



ISP_3A

开发指南

文档版本 02

发布日期 2017-11-15

版权所有 © 深圳市海思半导体有限公司 2016-2017。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

深圳市海思半导体有限公司

地址： 深圳市龙岗区坂田华为基地华为电气生产中心 邮编：518129

网址： <http://www.hisilicon.com>

客户服务电话： +86-755-28788858

客户服务传真： +86-755-28357515

客户服务邮箱： support@hisilicon.com



前 言

概述

本文档描述 ISP_3A 的功能、如何使用与开发。3A 功能包括 AE、AWB、AF。



说明

本文以 Hi3516CV300 为例，如未有特殊说明，Hi3516EV100 与 Hi3516CV300 完全一致。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3516C	V300
Hi3516E	V100


读者对象

本文档（本指南）主要适用于以下工程师：





- 技术支持工程师
- 软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。



符号	说明
 警告	表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
 注意	表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
 窍门	表示能帮助您解决某个问题或节省您的时间。
 说明	表示是正文的附加信息，是对正文的强调和补充。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 02 (2017-11-15)

第 2 次正式版本发布。

3.2、3.5.8 和 3.5.9 涉及修改

文档版本 01 (2017-08-30)

第 1 次正式版本发布。

3.2 小节，“统计信息 ISP_AE_INFO_S 的成员变量”涉及修改

3.4、3.5.7 和 3.5.11 小节涉及修改

文档版本 00B03 (2017-02-15)

第 3 次临时版本发布。

3.2、3.5.1、3.5.9 和 3.5.10 小节涉及修改。

文档版本 00B02 (2016-09-30)

第 2 次临时版本发布。

2.1、3.3、3.4 和 4.4 小节涉及修改。

文档版本 00B01 (2016-07-20)

第 1 次临时版本发布。



目 录

前 言.....	i
1 概述.....	1
1.1 概述.....	1
1.2 功能描述.....	1
1.2.1 设计思路	1
1.2.2 文件组织	2
1.2.3 开发模式	2
1.2.4 内部流程	3
2 使用者指南.....	5
2.1 软件流程.....	5
2.2 Sensor 对接.....	9
2.2.1 Sensor 注册 ISP 库.....	9
2.2.2 Sensor 注册 3A 算法库.....	11
3 开发者指南.....	15
3.1 概述.....	15
3.2 AE 算法注册 ISP 库.....	15
3.3 AWB 算法注册 ISP 库	18
3.4 开发用户 AF 算法.....	20
3.5 AF 统计信息使用说明	22
3.5.1 概述	22
3.5.2 输入图像的裁剪.....	23
3.5.3 Bayer 域配置.....	24
3.5.4 抑制光源对 FV 值的影响.....	24
3.5.5 Coring	25
3.5.6 统计模式配置.....	26
3.5.7 统计值的获取.....	28
3.5.8 FV 值计算	28
3.5.9 FV 计算参考代码	28
3.5.10 PQTools 滤波器设计工具使用说明	33



3.5.11 AF 同步回调.....	38
4 附录.....	43
4.1 注册函数的关系.....	43
4.2 扩展性的设计考虑.....	43
4.3 3A 架构的设计思路.....	44
4.4 外部寄存器的说明.....	44



插图目录

图 1-1 ISP firmware 设计思路	1
图 1-2 ISP firmware 文件组织	2
图 1-3 ISP firmware 内部流程	4
图 1-4 ISP firmware 软件结构	4
图 2-1 ISP firmware 使用流程	6
图 2-2 Sensor 适配示意图.....	9
图 2-3 Sensor 向 ISP 库注册的回调函数	9
图 2-4 Sensor 向 AE 库注册的回调函数.....	11
图 2-5 Sensor 向 AWB 库注册的回调函数	13
图 3-1 AE 算法向 ISP 库注册的回调函数	15
图 3-2 AWB 算法向 ISP 库注册的回调函数	18
图 3-3 统计信息参数示意图.....	20
图 3-4 AF 算法结构图.....	21
图 3-5 FV 曲线	22
图 3-6 AF 统计模块框图.....	23
图 3-7 AF 4 组滤波器.....	23
图 3-8 AF 输入图像裁剪.....	23
图 3-9 Bayer 域统计预处理	24
图 3-10 高亮统计值	24
图 3-11 Level Depend Gain 曲线	25
图 3-12 Coring 曲线	26
图 3-13 平方配置模式映射曲线图.....	27
图 3-14 滤波器后的图像.....	28
图 3-15 自动对焦参数仿真工具.....	34
图 3-16 通过截止频率获取滤波系数（窄带）	35



图 3-17 通过截止频率获取滤波系数（宽带）	36
图 3-18 参考组列表	37
图 3-19 参考组的曲线显示.....	37
图 3-20 选中参考组曲线高亮显示.....	38
图 3-21 AF 同步回调.....	39



表格目录

表 2-1 Sensor 向 ISP 库注册的回调函数	10
表 2-2 Sensor 向 AE 库注册的回调函数	12
表 2-3 Sensor 向 AWB 库注册的回调函数	14
表 3-1 AE 算法向 ISP 库注册的回调函数	16
表 3-2 统计信息 ISP_AE_INFO_S 的成员变量默认配置说明	17
表 3-3 AWB 算法向 ISP 库注册的回调函数	18



1 概述

1.1 概述

Hi3516CV300_ISP_3A 版本依赖于相应的 SDK 大版本，通过一系列数字图像处理算法完成对数字图像的效果处理。主要包含 Firmware 框架及海思 3A 库，Firmware 提供算法的基本框架，处理统计信息，驱动数字图像处理算法，并包含坏点校正、去噪、色彩增强、镜头阴影校正等处理。3A 库以注册的方式，添加到 Firmware 中，完成曝光、白平衡、色彩还原等处理。

1.2 功能描述

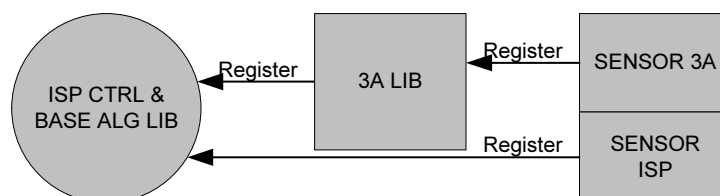
1.2.1 设计思路

ISP 的 Firmware 包含以下三部分：

- ISP 控制单元和基础算法单元，即 ISP 库；
- AE/AWB/AF 算法库；
- sensor 库。

Firmware 设计的基本思想是单独提供 3A 算法库，由 ISP 控制单元调度基础算法单元和 3A 算法库，同时 sensor 库分别向 ISP 库和 3A 算法库注册函数回调，以实现差异化的 sensor 适配。ISP firmware 设计思路如图 1-1 所示。

图1-1 ISP firmware 设计思路



不同的 sensor 都向 ISP 库和 3A 算法库注册控制函数，这些函数都以回调函数的形式存在。ISP 控制单元调度基础算法单元和 3A 算法库时，将通过这些回调函数获取初始化



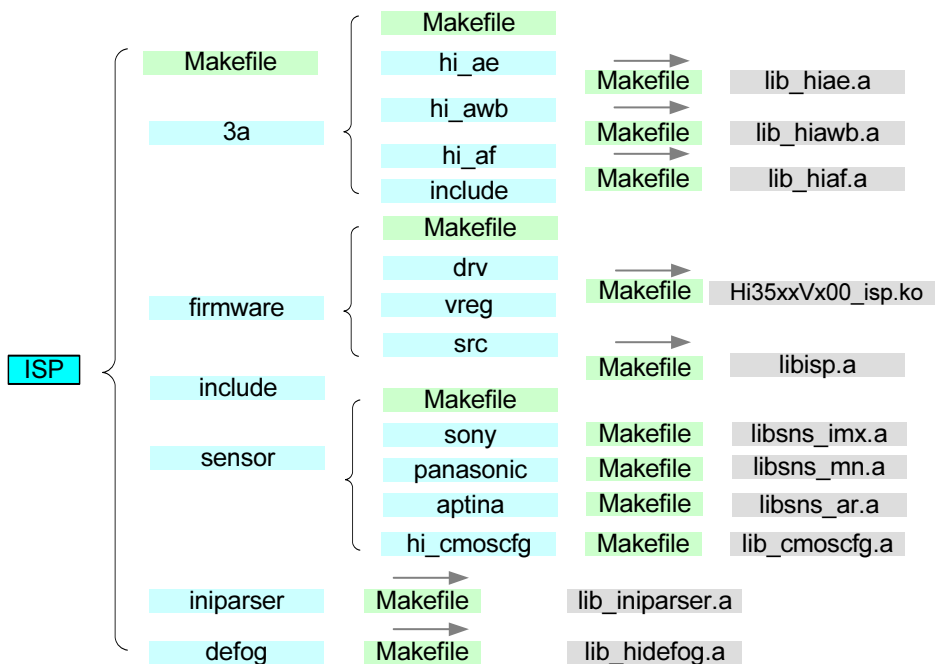
参数，并控制 sensor，如调节曝光时间、模拟增益、数字增益，控制 lens 步进聚焦或旋转光圈等。

1.2.2 文件组织

ISP Firmware 的文件组织结构如图 1-2 所示，ISP 库和 3A 库、sensor 库、iniparser 库、defog 库分别独立。Firmware 中的 drv 生成的驱动程序向用户态上报 ISP 中断，并以该中断驱动 Firmware 的 ISP 控制单元运转。ISP 控制单元从驱动程序中获取统计信息，并调度基础算法单元和 3A 算法库，最后通过驱动程序配置寄存器。

Src 文件夹中包含 ISP 控制单元和基础算法单元，编译后生成 libisp.a，即 ISP 库。3a 文件夹中包含 AE/AWB/AF 算法库，用户也可以基于统一的接口界面开发自己的 3a 算法。Sensor 文件夹中包含了各个 sensor 的驱动程序，该部分代码开源。hi_cmoscfg 文件夹中包含解析 ini 文件所需的公共程序，该部分代码开源。iniparser 文件夹包含 ini 解析函数库，也可用于其它应用开发。defog 文件对应去雾算法程序，该部分代码不开源。

图1-2 ISP firmware 文件组织



1.2.3 开发模式

SDK 中给出的形式支持用户的多种开发模式，用户可以使用海思的 3A 算法库，也可以根据 ISP 库提供的 3A 算法注册接口，实现自己的 3A 算法库开发，或者可以部分使用海思 3A 算法库，部分实现自己的 3A 算法库，例如 AE 使用 lib_hiae.a，AWB 使用自己的 3A 算法库。SDK 提供了灵活多变的支持方式。



1.2.3.1 使用海思 3A 算法库

用户需要根据 ISP 基础算法单元和海思 3A 算法库给出的 sensor 适配接口去适配不同的 sensor。Sensor 文件夹中包含两个主要文件：

- sensor_cmos.c
该文件中主要实现 ISP 需要的回调函数，这些回调函数中包含了 sensor 的适配算法，不同的 sensor 可能有所不同。
- sensor_ctrl.c
sensor 的底层控制驱动，主要实现 sensor 的读写和初始化动作。用户可以根据 sensor 的 datasheet 进行这两个文件的开发，必要的时候可以向 sensor 厂家寻求支持。

1.2.3.2 开发 3A 算法库

用户需要根据 ISP 基础算法单元给出的 sensor 适配接口去适配不同的 sensor，用户开发的 3A 算法库需要自定义数据接口和回调函数去适配和控制不同的 sensor。用户自行开发 3A 算法后，可以调用 mpi_isp.h 中的接口，不能调用 mpi_ae.h 和 mpi_awb.h 中的接口。



说明

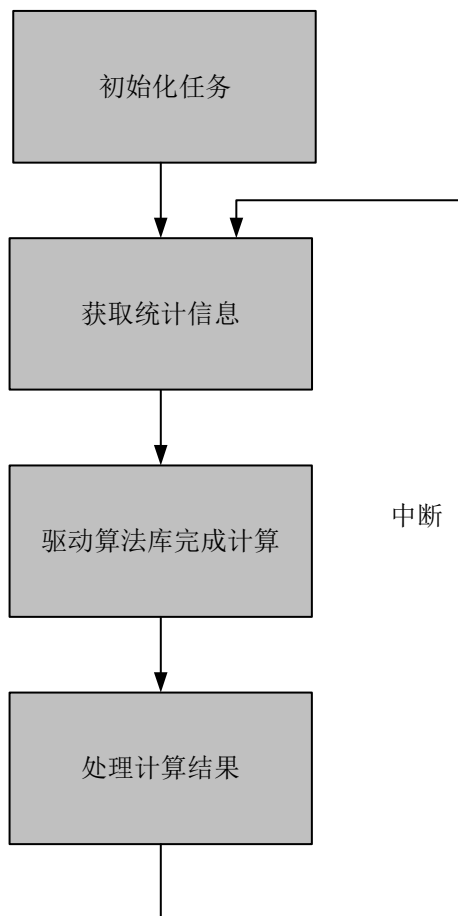
高级用户可以基于 Firmware 提供的统计信息进行自己的算法库开发，当然这需要对统计信息比较熟悉，同时具有算法开发能力。

1.2.4 内部流程

Firmware 内部流程，如图 1-3 所示。首先完成 Firmware 控制单元的初始化、基础算法单元的初始化、3A 算法库的初始化，包括调用 sensor 的回调获取 sensor 差异化的初始化参数。当初始化完成之后，Firmware 由中断驱动，每帧从内核态获取统计信息，并驱动基础算法单元和 3A 算法库完成计算，并反馈计算结果，配置 ISP 寄存器和 sensor 寄存器。

