.13 HB_MESH_SHADING_ATTR_S

[Structure Definition]

typedef struct HB_MESH_SHADING_ATTR_S {

uint32_t u32Enable;

uint32_t u32MeshScale;

uint32_t u32MeshAlphaMode;

uint32_t u32MeshWidth;

uint32_t u32MeshHeight;

uint32_t u32ShadingStrength;

} MESH_SHADING_ATTR_S;

[Function Description]

[Field Description]

Field	Description
u32Enable	Lens mesh shading correction enable: 0=off, 1=on.
u32MeshScale	Selects the precision and maximal gain range of mesh shading correction Gain range: 00 -> 0..2; 01 -> 0..4; 02 -> 0..8; 03 -> 0..16; 04 -> 1..2; 05 -> 1..3; 06 -> 1..5; 07 -> 1..9 (float).
u32MeshAlphaMode	Sets alpha blending between mesh shading tables. 0 = no alpha blending; 1 = 2 banks (odd/even bytes) 2 = 4 banks (one per 8-bit lane in each dword).
u32MeshWidth	Number of horizontal nodes

	Values: [0-63]
u32MeshHeight	Number of vertical nodes Values: [0-63]
u32ShadingStrength	Mesh strength in 4.12 format, e.g. 0 = no correction, 4096 = correction to match mesh data. Can be used to reduce shading correction based on AE. Values: [0-4096]

2.4.3.14 HB_MESH_SHADING_LUT_S

[Structure Definition]

typedef struct HB_MESH_SHADING_LUT_S {

uint32_t au32RGain[1024];

uint32_t au32GGain[1024];

uint32_t au32BGain[1024];

} MESH_SHADING_LUT_S;

[Function Description]

[Field Description]

Field	Description
au32Rgain	corrective R gain Different modes have different values
au32Ggain	corrective G gain Different modes have different values
au32Bgain	corrective B gain Different modes have different values

2.4.3.15 HB_RADIAL_SHADING_ATTR_S

[Structure Definition]

typedef struct HB_RADIAL_SHADING_ATTR_S {

uint32_t u32Enable;

uint32_t u32CenterRX;

uint32_t u32CenterRY;

```
    uint32_t u32CenterGX;
    uint32_t u32CenterGY;
    uint32_t u32CenterBX;
    uint32_t u32CenterBY;
    uint32_t u32OffCenterMultRX;
    uint32_t u32OffCenterMultRY;
    uint32_t u32OffCenterMultGX;
    uint32_t u32OffCenterMultGY;
    uint32_t u32OffCenterMultBX;
    uint32_t u32OffCenterMultBY;
} RADIAL_SHADING_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
u32Enable	Set to 1 to enable radial shading correction.
u32CenterRX	Center x coordinates for R shading map. Values: [0-65535]
u32CenterRY	Center y coordinates for R shading map. Values: [0-65535]
u32CenterGX	Center x coordinates for G shading map. Values: [0-65535]
u32CenterGY	Center y coordinates for G shading map. Values: [0-65535]
u32CenterBX	Center x coordinates for B shading map. Values: [0-65535]
u32CenterBY	Center y coordinates for B shading

	map. Values: [0-65535]
u32OffCenterMultRX u32OffCenterMultRY u32OffCenterMultGX u32OffCenterMultGY u32OffCenterMultBX u32OffCenterMultBY	Normalizing X/Y factor which scales the R, G, B radial table to the edge of the image. Calculated as 231/R2 where R is the furthest distance from the center coordinate to the edge of the image in pixels. Values: [0-65535]

2.4.3.16 HB_RADIAL_SHADING_LUT_S

[Structure Definition]

