Hi3518_ISP_3A

User Guide

Issue	**02**
Date	**2013-09-25**

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

About This Document

Purpose

This document describes the functions, principles, and workflow of the Hi3518_ISP_3A. 3A functions indicate automatic exposure (AE), automatic white balance (AWB), and automatic focus (AF).

Related Version

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

Product Name	Version
Hi3518_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
⚠ DANGER	Alerts you to a high risk hazard that could, if not avoided, result in serious injury or death.
⚠ WARNING	Alerts you to a medium or low risk hazard that could, if not avoided, result in moderate or minor injury.

Symbol	Description
⚠ CAUTION	Alerts you to a potentially hazardous situation that could, if not avoided, result in equipment damage, data loss, performance deterioration, or unanticipated results.
TIP	Provides a tip that may help you solve a problem or save time.
NOTE	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 02 (2013-09-25)

This issue is the second official release.

Chapter 2 Usage Guidance

In section 2.1, descriptions are added.

Chapter 3 Developer Guidance

In section 3.2, notes are added to the **Example** field.

Chapter 4 Appendix

This chapter is added.

Issue 01 (2013-07-16)

This issue is the first official release.

Contents

Figures

Tables

1 Introduction to the Hi3518_ISP_3A

1.1 Overview

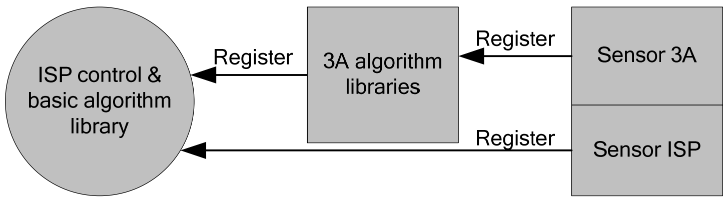
The Hi3518_ISP_3A depends on the software development kit (SDK) of a specific major release. It processes digital images by using a series of digital image processing algorithms. The Hi3518_ISP_3A includes the firmware framework and HiSilicon 3A libraries. The firmware processes statistics, drives digital 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