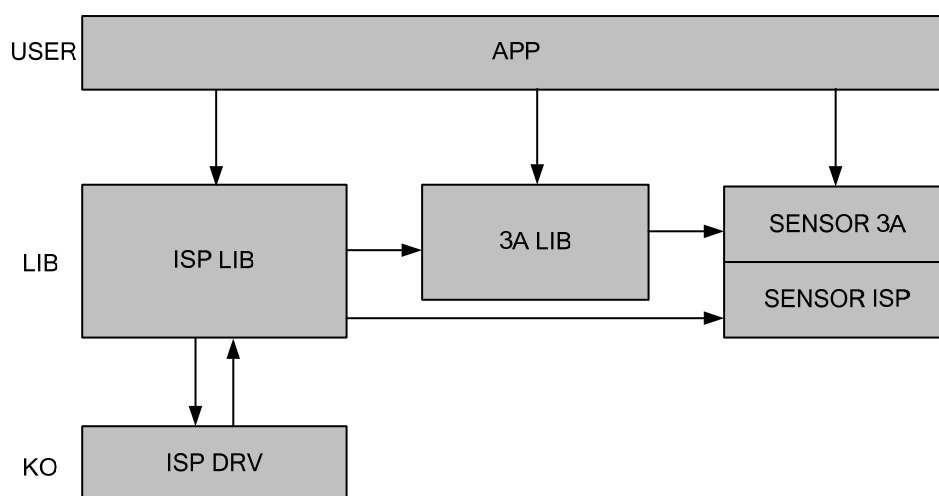
同时用户可以通过 ISP 的 MPI，控制和改变 Firmware 中包含的基础算法单元的内部数据和状态，定制自己的图像质量效果。如果用户使用海思提供的 3A 算法库，可以通过 3A 算法库的 MPI，改变 3A 算法库的内部数据和状态，调节曝光、白平衡和色彩还原。

图1-3 ISP firmware 内部流程



Firmware 的软件结构如图 1-4 所示。

图1-4 ISP firmware 软件结构





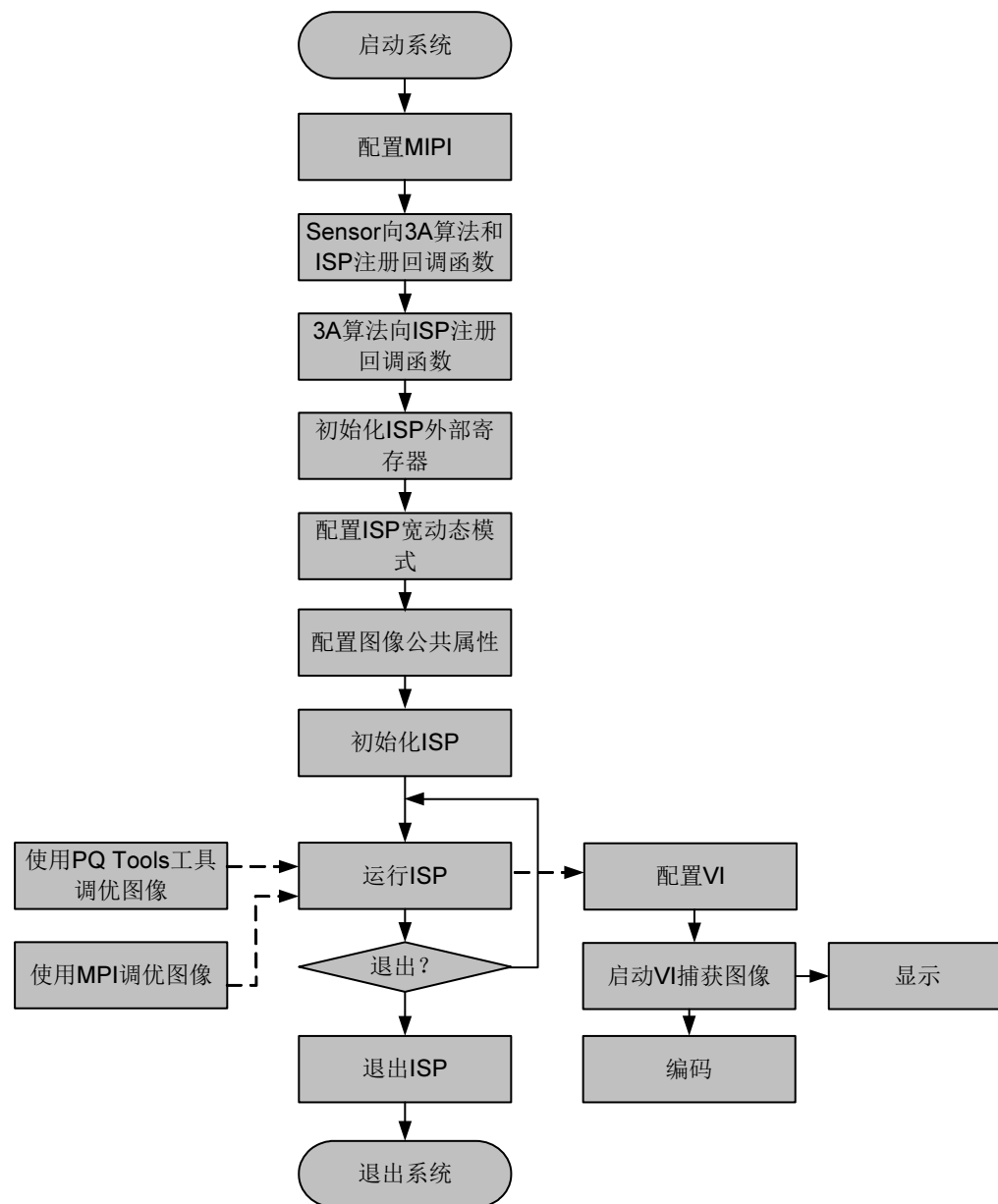
2 使用者指南

2.1 软件流程

ISP 作为图像前处理部分，需要和视频采集单元（VIU）协同工作。ISP 初始化和基本配置完成后，需要 VIU 进行接口时序匹配。一是为了匹配不同 sensor 的输入时序，二是为 ISP 配置正确的输入时序。待时序配置完成后，ISP 就可以启动 Run 来进行动态图像质量调节。此时输出的图像被 VIU 采集，进而送去显示或编码。软件使用流程如图 2-1 所示。

PQ Tools 工具主要完成在 PC 端进行动态图像质量调节，可以调节多个影响图像质量的因子，如去噪强度、色彩转换矩阵、饱和度等。

图2-1 ISP firmware 使用流程



如果用户调试好图像效果后，可以使用 PQ Tools 工具提供的配置文件保存功能进行配置参数保存。在下次启动时系统可以使用 PQ Tools 工具提供的配置文件加载功能加载已经调节好的图像参数。

代码示例：

```

HI_S32 s32Ret;
ALG_LIB_S stLib;
ISP_PUB_ATTR_S stPubAttr;
pthread_t isp_pid;
/* 注册sensor库 */
s32Ret = sensor_register_callback();
  
```



```
if (HI_SUCCESS != s32Ret)
{
    printf("register sensor failed!\n");
    return s32Ret;
}

/* 注册海思AE算法库 */
stLib.s32Id = 0;
strcpy(stLib.acLibName, HI_AE_LIB_NAME);
s32Ret = HI_MPI_AE_Register(IspDev, &stLib);
if (HI_SUCCESS != s32Ret)
{
    printf("register ae lib failed!\n");
    return s32Ret;
}

/* 注册海思AWB算法库 */
stLib.s32Id = 0;
strcpy(stLib.acLibName, HI_AWB_LIB_NAME);
s32Ret = HI_MPI_AWB_Register(IspDev, &stLib);
if (HI_SUCCESS != s32Ret)
{
    printf("register awb lib failed!\n");
    return s32Ret;
}

/* 注册海思AF算法库 */
stLib.s32Id = 0;
strcpy(stLib.acLibName, HI_AF_LIB_NAME);
s32Ret = HI_MPI_AF_Register(IspDev, &stLib);
if (HI_SUCCESS != s32Ret)
{
    printf("register af lib failed!\n");
    return s32Ret;
}

/* 初始化ISP外部寄存器 */
s32Ret = HI_MPI_ISP_MemInit(IspDev);
if (s32Ret != HI_SUCCESS)
{
    printf("%s: HI_MPI_ISP_Init failed!\n", __FUNCTION__);
    return s32Ret;
}
```




```
/* 配置ISP宽动态模式 */
ISP_WDR_MODE_S stWdrMode;
stWdrMode.enWDRMode = enWDRMode;
s32Ret = HI_MPI_ISP_SetWDRMode(0, &stWdrMode);
if (HI_SUCCESS != s32Ret)
{
    printf("start ISP WDR failed!\n");
    return s32Ret;
}

/* 配置图像公共属性 */
s32Ret = HI_MPI_ISP_SetPubAttr(IspDev, &stPubAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("%s: HI_MPI_ISP_SetPubAttr failed with %#x!\n", __FUNCTION__,
        s32Ret);
    return s32Ret;
}

/* 初始化ISP Firmware */
s32Ret = HI_MPI_ISP_Init(IspDev);
if (HI_SUCCESS != s32Ret)
{
    printf("isp init failed!\n");
    return s32Ret;
}

/* HI_MPI_ISP_Run单独启动线程运行 */
if (0 != pthread_create(&isp_pid, 0, ISP_Run, NULL))
{
    printf("create isp running thread failed!\n");
    return HI_FAILURE;
}

/* 启动VI/VO等业务 */
.....

/* 停止VI/VO等业务 */
s32Ret = HI_MPI_ISP_Exit(IspDev);
if (HI_SUCCESS != s32Ret)
{
    printf("isp exit failed!\n");
    return s32Ret;
}
```

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

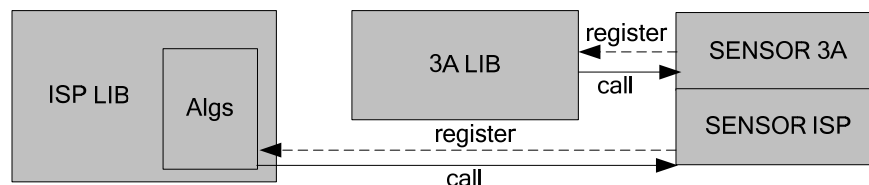
说明

AE 库有用到标准 C 库的数学库，请使用者在 Makefile 中增加 `-lm` 编译条件。

2.2 Sensor 对接

Sensor 库主要是为了提供差异化的 sensor 适配，里面的内容可以分为两部分：Sensor 向 ISP 库注册的差异化适配，这些差异化适配主要由 Firmware 中的基础算法单元决定；Sensor 向海思 3A 库注册的差异化适配。Sensor 的适配包括算法的初始化默认值，及 sensor 控制接口，sensor 的适配是通过接口回调的形式注册给 ISP 库和 3A 算法库。图 2-2 表示了 Sensor 与 ISP 库和 3A 算法库之间的关系。

