AX Sensor 调试指南

文档版本：**V2.3**

发布日期：**2024/11/08**

目 录

权利声明

注意

您购买的产品、服务或特性等应受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。除非商业合同另有约定，本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。

前 言

适用产品

- AX650A/N
- AX620E 系列产品（AX630C、AX620Q）

适读人群

- 软件开发工程师
- 技术支持工程师

符号与格式定义

符号/格式	说明
`xxx`	表示您可以执行的命令行。
斜体	表示变量。如，“*安装目录*/AXxxx_SDK_Vx.x.x/build 目录”中的“安装目录”是一个变量，由您的实际环境决定。
说明/备注：	表示您在使用产品的过程中，我们向您说明的事项。
注意：	表示您在使用产品的过程中，需要您特别注意的事项。

修订历史

文档版本	发布时间	修订说明
V1.0	2023/08/25	文档初版
V1.1	2023/10/25	添加 nSettingIndex 使用说明
V1.2	2024/01/03	更新 get black level 相关函数命名等
V1.3	2024/01/27	添加 sensor mclk 寄存器配置介绍
V1.4	2024/01/30	增加 FAQ 章节、增加部分新接口
V2.0	2024/02/23	文档合并，新增 AX650A/N 与 AX630C/AX620Q 区别内容说明
V2.1	2024/02/28	新增 AX650A/N 与 AX630C/AX620Q AX_SENSOR_DEFAULT_PARAM_T 区别内容说明
V2.2	2024/06/13	新增 mclk6、mclk7 使用方法描述
V2.3	2024/11/08	修改文档结构，增加 1.3 驱动架构章节
V2.4	2024/11/21	新增 FAQ,介绍 blc 定点数转换方法

1 概述

1.1 概述

Sensor 驱动模块是 ISP 系统的一个延伸，它一方面向用户提供接入具体 sensor 的驱动模型，另一方面需要与 ISP 3A 算法单元实现对接，配合算法完成对 sensor 曝光行为的控制。本模块的一个基本目标是定义一个 sensor 驱动模板，该模板需要能够充分发挥各种 sensor 的硬件能力，对于绝大多数流行的 sensor 都能够很好地完成与 ISP 系统的适配。

基于此需求，本模块声明了一组数据结构和一组接口函数的定义，用户在实现 sensor 驱动时需要向本模块以及 ISP 模块注册接口函数的具体实例。当系统开始运行后，本模块的控制逻辑以及 ISP 模块的控制逻辑将通过注册得到的回调函数控制物理 sensor，实现对 sensor 工作模式和曝光参数的准确调节。

1.1.1 功能概述

Sensor 驱动中，主要实现了三部分的功能：

- sensor 控制模块，包括配置 sensor 参数、使能 sensor 初始化序列、sensor 寄存器读写等；
- sensor 默认参数模块，用于存放用户自己调试好的模块默认参数，以及 sensor black level 参数等；
- AE 控制模块，用于控制 sensor 的曝光输出，包括配置 gain 增益、配置 shutter 曝光时间、HCG/LCG 模式、帧率、曝光限制参数等。

本文档将展开阐述每一个接口的使用方法，以方便用户快速的理解和调试。

1.1.2 重要概念

表1-1 重要概念

缩写	全称
曝光时间	Sensor 在一帧的时间内每个像素电荷积分时间，曝光时间越长，积累的电荷越多，相应的亮度也越亮。
模拟增益	模拟增益是指 Sensor 内部通过模拟电路将每个像素的电荷按倍数进行放大。每个 Sensor 都有对应的增益寄存器，每个 Sensor 所能支持最大模拟增益倍数和步长都不相同，具体需查看相应的用户手册。
数字增益	数字增益是指 Sensor 内部通过数字电路将每个像素的电荷按倍数进行放大，每个 Sensor 都有对应的增益寄存器，每个 Sensor 所能支持最大数字增益倍数和步长都不相同，具体需查看相应的用户手册。
曝光量	曝光时间、模拟增益、数字增益的乘积，指总亮度增益。
分贝	分贝 X 倍用 dB 表示为 20logX dB。

1.2 Sensor 驱动文件结构

SDK 软件开发包中提供了部分 sensor 驱动源码，在 msp/component/isp_proton/sensor 目录下。以 OS04A10 sensor 为例，驱动文件结构如下：