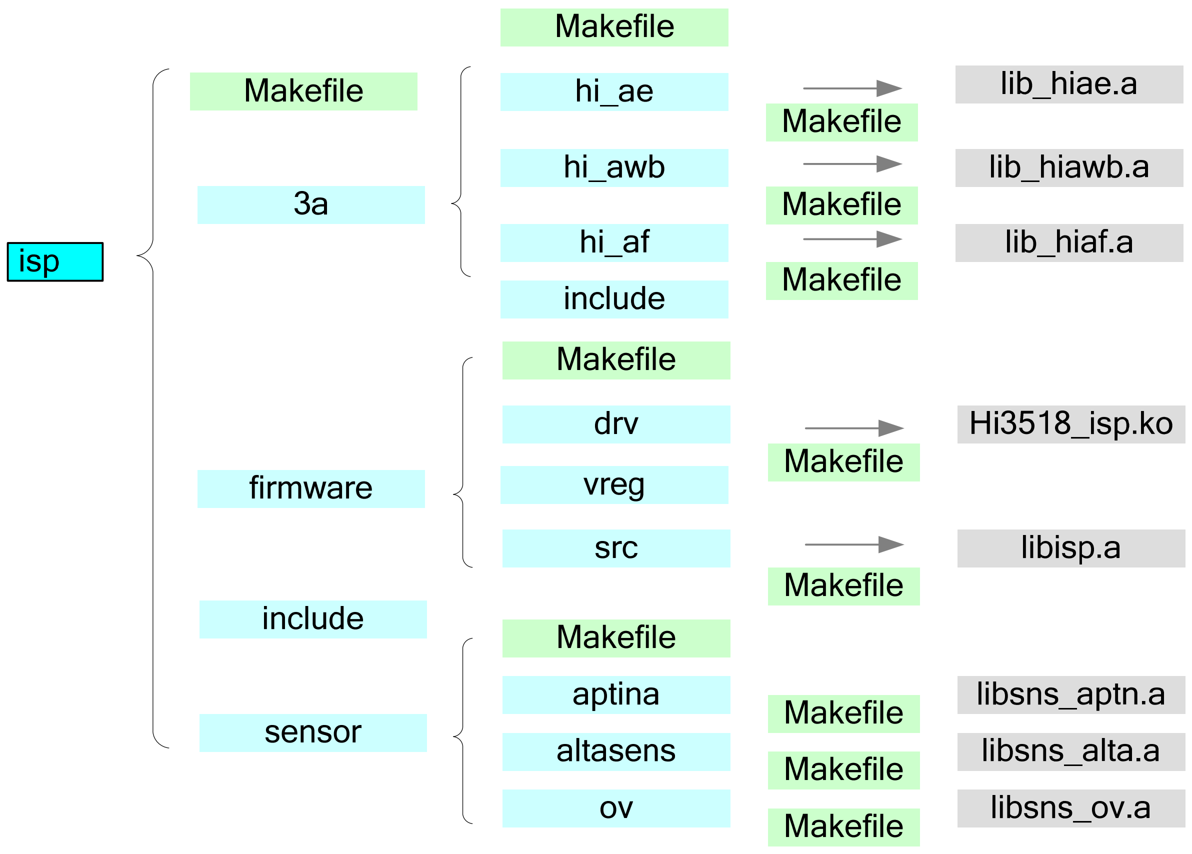


Different sensors register callback control functions with the ISP library and 3A algorithm libraries. When the ISP control unit schedules the basic algorithm unit and 3A algorithm libraries, the callback functions are called to obtain initialization parameters and control sensors, for example, adjusting 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 libraries, and sensor library are stored in different folders. The driver generated in the **drv** folder of the firmware reports the ISP interrupt. The interrupt is used to drive the ISP control unit. The ISP control unit obtains statistics from the driver, schedules the basic algorithm unit and 3A algorithm libraries, and notifies the driver of the registers to be configured. 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 driver code of each sensor is complied as a library to facilitate application compilation and connection between the ISP firmware and sensors. You can call related callback functions based on the features of the used sensor.

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.

NOTE

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

1.2.3.3 Differences Between the Original Hi3518 ISP and the New Hi3518 ISP with Separate 3A Algorithm Libraries

The previous Hi3518 SDK provides the ISP whose 3A algorithms are included in the ISP library. Note the following when you use the new Hi3518 ISP with separate 3A algorithm libraries:

- **Makefile** may need to be modified. Only **libisp.a** needs to be linked for the old ISP, and **libisp.a**, **lib_hiae.a**, **lib_hiawb.a**, and **lib_hiaf.a** need to be linked for the new ISP.
- The required header files are added. Only the header files **mpi_isp.h**, **hi_comm_isp.h**, **hi_comm_sns.h**, and **hi_sns_ctrl.h** are required for the old ISP. In addition to the preceding header files, extra header files **hi_comm_3a.h**, **hi_ae_comm.h**, **hi_af_comm.h**, **hi_awb_comm.h**, **mpi_ae.h**, **mpi_af.h**, and **mpi_awb.h** are required for the new ISP.
- The initialization process changes. In the old initialization process, sensor_init and sensor_register_callback need to be called. In the new initialization process, besides sensor_register_callback, HI_MPI_AE_Register, HI_MPI_AWB_Register, and HI_MPI_AF_Register need to be called to register AE, AWB, and AF algorithms respectively. For details about samples, see section 2.1 "Software Process."
- The process for switching between wide dynamic range (WDR) and linear modes changes. In the old switching processing, sensor_mode_set, sensor_register_callback, and HI_MPI_ISP_Init need to be called. In the new switching processing, only HI_MPI_ISP_SetWdrAttr needs to be called.
- The usage of other media processing platform interfaces (MPIs) is the same as that of old MPIs.

- If you have adjusted the parameters in the old ***xxx*_cmos.c** file, you need to modify the new ***xxx*_cmos.c** file based on the adjustments. For details about data structures, see section 2.2 "Sensor Interconnection" or the *HiISP Development Reference*.

