|  |  |  |
| --- | --- | --- |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | | |
| **Issue** | **01** | |
| **Date** | **2013-07-16** | |
|  | | |

|  |
| --- |
| Copyright © HiSilicon Technologies Co., Ltd. . 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  HI_logo, 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 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. |
| wanning | Alerts you to a medium or low risk hazard that could, if not avoided, result in moderate or minor injury. |
| 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 01 (2013-07-16)

This issue is the first official release.

Contents

[About This Document i](#_Toc361991284)

[1 Introduction to the Hi3518\_ISP\_3A 1](#_Toc361991285)

[1.1 Overview 1](#_Toc361991286)

[1.2 Function Description 1](#_Toc361991287)

[1.2.1 Design Methodology 1](#_Toc361991288)

[1.2.2 File Structure 2](#_Toc361991289)

[1.2.3 Development Mode 2](#_Toc361991290)

[1.2.4 Internal Process 4](#_Toc361991291)

[2 Usage Guidance 1](#_Toc361991292)

[2.1 Software Process 1](#_Toc361991293)

[2.2 Sensor Interconnection 5](#_Toc361991294)

[2.2.1 Registering Sensors with the ISP Library 5](#_Toc361991295)

[2.2.2 Registering Sensors with 3A Algorithm Libraries 10](#_Toc361991296)

[3 Developer Guidance 1](#_Toc361991297)

[3.1 Overview 1](#_Toc361991298)

[3.2 Registering the AE Algorithm with the ISP Library 1](#_Toc361991299)

[3.3 Registering the AWB Algorithm with the ISP Library 4](#_Toc361991300)

[3.4 Registering the AF Algorithm with the ISP Library 7](#_Toc361991301)

Figures

[**Figure 1-1** Design methodology of the ISP firmware 1](#_Toc361994974)

[**Figure 1-2** File structure of the ISP firmware 2](#_Toc361994975)

[**Figure 1-3** Internal process of the ISP firmware 4](#_Toc361994976)

[**Figure 1-4** Architecture of the ISP firmware 5](#_Toc361994977)

[**Figure 2-1** Workflow of the ISP firmware 2](#_Toc361994978)

[**Figure 2-2** Sensor adaptation diagram 5](#_Toc361994979)

[**Figure 2-3** Callback function for registering sensors with the ISP library 5](#_Toc361994980)

[**Figure 2-4** Callback function for registering sensors with the AE algorithm library 10](#_Toc361994981)

[**Figure 2-5** CallBack callback function for registering sensors with the AWB library 12](#_Toc361994982)

[**Figure 3-1** Callback function for registering the AE algorithm with the ISP library 1](#_Toc361994983)

[**Figure 3-2** Callback function for registering the AWB algorithm with the ISP library 4](#_Toc361994984)

[**Figure 3-3** Callback function for registering the AF algorithm with the ISP library 7](#_Toc361994985)

Tables

[**Table 2-1** Callback function for registering sensors with the ISP library 6](#_Toc361994956)

[**Table 2-2** ISP\_CMOS\_DEFAULT\_S members 6](#_Toc361994957)

[**Table 2-3** Callback functions for registering sensors with the AE algorithm library 11](#_Toc361994958)

[**Table 2-4** AE\_SENSOR\_DEFAULT\_S members 11](#_Toc361994959)

[**Table 2-5** Member of the HI\_MPI\_AWB\_SensorRegCallBackCallBack callback function for registering sensors with the AWB library 13](#_Toc361994960)

[**Table 2-6** AWB\_SENSOR\_DEFAULT\_S members 13](#_Toc361994961)

[**Table 3-1** Callback functions for registering the AE algorithm with the ISP library 2](#_Toc361994962)

[**Table 3-2** Members of the initialization parameter data structure ISP\_AE\_PARAM\_S 3](#_Toc361994963)

[**Table 3-3** Members of the statistics data structure ISP\_AE\_INFO\_S 3](#_Toc361994964)

[**Table 3-4** Members of the running result data structure ISP\_AE\_RESULT\_S 4](#_Toc361994965)

[**Table 3-5** Member of the callback function for registering the AWB algorithm with the ISP library 4](#_Toc361994966)

[**Table 3-6** Members of the initialization parameter data structure ISP\_AE\_PARAM\_S 5](#_Toc361994967)

[**Table 3-7** Members of the statistics data structure ISP\_AE\_INFO\_S 5](#_Toc361994968)

[**Table 3-8** Members of the running result data structure ISP\_AWB\_RESULT\_S 6](#_Toc361994969)

[**Table 3-9** Members of the callback function for registering the AF algorithm with the ISP library 7](#_Toc361994970)

[**Table 3-10** Members of the initialization parameter data structure ISP\_AF\_PARAM\_S 8](#_Toc361994971)

[**Table 3-11** Members of the statistics data structure ISP\_AF\_INFO\_S 8](#_Toc361994972)

[**Table 3-12** Members of the running result data structure ISP\_AF\_RESULT\_S 8](#_Toc361994973)

# Introduction to the Hi3518\_ISP\_3A

## 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.

## Function Description

### 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.

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.

### 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.

File structure of the ISP firmware



### 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.

#### 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.

#### 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.

#### 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*.

### 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.

Internal process of the ISP firmware



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

Architecture of the ISP firmware



# Usage Guidance

## 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.

Workflow of the ISP firmware



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;

## 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.

Sensor adaptation diagram



### 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*.

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\_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**.

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**.

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 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 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]. |
| 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.  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. |

### Registering Sensors with 3A Algorithm Libraries

#### 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*.

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**.

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**..

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) |
| 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 |

#### 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*.

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**.

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**.

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, 0xFFFF]. |
| au16LowCCM | Low color temperature CCM. The value range is [0, 0xFFFF]. |

#### 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.

# Developer Guidance

## 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.

## 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*.

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]

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);

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.

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.

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 |

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]. |

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 |
| 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]. |

## 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*.

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.

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.

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 |

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

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 **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. |

## 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*.

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.

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.

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. |
| s32Rsv | Reserved |

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 |

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.