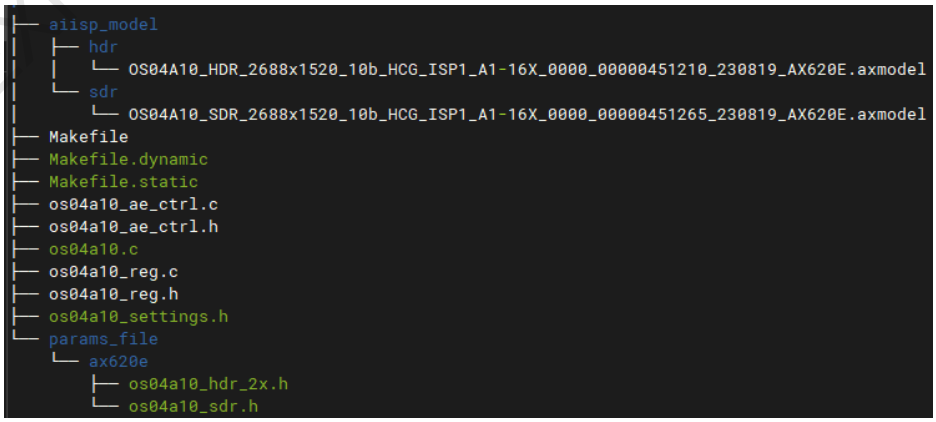

图1-1 Sensor 驱动源码结构图（os04a10 为例）

- os04a10.c 文件主要是 sensor 控制和默认参数加载功能的实现。
- os04a10_reg.c 文件主要是 sensor 基础功能的封装，比如 I2C 的控制、寄存器配置等。
- os04a10_ae_ctrl.c 文件主要实现与 sensor 曝光控制相关的功能
- os04a10_settings.h 文件用来存放 sensor 初始化序列 settings。
- os04a10_sdr.h/ os04a10_hdr_2x.h 用于存放不同模式下的默认参数。
- aiisp_model 文件夹用于存放不同模式下的模型文件。

每个 sensor 驱动都定义了一个 callback 函数集，通过 API 接口 AX_VIN_RegisterSensor()将 sensor driver handle 注册给 ISP FW，通过 API 接口 AX_ISP_ALG_AeRegisterSensor()将 sensor driver handle 注册给 AE。

