



Hi3516A/Hi3516D\_ISP\_3A

# Development Guide

Issue	05
Date	2016-10-28

**Copyright © HiSilicon Technologies Co., Ltd. 2014-2016. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of HiSilicon Technologies Co., Ltd.

## **Trademarks and Permissions**



**HISILICON**, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

## **HiSilicon Technologies Co., Ltd.**

Address: Huawei Industrial Base  
Bantian, Longgang  
Shenzhen 518129  
People's Republic of China

Website: <http://www.hisilicon.com>

Email: [support@hisilicon.com](mailto:support@hisilicon.com)



# About This Document

## Purpose

This document describes the functions, principles, and workflow of the Hi3516A/Hi3516D\_ISP\_3A. 3A functions indicate automatic exposure (AE), automatic white balance (AWB), and automatic focus (AF).



### NOTE

Unless otherwise specified, this document applies to the Hi3516A and Hi3516D.

## Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3516A_ISP_3A	-
Hi3516D_ISP_3A	-


## Intended Audience

This document is intended for:





- Technical support personnel
- R&D engineers

## Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
 <b>DANGER</b>	Alerts you to a high risk hazard that could, if not avoided, result in serious injury or death.



Symbol	Description
 <b>WARNING</b>	Alerts you to a medium or low risk hazard that could, if not avoided, result in moderate or minor injury.
 <b>CAUTION</b>	Alerts you to a potentially hazardous situation that could, if not avoided, result in equipment damage, data loss, performance deterioration, or unanticipated results.
 <b>TIP</b>	Provides a tip that may help you solve a problem or save time.
 <b>NOTE</b>	Provides additional information to emphasize or supplement important points in the main text.

## Change History

Changes between document issues are cumulative. Therefore, the latest document issue contains all changes made in previous issues.

### Issue 05 (2016-10-28)

This issue is the fifth official release, which incorporates the following changes:

#### **Chapter 2 Usage Guidance**

Section 2.2.2.1 is modified

#### **Chapter 3 Developer Guidance**

Section 3.2 and section 3.4 are modified.

### Issue 04 (2016-01-29)

This issue is the fourth official release, which incorporates the following changes:

Section 2.2.2, section 3.2, and section 3.3 are modified.

### Issue 03 (2015-06-15)

This issue is the third official release, which incorporates the following changes:

#### **Chapter 2 Usage Guidance**

In section 2.2.2.1, table 2-5 is modified.

#### **Chapter 3 Developer Guidance**

In section 3.2, table 3-4 is modified.

The description in section 3.3 is modified.

### Issue 02 (2015-02-10)

This issue is the second official release, which incorporates the following changes:

#### **Chapter 2 Usage Guidance**



The descriptions in section 2.2.1 and section 2.2.2 are modified.

### **Chapter 3 Developer Guidance**

In section 3.2, table 3-3 and table 3-4 are modified.

The description in section 3.3 is modified.

### **Issue 01 (2014-12-10)**

This issue is the first official release. The contents related to the Hi3516D are added.



# Contents

<b>About This Document</b>	<b>i</b>
<b>1 Introduction to the Hi3516A_ISP_3A</b>	<b>1</b>
1.1 Overview	1
1.2 Function Description	1
1.2.1 Design Methodology	1
1.2.2 File Structure	2
1.2.3 Development Mode	2
1.2.4 Internal Process	3
<b>2 Usage Guidance</b>	<b>5</b>
2.1 Software Process	5
2.2 Sensor Interconnection	9
2.2.1 Registering Sensors with the ISP Library	9
2.2.2 Registering Sensors with 3A Algorithm Libraries	17
<b>3 Developer Guidance</b>	<b>24</b>
3.1 Overview	24
3.2 Registering the AE Algorithm with the ISP Library	24
3.3 Registering the AWB Algorithm with the ISP Library	30
3.4 Developing User-defined AF Algorithm	39
3.4.1 Structure of the AF Algorithm	39
3.4.2 AF Synchronization Callback	40
<b>4 Appendix</b>	<b>46</b>
4.1 Relationships Between Registered Functions	46
4.2 Scalability Design Considerations	46
4.3 3A Architecture Design	47
4.4 External Register Description	47



## Figures

<b>Figure 1-1</b> Design methodology of the ISP firmware.....	1
<b>Figure 1-2</b> File structure of the ISP firmware.....	2
<b>Figure 1-3</b> Internal process of the ISP firmware.....	4
<b>Figure 1-4</b> Architecture of the ISP firmware .....	4
<b>Figure 2-1</b> Workflow of the ISP firmware .....	6
<b>Figure 2-2</b> Sensor adaptation diagram.....	9
<b>Figure 2-3</b> Callback function for registering sensors with the ISP library .....	9
<b>Figure 2-4</b> Callback functions for registering sensors with the AE algorithm library .....	17
<b>Figure 2-5</b> Callback function for registering sensors with the AWB library .....	21
<b>Figure 3-1</b> Callback function for registering the AE algorithm with the ISP library.....	24
<b>Figure 3-2</b> Description of the register with the address of 0x205a06c0 .....	27
<b>Figure 3-3</b> Callback function for registering the AWB algorithm with the ISP library .....	31
<b>Figure 3-4</b> Statistics parameters .....	37
<b>Figure 3-5</b> Structure of the AF algorithm .....	40
<b>Figure 3-6</b> AF synchronization callback.....	41



## Tables

<b>Table 2-1</b> Callback function for registering sensors with the ISP library .....	10
<b>Table 2-2</b> ISP_CMOS_DEFAULT_S members.....	11
<b>Table 2-3</b> ISP_SNS_REGS_INFO_S members.....	15
<b>Table 2-4</b> Callback functions for registering sensors with the AE algorithm library .....	18
<b>Table 2-5</b> AE_SENSOR_DEFAULT_S members .....	19
<b>Table 2-6</b> Member of the HI_MPI_AWB_SensorRegCallBackCallBack callback function for registering sensors with the AWB library.....	22
<b>Table 2-7</b> AWB_SENSOR_DEFAULT_S members.....	22
<b>Table 3-1</b> Callback functions for registering the AE algorithm with the ISP library .....	25
<b>Table 3-2</b> Members of the initialization parameter data structure ISP_AE_PARAM_S .....	26
<b>Table 3-3</b> Members of the statistics data structure ISP_AE_INFO_S .....	26
<b>Table 3-4</b> Members of the running result data structure ISP_AE_RESULT_S .....	28
<b>Table 3-5</b> Member of the callback function for registering the AWB algorithm with the ISP library .....	31
<b>Table 3-6</b> Members of the initialization parameter data structure ISP_AE_PARAM_S .....	32
<b>Table 3-7</b> Members of the statistics data structure ISP_AE_INFO_S .....	32
<b>Table 3-8</b> Members of the running result data structure ISP_AWB_RESULT_S.....	34





# 1 Introduction to the Hi3516A\_ISP\_3A

## 1.1 Overview

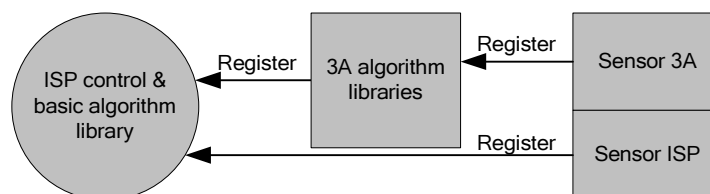
The Hi3516A\_ISP\_3A depends on the software development kit (SDK) of a specific major release. It processes digital images by using a series of digital image processing algorithms. The Hi3516A\_ISP\_3A includes the firmware framework and HiSilicon 3A libraries. The firmware processes statistics, drives digital image processing algorithms, and provides the basic algorithm framework and various functions such as defect pixel correction, denoising, color enhancement, and lens shading correction. The 3A libraries are registered with the firmware to implement exposure, white balance, and color reproduction.

## 1.2 Function Description

### 1.2.1 Design Methodology

The image signal processor (ISP) firmware consists of the ISP library (including the control unit and basic algorithm unit), AE/AWB/AF algorithm libraries (3A algorithm libraries), and sensor library. According to the firmware design methodology, separate 3A algorithm libraries are provided. The ISP control unit schedules the basic algorithm unit and 3A algorithm libraries, and the sensor library registers callback functions with the ISP library and 3A algorithm libraries to support various sensors. See [Figure 1-1](#).

**Figure 1-1** Design methodology of the ISP firmware



Different sensors register callback control functions with the ISP library and 3A algorithm libraries. When the ISP control unit schedules the basic algorithm unit and 3A algorithm libraries, the callback functions are called to obtain initialization parameters and control sensors, for example, adjusting the exposure time, analog gain, and digital gain, controlling the lens step focus, and rotating the iris.

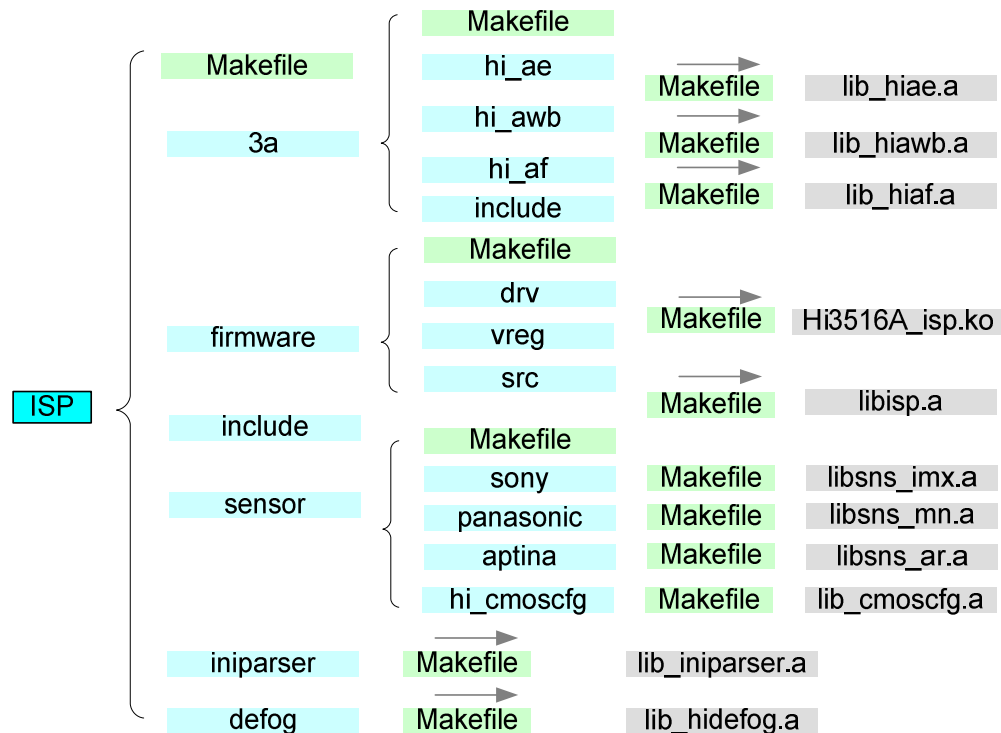


## 1.2.2 File Structure

As shown in Figure 1-2, the files of the ISP library, 3A algorithm library, sensor library, iniparser library, and defog library are stored in different folders. The driver in the **drv** folder of the firmware is used to report the ISP interrupt in user mode. The interrupt drives the ISP control unit of the firmware. The ISP control unit obtains statistics from the driver, schedules the basic algorithm unit and 3A algorithm library, and configures register by using the driver.

The **src** folder includes the code of the ISP control unit and basic algorithm unit. The ISP library **libisp.a** is generated after the code in the **src** folder is compiled. The **3a** folder includes the AE/AWB/AF algorithm library. You can develop your own 3A algorithms based on a unified interface. The **sensor** folder stores the drivers of all sensors as open-source code. The **hi\_cmoscfg** folder stores the common programs for parsing the .ini files. The code of the common programs is open source. The **iniparser** folder stores the ini parsing function library and can be used for developing other applications. The **defog** folder stores the anti-fog algorithm program that is not open source.

Figure 1-2 File structure of the ISP firmware



## 1.2.3 Development Mode

The SDK supports various development modes:

- Uses only the HiSilicon 3A algorithm libraries.
- Develops 3A algorithm libraries based on the 3A algorithm registration interfaces provided in the HiSilicon ISP library.
- Uses both the HiSilicon 3A algorithm libraries and your own 3A algorithm libraries. For example, you can use **lib\_hiae.a** to implement AE, and use your own 3A algorithm libraries to implement AWB.



### 1.2.3.1 Using the HiSilicon 3A Algorithm Libraries

You need to call the sensor adaptation interfaces provided in the ISP basic algorithm unit and HiSilicon 3A algorithm libraries to adapt to various sensors. The **sensor** folder includes the following two key files:

- **sensor\_cmos.c**  
This file is used to implement the callback functions required by the ISP. The callback functions include the adaptation algorithms of sensors and vary according to sensors.
- **sensor\_ctrl.c**  
This file is the underlying driver of sensors and is used to read, write to, and initialize sensors. You can develop these two files based on the data sheets of sensors and seek help from sensor vendors.

### 1.2.3.2 Developing 3A Algorithm Libraries

You need to call the sensor adaptation interfaces provided in the ISP basic algorithm unit to adapt to various sensors. Data interfaces and callback functions need to be customized for the developed 3A algorithm libraries to adapt to and control various sensors. After you develop your own 3A algorithm libraries, you can call the interfaces in **mpi\_isp.h** but not those in **mpi\_ae.h** and **mpi\_awb.h**.



#### NOTE

The senior engineers who are familiar with statistics and capable of algorithm development can develop algorithm libraries based on firmware statistics.