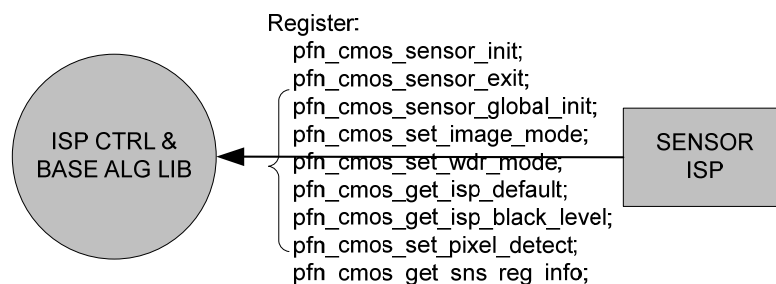
图2-2 Sensor 适配示意图



2.2.1 Sensor 注册 ISP 库

Sensor 注册 ISP 库调用 `HI_MPI_ISP_SensorRegCallBack`，如图 2-3 所示，详细说明参见《HiISP 开发参考》。

图2-3 Sensor 向 ISP 库注册的回调函数



【举例】

```
ISP_SENSOR_REGISTER_S stIspRegister;
ISP_SENSOR_EXP_FUNC_S *pstSensorExpFunc = &stIspRegister.stSnsExp;
memset(pstSensorExpFunc, 0, sizeof(ISP_SENSOR_EXP_FUNC_S));
pstSensorExpFunc->pfn_cmos_sensor_init = sensor_init;
pstSensorExpFunc->pfn_cmos_sensor_exit = sensor_exit;
pstSensorExpFunc->pfn_cmos_sensor_global_init = sensor_global_init;
```



```
pstSensorExpFunc->pfn_cmos_set_image_mode = cmos_set_image_mode;
pstSensorExpFunc->pfn_cmos_set_wdr_mode = cmos_set_wdr_mode;
pstSensorExpFunc->pfn_cmos_get_isp_default = cmos_get_isp_default;
pstSensorExpFunc->pfn_cmos_get_isp_black_level = cmos_get_isp_black_level;
pstSensorExpFunc->pfn_cmos_set_pixel_detect = cmos_set_pixel_detect;
pstSensorExpFunc->pfn_cmos_get_sns_reg_info = cmos_get_sns_regs_info;

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

需要在 xxx_cmos.c 中实现以下回调函数：

表2-1 Sensor 向 ISP 库注册的回调函数

成员名称	描述
pfn_cmos_sensor_init	初始化 sensor 的回调函数指针。
pfn_cmos_sensor_exit	退出 sensor 的回调函数指针。
pfn_cmos_sensor_global_init	初始化全局变量的回调函数指针。
pfn_cmos_set_image_mode	设置图像模式的回调函数指针。
pfn_cmos_set_wdr_mode	设置 wdr 模式和线性模式切换的回调函数指针。
pfn_cmos_get_isp_default	获取 ISP 基础算法的初始值的回调函数指针。
pfn_cmos_get_isp_black_level	获取 sensor 的黑电平值的回调函数指针。
pfn_cmos_set_pixel_detect	设置坏点校正开关的回调函数指针。
pfn_cmos_get_sns_reg_info	设置 sensor 曝光与增益延时的回调函数指针。



说明

如果回调函数暂不实现，可以实现空函数，或者将回调函数指针置为 NULL 即可。

一般情况下，cmos sensor 曝光时间相关寄存器从配置到生效的时间延迟最大。若能够在消隐区完成对 sensor 的配置，sensor 曝光时间通常会在配置后的第 2 帧生效，增益会在配置后的第 1 帧生效。pfn_cmos_get_sns_reg_info 用于保证 AE 计算出来的每帧 sensor 曝光时间和增益配置同步以避免闪烁现象。pfn_cmos_get_sns_reg_info 调用参数 ISP_SNS_REGS_INFO_S 支持 I2C 和 SPI 接口，成员变量具体含义请参考《HiISP 开发参考》，其中 u8Cfg2ValidDelayMax 可用于保证 ISP 寄存器与 sensor 寄存器同步生效，如 ISPDgain 和曝光比。

- 若 sensor 曝光时间在配置后的第 2 帧生效，增益在配置后的第 1 帧生效，那么 u8Cfg2ValidDelayMax 需设置为 2，表示所有 sensor 寄存器从配置到生效延迟的帧数最大值为 2，曝光时间的 u8DelayFrmNum 需设置为 0 表示曝光时间不需延迟配置，增益的 u8DelayFrmNum 需设置为 1 表示增益需延迟 1 帧配置，如此正好保证配置下去后 AE 计算出来的结果是同时生效的；
- 若 sensor 曝光时间和增益都在配置后的第 2 帧生效，则 u8Cfg2ValidDelayMax 需设置为 2，曝光时间和增益的 u8DelayFrmNum 都需设置为 0。除了曝光时间和增益外，可以通过增加变量个数，同步配置 sensor 每帧最大曝光时间 VMAX（以曝光行数为单位）或每行最大曝光时间 HMAX（以每行曝光对应的时钟个数为单位），如此在切换帧率时就不会出现闪烁现象，VMAX 或 HMAX 的 u8DelayFrmNum 一般与曝光时间的 u8DelayFrmNum 相同。对于不同 sensor，具体的参数配置需参考 sensor datasheet。在检验曝光时间与增益是否同步时，可以打开抗闪，如果不同步在抗闪时间改变时容易出现画面闪烁。

ISP_CMOS_DEFAULT_S 和 ISP_SNS_REGS_INFO_S 的成员变量

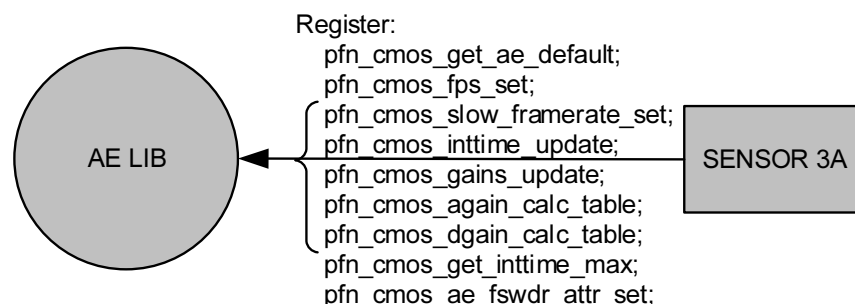
此数据结构请参考 Hi3516CV300 的《HiISP 开发参考》文档。

2.2.2 Sensor 注册 3A 算法库

2.2.2.1 Sensor 注册 AE 算法库

Sensor 注册 AE 算法库调用 HI_MPI_AE_SensorRegCallBack，如图 2-4 所示，详细说明参见《HiISP 开发参考》。

图2-4 Sensor 向 AE 库注册的回调函数



【举例】

```

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;
pstExpFuncs->pfn_cmos_again_calc_table  = cmos_again_calc_table;
pstExpFuncs->pfn_cmos_dgain_calc_table  = cmos_dgain_calc_table;
pstExpFuncs->pfn_cmos_get_inttime_max   = cmos_get_inttime_max;
pstExpFuncs->pfn_cmos_ae_fswdr_attr_set = cmos_ae_fswdr_attr_set;

stLib.s32Id = 0;
strcpy(stLib.acLibName, HI_AE_LIB_NAME);
s32Ret = HI_MPI_AE_SensorRegCallBack(IspDev, &stLib, IMX290_ID,
&stAeRegister);
if (s32Ret)
{
    printf("sensor register callback function to ae lib failed!\n");
    return s32Ret;
}
```

需要在 xxx_cmos.c 中实现以下回调函数：

表2-2 Sensor 向 AE 库注册的回调函数

成员名称	描述
pfn_cmos_get_ae_default	获取 AE 算法库的初始值的回调函数指针。
pfn_cmos_fps_set	设置 sensor 的帧率的回调函数指针。
pfn_cmos_slow_framerate_set	降低 sensor 的帧率的回调函数指针。
pfn_cmos_inttime_update	设置 sensor 的曝光时间的回调函数指针。
pfn_cmos_gains_update	设置 sensor 的模拟增益和数字增益的回调函数指针。
pfn_cmos_again_calc_table	查表方式计算 AE 模拟增益的回调函数指针。
pfn_cmos_dgain_calc_table	查表方式计算 AE 数字增益的回调函数指针。
pfn_cmos_get_inttime_max	WDR 模式下，计算各帧最大曝光时间和最小曝光时间的回调函数指针。
pfn_cmos_ae_fswdr_attr_set	2to1LineWDR 模式下，设置长帧模式。

说明

如果回调函数暂不实现，可以实现空函数，或者将回调函数指针置为 NULL 即可。

多帧合成 WDR 模式，需要调用 `pfn_cmos_get_inttime_max` 计算各帧的最大/最小曝光时间，AE 算法根据最大/最小曝光时间限制和曝光比计算得到各帧的曝光时间和增益，其余流程与线性模式一致。为了保证多帧合成 WDR 模式在正常场景的效果，海思 AE 算法可以根据直方图统计信息计算实现自动曝光比，在正常场景时设置曝光比为 1:1，如此 WDR 融合时只采用一帧数据，可以避免正常场景出现运动拖尾，同时也可减弱正常室内场景的工频闪现象。

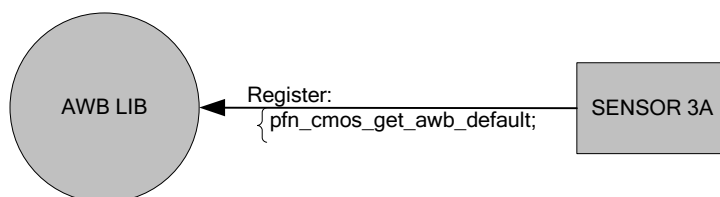
AE_SENSOR_DEFAULT_S 的成员变量

此数据结构请参考 Hi3516CV300 的《HiISP 开发参考》文档。

2.2.2.2 Sensor 注册 AWB 算法库

Sensor 注册 AWB 算法库调用 `HI_MPI_AWB_SensorRegCallBack`，如图 2-5 所示，详细说明参见《HiISP 开发参考》。

图2-5 Sensor 向 AWB 库注册的回调函数



【举例】

```
ALG_LIB_S stLib;
AWB_SENSOR_REGISTER_S stAwbRegister;
AWB_SENSOR_EXP_FUNC_S *pstExpFuncs = &stAwbRegister.stSnsExp;
memset(pstExpFuncs, 0, sizeof(AWB_SENSOR_EXP_FUNC_S));
pstExpFuncs->pfn_cmos_get_awb_default = cmos_get_awb_default;

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

需要在 `xxx_cmos.c` 中实现以下回调函数：



表2-3 Sensor 向 AWB 库注册的回调函数

成员名称	描述
pfn_cmos_get_awb_default	获取 AWB 算法库的初始值的回调函数指针。



说明

如果回调函数暂不实现，可以实现空函数，或者将回调函数指针置为 NULL 即可。

AWB_SENSOR_DEFAULT_S 的成员变量

此数据结构请参考 Hi3516CV300 的《HiISP 开发参考》文档。

2.2.2.3 Sensor 注册 AF 算法库

Sensor 注册 AF 算法库调用 HI_MPI_AF_SensorRegCallBack，AF 库暂未实现。

3 开发者指南

3.1 概述

用户可以基于 Firmware 框架开发定制 3A 库，并注册到 Firmware 中，Firmware 将会在中断的驱动下，获取每帧的统计信息，运行算法库，配置 ISP 寄存器。

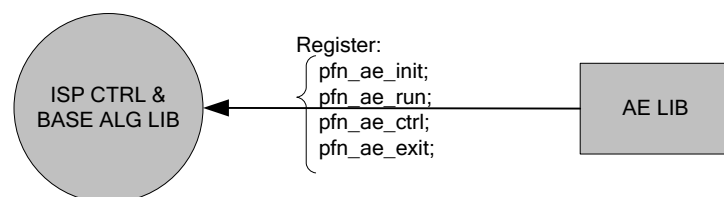
Sensor 注册到 ISP 库，以实现差异化适配 Firmware 中的基础算法单元的部分，仍需要参考使用者指南部分的内容实现，以保证 Firmware 中的坏点校正、去噪、色彩增强、镜头阴影校正等处理算法正常运行。Sensor 注册到 3A 算法库的部分，请用户根据自己开发定制的 3A 库，自行定义数据结构，实现 Sensor 曝光控制等。

如果用户只是使用海思 3A 算法库，并不自己开发 3A 算法库，可以忽略此章节内容。

3.2 AE 算法注册 ISP 库

AE 算法注册 ISP 库调用 HI_MPI_ISP_AeLibRegCallBack，如图 3-1 所示，详细说明参见《HiISP 开发参考》。

图3-1 AE 算法向 ISP 库注册的回调函数



海思 AE 算法实现了一个 HI_MPI_AE_Register 的注册函数，在这个函数中调用 ISP 提供的 HI_MPI_ISP_AeLibRegCallBack 回调接口，用户调用注册函数以实现向 ISP 注册 AE 算法，示例如下：

【举例】

```
/* 实现注册函数 */
```

```
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);

/* 调用钩子函数 */

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");
}
```

用户需要在自开发定制的 AE 库中实现以下回调函数：

表3-1 AE 算法向 ISP 库注册的回调函数

成员名称	描述
pfn_ae_init	初始化 AE 的回调函数指针。
pfn_ae_run	运行 AE 的回调函数指针。
pfn_ae_ctrl	控制 AE 内部状态的回调函数指针。
pfn_ae_exit	销毁 AE 的回调函数指针。

说明

- 调用 HI_MPI_ISP_Init 时将调用 pfn_ae_init 回调函数，以初始化 AE 算法库。
- 调用 HI_MPI_ISP_Run 时将调用 pfn_ae_run 回调函数，以运行 AE 算法库，计算得到 sensor 的曝光时间和增益、ISP 的数字增益。
- pfn_ae_ctrl 回调函数的目的是改变算法库内部状态。运行时 Firmware 会隐式调用 pfn_ae_ctrl 回调函数，通知 AE 算法库切换 WDR 和线性模式、设置 FPS。

当前 Firmware 定义的 ctrl 命令有：

```
typedef enum hiISP_CTRL_CMD_E
{
    ISP_WDR_MODE_SET = 8000,
    ISP_AE_FPS_BASE_SET,
    ISP_AWB_ISO_SET, /* set iso, change saturation when iso change
    */
    ...
    ISP_CTRL_CMD_BUTT,
} ISP_CTRL_CMD_E;
```



- 调用 HI_MPI_ISP_Exit 时将调用 pfn_ae_exit 回调函数，以销毁 AE 算法库。

初始化参数 ISP_AE_PARAM_S 的成员变量

此数据结构请参考 Hi3516CV300 的《HiISP 开发参考》文档。

统计信息 ISP_AE_INFO_S 的成员变量

此数据结构请参考 Hi3516CV300 的《HiISP 开发参考》文档。

说明

- pstAeStat3 表示全局 1024 段直方图统计信息。该统计信息是取输入数据流中的高 10bit 数据统计得到的，每个 bin 中数据表示该灰度值对应的像素个数。1024 个 bin 的数据之和即为参与统计的像素点个数。目前，海思 AE 算法默认只用了 Gr 通道的统计信息，在大面积红色时，会采用 R 和 Gb 通道的统计信息，在大面积蓝色时，会采用 B 和 Gr 通道的统计信息。1024 段直方图会受到分区间权重的影响。
- pstAeStat4 表示全局 R/Gr/Gb/B 4 分量的归一化均值，取值范围[0, 0xFFFF]。若统计数据是 12bit，0xFFFF 即表示某分量的均值为 12bit 的最大值 4095。全局 4 分量平均值会受分区间权重影响。在 WDR 模式下，默认开方使能，统计精度为 11bit，低 5bit 值为 0，因此最大值为 0xFFE0。
- pstAeStat5 表示 15*17 区间每个区间 R/Gr/Gb/B 4 分量的归一化均值，取值范围[0, 0xFFFF]，意义与全局 4 分量平均值相同。在 WDR 模式下，统计精度为 11bit，低 5bit 值为 0，因此最大值为 0xFFE0。
- pstAeStat6 表示多帧合成 WDR 模式数据合成前通道 1/2/3/4 路的全局 256 段直方图统计信息，该直方图会受分区间权重影响。Hi3516CV300 仅支持通道 1/2，直方图默认已减黑电平。

表3-2 统计信息 ISP_AE_INFO_S 的成员变量默认配置说明

成员名称	主要统计信息	默认位置	黑电平	权重表影响
pstAeStat3	WDR 合成后 1024 段直方图	WB 之后	已减	是
pstAeStat4	WDR 合成后全局均值	WB 之后	已减	是
pstAeStat5	WDR 合成后分块均值	WB 之后	已减	否
pstAeStat6	WDR 合成前 256 段直方图	WDR 合成前	已减	是



说明

- 表 3-2 中说明仅在 ISP 默认配置下生效，实际统计信息会受到黑电平配置及 AE 统计信息位置影响。
- 线性模式推荐使用 WDR 合成后 1024 段直方图。WDR 模式推荐使用 WDR 合成前 256 段直方图，或者使用 WDR 合成后开方模式的 1024 段直方图。