1.3 Sensor 驱动架构

AX 平台的 sensor 驱动在用户态，编译成动态库或者静态库的形式，每一个 sensor 对应一个驱动库，驱动与其他模块之间的层次关系如下图所示：

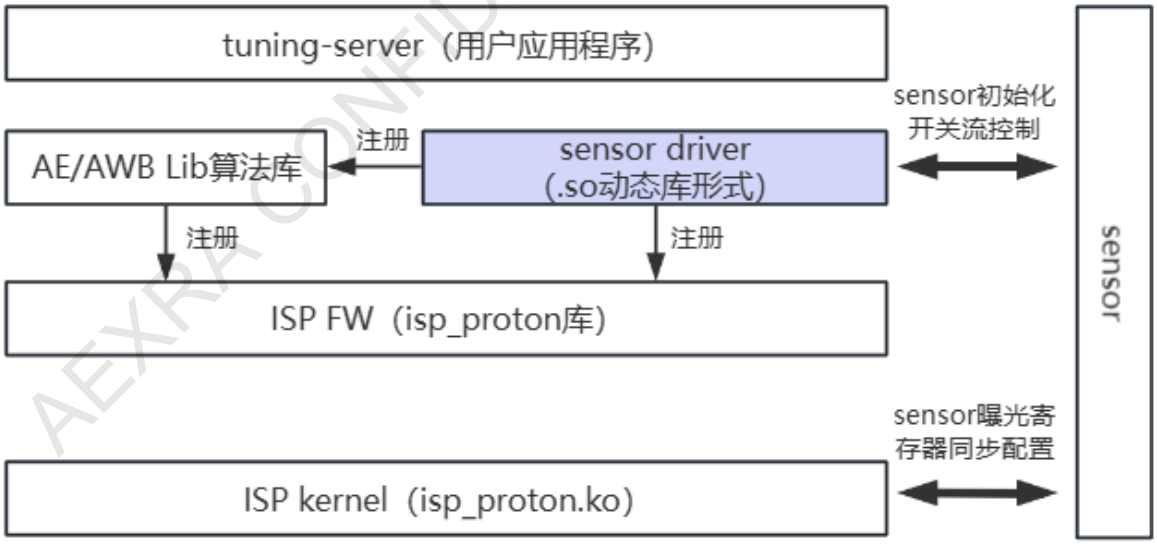

图1-2 Sensor 驱动架构图

以 AX 平台的 tuning-server 应用程序为例，对 sensor 驱动进行解析。

- 应用程序启动时，读取两级的 ini 文件，解析参数并完成初始化，ini 中有 i2c num、

mclk、reset gpio 等硬件特性相关的参数配置

- sensor 驱动在应用程序中被加载，并注册给 ISP FW 和 3A 算法
- ISP 初始化的时候会显式+隐式调用 sensor 驱动的部分 callback 函数，完成 sensor 的初始化
- 3A 算法注册到 ISP FW，应用程序启动时会创建 ISP_Run 线程，会实时调度算法的 run 函数
- ISP_Run 调度 ae_run 函数，AE 算法实时收敛，会将曝光参数下发到 sensor 驱动库中，sensor 驱动中会将参数更新到一个曝光配置 table 中，再由 ISP_Run 调用 pfn_sensor_get_sns_reg_info 将曝光参数统传递到 isp kernel 内核态驱动中，在内核驱动中时间曝光寄存器的同步配置。整个过程包含四步：

  1) ISP FW 先调度 ae_run
  2) ae_run 将曝光参数下发给 sensor 驱动
  3) ISP_FW 再通过回调函数获取 sensor 驱动库中的曝光参数
  4) 内核驱动完成曝光寄存器的同步配置

2 Sensor 调试流程

2.1 调试流程

请按照下图所示流程进行调试：

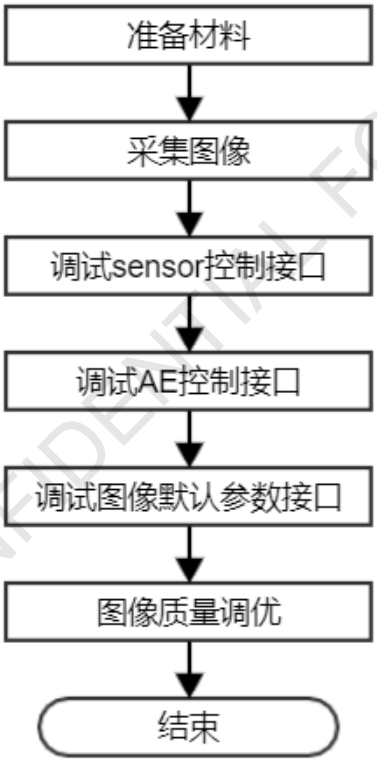

图2-1 Sensor 调试流程图

2.2 准备材料

2.2.1 确认主芯片规格

确认主芯片 sensor 时钟的频率范围、MIPI 等视频接口的能力、HDR 能力、sensor 通信方式（I2C/SPI）、ISP 工作频率上限等。

2.2.2 Sensor datasheet

- 确认图像传输接口模式和输出频率大小。
- 确认曝光时间、增益如何设置，帧率如何修改。
- 确认在 HDR 模式下的以上两项。
- LVDS 接口，需要确认同步码。

2.2.3 Initialize Settings

从 sensor 厂家那里获取 Sensor Initialize Settings，根据自己的业务需求，准备多组 settings，比如不同分辨率、不同帧率、不同数据位宽以及不同 HDR 模式下的 settings 序列。

2.3 调试 sensor 控制接口

2.3.1 确认 sensor 通信是否正常

步骤1　确认 I2C/SPI 的地址、寄存器位宽、数据位宽，这些信息可以固化在 sensor 驱动代码中；

步骤2　确认通信方式和 I2C/SPI 驱动的设备节点号。

如果用户使用 SDK 提供的 tuning-server 工具对接 sensor，需要在对应 sensor 的 ini 文件中配置通信方式和设备节点号。tuning-server ini 文件中定义的字段如下：

```
nRxDev = 0
nDevId = 0
nPipeId = 0
nBusType = 0        ; 0: i2c (default setting), 1: spi, 2: invalid value
nDevNode = 0        ; -1: Use this value only on AX demo&evb board, 0: i2c-0, 1: i2c-1
ePhySel = 0
nSnsClkIdx = 0    ;sensor clock index, support 0 to 5, 0-->mclk0,  1-->mclk1
eSnsClkRate = 24000000
```