## 1.2.4 Internal Process

Figure 1-3 shows the internal process of the ISP firmware. First, the ISP control unit, basic algorithm unit, and 3A algorithm libraries are initialized, and the sensor callback function is called to obtain sensor initialization parameters. After initialization, driven by the interrupt, the firmware obtains statistics of each frame in kernel mode, drives the basic algorithm unit and 3A algorithm libraries to complete calculation, and provides the calculation results for configuring ISP registers and sensor registers. You can call ISP MPIs to control and change the internal data and status of the basic algorithm unit of the firmware, achieving the required image quality. If you use the HiSilicon 3A algorithm libraries, you can call the MPIs related to 3A algorithm libraries to change its internal data and status, adjusting the exposure, white balance, and color reproduction effects.



**Figure 1-3** Internal process of the ISP firmware

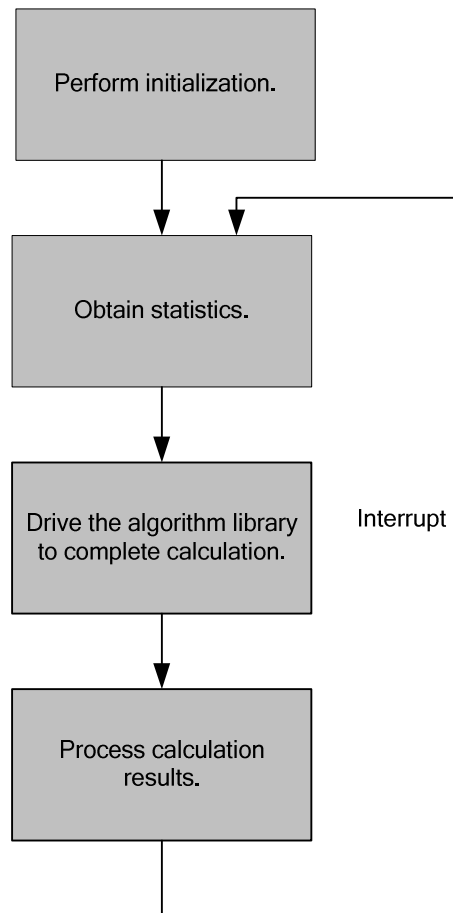
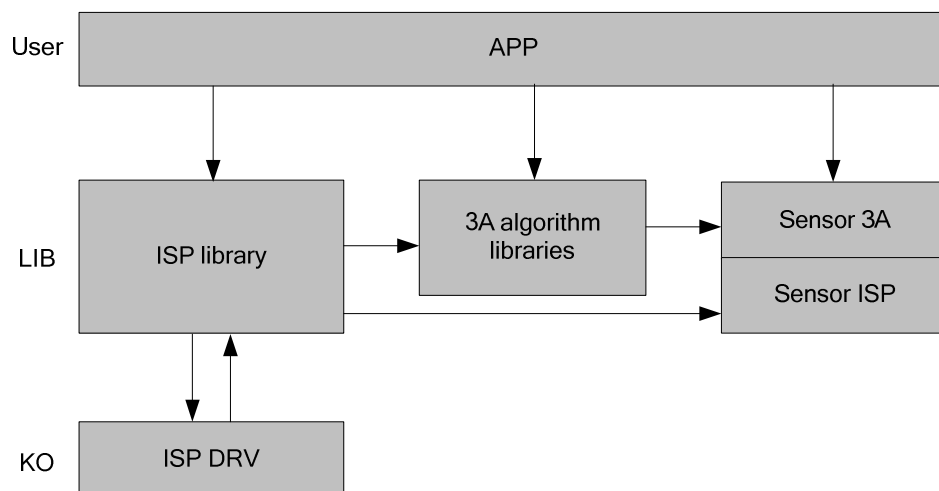


Figure 1-4 shows the architecture of the ISP firmware.

**Figure 1-4** Architecture of the ISP firmware





# 2 Usage Guidance

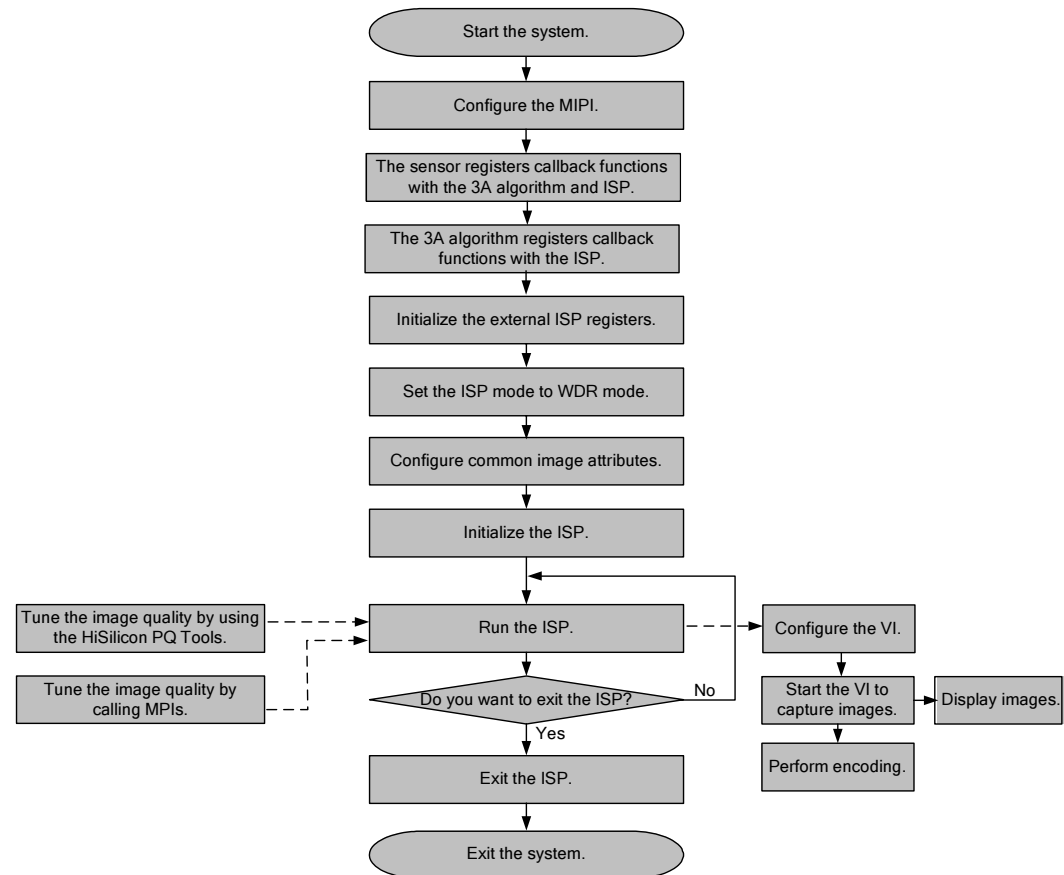
---

## 2.1 Software Process

As the front-end capturing unit, the ISP must work with the video input unit (VIU). After the ISP is initialized and configured, the interface timings of the VIU must be configured. This ensures that the input timings of different sensors are compatible, and the input timings of the ISP are correct. After timings are configured, the ISP can be started to dynamically adjust the image quality. The output images are captured by the VIU, and then transmitted for displaying or encoding. [Figure 2-1](#) shows the software process.

The HiSilicon PQ Tools dynamically adjusts the image quality at the PC end by tuning the values of related parameters such as the denoising strength, color conversion matrix, and saturation.

**Figure 2-1** Workflow of the ISP firmware



After adjusting images, you can save settings by using the configuration file of the HiSilicon PQ Tools. Then the system can load the configured image parameters from the configuration file when the HiSilicon PQ Tools starts next time.

The following is a code sample:

```

HI_S32 s32Ret;
ALG_LIB_S stLib;
ISP_PUB_ATTR_S stPubAttr;
pthread_t isp_pid;
/*Register the sensor library.*/
s32Ret = sensor_register_callback();
if (HI_SUCCESS != s32Ret)
{
    printf("register sensor failed!\n");
    return s32Ret;
}

/*Register the HiSilicon AE algorithm library.*/
stLib.s32Id = 0;
strcpy(stLib.acLibName, HI_AE_LIB_NAME);
  
```



```
s32Ret = HI_MPI_AE_Register(&stLib);
if (HI_SUCCESS != s32Ret)
{
    printf("register ae lib failed!\n");
    return s32Ret;
}

/*Register the HiSilicon AWB algorithm library.*/
stLib.s32Id = 0;
strcpy(stLib.acLibName, HI_AWB_LIB_NAME);
s32Ret = HI_MPI_AWB_Register(&stLib);
if (HI_SUCCESS != s32Ret)
{
    printf("register awb lib failed!\n");
    return s32Ret;
}

/*Register the HiSilicon AF algorithm library.*/
stLib.s32Id = 0;
strcpy(stLib.acLibName, HI_AF_LIB_NAME);
s32Ret = HI_MPI_AF_Register(&stLib);
if (HI_SUCCESS != s32Ret)
{
    printf("register af lib failed!\n");
    return s32Ret;
}

/*Initialize the external ISP registers.*/
s32Ret = HI_MPI_ISP_MemInit(IspDev);
if (s32Ret != HI_SUCCESS)
{
    printf("%s: HI_MPI_ISP_Init failed!\n", __FUNCTION__);
    return s32Ret;
}

/*Set the ISP mode to wide dynamic range (WDR) mode.*/
ISP_WDR_MODE_S stWdrMode;
stWdrMode.enWDRMode = enWDRMode;
s32Ret = HI_MPI_ISP_SetWDRMode(0, &stWdrMode);
if (HI_SUCCESS != s32Ret)
{
    printf("start ISP WDR failed!\n");
    return s32Ret;
}
```



```
/*Configure common image attributes.*/
    s32Ret = HI_MPI_ISP_SetPubAttr(IspDev, &stPubAttr);
    if (s32Ret != HI_SUCCESS)
    {
printf("%s: HI_MPI_ISP_SetPubAttr failed with %#x!\n", __FUNCTION__,
s32Ret);
        return s32Ret;
    }

/*Initialize the ISP firmware.*/
s32Ret = HI_MPI_ISP_Init();
if (HI_SUCCESS != s32Ret)
{
    printf("isp init failed!\n");
    return s32Ret;
}

/*Call HI_MPI_ISP_Run to separately start a thread.*/
if (0 != pthread_create(&isp_pid, 0, ISP_Run, NULL))
{
    printf("create isp running thread failed!\n");
    return HI_FAILURE;
}

/*Start services such as VI and VO.*/

...

/*Stop services such as VI and VO.*/

s32Ret = HI_MPI_ISP_Exit();
if (HI_SUCCESS != s32Ret)
{
    printf("isp exit failed!\n");
    return s32Ret;
}

pthread_join(isp_pid, 0);
return HI_SUCCESS;
```



#### NOTE

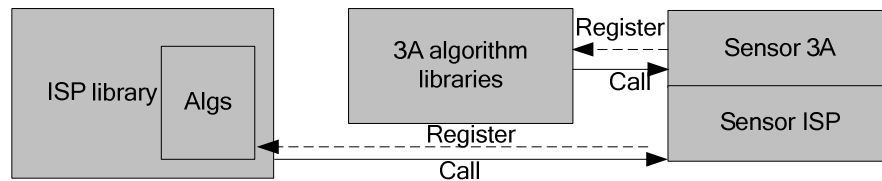
The AE library may use the mathematical library of the standard C library. Users must add `-lm` compilation requirements into **Makefile**.



## 2.2 Sensor Interconnection

The sensor library is used to adapt to different sensors. Differentiated adaptation is required when sensors are registered with the ISP library (which is determined by the basic algorithm unit of the firmware) or when sensors are registered with the HiSilicon 3A algorithm libraries. The interfaces for obtaining sensor initialization parameters and controlling sensors are involved in sensor adaptation. Sensor adaptation is implemented by registering callback functions with the ISP library and 3A algorithm libraries. [Figure 2-2](#) shows the relationship between sensors and the ISP library/3A algorithm libraries.

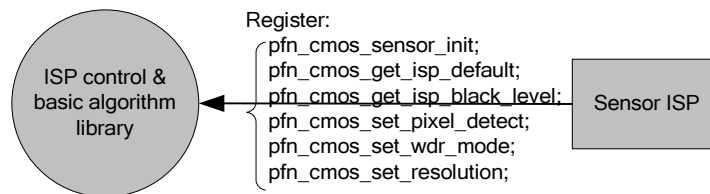
**Figure 2-2** Sensor adaptation diagram



### 2.2.1 Registering Sensors with the ISP Library

As shown in [Figure 2-3](#), sensors are registered with the ISP library by calling HI\_MPI\_ISP\_SensorRegCallBack. For details, see the *HiISP Development Reference*.

**Figure 2-3** Callback function for registering sensors with the ISP library



[Example]

```
ISP_SENSOR_REGISTER_S stIspRegister;
ISP_SENSOR_EXP_FUNC_S *pstSensorExpFunc = &stIspRegister.stSnsExp;

memset(pstSensorExpFunc, 0, sizeof(ISP_SENSOR_EXP_FUNC_S));
pstSensorExpFunc->pfn_cmos_sensor_init = sensor_init;
pstSensorExpFunc->pfn_cmos_sensor_exit = sensor_exit;
pstSensorExpFunc->pfn_cmos_sensor_global_init = sensor_global_init;
pstSensorExpFunc->pfn_cmos_set_image_mode = cmos_set_image_mode;
pstSensorExpFunc->pfn_cmos_set_wdr_mode = cmos_set_wdr_mode;
pstSensorExpFunc->pfn_cmos_get_isp_default = cmos_get_isp_default;
pstSensorExpFunc->pfn_cmos_get_isp_black_level = cmos_get_isp_black_level;
pstSensorExpFunc->pfn_cmos_set_pixel_detect = cmos_set_pixel_detect;
pstSensorExpFunc->pfn_cmos_get_sns_reg_info = cmos_get_sns_regs_info;
```



```
ISP_DEV IspDev = 0;
HI_S32 s32Ret;
s32Ret = HI_MPI_ISP_SensorRegCallBack(IspDev, IMX178_ID, &stIspRegister);
if (s32Ret)
{
    {
        printf("sensor register callback function failed!\n");
        return s32Ret;
    }
}
```

The following callback functions need to be implemented in **xxx\_cmos.c**.