```
typedef struct HB_RADIAL_SHADING_LUT_S {
    uint16_t au16RGain[129];
    uint16_t au16GGain[129];
    uint16_t au16BGain[129];
} RADIAL_SHADING_LUT_S;
```

[Function Description]

[Field Description]

Field	Description
au16Rgain	corrective R gain
au16Ggain	corrective G gain
au16Bgain	corrective B gain

2.4.3.17 HB_ISP_CSC_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_CSC_ATTR_S {
    uint32_t u32ClipMinY;
    uint32_t u32ClipMaxY;
    uint32_t u32ClipMinUV;
    uint32_t u32ClipMaxUV;
    uint32_t u32MaskRY;
    uint32_t u32MaskGU;
```

```
    uint32_t u32MaskBV;
    uint16_t aau16Coefft[12];
} ISP_CSC_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
u32ClipMinY	Minimal value for Y. Values below this value are clipped. Values: [0-1023]
u32ClipMaxY	Maximal value for Y. Values above this value are clipped. Values: [0-1023]
u32ClipMinUV	Minimal value for Cb, Cr. Values below this value are clipped Values: [0-1023]
u32ClipMaxUV	Maximal value for Cb, Cr. Values above this value are clipped Values: [0-1023]
u32MaskRY	Data mask for channel 1 (R or Y). Bit-wise and of this value and video data. Values: [0-1023]
u32MaskGU	Data mask for channel 2 (G or U). Bit-wise and of this value and video data. Values: [0-1023]
u32MaskBV	Data mask for channel 3 (B or V). Bit-wise and of this value and video data. Values: [0-1023]
aau16Coefft[12]	Coefficients in the 3x3 conversion matrix. And Offset coefficients. $\begin{pmatrix} coefft[0] & coefft[1] & coefft[2] \\ coefft[3] & coefft[4] & coefft[5] \\ coefft[6] & coefft[7] & coefft[8] \end{pmatrix} \begin{pmatrix} coefft[9] \\ coefft[10] \\ coefft[11] \end{pmatrix}$ Values: [0-65535] How to fill please refer to 1.12 Color Space

	Conversion。

2.4.3.18 HB_ISP_SCENE_MODES_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_SCENE_MODES_ATTR_S {
        uint32_t u32ColorMode;
        uint32_t u32BrightnessStrength;
        uint32_t u32ContrastStrength;
        uint32_t u32SaturationStrength;
        uint32_t u32HueTheta;
} ISP_SCENE_MODES_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
u32ColorMode	Select the color mode of the ISP. Values: {NORMAL} {BLACK_AND_WHITE} {NEGATIVE} {SEPIA} {VIVID} Default Value: {NORMAL}
u32BrightnessStrength	Control the exact brightness value. Values: [0-255] Key: 128 - Standard Brightness <128 - Decreased Brightness >128 - Increased Brightness Default Value: 128
u32ContrastStrength	Control the exact contrast value. Values: [0-255] Key: 128 - Standard Contrast <128 - Decreased Contrast >128 - Increased Contrast Default Value: 128
u32SaturationStrength	Control the exact saturation strength. Values: [0-255] Key: 128 - Standard Saturation <128 - Decreased Saturation >128 - Increased Saturation Default Value: 128
u32HueTheta	Control the exact hue value. Values: [0-255] Key: 128 - Standard

	Hue <128 - Decreased Hue >128 - Increased Hue Default Value: 128

2.4.3.19 HB_ISP_STATISTICS_AWB_ZONE_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_STATISTICS_AWB_ZONE_ATTR_S {
        uint16_t u16Rg;
        uint16_t u16Bg;
        uint32_t u32Sum;
} ISP_STATISTICS_AWB_ZONE_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
u16Rg	Average Green/Red or Red/Green
u16Bg	Average Green/Blue or Blue/Green
u32Sum	Number of pixels used for AWB

2.4.3.20 HB_ISP_STATISTICS_AE_5BIN_ZONE_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_STATISTICS_AE_5BIN_ZONE_ATTR_S {
        uint16_t u16Hist0;
        uint16_t u16Hist1;
        uint16_t u16Hist3;
        uint16_t u16Hist4;
} ISP_STATISTICS_AE_5BIN_ZONE_ATTR_S;
```

[Function Description]

AE-5bin metering data。

[Field Description]

Field	Description
u16Hist0	5bin-hist0 metering data
u16Hist1	5bin-hist1 metering data

u16Hist3	5bin-hist3 metering data
u16Hist4	5bin-hist4 metering data

2.4.3.21 HB_ISP_AE_PARAM_S

[Structure Definition]

```
typedef struct HB_ISP_AE_PARAM_S {
        uint32_t u32TotalGain;
        ISP_OP_TYPE_E GainOpType;
        uint32_t u32IntegrationTime;
        uint32_t u32ExposureRatio;
        ISP_OP_TYPE_E IntegrationOpType;
} ISP_AE_PARAM_S;
```

[Function Description]

[Field Description]

Field	Description
u32TotalGain	Total exposure time of ae，total_gain = sensor_again + sensor_dgain + isp_dgain Range: [0, 765]
GainOpType	gain auto/manual type
u32IntegrationTime	line control(unit us)
u32ExposureRatio	Exposure ratio, valid in HDR mode
IntegrationOpType	line auto/manual type

2.4.3.22 HB_ISP_AE_ROI_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_AE_ROI_ATTR_S {
        uint8_t u8XStart;
        uint8_t u8YStart;
        uint8_t u8XEnd;
        uint8_t u8YEnd;
} ISP_AE_ROI_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
u8XStart	ROI region x_start Range: [0,255]
u8YStart	ROI region y_start Range: [0,255]
u8XEnd	ROI region x_end Range: [0,255]
u8YEnd	ROI region y_end Range: [0,255]

2.4.3.23 ISP_ZONE_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_ZONE_ATTR_S {
        uint8_t u8Horiz;
        uint8_t u8Vert;
} ISP_ZONE_ATTR_S;
```

[Function Description]

Used to change AF/AWB zones.

[Field Description]

Field	Description
u8Horiz	Horizontal zones conut
u8Vert	Vertical zones count

2.4.3.24 HB_ISP_STATISTICS_LUMVAR_ZONE_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_STATISTICS_LUMVAR_ZONE_ATTR_S {
        uint16_t u16Var;
        uint16_t u16Mean;
```

} ISP_STATISTICS_LUMVAR_ZONE_ATTR_S;

[Function Description]

[Field Description]

Field	Description
u16Var	lumvarmetering data variance
u16Mean	lumvarmetering data mean

2.4.3.25 ISP_AWB_STAT_AREA_ATTR_S

[Structure Definition]

```
typedef struct HB_ISP_AWB_STAT_AREA_ATTR_S {
    uint32_t u32WhiteLevel;
    uint32_t u32BlackLevel;
    uint32_t u32CrRefMax;
    uint32_t u32CrRefMin;
    uint32_t u32CbRefMax;
    uint32_t u32CbRefMin;
    uint32_t u32CrRefHigh;
    uint32_t u32CrRefLow;
    uint32_t u32CbRefHigh;
    uint32_t u32CbRefLow;
} ISP_AWB_STAT_AREA_ATTR_S;
```

[Function Description]

[Field Description]

Field	Description
u32WhiteLevel;	Upper limit of valid data for AWB Range: [0,1023]
u32BlackLevel;	Lower limit of valid data for AWB Range: [0,1023]
u32CrRefMax; u32CrRefMin; u32CbRefMax; u32CbRefMin;	metering data, more detail refer to 1.14.1 Range: [0,4095]

u32CrRefHigh; u32CrRefLow; u32CbRefHigh; u32CbRefLow;	