1.2.4 Internal Process

Figure 1-3 shows the internal process of the ISP firmware. Firstly, 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

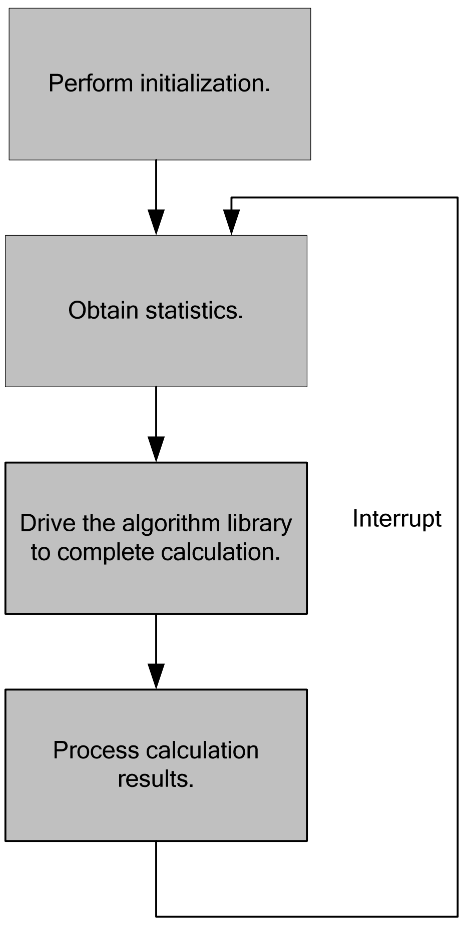
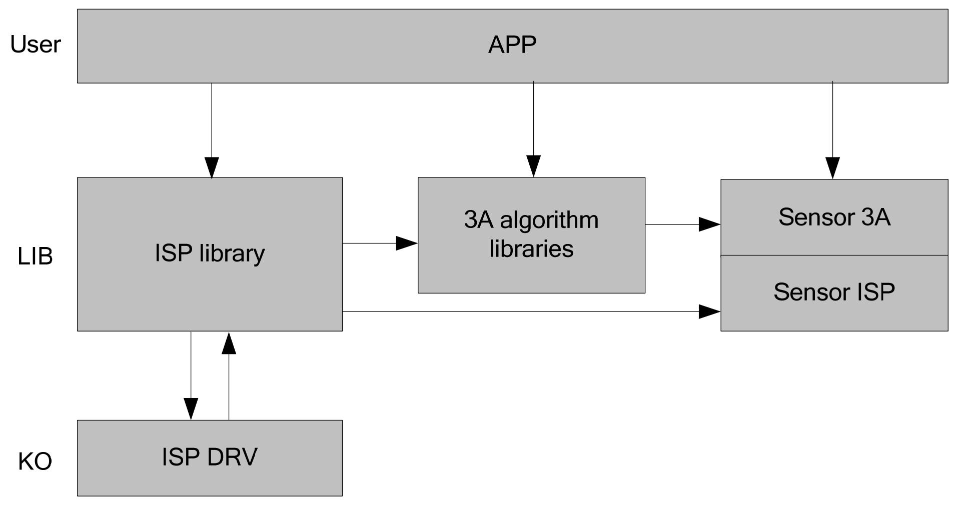

Perform initialization.
Obtain statistics.
Drive the algorithm library to complete calculation.
Interrupt
Process calculation results.