**Table 2-1** Callback function for registering sensors with the ISP library

Member	Description
pfn_cmos_sensor_init	Pointer to the callback function for initializing sensors
pfn_cmos_sensor_exit	Pointer to the callback function for exiting sensors
pfn_cmos_sensor_global_init	Pointer to the callback function for initializing global variables
pfn_cmos_set_image_mode	Pointer to the callback function for setting the image mode
pfn_cmos_set_wdr_mode	Pointer to the callback function for switching between the WDR mode and linear mode
pfn_cmos_get_isp_default	Pointer to the callback function for obtaining the initial value of the ISP basic algorithm
pfn_cmos_get_isp_black_level	Pointer to the callback function for obtaining the sensor black level
pfn_cmos_set_pixel_detect	Pointer to the callback function for enabling or disabling defect pixel correction
pfn_cmos_get_sns_reg_info	Pointer to the callback function for setting the exposure and gain delay of a sensor



**NOTE**

- If a callback function is not implemented currently, you can implement an empty function or set callback function pointers to **null**.



- Typically, the exposure time register of the CMOS sensor has the longest delay from the time the register is configured to the time the configuration takes effect. If the sensor configuration is complete in the blanking region, the setting of the sensor exposure time takes effect in the second frame after configuration and the gain setting takes effect in the first frame after configuration. `pfn_cmos_get_sns_reg_info` is used to ensure that the sensor exposure time and gain calculated by using the AE algorithm libraries are synchronized for each frame. In this way, flicker is avoided. `pfn_cmos_get_sns_reg_info` calls **ISP\_SNS\_REGS\_INFO\_S** to support the I2C and SPI interfaces. For details about **ISP\_SNS\_REGS\_INFO\_S**, see [Table 2-3](#). **u8Cfg2ValidDelayMax** is used to ensure that the settings of the ISP registers (for example, **ISPDgain**) and those of the sensor registers (for example, the exposure ratio) take effect synchronously. For example, if the setting of the sensor exposure time takes effect in the second frame after configuration and the gain setting takes effect in the first frame after configuration, **u8Cfg2ValidDelayMax** needs to be set to **2**, indicating that the maximum number of delayed frames from the time all the sensor registers are configured to the time the configurations take effect is 2; **u8DelayFrmNum** of the exposure time needs to be set to **0**, indicating that no exposure time delay is required; **u8DelayFrmNum** of the gain needs to be set to **1**, indicating that gain needs to be delayed by one frame. In this way, the sensor exposure time and gain calculated by using the AE algorithm libraries take effect synchronously. If the setting of the sensor exposure time and the gain setting both take effect in the second frame after configuration, **u8Cfg2ValidDelayMax** needs to be set to **2**, and **u8DelayFrmNum** of the exposure time as well as **u8DelayFrmNum** of the gain need to be set to **0**. Apart from synchronizing the exposure time with the gain, you can also avoid flicker during frame rate switching by adding the number of variables, synchronously setting the maximum exposure time VMAX (in the unit of exposure line) for each frame, or setting the maximum exposure time HMAX (in the unit of the clock corresponding to the exposure in each row) for each row. Typically **u8DelayFrmNum** of VMAX or HMAX is equal to that of the exposure time. For details about the parameter configuration, see the data sheet of the corresponding sensor. You can enable the anti-flicker function when you check whether the exposure time is synchronized with the gain. If not, picture flicker may occur when the anti-flicker time changes.

**Table 2-2** ISP\_CMOS\_DEFAULT\_S members

Member	Sub Member	Description
stDrc	bEnable	Dynamic range compression (DRC) enable HI_FALSE: disabled HI_TRUE: enabled  In linear mode, the default value is <b>HI_FALSE</b> . In WDR mode, the default value is <b>HI_TRUE</b> and it must be set to <b>HI_TRUE</b> .
	u32BlackLevel	Lower limit of the DRC algorithm. The pixels whose values are less than <b>u32Blacklevel</b> are not processed by the DRC algorithm.  The value range is [0, 0xFFFF], and the default value is 0.
	u32WhiteLevel	Upper limit of the DRC algorithm. The pixels whose values are greater than <b>u32Whitelevel</b> are not processed by the DRC algorithm.  The value range is [0, 0xFFFF]. The default value is 0x4FF in linear mode or 0xFFFF in WDR mode.
	u32SlopeMax	Control parameter for DRC tone curves, which is used to limit the maximum slope (gain) for the dark regions on the DRC curve  The value range is [0, 0xFF]. The default value is 0x30 in linear mode or 0x38 in WDR mode.



Member	Sub Member	Description
	u32SlopeMin	Control parameter for DRC tone curves, which is used to limit the minimum slope (gain) for the bright regions on the DRC curve. The value range is [0, 0xFF]. The default value is 0x0 in linear mode or 0xC0 in WDR mode.
	u32VarianceSpace	Space sensitivity of the DRC algorithm. A larger value indicates that more surrounding pixels are referenced when the tone_curve for each pixel is generated. The value range is [0x0, 0xF]. The default value is 0x4 in linear mode or 0xA in WDR mode.
	u32VarianceIntensity	Luminance sensitivity of the DRC algorithm. A larger value indicates that the difference between the tone_curve of each pixel and that of its surrounding pixels is smaller. The value range is [0x0, 0xF]. The default value is 0x1 in linear mode or 0x4 in WDR mode.
stAgcTbl	bValid	Data validity identifier of the data structure. The value is 0 or 1.
	au8SharpenAltD	Interpolation array for dynamically adjusting the sharpness of the large edges of images based on the gain. The value range is [0, 255].
	au8SharpenAltUd	Interpolation array for dynamically adjusting the sharpness of the small textures of images based on the gain. The value range is [0, 255].
	au8SnrThresh	Interpolation array for dynamically setting the image denoising strength based on the gain. The value range is [0, 255].
	au8DemosaicLumThresh	Array for setting the luminance threshold for the sharpness of large edges of images. The default value is recommended, and the value range is [0, 255].
	au8DemosaicNpOffset	Array for setting image noise parameters. The default value is recommended, and the value range is [0, 255].
	au8GeStrength	Array for setting the parameters of the green equalization parameter. The default value is recommended, and the value range is [0, 255].
stNoiseTbl	bValid	Data validity identifier of the data structure. The value is 0 or 1.
	au8NoiseProfileWeightLut	Array for setting the noise profile related to sensor features. The array value is used as the input of the denoising module. The default value is recommended, and the value range is [0, 255].



Member	Sub Member	Description
	au8DemosaicWeightLut	Array for setting the noise profile related to sensor features. The array value is used as the input of the demosaic module. The default value is recommended, and the value range is [0, 255].
stDemosaic	bValid	Data validity identifier of the data structure. The value is 0 or 1.
	u8VhSlope	Vertical/Horizontal slope threshold. The default value is recommended, and the value range is [0, 255].
	u8AaSlope	Angle slope threshold. The default value is recommended, and the value range is [0, 255].
	u8VaSlope	VH-AA slope threshold. The default value is recommended, and the value range is [0, 255].
	u8UuSlope	Undefined slope threshold. The default value is recommended, and the value range is [0, 255].
	u8SatSlope	Saturation slope threshold. The default value is recommended, and the value range is [0, 255].
	u8AcSlope	High-frequency component filtering slope threshold. The default value is recommended, and the value range is [0, 255].
	u16VhThresh	Vertical/Horizontal threshold. The default value is recommended, and the value range is [0, 0xFFFF].
	u16AaThresh	Angle threshold. The default value is recommended, and the value range is [0, 0xFFFF].
	u16VaThresh	VA threshold. The default value is recommended, and the value range is [0, 0xFFFF].
	u16UuThresh	Undefined threshold. The default value is recommended, and the value range is [0, 0xFFFF].
	u16SatThresh	Saturation threshold. The default value is recommended, and the value range is [0, 0xFFFF].
	u16AcThresh	High-frequency component filtering threshold. The default value is recommended, and the value range is [0, 0xFFFF].
stGammafe	bValid	Data validity identifier of the data structure. The value is 0 or 1. This member must be configured in WDR mode.
	au16Gammafe0	GammaFe table 0. It is generated by the correction tool, and is used to reduce the error during the interpolation of the lookup table and compress images by working with <b>au16Gammafe1</b> . The value range is [0, 0xFFFF].



Member	Sub Member	Description
	au16Gammafe1	GammaFe table 1. It is generated by the correction tool, and is used to compress images by working with <b>au16Gammafe0</b> . The value range is [0, 0xFFFF].
stGamma	bValid	Data validity identifier of the data structure. The value is 0 or 1. If the default gamma curve is used, the member can be set to <b>0</b> .
	au16Gamma	Gamma table. The value range is [0, 0xFFFF].
stShading	bValid	Data validity identifier of the data structure. The value is 0 or 1. If shading correction is not required, you can set this member to make data invalid.
	u16RCenterX	Horizontal coordinate of the R component center The value range is [0x0, 0xFFFF].
	u16RCenterY	Vertical coordinate of the R component center The value range is [0x0, 0xFFFF].
	u16GCenterX	Horizontal coordinate of the G component center The value range is [0x0, 0xFFFF].
	u16GCenterY	Vertical coordinate of the G component center The value range is [0x0, 0xFFFF].
	u16BCenterX	Horizontal coordinate of the B component center The value range is [0x0, 0xFFFF].
	u16BCenterY	Vertical coordinate of the B component center The value range is [0x0, 0xFFFF].
	au16RShadingTbl	Correction table of the R component The value range is [0x0, 0xFFFF].
	au16GShadingTbl	Correction table of the G component The value range is [0x0, 0xFFFF].
	au16BShadingTbl	Correction table of the B component The value range is [0x0, 0xFFFF].
	u16ROffCenter	Distance between the R component center and the farthest angle. A longer distance indicates a smaller value. The value range is [0x0, 0xFFFF].
	u16GOffCenter	Distance between the G component center and the farthest angle. A longer distance indicates a smaller value. The value range is [0x0, 0xFFFF].



Member	Sub Member	Description
	u16BOffCenter	Distance between the B component center and the farthest angle. A longer distance indicates a smaller value. The value range is [0x0, 0xFFFF].
	u16TblNodeNum	Number of used nodes in each component correction table The value range is [0x0, 0x81], and the default value is 0x81.
stRgbSharpen	bValid	Data validity identifier of the data structure. The value is 0 or 1. If the default SharpenRGB curve is used, the member can be set to 0.
	u8LutCore	The member affects the curve slope from the minimum edge magnitude of the SharpenRGB curve to the edge magnitude corresponding to the curve peak, and the curve slope from the edge magnitude corresponding to the curve peak to the maximum edge magnitude. The value range is [0, 255].
	u8LutStrength	Strength adjustment parameter of the sharpen curve. The value range is [0, 127].
	u8LutMagnitude	The member affects the curve slope from the edge magnitude corresponding to the peak value of the SharpenRGB curve to the maximum edge magnitude, and the curve slope from the minimum edge magnitude to the edge magnitude corresponding to the peak value of the curve. The value range is [0, 31].
stSensorMax Resolution	u32MaxWidth	Maximum width supported by the sensor, ensuring that the configured width does not exceed the maximum width when the resolution is switched
	u32MaxHeight	Maximum height supported by the sensor, ensuring that the configured height does not exceed the maximum height when the resolution is switched

**Table 2-3** ISP\_SNS\_REGS\_INFO\_S members

Member	Sub Member	Description
enSnsType	ISP_SNS_I2C_TYPE	Communication between the sensor and the ISP over the I <sup>2</sup> C interface
	ISP_SNS_SSP_TYPE	Communication between the sensor and the ISP over the SSP interface
u32RegNum		Number of registers that need to be configured when exposure results



Member	Sub Member	Description
		are written to the sensor. The member value cannot be dynamically changed.
u8Cfg2ValidDelayMax		Maximum number of delayed frames from the time all the sensor registers are configured to the time the configurations take effect. The unit is frame. This member ensures that the settings of the sensor registers and those of the ISP registers take effect synchronously. Typically, the exposure time register of the CMOS sensor has the longest delay of one frame or two frames. Therefore, this member is usually set to 1 or 2.
astl2cData	bUpdate	HI_TRUE: The sensor registers are written. HI_FALSE: The sensor registers are not written.
	u8DelayFrmNum	Number of delayed frames for the sensor register. This member is configured to ensure that the exposure time and gain take effect at the same time.
	u8DevAddr	Sensor device address
	u32RegAddr	Sensor register address
	u32AddrByteNum	Bit width of the sensor register address
	u32Data	Sensor register data
	u32DataByteNum	Bit width of sensor register data
astSspData	bUpdate	HI_TRUE: The sensor registers are written. HI_FALSE: The sensor registers are not written.
	u8DelayFrmNum	Number of delayed frames for the sensor register. This member is configured to ensure that the exposure time and gain take effect at the same time.
	u32DevAddr	Sensor device address
	u32DevAddrByteNum	Bit width of the sensor device address
	u32RegAddr	Sensor register address
	u32AddrByteNum	Bit width of the sensor register address





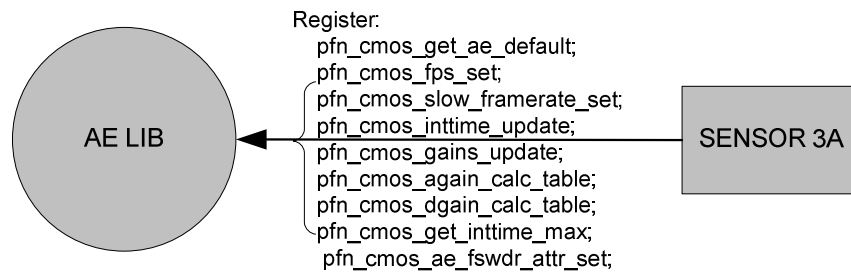
Member	Sub Member	Description
	u32Data	Sensor register data
	u32DataByteNum	Bit width of sensor register data

## 2.2.2 Registering Sensors with 3A Algorithm Libraries

### 2.2.2.1 Registering Sensors with the AE Algorithm Library

As shown in [Figure 2-4](#), sensors are registered with the AE algorithm library by calling HI\_MPI\_AE\_SensorRegCallBack. For details, see the *HiISP Development Reference*.