其中 nDevNode 的值就代表 I2C/SPI 驱动的设备节点号，用户可根据自己的硬件连接情况自行配置。

步骤3　验证是否可以读写 Sensor 寄存器，获取 sensor id，成功获取到 chip id 需要满足以下几个条件：

1. 硬件连接正确（和原理图对应）；
2. 管脚复用配置正确(参考《00 - AX SDK 使用说明.docx》的 pinmux 章节)
3. reset gpio 配置正确并成功复位；
4. i2c num 配置正确；
5. i2c slave addr 配置正确；
6. chip id 寄存器地址配置正确；
7. mclk num 配置正确，并成功开启；

2.3.2 完成初始化序列的配置

步骤1　准备驱动代码，可以基于一款规格相近的 sensor（master/slave, i2c/spi, hdr/linear）驱动修改，建立独立的文件夹，构建自己的 sensor 驱动库。具体可参考 msp/component/isp_proton/sensor/xxx 目录下的 xxx.c 和 xxx_ae_ctrl.c 文件。

步骤2　完成 pfn_sensor_init、pfn_sensor_exit 以及 pfn_sensor_streaming_ctrl 函数的赋值，将 settings 的初始化配置函数赋值给 pfn_sensor_init 这个回调接口。其中，pfn_sensor_streaming_ctrl 这个接口用来控制 sensor 开流和关流，以方便用户快速的控制 sensor 的输出，如果用户不需要使用该功能，也建议将空函数赋值给该回调函数，禁止将该接口置 NULL。

步骤3　完成 pfn_sensor_set_mode 和 pfn_sensor_get_mode 函数的赋值，这里用来维护 sensor 属性参数，包括分辨率、帧率、HDR 模式等，参数是通过 AX_ISP_SetSnsAttr 接口传递过来的。SDK 提供的 sensor 驱动中，在 pfn_sensor_set_mode 函数中初始化了

驱动内部使用的资源，建议用户复用这部分代码。

步骤4　在 AX_ISP_SetSnsAttr 接口中配置 sensor 分辨率、帧率、主从模式、位宽等参数。

4.3.2.1 关于 sensor mclk 寄存器配置介绍

MCLK	寄存器名称	寄存器地址	寄存器配置方法	寄存器描述
mclk0	clk freq 频率选择寄存器	0x02340000	先 read 寄存器的值，然后修改 bit0~bit3 的值 示例：ax_lookat 0x02340000 –s 0x3	MCLK0 mux select: 4'b0000 : cpll_12m 4'b0001 : cpll_19p2m 4'b0010 : hpll_20p48m 4'b0011 : cpll_24m 4'b0100 : hpll_24p576m 4'b0101 : epll_25m 4'b0110 : cpll_26m 4'b0111 : vpll0_27m 4'b1000 : vpll1_27m 4'b1001 : epll_50m 4'b1010 : vpll0_74p25m 4'b1011 : vpll1_74p25m 4'b1100 : epll_125m default : cpll_12m (4'b0000)
	clk en 使能寄存器	0x02340024	先 read 寄存器的值，然后修改 bit0 的值 示例：ax_lookat 0x02340024 –s 0x1	clock MCLK0 enable control: 1'b0 : disable 1'b1 : enable
mclk1	clk freq 频	0x02340000	先 read 寄存器	MCLK1 mux select:

	率选择寄存器		的值，然后修改 bit4~bit7 的值 示例：ax_lookat 0x02340000 –s 0x30	4'b0000 : cpll_12m 4'b0001 : cpll_19p2m 4'b0010 : hpll_20p48m 4'b0011 : cpll_24m 4'b0100 : hpll_24p576m 4'b0101 : epll_25m 4'b0110 : cpll_26m 4'b0111 : vpll0_27m 4'b1000 : vpll1_27m 4'b1001 : epll_50m 4'b1010 : vpll0_74p25m 4'b1011 : vpll1_74p25m 4'b1100 : epll_125m default : cpll_12m (4'b0000)
	clk en 使能寄存器	0x02340024	先 read 寄存器的值，然后修改 bit1 的值 示例：ax_lookat 0x02340024 –s 0x2	clock MCLK1 enable control: 1'b0 : disable 1'b1 : enable
mclk2	clk freq 频率选择寄存器	0x02340000	先 read 寄存器的值，然后修改 bit8~bit11 的值 示例：ax_lookat 0x02340000 –s 0x300	MCLK2 mux select: 4'b0000 : cpll_12m 4'b0001 : cpll_19p2m 4'b0010 : hpll_20p48m 4'b0011 : cpll_24m 4'b0100 : hpll_24p576m 4'b0101 : epll_25m 4'b0110 : cpll_26m 4'b0111 : vpll0_27m 4'b1000 : vpll1_27m