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

Figure 1-4 Architecture of the ISP firmware



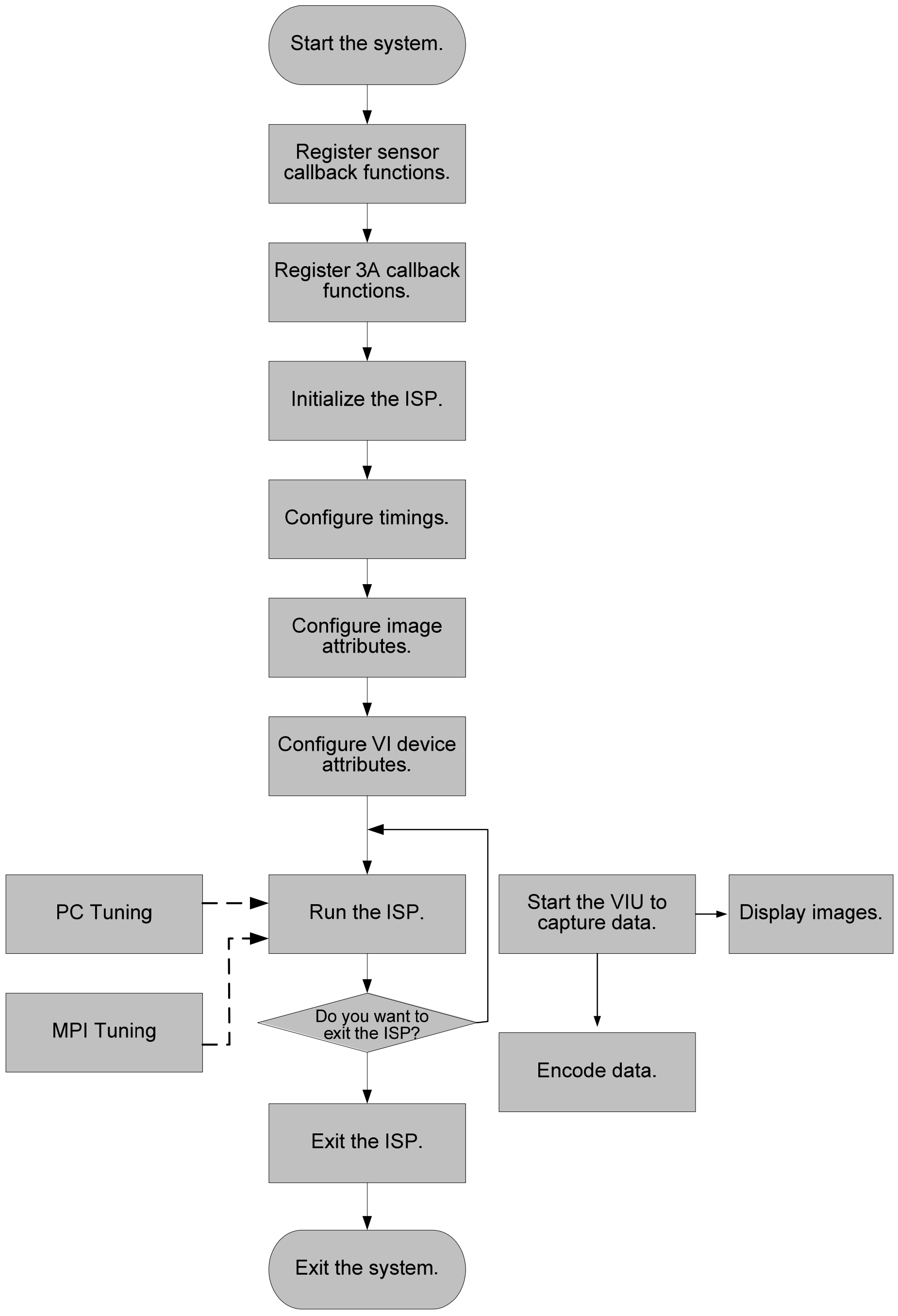
2 Usage Guidance

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, stored in the DDR, and then transmitted for displaying or encoding. Figure 2-1 shows the software process.

The PC Tuning tool 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. If the PC Tuning tool is not provided in the product release phase, you can call the image quality adjustment MPI to tune the image effect.

Figure 2-1 Workflow of the ISP firmware


Start the system.
Register sensor callback functions.
Register 3A callback functions.
Initialize the ISP.
Configure timings.
Configure image attributes.
Configure VI device attributes.
PC Tuning
MPI Tuning
Run the ISP.
Start the VIU to capture data.
Display images.
Do you want to exit the ISP?
Encode data.
Exit the ISP.
Exit the system.

After adjusting images, you can save settings by using the configuration file of the PC Tuning tool. Then the configured image parameters can be loaded when the ISP is started next time.

The following is a code sample:

```
HI_S32 s32Ret;
ALG_LIB_S stLib;
ISP_IMAGE_ATTR_S stImageAttr;
ISP_INPUT_TIMING_S stInputTiming;
```

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

```
    return s32Ret;
}

/*Set ImageAttr and InputTiming.*/
stImageAttr.enBayer        = BAYER_GRBG;
stImageAttr.u16FrameRate   = 30;
stImageAttr.u16Height      = 720;
stImageAttr.u16Width       = 1280;
s32Ret = HI_MPI_ISP_SetImageAttr(&stImageAttr);
if (HI_SUCCESS != s32Ret)
{
    printf("set image attr failed!\n");
    return s32Ret;
}

stInputTiming.enWndMode = ISP_WIND_NONE;
s32Ret = HI_MPI_ISP_SetInputTiming(&stInputTiming);
if (HI_SUCCESS != s32Ret)
{
    printf("set input timing 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;
```