**Figure 2-4** Callback functions for registering sensors with the AE algorithm library



[Example]

```
ALG_LIB_S stLib;
AE_SENSOR_REGISTER_S stAeRegister;
AE_SENSOR_EXP_FUNC_S *pstExpFuncs = &stAeRegister.stSnsExp;
pstExpFuncs->pfn_cmos_get_ae_default = cmos_get_ae_default;
pstExpFuncs->pfn_cmos_fps_set = cmos_fps_set;
pstExpFuncs->pfn_cmos_slow_framerate_set = cmos_slow_framerate_set;
pstExpFuncs->pfn_cmos_inttime_update = cmos_inttime_update;
pstExpFuncs->pfn_cmos_gains_update = cmos_gains_update;
pstExpFuncs->pfn_cmos_again_calc_table = cmos_again_calc_table;
pstExpFuncs->pfn_cmos_dgain_calc_table = cmos_dgain_calc_table;
pstExpFuncs->pfn_cmos_get_inttime_max = cmos_get_inttime_max;
pstExpFuncs->pfn_cmos_ae_fswdr_attr_set = cmos_ae_fswdr_attr_set;
ISP_DEV IspDev = 0;
stLib.s32Id = 0;
    strncpy(stLib.acLibName, HI_AE_LIB_NAME, sizeof(HI_AE_LIB_NAME));
    s32Ret = HI_MPI_AE_SensorRegCallBack(IspDev, &stLib, IMX178_ID,
&stAeRegister);
    if (s32Ret)
    {
        printf("sensor register callback function to ae lib failed!\n");
```



```
    return s32Ret;  
}
```

The following callback functions need to be implemented in **xxx\_cmos.c**.

**Table 2-4** Callback functions for registering sensors with the AE algorithm library

Member	Description
pfn_cmos_get_ae_default	Pointer to the callback function for obtaining the initial value of the AE algorithm library
pfn_cmos_fps_set	Pointer to the callback function for setting the frame rate of a sensor
pfn_cmos_slow_framerate_set	Pointer to the callback function for reducing the frame rate of a sensor
pfn_cmos_inttime_update	Pointer to the callback function for setting the exposure time of a sensor
pfn_cmos_gains_update	Pointer to the callback function for setting the analog and digital gains of a sensor
pfn_cmos_again_calc_table	Pointer to the callback function for calculating the AE analog gain by searching tables
pfn_cmos_dgain_calc_table	Pointer to the callback function for calculating the AE digital gain by searching tables
pfn_cmos_get_inttime_max	Pointer to the callback function for calculating the maximum exposure time of short exposure frames in 2To1_LINE or 2To1_FRAME WDR mode
pfn_cmos_ae_fswdr_attr_set	Pointer to the callback function for setting the FSWDR running mode



**NOTE**

- If a callback function is not implemented currently, you can implement an empty function or set callback function pointers to **null**.
- In multi-frame combination WDR mode, pfn\_cmos\_get\_inttime\_max needs to be called to calculate the maximum exposure time of the short exposure frame. Then the exposure time and gain of the long frame and those of the short frame are calculated based on the limitations and exposure ratio of the short exposure frame by using the AE algorithms. Other procedures in multi-frame combination WDR mode are the same as those in linear mode. To ensure the effect of the multi-frame combination WDR mode in normal scenarios, you can use the HiSilicon AE algorithms to automatically calculate the exposure ratio based on the histogram statistics and set the ratio of the long frame exposure to the short frame exposure to 1:1 in normal scenarios. Therefore, only one frame of data is used when multiple frames are combined in WDR mode. In this way, smearing is avoided in normal scenarios and the flicker at the power frequency is reduced in normal indoor scenarios.



**Table 2-5** AE\_SENSOR\_DEFAULT\_S members

Member	Description
au8HistThresh	Segmentation threshold array of the 5-segment histogram. The value range is [0, 255]. The default value is {0xD, 0x28, 0x60, 0x80} in linear mode, {0x20, 0x40, 0x60, 0x80} in BUILT_IN WDR mode, or {0xC, 0x18, 0x60, 0x80} in frame combination mode. The default value is recommended.
u8AeCompensation	Target AE luminance. The value range is [0, 255], and the value 0x38~0x40 is recommended.
u32LinesPer500ms	Total number of lines per 500 ms
u32FlickerFreq	Anti-flicker frequency, which is 256 times the current power frequency
f32Fps	Reference frame rate. When the HiSilicon AE algorithms and the callback function <code>cmos_fps_set</code> are used, this member must be assigned to return the actual frame rate.
u32InitExposure	Default initial exposure. It is the exposure time (in line) multiplied by the system gain (6-bit decimal precision). The HiSilicon AE algorithm uses this value as the exposure of the first five frames, which can be used for accelerating the startup of the motion DV. For the products that require fast boot, it is recommended that the initial exposure be configured based on common scenarios to reach fast AE convergence. Otherwise, the default value of this member is 0.
u32InitAESpeed	Default initial AE adjustment speed. The HiSilicon AE algorithm uses this value as the adjustment speed of the first 100 frames, which can be used for accelerating the startup of the motion DV. For the products that require fast boot, it is recommended that the value of this member configured set to <b>128</b> . Otherwise, the default value of this member is 0. The value range will be limited to [64, 255] by the internal AE algorithm.
u32InitAETolerance	Default initial AE tolerance. The HiSilicon AE algorithm uses this value as the exposure tolerance of the first 100 frames to obtain the luminance range for the flag indicating that the AE convergence becomes stable for the first time. When the AE statistical luminance falls within [Target luminance – u32InitAETolerance, Target luminance + u32InitAETolerance] for the first time, the flag indicating that the AE convergence becomes stable for the first time is obtained. The default value 0 is used if this member is not configured. If this member is not set or is set to <b>0</b> in <code>cmos.c</code> , the AE algorithm internally sets it to <b>u8AeCompensation/10</b> . The AE algorithm limits the value of this member to [0, 255] internally.
u32FullLinesStd	Number of valid lines in a frame at the reference frame rate
u32FullLinesMax	Maximum number of rows in one frame after the frame rate of the sensor is reduced. <b>u32FullLinesMax</b> is generally set to the maximum number of rows supported by the sensor.



Member	Description
u32FullLines	Total number of actual rows in each frame for the sensor. When the HiSilicon AE algorithms and the callback functions <code>cmos_fps_set</code> and <code>cmos_slow_framerate_set</code> are used, this member must be assigned to return the total number of actual rows in each frame.
u32MaxIntTime	Maximum exposure time (in line)
u32MinIntTime	Minimum exposure time (in line)
u32MaxIntTimeTarget	Target maximum exposure time (in line)
u32MinIntTimeTarget	Target minimum exposure time (in line)
stIntTimeAccu	Exposure time accuracy
u32MaxAgain	Maximum analog gain (measured by multiple)
u32MinAgain	Minimum analog gain (measured by multiple)
u32MaxAgainTarget	Target maximum analog gain (measured by multiple)
u32MinAgainTarget	Target minimum analog gain (measured by multiple)
stAgainAccu	Analog gain accuracy
u32MaxDgain	Maximum digital gain (measured by multiple)
u32MinDgain	Minimum digital gain (measured by multiple)
u32MaxDgainTarget	Target maximum digital gain (measured by multiple)
u32MinDgainTarget	Target minimum digital gain (measured by multiple)
stDgainAccu	Digital gain accuracy
u32MaxISPDgainTarget	Maximum target ISP digital gain (measured by multiple)
u32MinISPDgainTarget	Minimum target ISP digital gain (measured by multiple)
u32ISPDgainShift	Precision of the ISP digital gain
stAERouteAttr	Default AE route
bAERouteExValid	Whether the extended AE allocation route is valid. When <b>bAERouteExValid</b> is set to <b>HI_TRUE</b> , the extended AE allocation route is used; otherwise, the common AE allocation route is used.
stAERouteAttrEx	Default extended AE allocation route. If this member is set to an appropriate value, the picture effect in WDR mode can be optimized.
u16ManRatioEnable	Manual exposure ratio enable in 2-frame combination WDR mode. When <b>u16ManRatioEnable</b> is set to <b>HI_TRUE</b> , the exposure ratio is fixed and is not automatically updated based on the scenario.

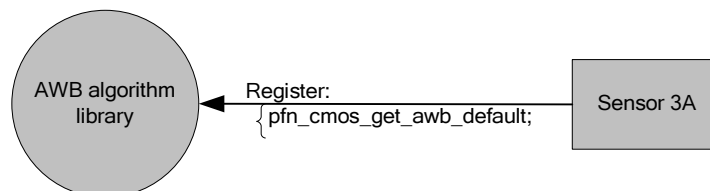


Member	Description
u32Ratio	Default exposure ratio in 2-frame combination WDR mode (6-bit decimal precision). When <b>u16ManRatioEnable</b> is set to <b>HI_TRUE</b> , <b>u32Ratio</b> is used as the default exposure ratio. Value range: [0x40, 0xFFFF]
enIrisType	Default iris type (DC iris or P iris). The AI algorithm fails to work properly when the default iris type is inconsistent with the iris type of the connected lens.
stPirisAttr	P iris parameter, which is related to the lens. This member is required only when the default iris type is P iris.
enMaxIrisFNO	Maximum F value that can be used by the P iris, which is related to the lens and must be configured when the P iris is used. <b>enMaxIrisFNO</b> is used to limit the values of <b>enMaxIrisFNOTarget</b> and <b>enMinIrisFNOTarget</b> that actually take effect to avoid AE allocation exceptions caused when the iris target value falls within an inappropriate range. Value range: [ISP_IRIS_F_NO_32_0, ISP_IRIS_F_NO_1_0]
enMinIrisFNO	Minimum F value that can be used by the P iris, which is related to the lens and must be configured when the P iris is used. <b>enMinIrisFNO</b> is used to limit the values of <b>enMaxIrisFNOTarget</b> and <b>enMinIrisFNOTarget</b> that actually take effect to avoid AE allocation exceptions caused when the iris target value falls within an inappropriate range. Value range: [ISP_IRIS_F_NO_32_0, enMaxIrisFNO]

## 2.2.2.2 Registering Sensors with the AWB Algorithm Library

As shown in [Figure 2-5](#), sensors are registered with the AWB algorithm library by calling `HI_MPI_AWB_SensorRegCallBack`. For details, see the *HiISP Development Reference*.

**Figure 2-5** Callback function for registering sensors with the AWB library



[Example]

```
ALG_LIB_S stLib;  
AWB_SENSOR_REGISTER_S stAwbRegister;  
AWB_SENSOR_EXP_FUNC_S *pstExpFuncs = &stAwbRegister.stSnsExp;
```



```
memset(pstExpFuncs, 0, sizeof(AWB_SENSOR_EXP_FUNC_S));  
pstExpFuncs->pfn_cmos_get_awb_default = cmos_get_awb_default;  
  
ISP_DEV IspDev = 0;  
stLib.s32Id = 0;  
strncpy(stLib.acLibName, HI_AWB_LIB_NAME, sizeof(HI_AWB_LIB_NAME));  
s32Ret = HI_MPI_AWB_SensorRegCallBack(IspDev, &stLib, IMX178_ID,  
&stAwbRegister);  
if (s32Ret)  
{  
  
    printf("sensor register callback function to awb lib failed!\n");  
    return s32Ret;  
}
```

The following callback functions need to be implemented in **xxx\_cmos.c**.

**Table 2-6** Member of the HI\_MPI\_AWB\_SensorRegCallBack callback function for registering sensors with the AWB library

Member	Description
pfn_cmos_get_awb_default	Pointer to the callback function for obtaining the initial value of the AWB algorithm library



**NOTE**

If a callback function is not implemented currently, you can implement an empty function or set callback function pointers to **NULL**.

**Table 2-7** AWB\_SENSOR\_DEFAULT\_S members

Member	Sub Member	Description
u16WbRefTemp	None	Color temperature for correcting static white balance. The value range is [0, 0xFFFF].
au16GainOffset	None	Gain of the R, Gr, Gb, and B color channels for static white balance. The value range is [0, 0xFFFF].
as32WbPara	None	White balance parameter provided by the correction tool. The value range is [0, 0xFFFFFFFF].
stAgcTbl	bValid	Data validity identifier of the data structure. The value is 0 or 1.
	au8Saturation	Interpolation array for dynamically adjusting the saturation based on the gain. The value range is [0, 255].