运算结果 ISP_AE_RESULT_S 的成员变量

此数据结构请参考 Hi3516CV300 的《HiISP 开发参考》文档。

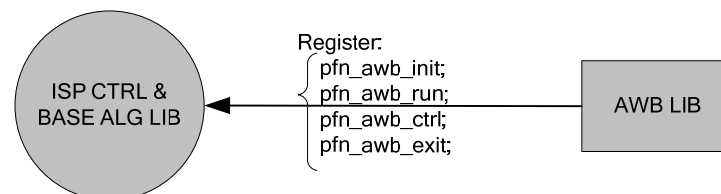
说明

- 将曝光时间 au32IntTime 由行转换成 us 时，可以通过 cmos.c 中的 u32LinesPer500ms 进行，转换关系如： $au32IntTime[0] = (((HI_U64)au32IntTimeRst[0] * 1024 - u32Offset) * 500000 / pstAeSnsDft -> u32LinesPer500ms) >> 10$
- 上式中 au32IntTimeRst[0] 是以行数为单位的曝光时间，u32Offset=f32Offset * 1024，f32Offset 即为曝光时间的偏移量，详见《HiISP 开发参考》中 AE_ACCURACY_S 部分描述。

3.3 AWB 算法注册 ISP 库

AWB 算法注册 ISP 库调用 HI_MPI_ISP_AwbLibRegCallBack，如图 3-2 所示，详细说明参见《HiISP 开发参考》。

图3-2 AWB 算法向 ISP 库注册的回调函数



海思 AWB 算法实现了一个 HI_MPI_AWB_Register 的注册函数，在这个函数中调用 ISP 提供的 HI_MPI_ISP_AwbLibRegCallBack 回调接口，用户调用注册函数以实现向 ISP 注册 AWB 算法，示例和 AE 算法库注册类似。

用户需要在自开发定制的 AWB 库中实现以下回调函数：

表3-3 AWB 算法向 ISP 库注册的回调函数

成员名称	描述
pfn_awb_init	初始化 AWB 的回调函数指针。
pfn_awb_run	运行 AWB 的回调函数指针。
pfn_awb_ctrl	控制 AWB 内部状态的回调函数指针。



pfn_awb_exit	销毁 AWB 的回调函数指针。
--------------	-----------------

说明

- 调用 HI_MPI_ISP_Init 时将调用 pfn_awb_init 回调函数，以初始化 AWB 算法库。
- 调用 HI_MPI_ISP_Run 时将调用 pfn_awb_run 回调函数，以运行 AWB 算法库，计算得到白平衡增益、色彩校正矩阵。

pfn_awb_ctrl 回调函数的目的是改变算法库内部状态。运行时 Firmware 会隐式调用 pfn_awb_ctrl 回调函数，通知 AWB 算法库切换 WDR 和线性模式、设置 ISO。设置 ISO 的目的是为了实现 ISO 与饱和度的联动，增益大时色度噪声也会比较大，所以需要调节饱和度。当前 Firmware 定义的 ctrl 命令有：

```
typedef enum hiISP_CTRL_CMD_E
{
    ISP_WDR_MODE_SET = 8000,
    ISP_PROC_WRITE,
    ISP_AE_FPS_BASE_SET,
    ISP_AWB_ISO_SET,
    ISP_CHANGE_IMAGE_MODE_SET,
    ISP_UPDATE_INFO_GET,
    ISP_AWB_INTTIME_SET,
    ISP_BAS_MODE_SET,
    ISP_AWB_PIRIS_SET,
    ISP_CTRL_CMD_BUTT,
} ISP_CTRL_CMD_E;
```

- 调用 HI_MPI_ISP_Exit 时将调用 pfn_awb_exit 回调函数，以销毁 AWB 算法库。

初始化参数 ISP_AWB_PARAM_S 的成员变量

此数据结构请参考 Hi3516CV300 的《HiISP 开发参考》文档。

统计信息 ISP_AWB_INFO_S 的成员变量

此数据结构请参考 Hi3516CV300 的《HiISP 开发参考》文档。

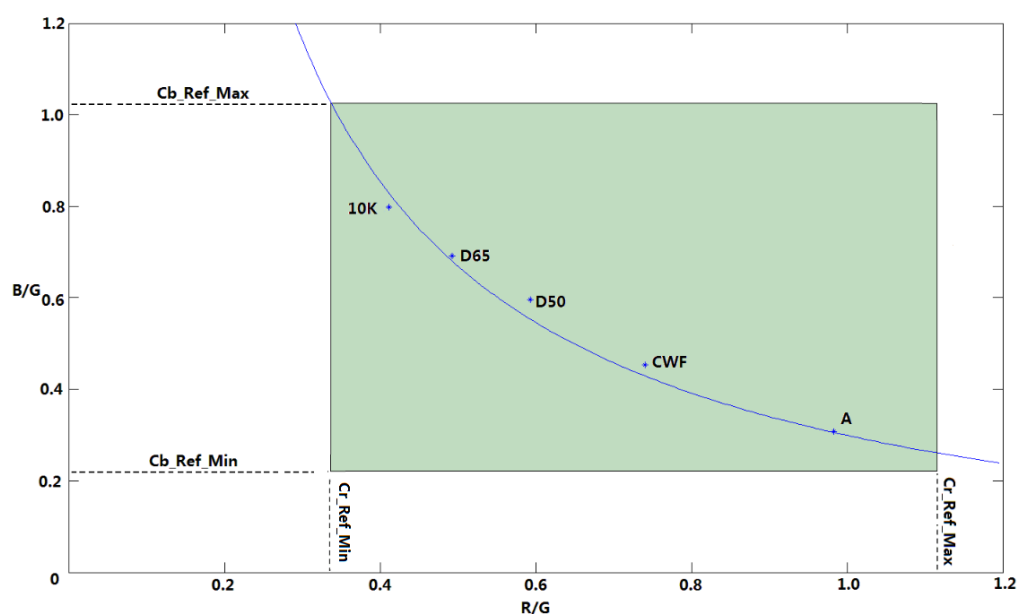
运算结果 ISP_AWB_RESULT_S 的成员变量

此数据结构请参考 Hi3516CV300 的《HiISP 开发参考》文档。

说明

- 无论是线性模式或 WDR 模式，Hi3516CV300 Bayer 域统计输出是线性的，不需要做特殊处理。
- 统计信息异常：为了避免统计信息异常对算法的影响，建议白点个数满足一定条件的统计信息参与计算。
- AWB 参数生效时间点：下一帧生效。
- 建议 AWB 统计参数根据 sensor 的光谱特性和产品支持的色温范围做调整，以达到更好的效果。
- Hi3516CV300 AWB 统计输出 16bit 精度的 RGB 均值，均值已减掉黑电平。

图3-3 统计信息参数示意图



3.4 开发用户 AF 算法

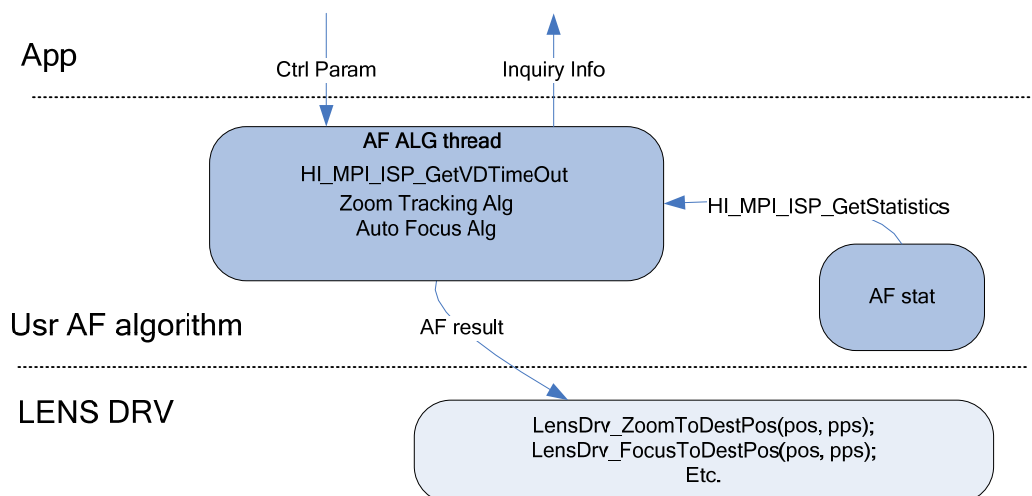
用户开发 AF 算法时不用关注 lib_hiaf.a 和 lib_hiaf.so 库，因为 AF 算法注册 ISP 库（通过 lib_hiaf.a 和 lib_hiaf.so 库实现）的方式实时性不好。

说明

HI_MPI_ISP_GetVDEndTimeOut 和 HI_MPI_ISP_GetStatistics 请参考《HiISP 开发参考》。

用户开发的 AF 算法可以参考图 3-4 所示的结构，AF 算法在一个 thread 内运行，使用 HI_MPI_ISP_GetVDEndTimeOut 接口将算法同步在 VD 下运行。通过调用 HI_MPI_ISP_GetStatistics 来获取 AF 相关的统计信息，相关算法参考统计信息完成目标 zoom 和 focus 位置的计算，AF 算法输出 result，包括 lens 的位置信息和速度。调用 Lens Driver 将镜头镜片驱动到设定的位置即完成当前帧的操作。AF Alg 向上提供控制接口和查询接口，方便用户完成一键聚焦，变焦，手动聚焦，查询算法状态等操作。

图3-4 AF 算法结构图



【举例】

如果用户使用的是 Linux 系统，推荐将算法放在 user space 完成，lens driver 放在 kernel space，ISP 驱动提供同步回调功能，参考 3.5.11 “AF 同步回调” 章节。

```

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

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

    // User Auto Focus Alg Source
    UsrAfAlgRun();

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

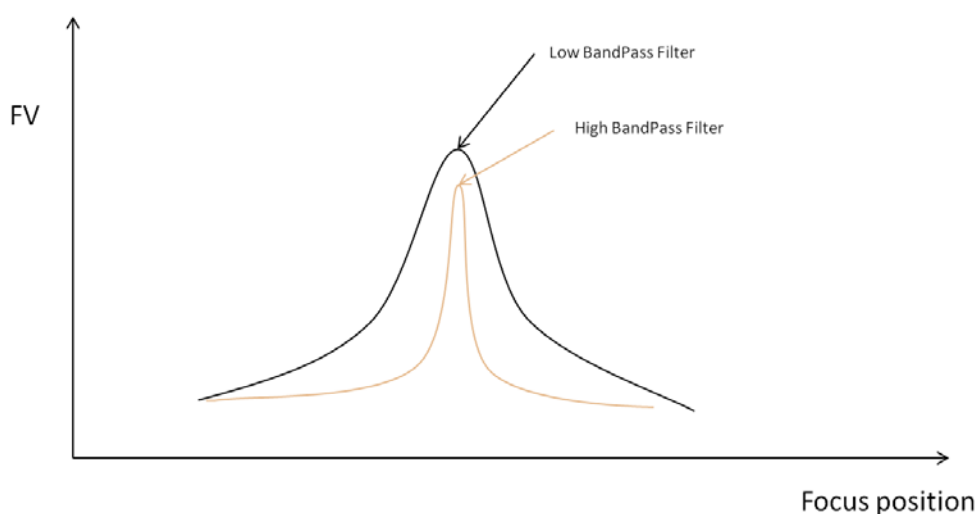
```

3.5 AF 统计信息使用说明

3.5.1 概述

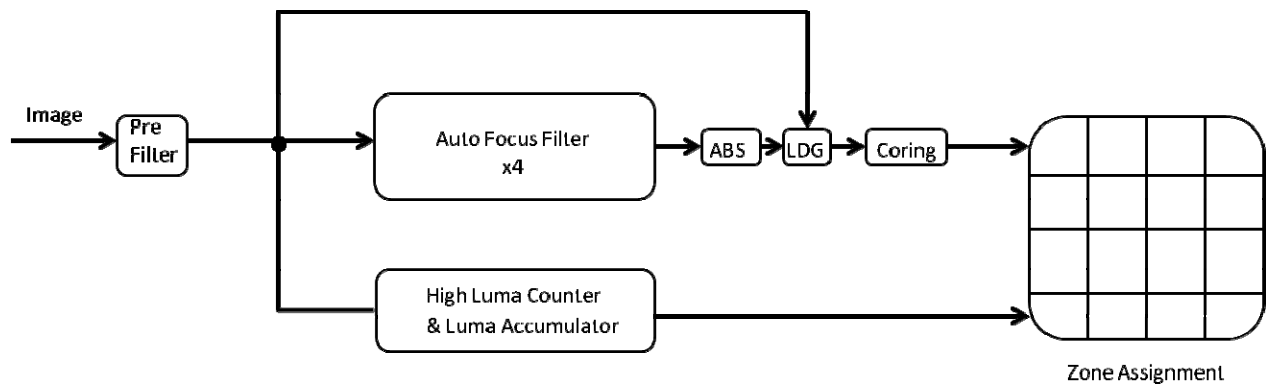
被动式自动对焦一般是通过分析图像特征得出图像清晰度值 FV(Focus Value)，通过驱动对焦马达调节焦点到最佳位置，如图 3-5，图像越清晰 FV 值越大，Peak 点对应聚焦清晰点。获取图像清晰度算法有多种，如灰度梯度法，高频分量法等，Hi3516CV300 采用高频分量法来计算 FV，即图像越清晰的时候高频部分幅值越大，将图像通过高通滤波器便可以得到高频分量，Hi3516CV300 一共提供四个滤波器和亮度信息，分别为水平方向滤波 H1，H2，垂直方向滤波 V1，V2，以及 Y 和高亮计数器 HLCnt。FV 曲线如图 3-5 所示。