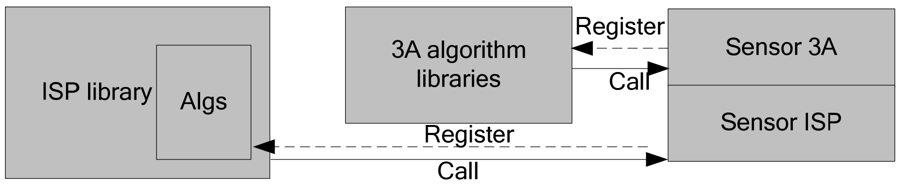
NOTE

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.

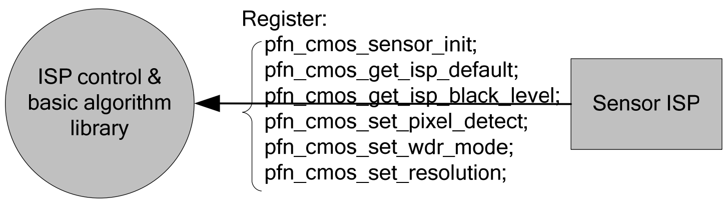
Figure 2-2 Sensor adaptation diagram



2.2.2 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


ISP control & basic algorithm library
Register:
pfn_cmos_sensor_init;
pfn_cmos_get_isp_default;
pfn_cmos_get_isp_black_level;
pfn_cmos_set_pixel_detect;
pfn_cmos_set_wdr_mode;
pfn_cmos_set_resolution;
Sensor ISP

[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_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_set_wdr_mode = cmos_set_wdr_mode;
s32Ret = HI_MPI_ISP_SensorRegCallBack(IMX104_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_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_set_wdr_mode	Pointer to the callback function for switching between the WDR mode and linear mode
pfn_cmos_set_resolution	Pointer to the callback function for setting the resolution

NOTE

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

Table 2-2 ISP_CMOS_DEFAULT_S members

Member	Sub Member	Description
stComm	u8Rggb	RGrGbB output sequence of the sensor. The value range is [0, 3].
	u8BalanceFe	The default value is 0x1. You are advised not to change the default value.
stDenoise	u8SinterThresh	The default value is 0x15. You are advised not to change the default value.
	u8NoiseProfile	Noise profile. The default value 0 indicates that the default ISP noise profile is used. You are advised to use the default value 0.
	u16Nr0	The default value is 0x0. You are advised not to change the default value.
	u16Nr1	The default value is 0x0. You are advised not

Member	Sub Member	Description
		to change the default value.
stDrc	u8DrcBlack	The default value is 0x0. You are advised not to change the default value.
	u8DrcVs	The default value is 0x04 in linear mode or 0x08 in WDR mode. You are advised not to change the default value.
	u8DrcVi	The default value is 0x08 in linear mode or 0x01 in WDR mode. You are advised not to change the default value.
	u8DrcSm	The default value is 0xA0 in linear mode or 0x3C in WDR mode. You are advised not to change the default value.
	u16DrcWl	The default value is 0x4FF in linear mode or 0xFFF in WDR mode. You are advised not to change the default value.
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

Member	Sub Member	Description
		input of the denoise module. The default value is recommended, and the value range is [0, 255].
	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].

Member	Sub Member	Description
	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 needs to be configured when the sensor supports the WDR mode.
	au16Gammafe	GammaFe 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.

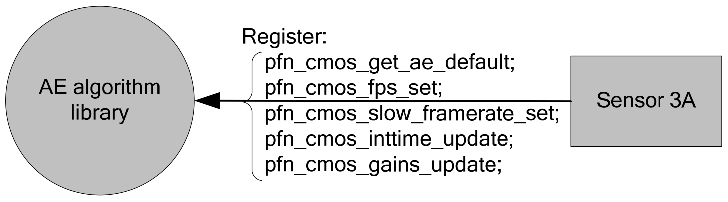
Member	Sub Member	Description
		The value range is [0x0, 0xFFFF].
	u16GOffCenter	Distance between the G component center and the farthest angle. A longer distance indicates a smaller value. The value range is [0x0, 0xFFFF].
	u16BOffCenter	Distance between the B component center and the farthest angle. 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.