Member	Sub Member	Description
stCcm	u16HighColorTemp	High color temperature. The value range is [0, 0xFFFF].
	au16HighCCM	High color temperature color correction matrix (CCM). The value range is [0, 0xFFFF].
	u16MidColorTemp	Middle color temperature. The value range is [0, 0xFFFF].
	au16MidCCM	Middle color temperature CCM. The value range is [0, 0xFFFF].
	u16LowColorTemp	Low color temperature. The value range is [0, 0xFFFF].
	au16LowCCM	Low color temperature CCM. The value range is [0, 0xFFFF].

### 2.2.2.3 Registering Sensors with the AF Algorithm Library

Sensors are registered with the AF algorithm library by calling HI\_MPI\_AF\_SensorRegCallBack. The AF algorithm library is not implemented currently.

# 3 Developer Guidance

## 3.1 Overview

You can develop user-defined 3A algorithm libraries based on the firmware framework, and register the library with the firmware. Driven by the interrupt, the firmware obtains statistics of each frame, runs the algorithm library, and configures ISP registers.

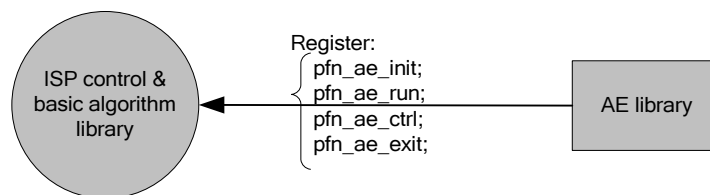
Sensors are registered with the ISP library to implement differentiated adaptation to the basic algorithm unit of the firmware. You are advised to perform operations by referring to chapter 2 "Usage Guidance", ensuring that the algorithms for defect pixel correction, denoising, color enhancement, and lens shading correction properly run. When you register sensors with user-defined 3A algorithm libraries, you need to define data structures to control sensor exposure.

If you use only the HiSilicon 3A algorithm libraries, skip this chapter.

## 3.2 Registering the AE Algorithm with the ISP Library

As shown in Figure 3-1, the AE algorithm is registered with the ISP library by calling HI\_MPI\_ISP\_AeLibRegCallBack. For details, see the *HiISP Development Reference*.

**Figure 3-1** Callback function for registering the AE algorithm with the ISP library



The HI\_MPI\_AE\_Register function is implemented in the HiSilicon AE algorithm. You can register the AE algorithm with the ISP library by calling HI\_MPI\_ISP\_AeLibRegCallBack in HI\_MPI\_AE\_Register. See the following example:

[Example]

```
/* Implement function registration. */  
ISP_AE_REGISTER_S stRegister;
```





```
HI_S32 s32Ret = HI_SUCCESS;

AE_CHECK_POINTER(pstAeLib);
AE_CHECK_HANDLE_ID(pstAeLib->s32Id);
AE_CHECK_LIB_NAME(pstAeLib->acLibName);

/* Call the hook function. */
stRegister.stAeExpFunc.pfn_ae_init = AeInit;
stRegister.stAeExpFunc.pfn_ae_run  = AeRun;
stRegister.stAeExpFunc.pfn_ae_ctrl = AeCtrl;
stRegister.stAeExpFunc.pfn_ae_exit = AeExit;
s32Ret = HI_MPI_ISP_AeLibRegCallBack(pstAeLib, &stRegister);
if (HI_SUCCESS != s32Ret)
{
    printf("Hi_ae register failed!\n");
}
```

You need to implement the following callback functions in the user-defined AE algorithm library.

**Table 3-1** Callback functions for registering the AE algorithm with the ISP library

Member	Description
pfn_ae_init	Pointer to the callback function for initializing the AE algorithm library
pfn_ae_run	Pointer to the callback function for running the AE algorithm library
pfn_ae_ctrl	Pointer to the callback function for controlling the internal status of the AE algorithm library
pfn_ae_exit	Pointer to the callback function for destroying the AE algorithm library



**NOTE**

- When HI\_MPI\_ISP\_Init is called, pfn\_ae\_init is called to initialize the AE algorithm library.
- When HI\_MPI\_ISP\_Run is called, pfn\_ae\_run is called to run the AE algorithm library and calculate the exposure time and gain of the sensor and the digital gain of the ISP.
- pfn\_ae\_ctrl is used to change the internal status of the AE algorithm library. When the ISP runs, the firmware implicitly calls pfn\_ae\_ctrl to prompt the AE algorithm library to switch the WDR or linear mode and set the frame rate.

The following is the current ctrl command defined by the firmware:

```
typedef enum hiISP_CTRL_CMD_E
{
    ISP_WDR_MODE_SET = 8000,
    ISP_AE_FPS_BASE_SET,
    ISP_AWB_ISO_SET, /*Set the ISO value. Change the saturation when
the ISO value changes.*/
    ...
}
```



```
    ISP_CTRL_CMD_BUTT,  
} ISP_CTRL_CMD_E;
```

- When HI\_MPI\_ISP\_Exit is called, pfn\_ae\_exit is called to destroy the AE algorithm library.

**Table 3-2** Members of the initialization parameter data structure ISP\_AE\_PARAM\_S

Member	Description
SensorId	ID of the sensor that is registered with the ISP library. This ID is used to check whether the sensor registered with the ISP library is the same as the one registered with the AE algorithm library.
u8WDRMode	WDR mode. The ISP provides the WDR mode information for the AE.
f32Fps	Frame rate. The ISP provides the frame rate information for the AE.

**Table 3-3** Members of the statistics data structure ISP\_AE\_INFO\_S

Member	Sub Member	Description
u32FrameCnt	None	Total number of frames. The value range is [0, 0xFFFFFFFF].
pstAeStat1	au8MeteringHistThresh	Segmentation threshold array of the 5-segment histogram. The value range is [0, 255].
	au16MeteringHistogram	Statistics array of the 5-segment histogram. The value range is [0, 0xFFFF]. <b>au16MeteringHist[0]</b> indicates the first segment, <b>au16MeteringHist[1]</b> indicates the second segment, <b>au16MeteringHist[2]</b> indicates the fourth segment, and <b>au16MeteringHist[3]</b> indicates the fifth segment. The value of the third segment is calculated as follows: $0xFFFF - (\text{au16MeteringHist}[0] + \text{au16MeteringHist}[1] + \text{au16MeteringHist}[2] + \text{au16MeteringHist}[3])$ .
pstAeStat2	au8MeteringHistThresh	Segmentation threshold array of the 5-segment histogram. The value range is [0, 255].
	au16MeteringMemArray	Zoned statistics array of the 5-segment histogram. The value range is [0, 0xFFFF].
pstAeStat3	au32HistogramMemArray	Statistics array of the 256-segment histogram. The value range is [0, 0xFFFFFFFF].
pstAeStat4	u16GlobalAvgR	Average value of R component in global statistics. The value range is [0, 0xFFFF].
	u16GlobalAvgGr	Average value of Gr component in global statistics. The value range is [0, 0xFFFF].
	u16GlobalAvgG	Average value of Gb component in global statistics.

Member	Sub Member	Description
	b	The value range is [0, 0xFFFF].
	u16GlobalAvgB	Average value of B component in global statistics. The value range is [0, 0xFFFF].
pstAeStat5	au16ZoneAvg	Average values of components R, Gr, Gb, and B in zoned statistics. The value range is [0, 0xFFFF].

#### NOTE

- pstAeStat1** and **pstAeStat2** indicate the normalized global and zoned 5-segment histogram statistics respectively obtained according to the segmentation thresholds of the histogram. The value ranges of the two parameters are [0, 0xFFFF]. Taking the global 5-segment histogram for example, if the values of all the pixels in the picture are greater than the maximum threshold, the data of the fifth segment in the histogram is 0xFFFF and the data of the other four segments in the histogram is 0. The global 5-segment histogram is affected by the zone weights and is irrelevant to the position of the statistics module in the ISP pipeline.
- pstAeStat3** indicates the global 256-segment histogram statistics. The statistics is obtained by collecting the upper eight bits of the data in the input data stream. The data of each segment indicates the number of pixel points corresponding to a specific gray scale. The sum of the data of the 256 segments is the number of pixel points involved in the statistics. The sum is determined by the register with the address of 0x205a06c0, as shown in Figure 3-2. Once the value of the register with the address of 0x205a06c0 is determined, the sum of the data of the 256 segments is determined. For the HiSilicon AE algorithm, the statistics on the Gr channel is used by default, the statistics on the R and Gb channels is used in scenarios with a large red area, and the statistics on the B and Gr channels is used in scenarios with a large blue area. If the configured position of the statistics module for the 256-segment histogram is the same as the position of the statistics module for the 5-segment histogram in the ISP pipeline, the 256-segment histogram will be affected by the zone weights.

**Figure 3-2** Description of the register with the address of 0x205a06c0

Addr	Mode	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Default
0x06C0	R/W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0x00000000
										offset y	skip y			offset x	skip x			

**skip x [2:0]** Histogram decimation in horizontal direction: 0=every 2nd pixel; 1=every 3rd pixel; 2=every 4th pixel; 3=every 5th pixel; 4=every 8th pixel; 5+=every 9th pixel  
**offset x** 0= start from the first column; 1=start from second column  
**skip y [2:0]** Histogram decimation in vertical direction: 0=every pixel; 1=every 2nd pixel; 2=every 3rd pixel; 3=every 4th pixel; 4=every 5th pixel; 5=every 8th pixel; 6+=every 9th pixel  
**offset y** 0= start from the first row; 1= start from second row

- pstAeStat4** indicates the normalized average values of the global components R, Gr, Gb, and B. The value range of **pstAeStat4** is [0, 0xFFFF]. If the statistics is a 12-bit number, the value 0xFFFF indicates that the average value of a component is 4095 (maximum value of a 12-bit number). The average values of the global four components are affected



by the zone weights and are irrelevant to the position of the statistics module in the ISP pipeline.

- **pstAeStat5** indicates the normalized average values of the components R, Gr, Gb, and B for each zone among the 15 x 17 zones. The value range of **pstAeStat5** is [0, 0xFFFF], and the value meaning is the same as that of **pstAeStat4**.
- If the preceding statistics modules are after GammaFE in the ISP pipeline in WDR mode, data after root extraction is collected for statistics. That is, root extraction is performed after the input data is normalized to 1. Taking the 256-segment histogram statistics for example, if the input is 12-bit data and a pixel value is 2048, the upper eight bits are used for statistics in linear mode. That is, the statistics in the histogram is the number of bin pixel points corresponding to the gray scale of 128 plus 1. In WDR mode, the value 0.5 is obtained after the pixel value 2048 is normalized to 1. The value 0.707 is obtained after the root extraction of 0.5. The value 0.707 is represented by the 8-bit data of 181. The statistics in the histogram is the number of bin pixel points corresponding to the gray scale of 181 plus 1. A small pixel value is greatly increased after root extraction. That is, the statistical precision of the dark regions is improved by decreasing the statistical precision of the bright regions in WDR mode. For details about how to change the positions of the preceding statistics modules in the ISP pipeline, see the *HiISP Development Reference*. In WDR mode, assume that the exposure ratio is  $N$ , if the root of the input data for the AE statistical module is not extracted, the first  $256/N$  bins in the global 256-segment histogram are the statistics of long frames, and the remaining bins are the statistics of short frames. If the root of the input data for the AE statistical module is extracted, the first  $256/\sqrt{N}$  bins in the global 256-segment histogram are the statistics of long frames, and the remaining bins are the statistics of short frames. Therefore, when the exposure ratio remains unchanged, more bins in the histogram are used to represent long frames after root extraction. This conclusion is consistent with the conclusion that the statistical precision of the dark region is improved by decreasing the statistical precision of the bright region in WDR mode after root extraction, because long frame data is generally used for the statistics of the dark region.