图3-5 FV 曲线



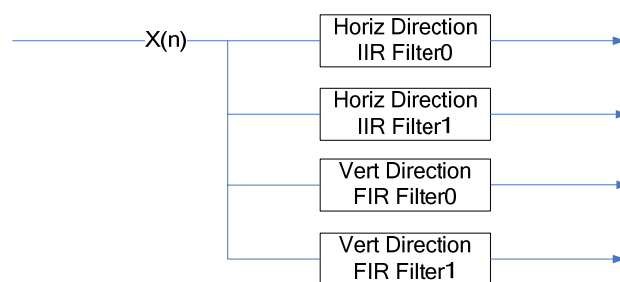
Hi3516CV300 提供 Bayer 域或者 YUV 域进行统计，配置 MPI 中的 enStatisticsPos 字段即可以进行统计位置的调整。WDR 模式下，非宽动态场景的 Bayer 域（即 RAW 域）统计信息无明显变化趋势，推荐使用 YUV 域统计信息。首先对图像进行预处理滤波，去除椒盐噪声对统计值的干扰，用户可以设置 stPreFltCfg.bEn 来使能 PreFilter。配置 stPreFltCfg.u16strength 来控制滤波强度。在经过水平垂直四组滤波器后会对输出做 Level Depend Gain 和 Coring，关于这两个模块将在后面的章节详述。模块框图如图 3-6 所示。

图3-6 AF 统计模块框图



Hi3516CV300 水平垂直滤波器分别采用 IIR 和 FIR 实现如[图 3-7](#) 所示。

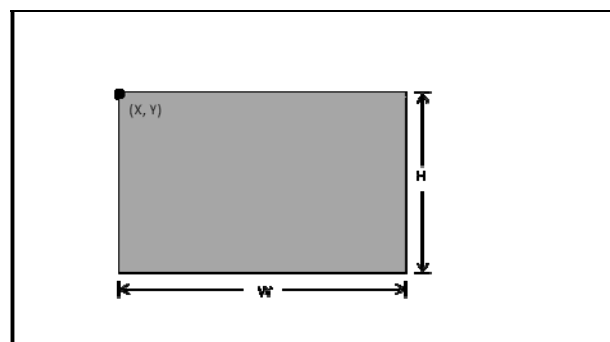
图3-7 AF 4 组滤波器



3.5.2 输入图像的裁剪

Hi3516CV300 AF 模块支持对输入图像的裁剪，如[图 3-8](#) 所示，用户可以根据实际需求配置 stCrop 中 X, Y, W, H 四个参数对指定区域进行统计。

图3-8 AF 输入图像裁剪

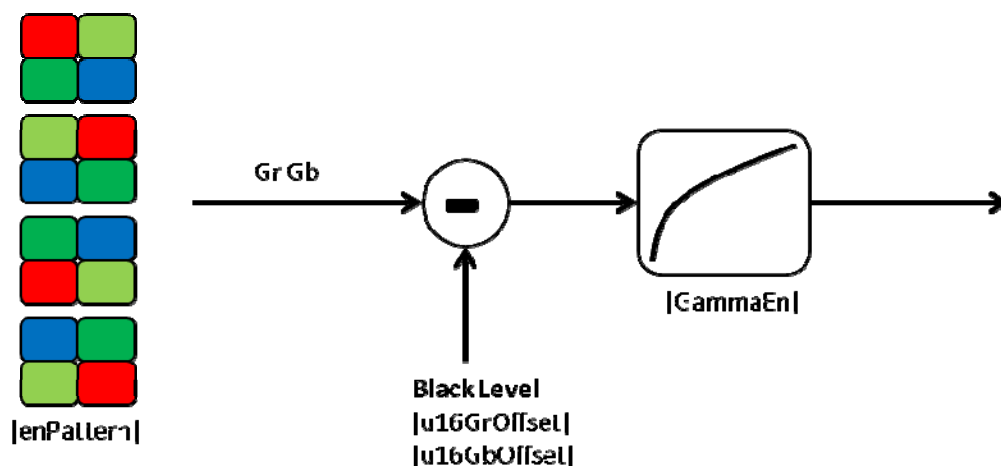


3.5.3 Bayer 域配置

如果用户将统计位置调整到 Bayer 域则需要配置以下几点。

- 配置 enPattern 来指定 bayer 数据的 pattern。Hi3516CV300 支持如图 3-9 所示的 4 种 pattern。
- 配置 AF black level 来去掉黑电平。因统计模块只对 G 通道做统计，因此只需要配置 u16GrOffset 和 u16GbOffset 即可。
- 配置 AF 模块 Gamma 使能，raw 上的线性数据需要做 Gamma 处理。

图3-9 Bayer 域统计预处理



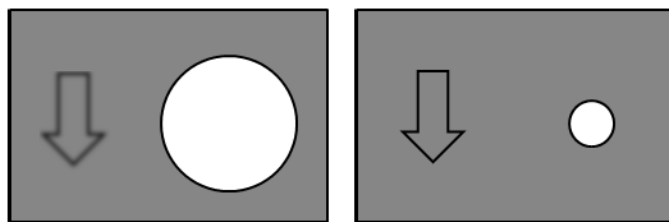
3.5.4 抑制光源对 FV 值的影响

因为 FV 值容易受到 SpotLight 的影响，在聚焦模糊的时候因为光晕扩散，图像中低频分量会增加，会出现图像模糊反而 FV 值变大的现象。为了抑制这一现象，Hi3516CV300 提供以下方法。

a. 高亮计数器

如图 3-10 所示，在聚焦模糊的时候因为光晕扩大画面中高亮点的个数增加，清晰的时候高亮点的个数最小，通过配置合适的高亮点判断门限 u16HighLumaTh 可以辅助实现 SpotLight 场景聚焦。

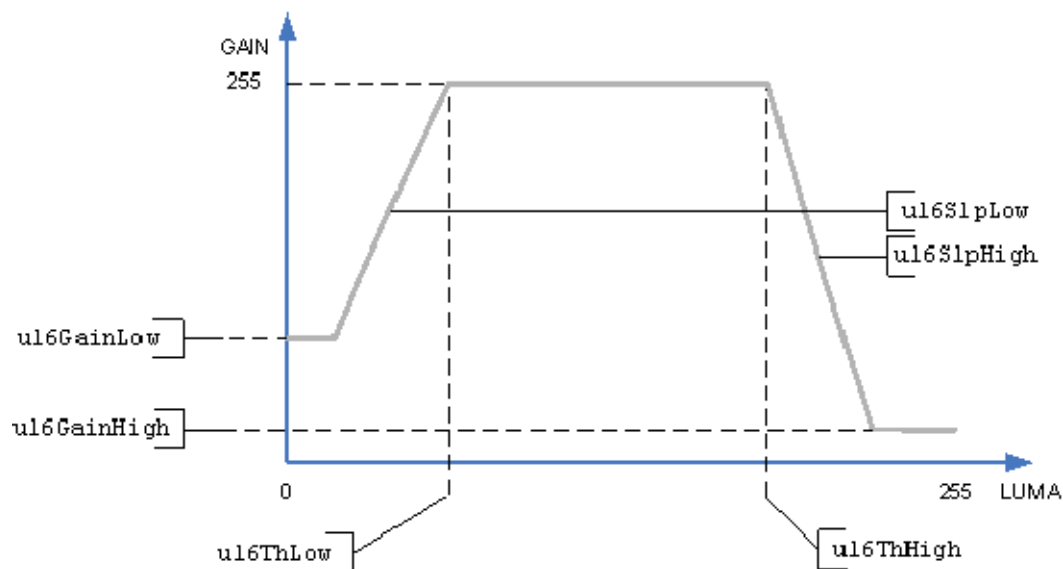
图3-10 高亮统计值



b. Level Depend Gain

LDG 的主要原理为通过参考画面像素亮度来衰减滤波器的输出，从而抑制 SpotLight 对 FV 值的影响。如图 3-11 所示，横坐标为像素亮度值，纵坐标为对滤波器输出的 Gain，此处的 Gain 为 8bit 定点小数 Q8。用户可以通过调整图中 6 个参数控制曲线的形状。

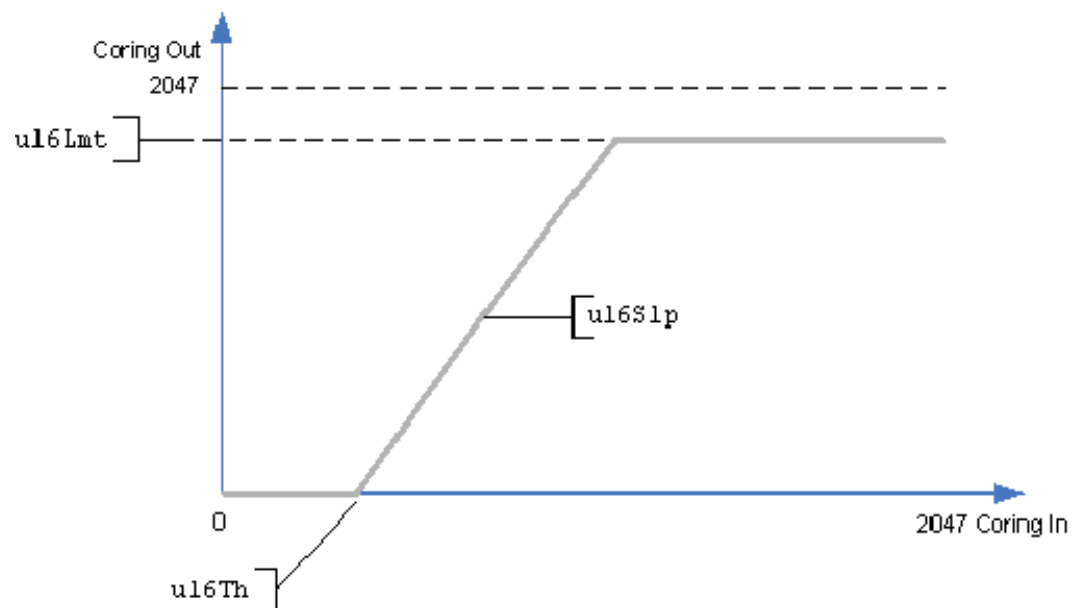
图3-11 Level Depend Gain 曲线



3.5.5 Coring

将滤波器输出通过 Coring 模块可以提高 FV 曲线的抗噪声能力，如图 3-12 所示，设置图中 3 个参数便可以确定 coring 曲线的形状，通过设置合理的参数便可以实现不统计噪声，以及增加有效信号对统计信息的贡献，从而提升统计信息在低照度场景中的性能。

图3-12 Coring 曲线



3.5.6 统计模式配置

统计模块支持以下两种模式：

- 峰值配置模式

设置为 peak 模式 block 的统计值为滤波后图像每行最大值之和，否则直接将每个点的值求和。求和模式适用于噪声较大场景。

- 平方配置模式，有以下两种选择。

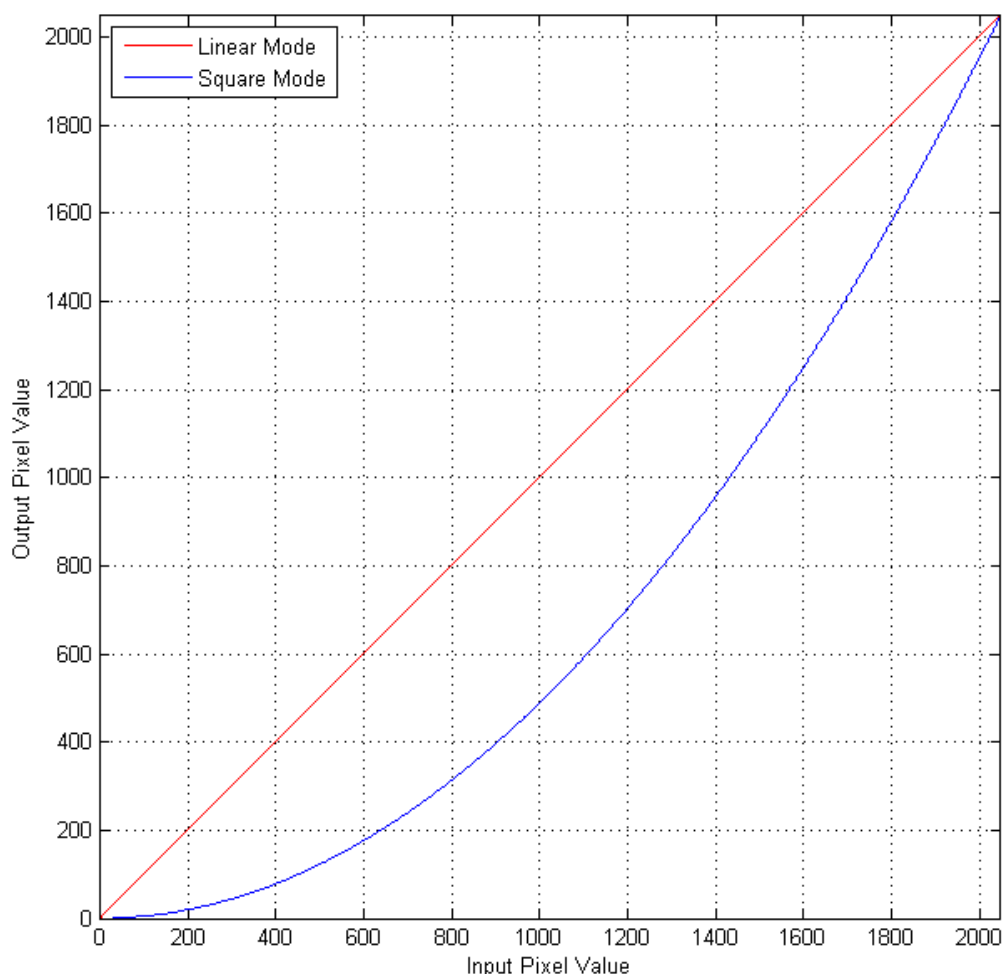
- 0: Linear mode

- 1: Square mode

设置为 Square 模式后，会先对滤波器输出进行归一化后进行平方再做统计。

Square 模式的 FV 曲线在焦点附近会更加陡峭，相应的映射曲线如[图 3-13](#)所示。

图3-13 平方配置模式映射曲线图



同时也支持以下配置：

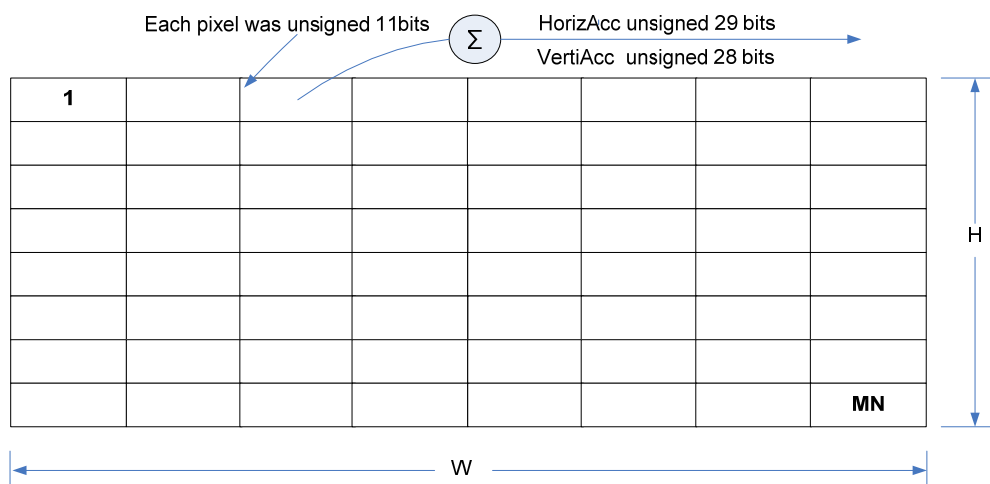
- 输出 shift

对输出值 right shift 缩小，防止值溢出。

- Block 大小的配置

对通过滤波器后的图像可以进行分块统计，如图 3-14 所示，块的大小可以设置，Hi3516CV300 最大支持 17*15 个 blocks，最终块的统计值是对每个 pixel 的累加或者对每行最大值累加。以水平方向为例，因为累加器的宽度为 unsigned 29bits，滤波器输出 pixel 宽度为 unsigned 11bits，所以当块内输出值都为 2047 的时候，最大支持累加的 pixel 个数为 2^{18} ，否则会发生溢出，用户设置大小时应该注意。