2.2.3 Registering Sensors with 3A Algorithm Libraries

2.2.3.1 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 function 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;

memset(pstExpFuncs, 0, sizeof(AE_SENSOR_EXP_FUNC_S));
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;

stLib.s32Id = 0;
```

```
strcpy(stLib.acLibName, HI_AE_LIB_NAME);
s32Ret = HI_MPI_AE_SensorRegCallBack(&stLib, IMX104_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 2-3 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

NOTE

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

Table 2-4 AE_SENSOR_DEFAULT_S members

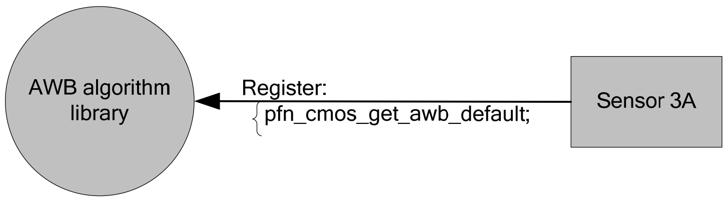
Member	Description
au8HistThresh	Segmentation threshold array of the 5-segment histogram. The value range is [0, 255]. The default value is {0xd, 0x28, 0x60, 0x80} in linear mode or {0x20, 0x40, 0x60, 0x80} in WDR mode. The default value is recommended.
u8AeCompensation	Target AE luminance. The value range is [0, 255], and the value 0x80 is recommended.
u32LinesPer500ms	Total number of lines per 500 ms
u32FlickerFreq	Anti-flicker frequency, which is 256 times the current power frequency
u32MaxIntTime	Maximum exposure time (in line)
u32MinIntTime	Minimum exposure time (in line)
u32MaxIntTimeTarget	Target maximum exposure time (in line)

Member	Description
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

2.2.3.2 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 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;

stLib.s32Id = 0;
strcpy(stLib.acLibName, HI_AWB_LIB_NAME);
s32Ret = HI_MPI_AWB_SensorRegCallBack(&stLib, IMX104_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 2-5 Member of the HI_MPI_AWB_SensorRegCallBackCallBack 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 2-6 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].
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,

Member	Sub Member	Description
		0xFFFF].
	au16LowCCM	Low color temperature CCM. The value range is [0, 0xFFFF].

2.2.3.3 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.

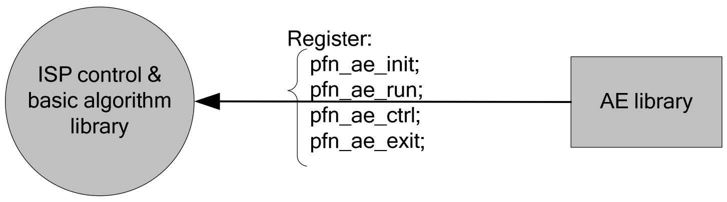
Sensors are registered with the ISP library to implement differentiated adaptation to the basic algorithm unit of the firmware. You are advised to perform operations by referring to chapter 2 "Usage Guidance", ensuring that the algorithms for defect pixel correction, denoising, color enhancement, and 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, because the firmware does not 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

- pfn_ae_init is called when HI_MPI_ISP_Init is called to initialize the AE algorithm library.
- pfn_ae_run is called when cHI_MPI_ISP_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 inexplicitly 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 and change the saturation when the ISO changes.*/