**Table 3-4** Members of the running result data structure ISP\_AE\_RESULT\_S

Member	Sub Member	Description
au32IntTime	-	Exposure time calculated by the AE (in $\mu$ s). <b>f32Offset</b> in <b>cmos.c</b> must be considered when the exposure time in the unit of line is converted into that in the unit of $\mu$ s. In linear mode and sensor built-in WDR mode, only <b>au32IntTime[0]</b> is valid. It is recommended that <b>au32IntTime[1]</b> to <b>au32IntTime[3]</b> be set to the same value as <b>au32IntTime[0]</b> . In $N$ -frame combination WDR mode, <b>au32IntTime[0]</b> to <b>au32IntTime[N - 1]</b> are valid and their configured values are in ascending order, indicating the shortest exposure time to the longest exposure time. The values are used for calculating the ratio of the long frame exposure to the short frame exposure. It is recommended that <b>au32IntTime[N - 1]</b> to <b>au32IntTime[3]</b> be set to the same value as <b>au32IntTime[N - 1]</b> . The value of <b>au32IntTime[0]</b> needs to be transferred to other modules for associated control related to the exposure time, which affects the HiSilicon AWB



Member	Sub Member	Description
		effect. This data structure must be configured when the HiSilicon AWB algorithms and multi-frame combination WDR mode are used.
u32IspDgain	None	ISP digital gain. This member must be configured when the ISP digital gain is used and is set to <b>0x100</b> when the ISP digital gain is not used.
u32Iso	None	Total gain calculated by the AE. The ISO indicates the system gain and is multiplied by 100. For example, if the system sensor gain is 2x and the ISP gain is 1x, the ISO of the entire system is 200 (2 x 1 x 100). This calculation method applies to all the ISO values in this document. This member affects the adaptation (such as denoising and sharpening) effect and therefore must be configured.
u8AERunInterval	None	Running interval of the AE algorithm. The value range is [1, 255]. The value 1 indicates that the AE algorithm runs for every frame, the value 2 indicates that the AE algorithm runs for every two frames, and so on. The recommended maximum value is 2; otherwise, the AE adjustment speed is affected. In WDR mode, the recommended value is 1, which ensures smooth AE convergence. This member determines the number of interval frames during the time the calculation results of the AE algorithms are configured to the sensor registers and ISP registers. Therefore, this member must be configured.
bPirisValid	None	P iris validity flag. When <b>bPirisValid</b> is set to <b>HI_TRUE</b> , the P iris drive is called back to configure the position of the stepper motor in kernel mode. When <b>bPirisValid</b> is set to <b>HI_FALSE</b> , the P iris drive is not called back. When the HiSilicon AE algorithms are used, <b>bPirisValid</b> must be set to <b>HI_TRUE</b> if the HiSilicon P iris drive connects to the P-iris lens, and it must be set to <b>HI_FALSE</b> if the HiSilicon P iris drive connects to a non-P-iris lens.
s32PirisPos	None	Position of the P iris stepper motor. The value range depends on the specifications of the P-iris lens. <b>s32PirisPos</b> must be configured when the HiSilicon P iris drive connects to the P-iris lens.
u32PirisGain	None	Equivalent gain of the P iris. The value range depends on the specifications of the P-iris lens. This parameter is used to calculate the equivalent exposure when the P iris is working. The equivalent exposure serves as a reference for other modules. The value range is [0, 1024]. When the AE algorithm developed by customers is used and a non-P-iris lens is connected, this value can be set to



Member	Sub Member	Description
		<b>0.</b>
enFSWDRMode	None	FSWDR running mode, which can be normal WDR mode or long frame mode and serves as a reference for other modules. If the long frame mode is enabled, the optimization parameters can be configured. If the AE algorithm developed by the customer does not support the long frame mode, set this member to <b>ISP_FSWDR_NORMAL_MODE</b> . To ensure that the FSWDR module only outputs the long frame data, set this member to <b>ISP_FSWDR_LONG_FRAME_MODE</b> . Note: The long frame mode is valid only in row WDR mode.
stStatAttr	bChange	Whether the sub members of stStatAttr need to be configured again
	au8MeteringHistThresh	Segmentation threshold array of the 5-segment histogram Value range: [0, 255]
	au8WeightTable	AE weight table of 15 x 17 zones Value range: [0, 255]



#### NOTE

The exposure time (**au32IntTime**) in the unit of line can be converted into that in the unit of  $\mu\text{s}$  by using **u32LinesPer500ms** in **cmos.c**. The conversion relationship is as follows:

$$\text{au32IntTime}[0] = \{[(\text{HI\_U64})\text{au32IntTimeRst}[0] \times 1024 - \text{u32Offset}] \times 500000 / \text{pstAeSnsDft} \rightarrow \text{u32LinesPer500ms}\} \gg 10$$

In the preceding formula, **au32IntTimeRst[0]** indicates the exposure time in the unit of line. **u32Offset** is calculated as follows:  $\text{u32Offset} = \text{f32Offset} \times 1024$ . **f32Offset** indicates the exposure time offset. For details, see the description of **AE\_ACCURACY\_S** in the *HiISP Development Reference*.

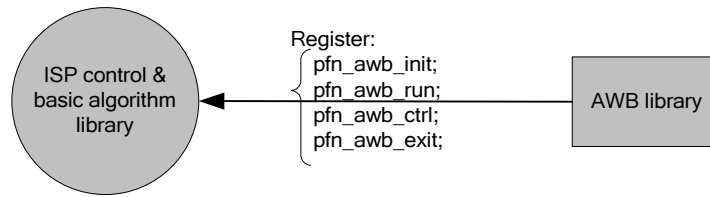
In 2-frame combination WDR mode, if **au32IntTime[0]** is equal to **au32IntTime[1]** (that is, the ratio of the long frame exposure to the short frame exposure is 1:1), the frames are not combined and the long frame data is directly used as the output.

In FSWDR mode, if **enFSWDRMode** is set to **ISP\_FSWDR\_LONG\_FRAME\_MODE**, the AE sets the exposure time of the short frame to the minimum value and that of the long frame close to the maximum value allowed by a frame. In this case, the combination module outputs only the long frame data. **enFSWDRMode** can be used to optimize the picture quality in the weak WDR scenario or low illumination scenario. Note that the long frame mode is valid only in row WDR mode because the exposure time of the long frame in frame WDR mode can reach the maximum value allowed by a frame. In long frame mode, assign values to **au32IntTimeMax[0]** and **au32IntTimeMin[0]** in the callback function **cmos\_get\_inttime\_max** in **cmos.c**, and use **au32IntTimeMin[0]** as the exposure time of the short frame. The default value of **enFSWDRMode** is **ISP\_FSWDR\_NORMAL\_MODE**.

## 3.3 Registering the AWB Algorithm with the ISP Library

As shown in [Figure 3-3](#), the AWB algorithm is registered with the ISP library by calling **HI\_MPI\_ISP\_AwbLibRegCallBack**. For details, see the *HiISP Development Reference*.

**Figure 3-3** Callback function for registering the AWB algorithm with the ISP library



The HI\_MPI\_AWB\_Register function is implemented in the HiSilicon AWB algorithm. You can register the AWB algorithm with the ISP library by calling HI\_MPI\_ISP\_AwbLibRegCallBack in HI\_MPI\_AWB\_Register. The example is similar to that of registering the AE algorithm library.

You need to implement the following callback functions in the user-defined AWB algorithm library.

**Table 3-5** Member of the callback function for registering the AWB algorithm with the ISP library

Member	Description
pfn_awb_init	Pointer to the callback function for initializing the AWB algorithm library
pfn_awb_run	Pointer to the callback function for running the AWB algorithm library
pfn_awb_ctrl	Pointer to the callback function for controlling the internal status of the AWB algorithm library
pfn_awb_exit	Pointer to the callback function for destroying the AWB algorithm library



#### NOTE

- When HI\_MPI\_ISP\_Init is called, pfn\_awb\_init is called to initialize the AWB algorithm library.
- When HI\_MPI\_ISP\_Run is called, pfn\_awb\_run is called to run the AWB algorithm library and calculate the white balance gain and CCM.
- pfn\_awb\_ctrl is used to change the internal status of the AWB algorithm library. When the ISP runs, the firmware implicitly calls pfn\_awb\_ctrl to prompt the AWB algorithm library to switch the WDR or linear mode and set the ISO (sensor gain). The ISO is related to the saturation. The chrominance noises are large when the gain is large. The saturation needs to be adjusted to set the ISO. The following is the current ctrl command defined by the firmware:  

```

typedef enum hiISP_CTRL_CMD_E
{
    ISP_WDR_MODE_SET = 8000,
    ISP_AE_FPS_BASE_SET,
    ISP_AWB_ISO_SET, /*Set the ISO and change the saturation when the ISO changes.*/

    ISP_CTRL_CMD_BUTT,
} ISP_CTRL_CMD_E;

```
- When HI\_MPI\_ISP\_Exit is called, pfn\_awb\_exit is called to destroy the AWB algorithm library.





**Table 3-6** Members of the initialization parameter data structure ISP\_AE\_PARAM\_S

Member	Description
SensorId	ID of the sensor that is registered with the ISP library. This ID is used to check whether the sensor registered with the ISP library is the same as the one registered with the AWB algorithm library.
u8WDRMode	WDR mode. The ISP provides the WDR mode information for the AWB.
s32Rsv	Reserved

**Table 3-7** Members of the statistics data structure ISP\_AE\_INFO\_S

Member	Sub Member	Description
u32FrameCnt	None	Total number of frames. The value range is [0, 0xFFFFFFFF].
pstAwbStat1	u16MeteringAwbRg	Average value of the G/R ratio of white points in the statistics for the RGB field. The value range is [0, 0xFFFF].
	u16MeteringAwbBg	Average value of the G/B ratio of white points in the statistics for the RGB field. The value range is [0, 0xFFFF].
	u32MeteringAwbSum	Number of white points in the statistics for the RGB field. The value range is [0, 0xFFFFFFFF].
pstAwbStat2	au16MeteringMemArrayRg	Average value of the G/R ratio of white points in the zoned statistics for the RGB field. The value range is [0, 0xFFFF].
	au16MeteringMemArrayBg	Average value of the G/B ratio of white points in the zoned statistics for the RGB field. The value range is [0, 0xFFFF].
	au16MeteringMemArraySum	Number of white points in the zoned statistics for the RGB field. The value range is [0, 0xFFFFFFFF].
pstAwbStat3	u16MeteringAwbAvgR	Average R value of white points in the global statistics for the Bayer field (4-bit decimal precision). The black level is subtracted. The value range is [0, 0xFFFF].
	u16MeteringAwbAvgG	Average G value of white points in the





Member	Sub Member	Description
		global statistics for the Bayer field (4-bit decimal precision). The black level is subtracted. The value range is [0, 0xFFFF].
	u16MeteringAwbAvgB	Average B value of white points in the global statistics for the Bayer field (4-bit decimal precision). The black level is subtracted. The value range is [0, 0xFFFF].
	u16MeteringAwbCountAll	Number of white points in the global statistics for the Bayer field, which has been normalized. The value range is [0, 0xFFFF].
	u16MeteringAwbCountMin	Number of pixels less than <b>u16MeteringBlackLevelAwb</b> in the global statistics for the Bayer field, which has been normalized. The value range is [0, 0xFFFF].
	u16MeteringAwbCountMax	Number of pixels greater than <b>u16MeteringWhiteLevelAwb</b> in the global statistics for the Bayer field, which has been normalized. The value range is [0, 0xFFFF].
pstAwbStat4	au16MeteringMemArrayAvgR	Average R value of white points in the zoned statistics for the Bayer field (4-bit decimal precision). The black level is subtracted. The value range is [0, 0xFFFF].
	au16MeteringMemArrayAvgG	Average G value of white points in the zoned statistics for the Bayer field (4-bit decimal precision). The black level is subtracted. The value range is [0, 0xFFFF].
	au16MeteringMemArrayAvgB	Average B value of white points in the zoned statistics for the Bayer field (4-bit decimal precision). The black level is subtracted. The value range is [0, 0xFFFF].
	au16MeteringMemArrayCountAll	Number of white points in the zoned statistics for the Bayer field, which has been normalized. The value range is [0, 0xFFFF].
	au16MeteringMemArrayCountMin	Number of pixels less than <b>u16MeteringBlackLevelAwb</b> in the zoned statistics for the Bayer field, which has been normalized. The



Member	Sub Member	Description
		value range is [0, 0xFFFF].
	au16MeteringMemArrayCountMax	Number of pixels greater than <b>u16MeteringWhiteLevelAwb</b> in the zoned statistics for the Bayer field, which has been normalized. The value range is [0, 0xFFFF].

**Table 3-8** Members of the running result data structure ISP\_AWB\_RESULT\_S

Member	Sub Member	Description
au32WhiteBalanceGain	None	Gain of the R, Gr, Gb, and B color channels obtained by using the AWB algorithm (16-bit precision)
au16ColorMatrix	None	CCM (8-bit precision)
stStatAttr	bChange	Whether the sub members of <b>stStatAttr</b> need to be configured again
	u16MeteringWhiteLevelAwb	Upper luminance limit for searching for white points during white point statistics for the RGB field. The value range is [0x0, 0x3FF], and the default value is 0x3AC.
	u16MeteringBlackLevelAwb	Lower luminance limit for searching for white points during white point statistics for the RGB field. The value range is [0x0, 0x3FF], and the default value is 0x40.
	u16MeteringCrRefMaxAwb	Maximum R/G color difference during white point statistics for the RGB field (8-bit precision). The default value is 0x200.
	u16MeteringCbRefMaxAwb	Maximum B/G color difference during white point statistics for the RGB field (8-bit precision). The default value is 0x200.
	u16MeteringCrRefMinAwb	Minimum R/G color difference during white point statistics for the RGB field (8-bit precision). The default value is 0x80.
	u16MeteringCbRefMinAwb	Minimum B/G color difference during white point statistics for the RGB field (8-bit precision). The



Member	Sub Member	Description
		default value is 0x80.
	u16MeteringCrRefHighAwb	Cr value (8-bit precision) corresponding to CbMax limited in the white point area in the hexagon during white point statistics for the RGB field. The default value is 0x200.
	u16MeteringCrRefLowAwb	Cr value (8-bit precision) corresponding to CbMin limited in the white point area in the hexagon during white point statistics for the RGB field. The default value is 0x80.
	u16MeteringCbRefHighAwb	Cb value (8-bit precision) corresponding to CrMax limited in the white point area in the hexagon during white point statistics for the RGB field. The default value is 0x200.
	u16MeteringCbRefLowAwb	Cb value (8-bit precision) corresponding to CrMin limited in the white point area in the hexagon during white point statistics for the RGB field. The default value is 128.
stRawStatAttr	bChange	Whether the sub members of <b>stRawStatAttr</b> need to be configured again
	u16MeteringWhiteLevelAwb	Upper luminance limit for searching for white points during white point statistics for the Bayer field. The value range is [0x0, 0xFFFF] and the default value is 0xFFFF.
	u16MeteringBlackLevelAwb	Lower luminance limit for searching for white points during white point statistics for the Bayer field. The value range is [0x0, 0xFFFF] and the default value is 0x0.
	u16MeteringCrRefMaxAwb	Maximum R/G color difference during white point statistics for the Bayer field (8-bit precision). The default value is 0x180.
	u16MeteringCbRefMaxAwb	Maximum B/G color difference during white point statistics for the Bayer field (8-bit precision). The



Member	Sub Member	Description
		default value is 0x180.
	u16MeteringCrRefMinAwb	Minimum R/G color difference during white point statistics for the Bayer field (8-bit precision). The default value is 0x40.
	u16MeteringCbRefMinAwb	Minimum B/G color difference during white point statistics for the Bayer field (8-bit precision). The default value is 0x40.
	u16MeteringCrRefHighAwb	Cr value (8-bit precision) corresponding to CbMax limited in the white point area in the hexagon during white point statistics for the Bayer field. The default value is 0x180.
	u16MeteringCrRefLowAwb	Cr value (8-bit precision) corresponding to CbMin limited in the white point area in the hexagon during white point statistics for the Bayer field. The default value is 0x40.
	u16MeteringCbRefHighAwb	Cb value (8-bit precision) corresponding to CrMax limited in the white point area in the hexagon during white point statistics for the Bayer field. The default value is 0x180.
	u16MeteringCbRefLowAwb	Cb value (8-bit precision) corresponding to CrMin limited in the white point area in the hexagon during white point statistics for the Bayer field. The default value is 0x40.