图3-14 滤波器后的图像



3.5.7 统计值的获取

当一帧图像最后一个 pixel 通过 AF 模块后，统计值即更新，推荐用户通过 HI_MPI_ISP_GetVDEndTimeOut 同步获取统计值，然后 AutoFocus 和 ZoomTracking 算法完成目标 focus 和 zoom position 的计算，需要注意的是因为 LINUX user space 任务调度不能保证一致的实时性，建议将需要保证实时性的驱动配置放在 kernel space 完成。

3.5.8 FV 值计算

一般的推荐最终生成如图 3-5 所示的两条曲线，我们将 low BandPass filter 生成的 FV 命名为 FV1，High BandPass filter 生成的命名为 FV2，3.5.9 参考代码中变量 stFocusCfg 的值为默认推荐使用的 AF 滤波器参数(以 sony imx290 为例),FV1 使用在低照度场合，FV2 使用在正常照度场合。对于每一个 block，FV1，FV2 的计算方式为：

$$\begin{cases} FV1_n = \alpha * H1_n + (1 - \alpha) * V1_n \\ FV2_n = \beta * H2_n + (1 - \beta) * V2_n \end{cases}$$

H1, H2, V1, V2 分别为水平垂直四组滤波器的统计值，可设置适当的权重对水平和垂直方向的滤波器输出统计值进行混合。

最终的 FV 值还需要对每个 block 进行加权，即 FV1，FV2 采用如下公式计算。

$$FV = \frac{\sum_{n=1}^{BLOCKS} (FV_n * Weight_n)}{\sum Weight_n}$$

3.5.9 FV 计算参考代码

```
#include <stdio.h>
#include <string.h>
#include "hi_type.h"
#include "mpi_isp.h"
```



```
#include "iniparser.h"

#define BLEND_SHIFT 6
#define ALPHA      64          // 1
#define BELTA      54          // 0.85

static int AFWeight[8][8] = {{1,1,1,1,1,1,1,1},
                              {1,2,2,2,2,2,2,1},
                              {1,2,2,2,2,2,2,1},
                              {1,2,2,2,2,2,2,1},
                              {1,2,2,2,2,2,2,1},
                              {1,2,2,2,2,2,2,1},
                              {1,2,2,2,2,2,2,1},
                              {1,1,1,1,1,1,1,1}},
};

int main(int argc, char* argv[])
{
    HI_S32 s32Ret = HI_SUCCESS;
    HI_U32 u32FrmCnt = 0;
    HI_U32 i, j;
    HI_U16 u16StatData;
    ISP_DEV IspDev;
    ISP_STATISTICS_CFG_S stIspStaticsCfg;
    ISP_STATISTICS_S stIspStatics;
    ISP_VD_INFO_S stVdInfo;
    ISP_PUB_ATTR_S stPubattr;

    ISP_FOCUS_STATISTICS_CFG_S stFocusCfg =
    {
        {1, 8, 8, 1920, 1080, 1, 0, {0}, 1, {1, 1, 240, 240, 0}, {1,
0x9bff}, 0xf0},
        {1, {1, 1, 1}, {188, 414, -330, 486, -461, 400, -328}, {7, 0, 3,
1}, {1, 0, 255, 0, 220, 8, 14}, {127, 12, 2047} },
        {0, {1, 1, 0}, {200, 200, -110, 461, -415, 0, 0}, {6, 0, 1,
0}, {0, 0, 0, 0, 0, 0 }, {15, 12, 2047} },
        {{20, 16, 0, -16, -20}, {1, 0, 255, 0, 220, 8, 14}, {38, 12,
1800} },
        {{-12, -24, 0, 24, 12}, {1, 0, 255, 0, 220, 8, 14}, {15, 12,
2047} },
        {4, {0, 0}, {1, 1}, 0}
    };
    char param1, param2;
```



```
if (argc < 2)
{
    printf("use like. ./sample_af c    -> Fv curve\n");
    printf("..... ./sample_af h 0  -> h0 blocks\n");
    printf("..... ./sample_af h 1  -> h1 blocks\n");
    printf("..... ./sample_af v 0  -> v0 blocks\n");
    printf("..... ./sample_af v 1  -> v1 blocks\n");
    printf("..... ./sample_af y 0  -> y  blocks\n");
    printf("..... ./sample_af y 1  -> hlcnt blocks\n");
    return -1;
}

if ((param1 = *argv[1]) != 'c' && !argv[2])
{
    printf("args was too less!, should be %c + X\n", *argv[1]);
    return -1;
}
else if (argv[2])
{
    param2 = *argv[2];
}

//step1: set AF static widnow to 8x8 block
IspDev = 0;
s32Ret = HI_MPI_ISP_GetStatisticsConfig(IspDev, &stIspStaticsCfg);
s32Ret |= HI_MPI_ISP_GetPubAttr(IspDev, &stPubattr);

stFocusCfg.stConfig.ul6Vsize = stPubattr.stWndRect.u32Height;
stFocusCfg.stConfig.ul6Hsize = stPubattr.stWndRect.u32Width;

stIspStaticsCfg.unKey.bit1AfStat = 0x1;

if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_ISP_GetStatisticsConfig error!(s32Ret = 0x%x)\n",
s32Ret);
    return HI_FAILURE;
}
memcpy(&stIspStaticsCfg.stFocusCfg, &stFocusCfg,
sizeof(ISP_FOCUS_STATISTICS_CFG_S));
s32Ret = HI_MPI_ISP_SetStatisticsConfig(IspDev, & stIspStaticsCfg);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_ISP_SetStatisticsConfig error!(s32Ret = 0x%x)\n",
```



```
s32Ret);  
    return HI_FAILURE;  
}  
  
//step2:calculate FV for every frame  
while (1)  
{  
    s32Ret = HI_MPI_ISP_GetVDEndTimeOut(IspDev, &stVdInfo, 5000);  
    s32Ret |= HI_MPI_ISP_GetStatistics(IspDev, &stIspStatics);  
  
    if (HI_SUCCESS != s32Ret)  
    {  
        printf("HI_MPI_ISP_GetStatistics error!(s32Ret = 0x%x)\n",  
s32Ret);  
        return HI_FAILURE;  
    }  
    if (param1 != 'c')  
    {  
        if ((++u32FrmCnt % 30))  
        {  
            continue;  
        }  
  
        for (i = 0; i < 15; i++)  
        {  
            for (j = 0; j < 17; j++)  
            {  
                if (param1 == 'h' && param2 == '0')  
                {  
                    u16StatData =  
stIspStatics.stFocusStat.stZoneMetrics[i][j].ul6h1;  
                }  
                else if (param1 == 'h' && param2 == '1')  
                {  
                    u16StatData =  
stIspStatics.stFocusStat.stZoneMetrics[i][j].ul6h2;  
                }  
                else if (param1 == 'v' && param2 == '0')  
                {  
                    u16StatData =  
stIspStatics.stFocusStat.stZoneMetrics[i][j].ul6v1;  
                }  
                else if (param1 == 'v' && param2 == '1')  
                {  

```




```
        u16StatData =
stIspStatics.stFocusStat.stZoneMetrics[i][j].u16v2;
    }
    else if (param1 == 'y' && param2 == '0')
    {
        u16StatData =
stIspStatics.stFocusStat.stZoneMetrics[i][j].u16y;
    }
    else
    {
        u16StatData =
stIspStatics.stFocusStat.stZoneMetrics[i][j].u16H1Cnt;
    }
    printf("%6d", u16StatData);
}
printf("\n");
}
printf("-----%c-%c--
-----\n\n\n\n\n", param1, param2);
    continue;
}
HI_U32 u32SumFv1 = 0;
HI_U32 u32SumFv2 = 0;
HI_U32 u32WgtSum = 0;
HI_U32 u32Fv1_n, u32Fv2_n, u32Fv1, u32Fv2;

if ((++u32FrmCnt % 2))
{
    continue;
}

for ( i = 0 ; i < stFocusCfg.stConfig.u16Vwnd; i++ )
{
    for ( j = 0 ; j < stFocusCfg.stConfig.u16Hwnd; j++ )
    {

        HI_U32 u32H1 =
stIspStatics.stFocusStat.stZoneMetrics[i][j].u16h1;
        HI_U32 u32H2 =
stIspStatics.stFocusStat.stZoneMetrics[i][j].u16h2;
        HI_U32 u32V1 =
stIspStatics.stFocusStat.stZoneMetrics[i][j].u16v1;
        HI_U32 u32V2 =
stIspStatics.stFocusStat.stZoneMetrics[i][j].u16v2;
```



```
        u32Fv1_n = (u32H1 * ALPHA + u32V1 * ((1<<BLEND_SHIFT) -  
        ALPHA)) >> BLEND_SHIFT;  
  
        u32Fv2_n = (u32H2 * BELTA + u32V2 * ((1<<BLEND_SHIFT) -  
        BELTA)) >> BLEND_SHIFT;  
  
        u32SumFv1 += AFWeight[i][j] * u32Fv1_n;  
        u32SumFv2 += AFWeight[i][j] * u32Fv2_n;  
        u32WgtSum += AFWeight[i][j];  
    }  
}  
  
u32Fv1 = u32SumFv1 / u32WgtSum;  
u32Fv2 = u32SumFv2 / u32WgtSum;  
  
printf("%4d    %4d\n", u32Fv1, u32Fv2);  
}  
  
return HI_SUCCESS;  
}
```

3.5.10 PQTools 滤波器设计工具使用说明

HiPQ 工具提供了自动对焦参数仿真插件，使图像质量调试人员可以如下操作：

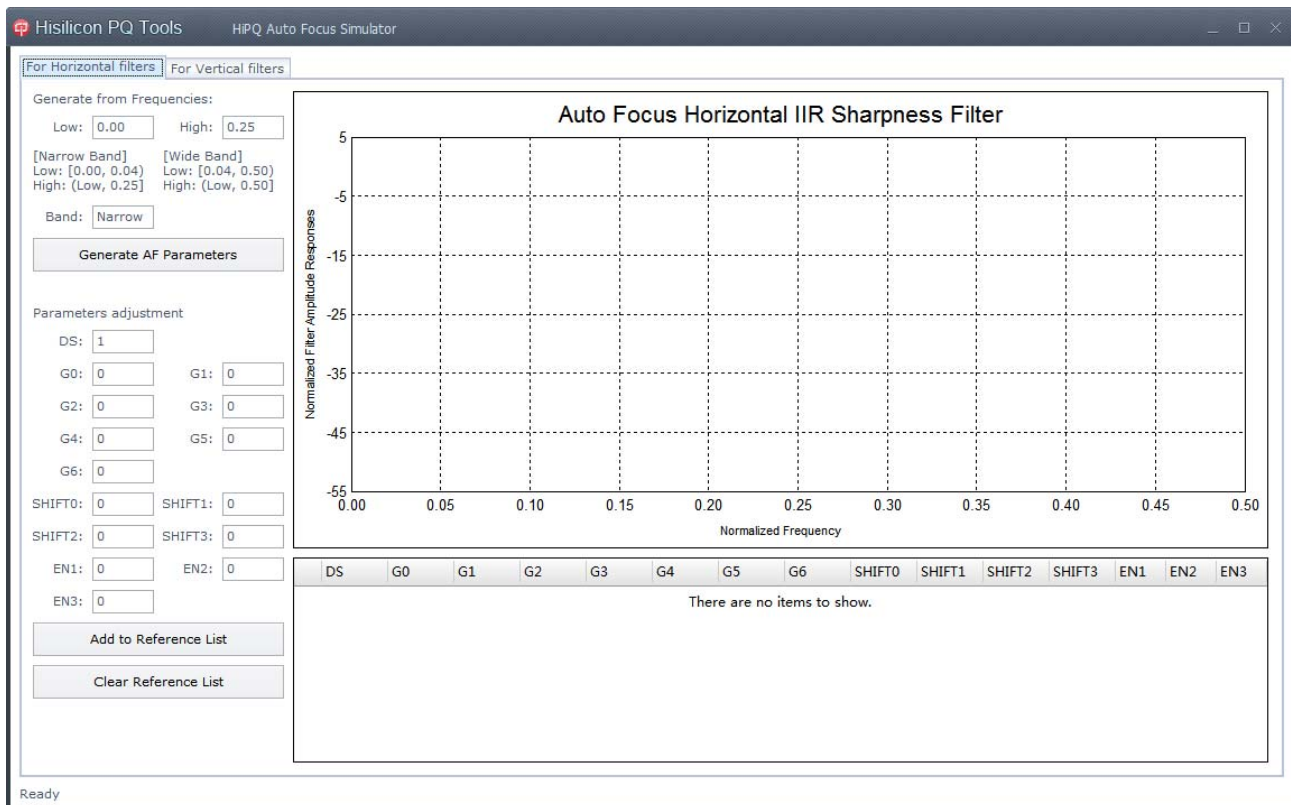
- 根据截止频率自动获取较适合的 AF 参数，并查看对应的频率响应曲线；
- 自行配置 AF 参数，并查看对应的频率响应曲线。