      ISP_CTRL_CMD_BUTT,
  } ISP_CTRL_CMD_E;
  ```
- pfn_ae_exit is called when HI_MPI_ISP_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.
u32MaxIspDgain	Maximum digital gain of the ISP with the accuracy of u32IspDgainShift
u32MinIspDgain	Minimum digital gain of the ISP with the accuracy of u32IspDgainShift
u32IspDgainShift	ISP digital gain accuracy

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].
	au16MeteringHist	Statistics array of the 5-segment histogram. The value range is [0, 0xFFFF].
pstAeStat2	au8MeteringHistThresh	Segmentation threshold array of the 5-segment histogram. The value range is [0, 255].
	au16MeteringMemArrary	Zone statistics array of the 5-segment histogram. The value range is [0, 0xFFFF].
pstAeStat3	au16HistogramMemArray	Statistics array of the 256-segment histogram. The value range is [0, 0xFFFF].

Table 3-4 Members of the running result data structure ISP_AE_RESULT_S

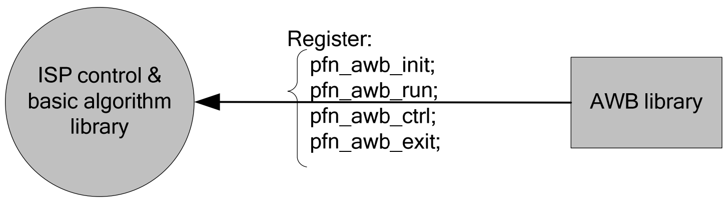
Member	Sub Member	Description
u32IspDgain	None	ISP digital gain
u32IspDgainShift	None	ISP digital gain accuracy
u32Iso	None	Total gain calculated by the AE algorithm library
stStatAttr	bChange	Whether the sub members of stStatAttr need to be configured again

Member	Sub Member	Description
	au8MeteringHistThresh	Segmentation threshold array of the 5-segment histogram. The value range is [0, 255].
	au8WeightTable	AE weight table of 15 x 17 zones. The value range is [0, 255].

3.3 Registering the AWB Algorithm with the ISP Library

As shown in Figure 3-2, the AWB algorithm is registered with the ISP library by calling HI_MPI_ISP_AwbLibRegCallBack. For details, see the *HiISP Development Reference*.

Figure 3-2 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

- pfn_awb_init is called when HI_MPI_ISP_Init is called to initialize the AWB algorithm library.
- pfn_awb_run is called when HI_MPI_ISP_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 inexplicitly 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;
  ```
- pfn_awb_exit is called when HI_MPI_ISP_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.
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	Ratio of the average R value to the average G value of white points in statistics. The value range is [0, 0xFFFF].
	u16MeteringAwbBg	Ratio of the average B value to the average G value of white points in statistics. The value range is [0, 0xFFFF].
	u32MeteringAwbSum	Number of white points in the statistics. The value range is [0, 0xFFFFFFFF].
pstAwbStat2	au16MeteringMemArrayRg	Ratio of the average R value to the average G value of white points in statistics by zone. The value range is [0, 0xFFFF].
	au16MeteringMemArrayBg	Ratio of the average B value to the average G value of white points in statistics by zone. The value range is [0, 0xFFFF].

Member	Sub Member	Description
	au16MeteringMemArraySum	Number of white points in the statistics by zone. The value range is [0, 0xFFFFFFFF].

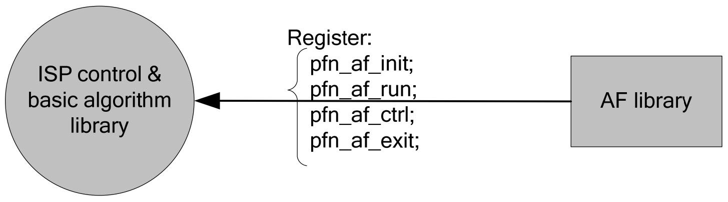
Table 3-8 Members of the running result data structure ISP_AWB_RESULT_S

Member	Sub Member	Description
au32WhiteBalance Gain	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 **stStatAttr** need to be configured again
	u16MeteringWhiteLevelAwb	Upper limit for searching for white points during white point statistics. The default value is 0x3ac.
	u16MeteringBlackLevelAwb	Lower limit for searching for white points during white point statistics. The default value is 0x40.
	u16MeteringCrRefMaxAwb	Maximum R/G value of the white point area during white point statistics. The default value is 512.
	u16MeteringCbRefMaxAwb	Maximum B/G value of the white point area during white point statistics. The default value is 512.
	u16MeteringCrRefMinAwb	Minimum R/G value of the white point area during white point statistics. The default value is 128.
	u16MeteringCbRefMinAwb	Minimum B/G value of the white point area during white point statistics. The default value is 128.

3.4 Registering the AF Algorithm with the ISP Library

As shown in Figure 3-3, the AF algorithm is registered with the ISP library by calling HI_MPI_ISP_AfLibRegCallBack. For details, see the *HiISP Development Reference*.