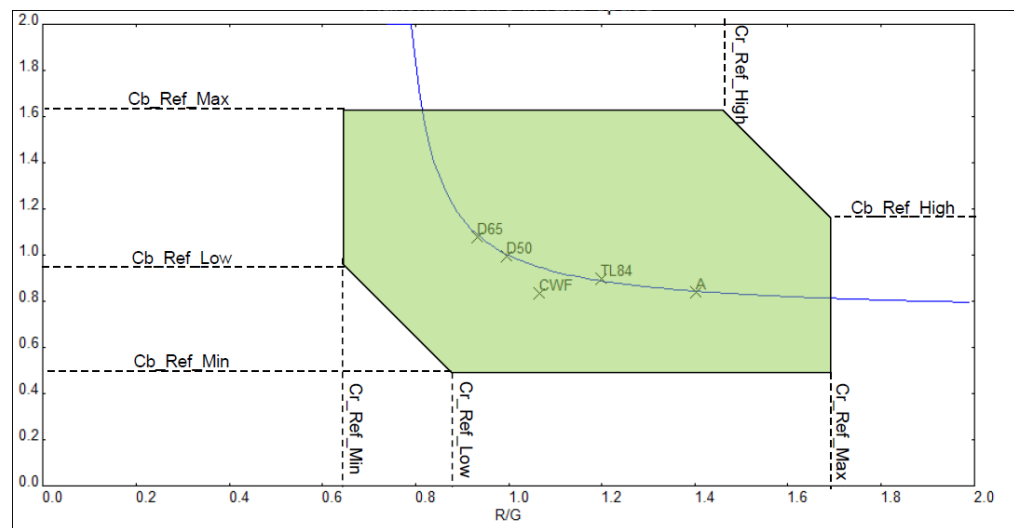


#### NOTE

- The WB statistics for the Bayer field is collected before the WB gain. The WB statistics for the RGB field is collected after the WB gain. The AWB algorithm is implemented based on the statistics for the RGB field. The statistics for the RGB field is affected by the WB gain because the AWB module is a feedback system.
- **u16MeteringAwbRg** indicates the average value of the G/R ratio of white points in the statistics for the RGB field. **u16MeteringCrRefMaxAwb** indicates the maximum R/G color difference during white point statistics for the RGB field.
- To avoid the impact on the algorithm exerted by statistics exceptions, it is recommended that the number of white points in the statistics involved in calculation meet the requirements. The number of white points in the statistics for the RGB field is restricted to 0x40. There is no limitation on the number of white points in the statistics for the Bayer field.
- The AWB parameters take effect in the next frame.

- You are advised to adjust the AWB statistics parameters according to the sensor spectral characteristics and color temperature range supported by the product to achieve better effect.
- In sensor WDR mode, root extraction is performed on the pipeline data. Root extraction is required when the AWB statistical parameters (such as **CrMax**) are configured by calling the corresponding MPI, and square operation is required when the AWB statistical parameters are obtained by calling the corresponding MPI.
- If the AWB statistical parameter **u16MeteringCrRefMaxAwb** is configured in **ISP\_AWB\_RESULT\_S**, the ISP automatically performs root extraction on this parameter. Therefore, manual processing is not required. Square operation is required when the AWB statistical result is obtained by calling **ISP\_AWB\_INFO\_S**. For details about the process, see the sample code.
- The AWB statistics for the Bayer field provided by **HI\_MPI\_ISP\_GetStatistics** remove the black level. However, the AWB statistics are not decompressed in WDR mode.
- The gain of the G channel is set to **1** by the AWB algorithm, while the gains of the R and B channels are greater than 1. Therefore, a bright pink region occurs when the G channel is unsaturated after the G channel data subtracts the black level and the R and B channels remain saturate. To solve this case, the minimum gains of R, G, and B channels need to be normalized to:  $4095/(4095 - \text{Black level})$ .

**Figure 3-4** Statistics parameters



[Example]

The following is a sample example for processing the AWB statistical parameters and result (the AWB statistical result processing method of **ISP\_AWB\_INFO\_S** is the same as that of the corresponding MPI):

```
ISP_DEV IspDev;
ISP_STATISTICS_CFG_S stStatCfg;
ISP_STATISTICS_S stStat;
HI_MPI_ISP_GetStatisticsConfig(IspDev, &stStatCfg);

/*Hi3516A platform, in WDR mode, user should configure awb parameter's
square root, because the pipeline data is non-linear. */

/*WhiteLevel is 10 bits valid. For example, in linear mode, user configure
WhiteLevel = 0x3AC, in wdr mode, user should configure square root of 0x3AC*/
stStatCfg.stWBCfg.stRGCfg.ul6WhiteLevel = (HI_U16)(sqrt(0x3AC << 10));
```



```
/*BlackLevel is 10 bits valid. For example, in linear mode, user configure
BlackLevel = 0x40, in wdr mode, user should configure square root of 0x40*/
stStatCfg.stWBCfg.stRGCfg.ul6BlackLevel = (HI_U16)(sqrt(0x40 << 10));
/*Cr Cb are 4.8-bit fixed-point*/
stStatCfg.stWBCfg.stRGCfg.ul6CrMax = (HI_U16)(sqrt(0x200 << 8));
stStatCfg.stWBCfg.stRGCfg.ul6CrMin = (HI_U16)(sqrt(0x80 << 8));
stStatCfg.stWBCfg.stRGCfg.ul6CbMax = (HI_U16)(sqrt(0x200 << 8));
stStatCfg.stWBCfg.stRGCfg.ul6CbMin = (HI_U16)(sqrt(0x80 << 8));
/*WhiteLevel is 12 bits valid. For example, in linear mode, user configure
WhiteLevel = 0xF00, in wdr mode, user should configure square root of 0xF00
*/
stStatCfg.stWBCfg.stBayerCfg.ul6WhiteLevel = (HI_U16)(sqrt(0xF00 <<
12));

/*BlackLevel is 12 bits valid, Warning: The min value of ul6BlackLevel is
sensor offset.
For example, in linear mode, user configure BlackLevel = 0x100, in wdr mode,
user should configure square root of 0x100*/
stStatCfg.stWBCfg.stBayerCfg.ul6BlackLevel = (HI_U16)(sqrt(0x100 <<
12));

/*Cr Cb are 4.8-bit fixed-point*/
stStatCfg.stWBCfg.stBayerCfg.ul6CrMax = (HI_U16)(sqrt(0x130 << 8));
stStatCfg.stWBCfg.stBayerCfg.ul6CrMin = (HI_U16)(sqrt(0x40 << 8));
stStatCfg.stWBCfg.stBayerCfg.ul6CbMax = (HI_U16)(sqrt(0x120 << 8));
stStatCfg.stWBCfg.stBayerCfg.ul6CbMin = (HI_U16)(sqrt(0x40 << 8));

HI_MPI_ISP_SetStatisticsConfig(IspDev, &stStatCfg);

/*Get AWB statistics*/
HI_MPI_ISP_GetStatistics(IspDev, &stStat);
/*RGB domain statistics, pow 2 to get linear statistics*/
stStat.stWBStat.stRGBStatistics.ul6GlobalGR =
POW2((HI_U32)stStat.stWBStat.stRGBStatistics.ul6GlobalGR) >> 8;
stStat.stWBStat.stRGBStatistics.ul6GlobalGB =
POW2((HI_U32)stStat.stWBStat.stRGBStatistics.ul6GlobalGB) >> 8;
for(i = 0; i < AWB_ZONE_ROW; i++)
{
    for(j = 0; j < AWB_ZONE_COLUMN; j++)
    {
        stStat.stWBStat.stRGBStatistics.au16ZoneGR[i][j] =
POW2((HI_U32)stStat.stWBStat.stRGBStatistics.au16ZoneGR[i][j]) >> 8;
        stStat.stWBStat.stRGBStatistics.au16ZoneGB[i][j] =
POW2((HI_U32)stStat.stWBStat.stRGBStatistics.au16ZoneGB[i][j]) >> 8;
```

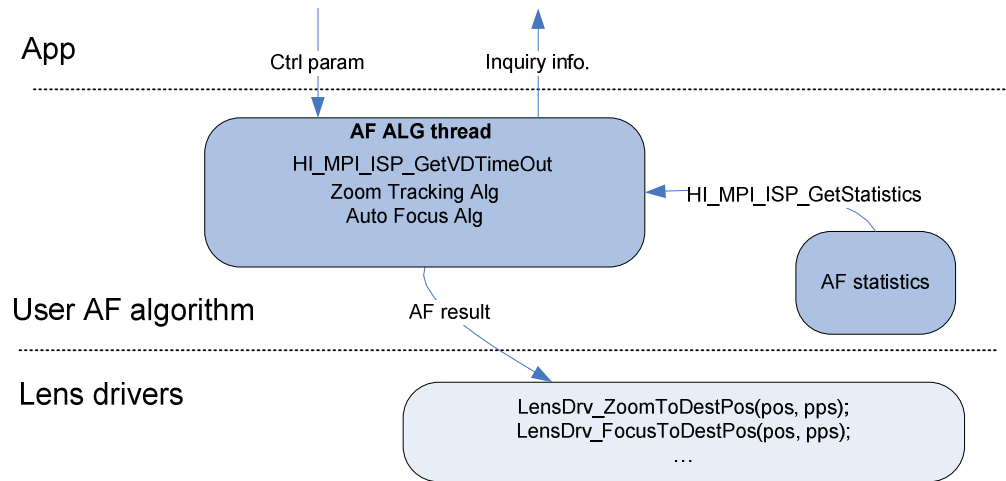


```
    }  
}  
/*Bayer domain statistics, pow2 to get linear data. RGB average value are  
16 bit valid*/  
stStat.stWBStat.stBayerStatistics.ul6GlobalR =  
POW2((HI_U32)stStat.stWBStat.stBayerStatistics.ul6GlobalR) >> 16;  
stStat.stWBStat.stBayerStatistics.ul6GlobalG =  
POW2((HI_U32)stStat.stWBStat.stBayerStatistics.ul6GlobalG) >> 16;  
stStat.stWBStat.stBayerStatistics.ul6GlobalB =  
POW2((HI_U32)stStat.stWBStat.stBayerStatistics.ul6GlobalB) >> 16;  
for(i = 0; i < AWB_ZONE_ROW; i++)  
{  
    for(j = 0; j < AWB_ZONE_COLUMN; j++)  
    {  
        stStat.stWBStat.stBayerStatistics.aul6ZoneAvgR[i][j] =  
POW2((HI_U32)stStat.stWBStat.stBayerStatistics.aul6ZoneAvgR[i][j]) >> 16;  
        stStat.stWBStat.stBayerStatistics.aul6ZoneAvgG[i][j] =  
POW2((HI_U32)stStat.stWBStat.stBayerStatistics.aul6ZoneAvgG[i][j]) >> 16;  
        stStat.stWBStat.stBayerStatistics.aul6ZoneAvgB[i][j] =  
POW2((HI_U32)stStat.stWBStat.stBayerStatistics.aul6ZoneAvgB[i][j]) >> 16;  
    }  
}
```

## 3.4 Developing User-defined AF Algorithm

### 3.4.1 Structure of the AF Algorithm

Figure 3-5 shows the structure of the user-defined AF algorithm. The AF algorithm runs within one thread and can be synchronized with the VD by using the HI\_MPI\_ISP\_GetVDTimeOut interface. The AF statistics can be obtained by calling HI\_MPI\_ISP\_GetStatistics, and related algorithms calculate the target zoom and focus position by referring to the obtained statistics. The AF algorithm outputs the result, which includes the position and speed information of the lens. The camera lens can be driven to the specified position by calling the lens driver. Then the operation of the current frame is completed. The AF algorithm provides the control interface and query interface, through which users can perform a series of operations, including one-click focusing, zooming, manual focusing, and querying the algorithm state.