目前版本的插件可用于图像质量调测，使用该工具之前，用户需要安装 Matlab 运行时的 2012a（32 位）版本。

3.5.10.1 工具界面

从 HiPQ 工具主界面工具栏的外挂插件下拉框中选择“HiPQ Auto Focus Simulator”，可以打开自动对焦参数仿真工具，如图 3-15 所示。

图3-15 自动对焦参数仿真工具



工具分为 For Horizontal filters 和 For Vertical filters 两个页签，相互独立，分别对应水平和垂直方向的滤波器参数仿真。下面叙述的内容对两个页签均有效。

Hi3516CV300 芯片上的 AF 水平滤波器具备以下两种模式：

- 窄带模式（Narrow Band）：此种模式下，AF 水平滤波器将在一个较小的频率范围内进行对焦。当用户设置的截止频率的低频小于 0.04 时，工具将推荐用户使用窄带模式。这个模式下，截止频率高频最大值为 0.25。
- 宽带模式（Wide Band）：此种模式下，AF 水平滤波器将在一个较大的频率范围内进行对焦。当截止频率的低频大于或等于 0.04 时，工具将推荐用户使用宽带模式。这个模式下，截止频率高频最大值为 0.5。

用户可以在 PQ 工具主界面（已经加载调试表）上找到 StatisticsConfig -> AFConfig_HPParam_IIR0 的寄存器分组来对模式和滤波系数进行设置。

3.5.10.2 通过截止频率获取 AF 滤波系数

在每个页签中，用户均可以用截止频率来生成滤波系数。在 Generate from Frequencies 处输入截止频率的低频和高频后，点击“Generate AF Parameters”，即可生成截止频率对应的滤波系数，显示在 Parameters adjustment 下方的各个文本框中，并在右侧的坐标图上显示对应该组滤波系数的频率响应曲线，曲线以红色表示，如图 3-16 和图 3-17 所示。



图3-16 通过截止频率获取滤波系数（窄带）

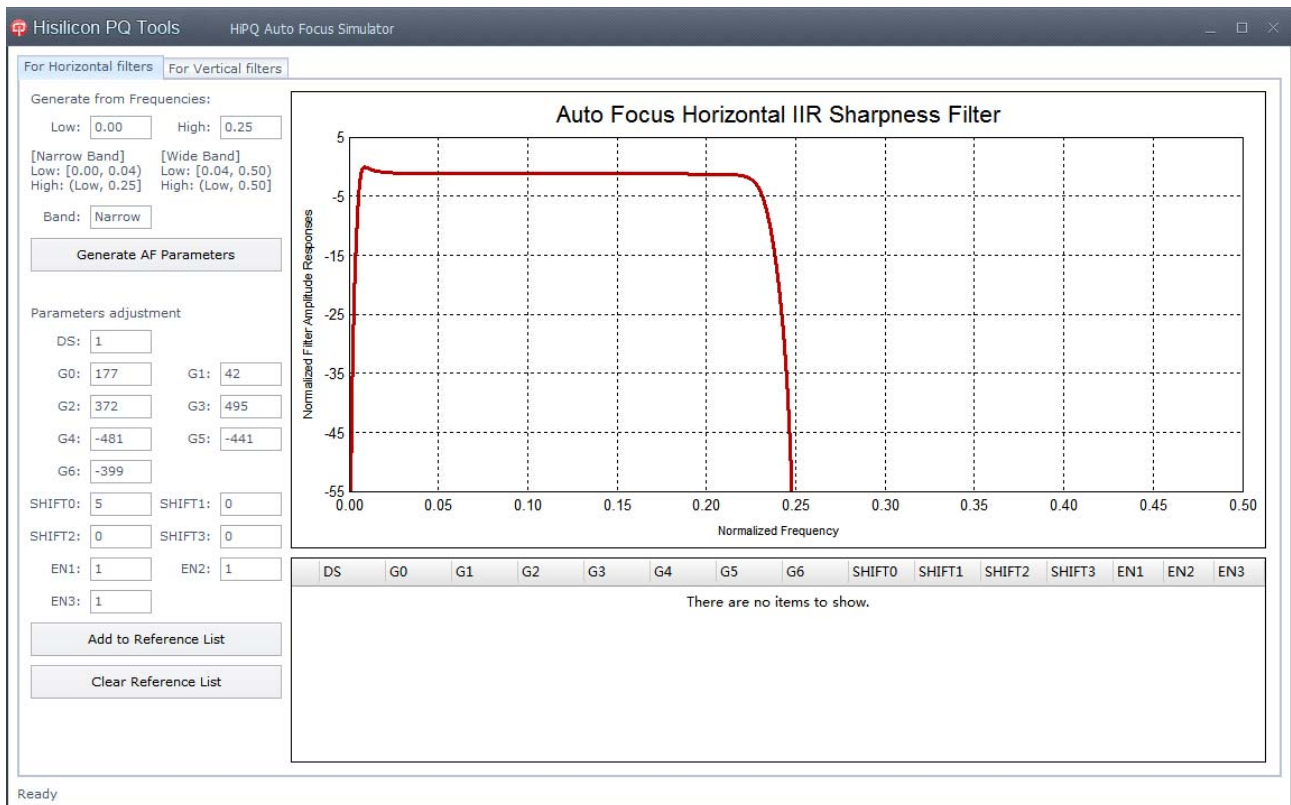
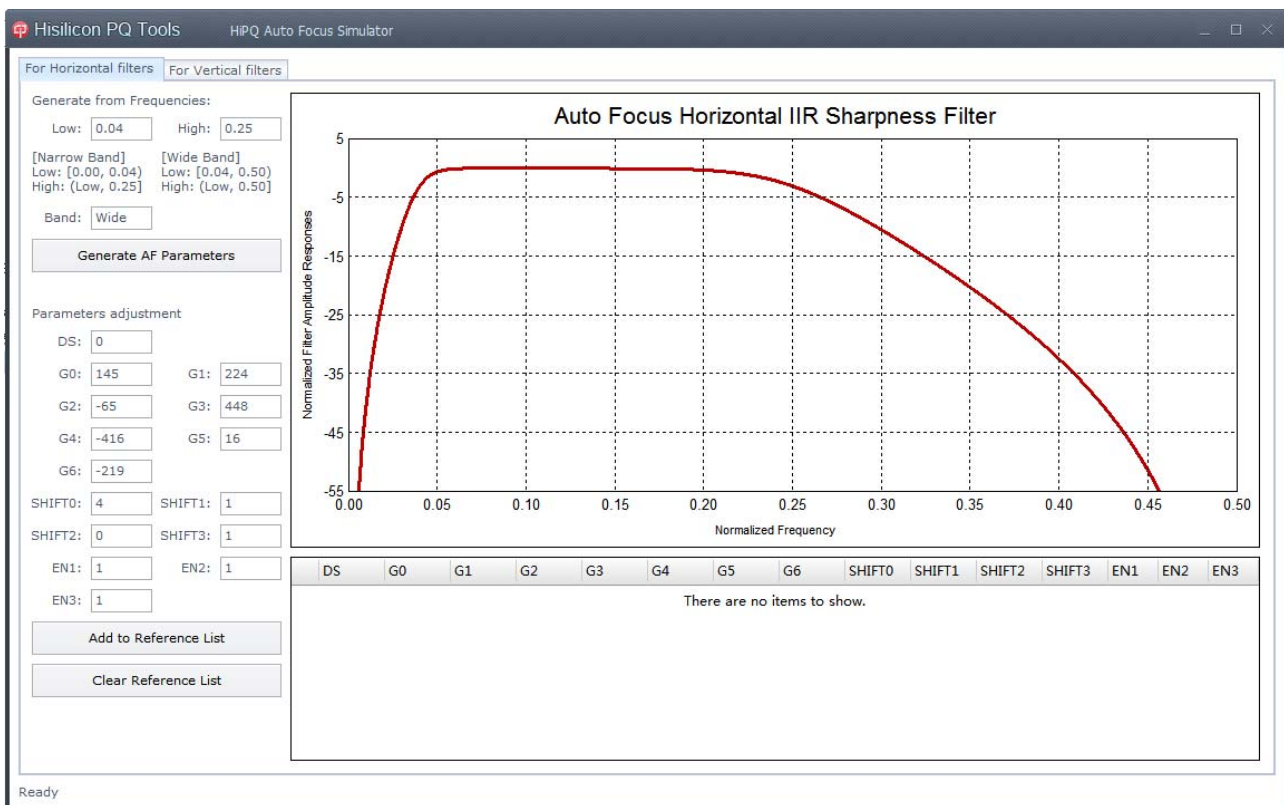


图3-17 通过截止频率获取滤波系数（宽带）



注意

截止频率在工具中会自动适配精度。水平滤波器的截止频率取小数点后两位，垂直滤波器的截止频率取小数点后一位。取精度时，多余的小数位数直接舍弃。

3.5.10.3 手动配置滤波系数

除了通过截止频率获取对应的滤波系数，用户还可以手动进行每个滤波系数项进行设置。在 Parameters adjustment 处，找到对应的参数并进行修改后，工具将自动计算这组参数的频率响应曲线并绘制在右侧。曲线以红色表示。

3.5.10.4 设置参考组

如果用户想要在不同组的滤波系数间对比，可以使用参考组功能。当用户需要将某一组滤波系数列为参考组时，点击“Add to Reference List”按钮，工具即会在右下角的列表中添加一行，列出该参考组的参数项，如图 3-18 所示。

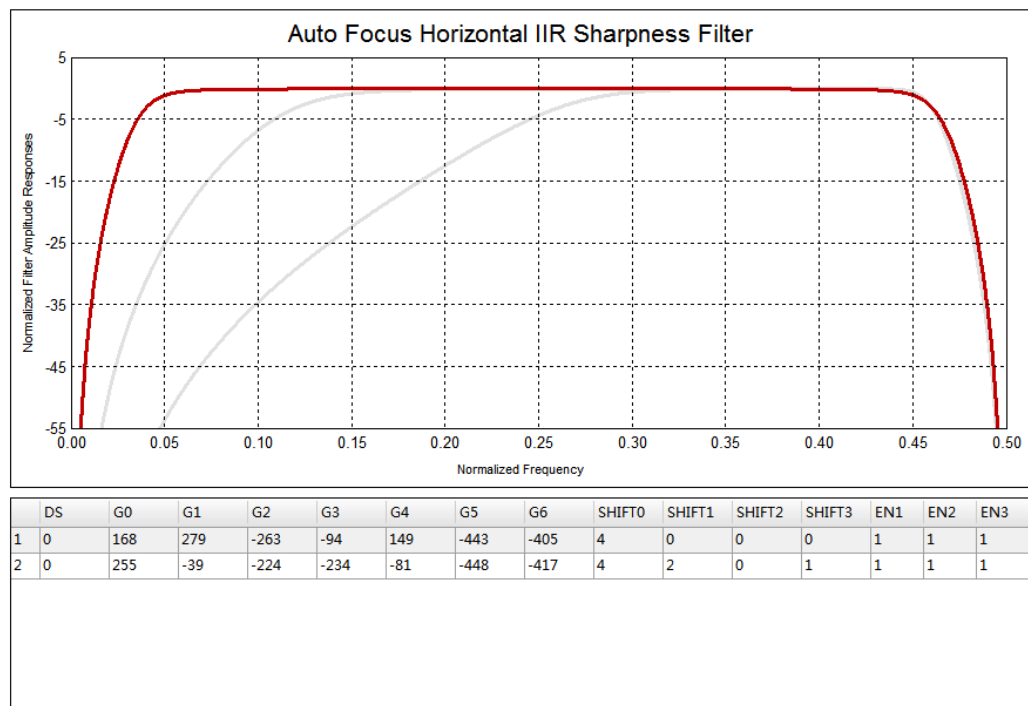


图3-18 参考组列表

	DS	G0	G1	G2	G3	G4	G5	G6	SHIFT0	SHIFT1	SHIFT2	SHIFT3	EN1	EN2	EN3
1	0	145	224	-65	448	-416	16	-219	4	1	0	1	1	1	1

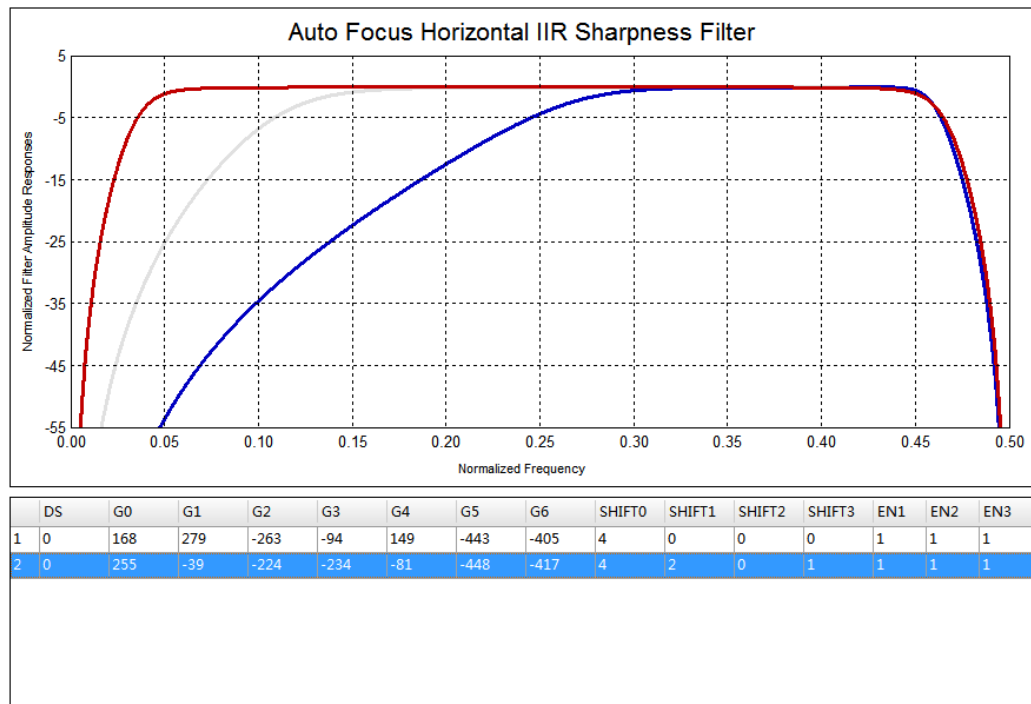
对于每一个参考组，工具还将在上方的坐标图上生成一条曲线，以灰色表示，如图 3-19 所示：