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



The HiSilicon AF algorithm is not implemented and there is only a registration framework. The sample is similar to that of registering the AE algorithm.

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

Table 3-9 Members of the callback function for registering the AF algorithm with the ISP library

Member	Description
pfn_af_init	Pointer to the callback function for initializing the AF algorithm library
pfn_af_run	Pointer to the callback function for running the AF algorithm library
pfn_af_ctrl	Pointer to the callback function for controlling the internal status of the AF algorithm library
pfn_af_exit	Pointer to the callback function for destroying the AF algorithm library

NOTE

- pfn_af_init is called when HI_MPI_ISP_Init is called to initialize the AF algorithm library.
- pfn_af_ru is called when HI_MPI_ISP_Run is called to run the AF algorithm library and calculate the white balance gain and CCM.
- pfn_af_ctrl is used to change the internal status of the AF algorithm library.
- pfn_af_exit is called when HI_MPI_ISP_Exit is called to destroy the AF algorithm library.

Table 3-10 Members of the initialization parameter data structure ISP_AF_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 AF algorithm library.

Member	Description
s32Rsv	Reserved

Table 3-11 Members of the statistics data structure ISP_AF_INFO_S

Member	Description
u32FrameCnt	Total number of frames. The value range is [0, 0xFFFFFFFF].
pstAfStat	Pointer to AF statistics

Table 3-12 Members of the running result data structure ISP_AF_RESULT_S

Member	Description
s32Rsv	Reserved

NOTE

The peripheral interfaces such as the pulse width modulation (PWM) interface need to be called to implement AF. No result needs to be returned to the ISP.

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_AXX_Register interface on their own. In addition, the users must implement the HI_MPI_AXX_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 cmos.c File Modification

In 3A version, the **cmos.c** files of other versions are sorted out. For the description of the main structure, see 2 "Usage Guidance". This section provides some additional descriptions.

The parameters of the algorithms delivered along with the ISP firmware are sorted out and added to the ISP_CMOS_DEFAULT_S structure, for example, RGGB sequence, denoising, DRC, demosaic, and Gamma. Black level parameters are added to ISP_CMOS_BLACK_LEVEL_S because the black levels of some sensors dynamically change.

Parameters related to HiSilicon AE are sorted out and added to the AE_SENSOR_DEFAULT_S structure, for example, exposure duration, analog gain, digital gain, and anti-flicker. In addition, the expression mode is changed to floating-point precision mode. After the floating-point precision mode is used in unified manner, all codes allocated by exposure no longer appear in **cmos.c**. The precision is classified into two types, that is, multiplier precision and decibel precision. The multiplier precision increases in linear mode and the decibel precision increases in power series mode. For example, the 0.125 multiplier precision indicates that the minimum adjustable unit of the sensor is 1/8 times, and the 0.3 decibel precision indicates that the minimum adjustable unit is 0.3 db. The precision of the sensor can always be expressed in multiplier precision and decibel precision modes. For example, the gain of OV9712 is a (1 + b/16), and a can be 1, 2, 4, 8, or 16 and b can be a value from 1 to 15, you can regard that the analog gain uses decibel precision and the minimum adjustable unit is 6 db (2 times), the digital gain uses linear precision and the minimum adjustable unit is 0.0625 (1/16) times, and u32Again and u32Dgain in cmos_gains_update respectively use the units of 6 db and 0.0625 times.

Parameters related to HiSilicon AWB are sorted out and added to the AWB_SENSOR_DEFAULT_S structure, for example, white balance correction curve, static white balance gain, and saturation.

Other contents conduct some operations. For example, cmos_set_pixel_detect sets defect pixel correction, cmos_set_wdr_mode switches the WDR mode, cmos_fps_set sets the standard frame rate, and cmos_slow_framerate_set sets frame rate reduction.

If a user develops 3A, the AE and AWB parts in **cmos.c** also need to be developed, for example, exposure duration and gain precision of AE and parameters of AWB. The cmos_get_isp_default, cmos_get_isp_black_level, cmos_set_pixel_detect, and cmos_set_wdr_mode can be reused.

4.4 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.

4.5 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. 0x30000–0x3FFFF 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.