**Figure 3-5** Structure of the AF algorithm


#### [Example]

If the user uses the Linux system, it is recommended that the algorithm be completed in the user space and lens driver be placed in the kernel space. The ISP driver provides synchronization callback function. For details, see section [3.4.2 "AF Synchronization Callback."](#)

```

while (1)
{
    s32Ret = HI_MPI_ISP_GetVDTimeOut(IspDev, &stVdInfo, 5000);
    s32Ret |= HI_MPI_ISP_GetStatistics(IspDev, &stIspStatics);

    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_ISP_GetStatistics error!(s32Ret = 0x%x)\n", s32Ret);
        return HI_FAILURE;
    }

    // User Auto Focus Alg Source
    UsrAfAlgRun();

    // Update Lens Position
    LensDrv_ZoomToDestPos(pos, pps);
    LensDrv_FocusToDestPos(pos, pps);
}

```

## 3.4.2 AF Synchronization Callback

After the last pixel in a frame of image passes the AF module, the statistics are updated. You are advised to call **HI\_MPI\_ISP\_GetVDTimeOut** to obtain the statistics synchronously. Then the AutoFocus and ZoomTracking algorithms calculate the target focus and zoom position. If

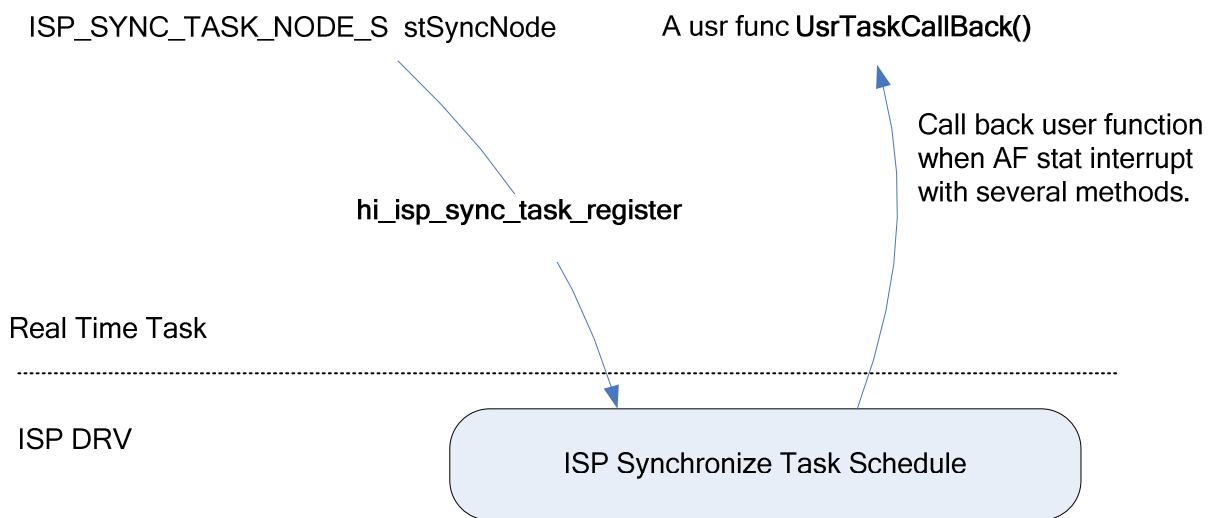


the Linux system is used, you are advised to implement the driver configuration that ensures real-time performance in the kernel space. This is because consistent real-time performance cannot be guaranteed by task scheduling in the Linux user space. The ISP supports the function of registering the synchronization callback function to implement synchronization with the VD. The corresponding sample is provided in `extdrv/sample_ist`. The user can put the tasks that have high requirements on the real-time performance in the synchronization callback function. The bottom layer provides three implementation methods: HwIRQ, Tasklet, and Workqueue. The real-time performance level can be determined by selecting the implementation method.


**NOTE**

For details about `HI_MPI_ISP_GetVDTimeOut`, see the *HiISP Development Reference*.

**Figure 3-6** AF synchronization callback



## API Reference

- [hi\\_isp\\_sync\\_task\\_register](#): Registers the synchronization callback function with the ISP.
- [hi\\_isp\\_sync\\_task\\_unregister](#): Deregisters the synchronization callback function from the ISP.

### hi\_isp\_sync\_task\_register

[Description]

Registers the synchronization callback function with the ISP.

[Syntax]

```
HI_S32 hi_isp_sync_task_register(ISP_DEV dev, ISP_SYNC_TASK_NODE_S *
pstNewNode);
```

[Parameter]



Parameter	Description	Input/Output
dev	ISP device ID	Input
pstNewNode	Synchronization callback node that is newly inserted	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: isp\_ext.h.

[Note]

- Ensure that the ISP driver is loaded before calling this MPI.
- As the entity to which **pstNewNode** (transferred by the user) points is not stored in the internal implementation of the ISP synchronization callback function, the entity defined in [ISP\\_SYNC\\_TASK\\_NODE\\_S](#) cannot be a local variable.

[Example]

None

[See Also]

[hi\\_isp\\_sync\\_task\\_unregister](#)

## hi\_isp\_sync\_task\_unregister

[Description]

Deregisters the synchronization callback function from the ISP.

[Syntax]

```
HI_S32 hi_isp_sync_task_unregister(ISP_DEV dev, ISP\_SYNC\_TASK\_NODE\_S
*pstDelNode)
```

[Parameter]

Parameter	Description	Input/Output
dev	ISP device ID	Input
pstDelNode	Synchronization callback node that needs to be deleted	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: isp\_ext.h.

[Note]

Ensure that the ISP driver is loaded before calling this MPI.

[Example]

None

[See Also]

[hi\\_isp\\_sync\\_task\\_register](#)

## Data Structures

- [ISP\\_SYNC\\_TSK\\_METHOD\\_E](#): Defines the synchronization callback method to determine the real-time performance.
- [ISP\\_SYNC\\_TASK\\_NODE\\_S](#): Defines information about the synchronization callback node.

## ISP\_SYNC\_TSK\_METHOD\_E

[Description]

Defines the synchronization callback method to determine the real-time performance.

[Syntax]

```
typedef enum hiISP_SYNC_TSK_METHOD_E
{
    ISP_SYNC_TSK_METHOD_HW_IRQ = 0,
    ISP_SYNC_TSK_METHOD_TSKLET,
    ISP_SYNC_TSK_METHOD_WORKQUE,
    ISP_SYNC_TSK_METHOD_BUTT
} ISP_SYNC_TSK_METHOD_E;
```

[Member]

Member	Description
ISP_SYNC_TSK_METHOD_HW_IRQ	Hardware interrupt method
ISP_SYNC_TSK_METHOD_TSKLET	TaskLet method
ISP_SYNC_TSK_METHOD_WORKQUE	Work queue method



[Note]

None

[See Also]

None

## ISP\_SYNC\_TASK\_NODE\_S

[Description]

Defines information about the synchronization callback node.

[Syntax]

```
typedef struct hiISP_SYNC_TASK_NODE_S
{
    ISP_SYNC_TSK_METHOD_E enMethod;
    HI_S32 ( *pfnIspSyncTskCallBack ) (HI_U64 u64Data);
    HI_U64 u64Data;
    const char *pszId;
    struct list_head list;
} ISP_SYNC_TASK_NODE_S;
```

[Member]

Member	Description
enMethod	Callback method
pfnIspSyncTskCallBack	Callback function, transferred by the user during registration
u64Data	Callback function parameter, transferred by the user during registration
pszId	Node ID
list	List node used to manage multiple callback nodes. This member can be ignored.

[Example]

```
ISP_SYNC_TASK_NODE_S stSyncNode =
{
    .enMethod = ISP_SYNC_TSK_METHOD_HW_IRQ,
    .pfnIspSyncTskCallBack = UsrTaskCallBack,
    .u64Data = 0,
    .pszId = "HardwareInterrupt "
};
```

[Note]



None

[See Also]

[ISP\\_SYNC\\_TSK\\_METHOD\\_E](#)



# 4 Appendix

## 4.1 Relationships Between Registered Functions

The HI\_MPI\_ISP\_AeLibRegCallBack, HI\_MPI\_ISP\_AwbLibRegCallBack, and HI\_MPI\_ISP\_AfLibRegCallBack interfaces are hook functions provided by the ISP firmware library. They are used to implement registration during development of 3A algorithm libraries. For example, during implementation of the HI\_MPI\_AE\_Register, HI\_MPI\_AWB\_Register, and HI\_MPI\_AF\_Register interfaces for the 3A algorithm library provided by HiSilicon, relevant hook functions are called. This means that the HI\_MPI\_AE\_Register interface can be called to make the AE algorithm library register with the ISP firmware library.

The 3A algorithm library of HiSilicon also provides hook functions to make the sensor library register with the 3A algorithm library. For example, the sensor\_register\_callback function that calls the HI\_MPI\_AE\_SensorRegCallBack, HI\_MPI\_AWB\_SensorRegCallBack, and HI\_MPI\_AF\_SensorRegCallBack hook functions can be seen in **xxx\_cmos.c**. When developing a 3A algorithm library, users can also provide hook functions to make the sensor library register with the 3A algorithm library.

Without doubt, the ISP firmware library also provides hook functions to make the sensor library register with the ISP firmware library. For example, the sensor\_register\_callback function that calls the HI\_MPI\_ISP\_SensorRegCallBack hook function can be seen in **xxx\_cmos.c**.

In conclusion, users only need to call HI\_MPI\_AE\_Register, HI\_MPI\_AWB\_Register, HI\_MPI\_AF\_Register, and sensor\_register\_callback to make the 3A algorithm library register with the ISP firmware library and make the sensor library register with the 3A algorithm library and ISP firmware library.



### NOTE

When developing a 3A algorithm library, users must implement the HI\_MPI\_XXX\_Register interface on their own. In addition, the users must implement the HI\_MPI\_XXX\_SensorRegCallBack hook function and add relevant codes to sensor\_register\_callback for invoking this hook function. For details about the relevant codes, refer to the open source codes of the ISP firmware library.

## 4.2 Scalability Design Considerations

Relevant concepts, such as ISP\_DEV, ALG\_LIB\_S, and SENSOR\_ID, exist in the codes. They are proposed for architecture scalability.



ISP\_DEV considers the scenario where multiple ISP units need to be supported. No matter multiple ISP hardware units or time division multiplexing of one ISP hardware unit, scalability needs to be reserved in terms of software. Currently, ISP\_DEV needs to be set to 0.

ALG\_LIB\_S considers the scenario where multiple algorithm libraries need to be supported and dynamic switching of them needs to be supported. For example, if a user implements a set of AE algorithm codes, but registers with two libraries for the normal scenario and snapshot scenario, s32Handle in the structure needs to be used for making a distinction. If a user implements a set of AWB algorithm codes, but this user also wants to use HiSilicon AWB algorithm libraries in some scenarios, this user can use the acLibName in the structure to make a distinction. If a user registers with multiple AE or AWB libraries, the ISP firmware initializes all of the libraries, but calls only valid libraries during the running. The HI\_MPI\_ISP\_SetBindAttr interface is used for setting valid libraries. This interface can also be used to quickly switch operating libraries.

SENSOR\_ID only provides the verification function. It verifies that the same sensor is registered with the ISP firmware library and 3A algorithm library.

These concepts are only redundancy reserved during the design, which can be removed during the development if they are not required at all.

## 4.3 3A Architecture Design

The ISP firmware initializes and destroys all algorithm units. During the running, the ISP firmware provides the statistics information of the previous frame and configures the register based on the return value. Other contents are all developed by users. Therefore, after a user replaces his/her 3A algorithm, the MPI of the current AE/AWB/AF, the contents related to AE/AWB/AF in **cmos.c**, AE weight configurations, threshold configurations of the 5-segment histogram, and AWB white point search configuration cannot be reused. In theory, these configurations are configured by the 3A algorithm, but not obtained from the ISP firmware. The ISP firmware only contains simple initialization values.

When configuring an ISP register, the 3A algorithm does not need to display the configurations. It only needs to write the values to be set for relevant parameters of the ISP register into the ISP\_AE\_RESULT\_S, ISP\_AWB\_RESULT\_S, and ISP\_AF\_RESULT\_S structures. When reading contents in an ISP register, the 3A algorithm does not need to display the read contents either. It only needs to read relevant contents in the ISP\_AE\_INFO\_S, ISP\_AWB\_INFO\_S, and ISP\_AF\_INFO\_S structures.

## 4.4 External Register Description

In IPC applications, in addition to the processes of the main service program, other processes are available on the board for allowing the tools on a PC to adjust the image quality. The status of various algorithms and the parameters of the ISP all persist in global variables. Consequently, existing registers are not capable enough to support the access of multiple processes. In this case, external registers are introduced to support the multi-process service scenario. Users use tools on the PC to communicate with the processes on the board and use the MPI provided by HiSilicon to change the configurations in external registers, thereby changing the status and parameters of various algorithms of the ISP in the main service program.

External registers and actual hardware registers support read/write through a unified interface. External registers are the same as actual hardware registers in terms of functions.



The VReg\_Init, VReg\_Exit, IORD\_32DIRECT, IORD\_16DIRECT, IORD\_8DIRECT, IOWR\_32DIRECT, IOWR\_16DIRECT, and IOWR\_8DIRECT interfaces are encapsulated into external registers. The address settings are as follows:

0x0-0xFFFF correspond to hardware registers of the ISP. For example, IORD\_32DIRECT(0x0008) reads the values of the 0x205A0008 hardware register.

0x10000-0x1FFFF correspond to external registers of the ISP firmware, for example, IOWR\_16DIRECT(0x10020).

0x20000-0x2FFFF correspond to external registers of the AE. A total of 16 groups of such external registers are allocated, and those external registers correspond 0x20000-0x21FFF are used by HiSilicon AE. 0x40000-0x4FFFF correspond to external registers of the AWB. 0x30000-0x3FFFF correspond to external registers of the AF. A total of 16 groups of such external registers are allocated.

When using external registers, user can use the encapsulated interfaces. However, they can also use other solutions to support multi-process access. The address spaces of external registers are defined by users. Users only need to ensure that no conflict exists between them. For details, refer to the open source codes. Interface definition is provided in the **hi\_vreg.h** file.