图3-19 参考组的曲线显示



用户还可以将某一参考组的曲线高亮显示出来。在列表中点击需要高亮显示的参考组，此时对应的曲线会在上方的坐标图中变成蓝色，如图 3-20 所示：

图3-20 选中参考组曲线高亮显示



通过这样的方式，用户可以在不同的参考组间进行频率响应曲线对比，从而发现更优的方案。



注意

- 在坐标图上，红色的当前参数曲线和蓝色的参考组高亮曲线相对其他的灰色参考组曲线处于更上方的位置。因此，会出现刚刚添加参考组时看不到灰色曲线的状况。此时只要修改滤波系数，使红色曲线重新绘制，就能看到灰色的参考曲线。
- 每个页签上最多只能添加 6 组参考组。如果需要添加更多参考组，请先点击 Clear Reference List 删除所有参考组。

3.5.11 AF 同步回调

当一帧图像最后一个 pixel 通过 AF 模块后，统计值即更新，推荐用户通过 HI_MPI_ISP_GetVDEndTimeOut 同步获取统计值，AutoFocus 和 ZoomTracking 算法完成目标 focus 和 zoom position 的计算，如果用户使用的是 LINUX 系统，因为 user space 任务调度不能保证一致的实时性，建议将需要保证实时性的驱动配置放在 kernel space 完成。ISP 提供同步回调接口的注册，可以实现与 VD 同步。在 extdrv/sample_ist 有相应的 sample，用户可以将实时性要求较高的任务放在同步回调里面，底层提供 HwIRQ 和 Workqueue 两种方式实现，可以选择相应的实现方式以确定实时级别。HwIRQ 是指



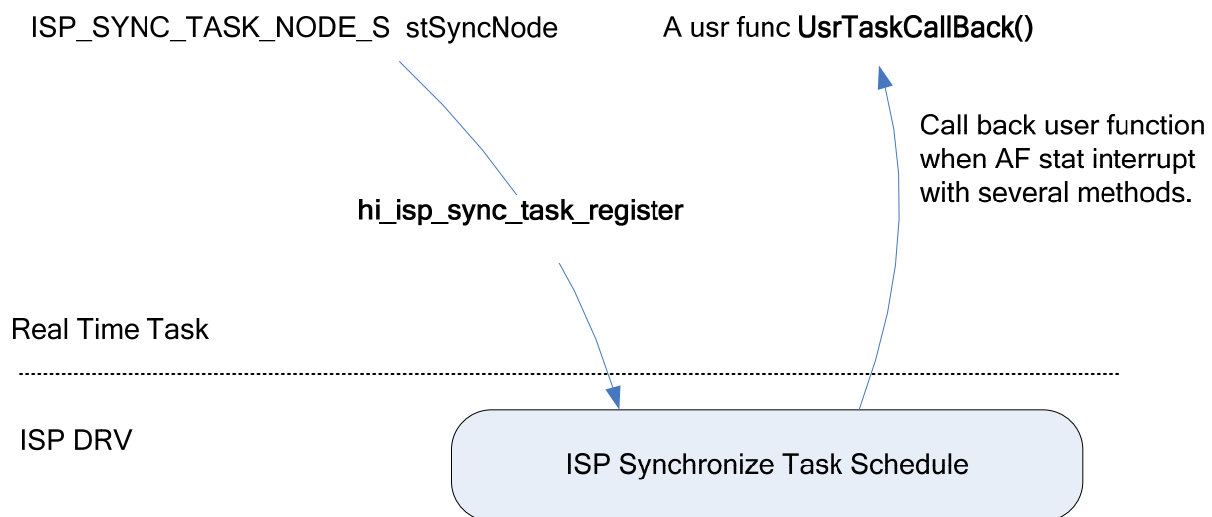
任务放在中断服务程序中实现，实时性最高；Workqueue 的实时性取决于 LINUX 系统调用。



说明

HI_MPI_ISP_GetVDEndTimeOut 请参考《HiISP 开发参考》。

图3-21 AF 同步回调



API 参考

- [hi_isp_sync_task_register](#): 向 ISP 注册同步回调接口。
- [hi_isp_sync_task_unregister](#): 向 ISP 反注册同步回调接口。

hi_isp_sync_task_register

【描述】

向 ISP 注册同步回调接口。

【语法】

```
HI_S32 hi_isp_sync_task_register(ISP_DEV dev, ISP\_SYNC\_TASK\_NODE\_S *  
pstNewNode);
```

【参数】

参数名称	描述	输入/输出
dev	ISP 设备号。	输入
pstNewNode	新插入的同步回调节点	输入

【返回值】



返回值	描述
0	成功
非 0	失败

【需求】

头文件：isp_ext.h.

【注意】

- 使用前需要确保 ISP 驱动已加载。
- 因为 ISP 同步回调内部实现不保存用户传入的 pstNewNode 指向的实体，所以要求使用 [ISP_SYNC_TASK_NODE_S](#) 定义实体时不能为局部变量。

【举例】

无

【相关主题】

[hi_isp_sync_task_unregister](#)

hi_isp_sync_task_unregister

【描述】

向 ISP 反注册同步回调接口。

【语法】

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

【参数】

参数名称	描述	输入/输出
dev	ISP 设备号。	输入
pstDelNode	需要删除的同步回调节点	输入

【返回值】

返回值	描述
0	成功
非 0	失败



【需求】

头文件：isp_ext.h.

【注意】

使用前需要确保 ISP 驱动已加载。

【举例】

无

【相关主题】

[hi_isp_sync_task_register](#)

数据类型

- [ISP_SYNC_TSK_METHOD_E](#): 定义同步回调方法，决定实时性。
- [ISP_SYNC_TASK_NODE_S](#): 定义同步回调节点信息。

ISP_SYNC_TSK_METHOD_E

【说明】

定义同步回调方法，决定实时性。

【定义】

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

【成员】

成员名称	描述
ISP_SYNC_TSK_METHOD_HW_IRQ	使用硬件中断方式回调。
ISP_SYNC_TSK_METHOD_TSKLET	使用 TaskLet 方式回调。
ISP_SYNC_TSK_METHOD_WORKQUE	使用工作队列方式回调。

【注意事项】

无

【相关数据类型及接口】

无



ISP_SYNC_TASK_NODE_S

【说明】

定义同步回调节点信息。

【定义】

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

【成员】

成员名称	描述
enMethod	回调方式。
pfnIspSyncTskCallBack	回调函数，用户注册时传入。
u64Data	回调函数参数，用户注册时传入。
pszId	节点 ID。
list	list 节点，用于管理多个回调节点，无需关注。

【举例】

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

【注意事项】

无

【相关数据类型及接口】

[ISP_SYNC_TSK_METHOD_E](#)



4 附录

4.1 注册函数的关系

- HI_MPI_ISP_AeLibRegCallBack、HI_MPI_ISP_AwbLibRegCallBack、HI_MPI_ISP_AfLibRegCallBack 这三个接口是 ISP firmware 库提供的钩子函数，用于开发 3A 算法库时实现注册动作。例如海思提供的 3A 算法库的 HI_MPI_AE_Register、HI_MPI_AWB_Register、HI_MPI_AF_Register 接口，在实现时调用了相应的钩子函数，所以调用 HI_MPI_AE_Register 能实现 AE 算法库向 ISP firmware 库注册。
- 海思 3A 算法库同样也提供了钩子函数，用于 Sensor 库实现向 3A 算法库注册的动作。例如 HI_MPI_AE_SensorRegCallBack、HI_MPI_AWB_SensorRegCallBack、HI_MPI_AF_SensorRegCallBack，在 xxx_cmos.c 中可以看到调用了这些钩子函数的函数 sensor_register_callback。用户在开发 3A 算法库时，也可以通过提供钩子函数的方式，实现 Sensor 库向 3A 算法库的注册。
- ISP firmware 库也提供了钩子函数，用于 Sensor 库实现向 ISP firmware 库注册的动作。例如 HI_MPI_ISP_SensorRegCallBack，在 xxx_cmos.c 中可以看到调用了该钩子函数的函数 sensor_register_callback。
- 所以，当用户调用 HI_MPI_AE_Register、HI_MPI_AWB_Register、HI_MPI_AF_Register 和 sensor_register_callback 就完成了 3A 算法库向 ISP firmware 库注册、Sensor 库向 3A 算法库和 ISP firmware 库注册。



说明

用户开发 3A 算法库时，请自行实现 HI_MPI_XXX_Register 接口。同时也请自行实现 HI_MPI_XXX_SensorRegCallBack 钩子函数，并在 sensor_register_callback 中增加调用该钩子函数的代码，相关代码可以参考 ISP firmware 库的开源代码。

4.2 扩展性的设计考虑

在代码中有 ISP_DEV、ALG_LIB_S、SENSOR_ID 这样一些概念，这些概念是出于架构扩展性的考虑。

- ISP_DEV 主要考虑的是支持多个 ISP 单元的情形。无论是多个 ISP 硬件单元，或是一个 ISP 硬件单元分时复用，从软件意义上讲，需要预留出扩展性。目前 ISP_DEV 只需设置为 0 即可。



- ALG_LIB_S 主要考虑的是支持多个算法库，并动态切换的情形。例如用户实现了一套 AE 算法代码，但注册两个库，分别用于正常场景和抓拍场景，那么这时候需要用结构体中的 s32Handle 来进行区分。例如用户实现了一套 AWB 算法代码，同时又想在某些场景下使用海思 AWB 算法库，那么这时候可以用结构体中的 acLibName 进行区分。当用户注册多个 AE 库，或 AWB 库时，ISP firmware 将会全部对它们进行初始化，但是在运行时，仅会调用有效的库，设置有效库的接口是 HI_MPI_ISP_SetBindAttr，通过此接口可以快速切换运算的库。
- SENSOR_ID 仅起一个校验作用，确认注册给 ISP firmware 库和 3A 算法库的是同一款 sensor。

这些概念仅是设计时预留的冗余，如果完全不需要这些概念，可以在开发时去掉这些概念。

4.3 3A 架构的设计思路

设计思路基本是这样，ISP firmware 初始化并销毁各个算法单元；在运行时，提供前一帧的统计信息，并根据返回值配置寄存器，其他内容，均由用户开发。所以当用户替换自己的 3A 算法后，当前的 AE/AWB/AF 的 MPI 不可复用，cmos.c 中的 AE/AWB/AF 相关的内容不可复用，对于 AE 的权重配置、五段直方图 Thresh 配置和 AWB 的找白点配置的内容不可复用，这几个配置理论上是由 3A 算法配置，而不是从 ISP firmware 获取，ISP firmware 中仅有简单的初始化值。

3A 算法并不需要显式地去配置 ISP 寄存器，只需将需要配置的 ISP 寄存器值写到 ISP_AE_RESULT_S、ISP_AWB_RESULT_S、ISP_AF_RESULT_S 结构体中即可；也不需要显式地去读取 ISP 寄存器，只需从 ISP_AE_INFO_S、ISP_AWB_INFO_S、ISP_AF_INFO_S 结构体中读取即可。

4.4 外部寄存器的说明

在 IPC 的应用中，通常除了业务主程序进程外，板端还会有另外的进程去支持 PC 端的工具来调节图像质量。ISP 的各个算法的许多状态、参数均驻留于全局变量中，不足以支持多进程的访问，所以引入外部寄存器的概念，用以支持多进程的业务场景。用户通过 PC 端工具与板端进程通信，调用海思提供的 MPI，实际是改变外部寄存器中的内容，从而改变业务主程序中的 ISP 的各个算法的状态和参数。

外部寄存器还能与实际的硬件寄存器通过统一的接口读写，形式上与实际的硬件寄存器无差别。

外部寄存器封装的接口为 VReg_Init、VReg_Exit、IORD_32DIRECT、IORD_16DIRECT、IORD_8DIRECT、IOWR_32DIRECT、IOWR_16DIRECT、IOWR_8DIRECT，地址设定如下：

0x0~0xFFFF 对应 ISP 的硬件寄存器，例如 IORD_32DIRECT(0x0008)读取 0x113A0008 硬件寄存器的值。

0x10000~0x1FFFF 对应 ISP firmware 的外部寄存器，例如 IOWR_16DIRECT(0x10020)。



0x20000~0x2FFFF 对应 AE 的外部寄存器，其中分成了 16 份，海思 AE 用了 0x20000~0x21FFF。0x30000~0x3FFFF 对应 AWB 的外部寄存器，0x40000~0x4FFFF 对应 AF 的外部寄存器，同样的也分成了 16 份。

用户如果使用外部寄存器的话，有这些已封装好的接口可以使用，当然用户也可以有其他方案实现多进程的支持。外部寄存器的地址空间是自定义的，只要不冲突即可。详细请参考开源代码，接口定义在 hi_vreg.h 文件中。