				4'b1001 : epll_50m 4'b1010 : vpll0_74p25m 4'b1011 : vpll1_74p25m 4'b1100 : epll_125m default : cpll_12m (4'b0000)
	clk en 使能寄存器	0x02340024	先 read 寄存器的值，然后修改 bit2 的值 示例：ax_lookat 0x02340024 –s 0x4	clock MCLK2 enable control: 1'b0 : disable 1'b1 : enable
mclk3	clk freq 频率选择寄存器	0x02340000	先 read 寄存器的值，然后修改 bit12~bit15 的值 示例：ax_lookat 0x02340000 –s 0x3000	MCLK3 mux select: 4'b0000 : cpll_12m 4'b0001 : cpll_19p2m 4'b0010 : hpll_20p48m 4'b0011 : cpll_24m 4'b0100 : hpll_24p576m 4'b0101 : epll_25m 4'b0110 : cpll_26m 4'b0111 : vpll0_27m 4'b1000 : vpll1_27m 4'b1001 : epll_50m 4'b1010 : vpll0_74p25m 4'b1011 : vpll1_74p25m 4'b1100 : epll_125m default : cpll_12m (4'b0000)
	clk en 使能寄存器	0x02340024	先 read 寄存器的值，然后修改 bit3 的值	clock MCLK3 enable control: 1'b0 : disable 1'b1 : enable

			示例：ax_lookat 0x02340024 –s 0x8	
mclk4	clk freq 频率选择寄存器	0x02340000	先 read 寄存器的值，然后修改 bit19~bit15 的值 示例：ax_lookat 0x02340000 –s 0x30000	MCLK4 mux select: 4'b0000 : cpll_12m 4'b0001 : cpll_19p2m 4'b0010 : hpll_20p48m 4'b0011 : cpll_24m 4'b0100 : hpll_24p576m 4'b0101 : epll_25m 4'b0110 : cpll_26m 4'b0111 : vpll0_27m 4'b1000 : vpll1_27m 4'b1001 : epll_50m 4'b1010 : vpll0_74p25m 4'b1011 : vpll1_74p25m 4'b1100 : epll_125m default : cpll_12m (4'b0000)
	clk en 使能寄存器	0x02340024	先 read 寄存器的值，然后修改 bit4 的值 示例：ax_lookat 0x02340024 –s 0x10	clock MCLK4 enable control: 1'b0 : disable 1'b1 : enable
mclk5	clk freq 频率选择寄存器	0x02340000	先 read 寄存器的值，然后修改 bit20~bit23 的值 示例：ax_lookat	MCLK5 mux select: 4'b0000 : cpll_12m 4'b0001 : cpll_19p2m 4'b0010 : hpll_20p48m 4'b0011 : cpll_24m

			0x02340000 –s 0x300000	4'b0100 : hpll_24p576m 4'b0101 : epll_25m 4'b0110 : cpll_26m 4'b0111 : vpll0_27m 4'b1000 : vpll1_27m 4'b1001 : epll_50m 4'b1010 : vpll0_74p25m 4'b1011 : vpll1_74p25m 4'b1100 : epll_125m default : cpll_12m (4'b0000)
	clk en 使能寄存器	0x02340024	先 read 寄存器的值，然后修改 bit5 的值 示例：ax_lookat 0x02340024 –s 0x20	clock MCLK5 enable control: 1'b0 : disable 1'b1 : enable
mclk6	同 mclk0	同 mclk0	MCLK6(AX620Q 没有该 PIN, AX630C 才有该 PIN) 需要将 PINMUX 的 THM_AINA 配置成 MCLK6（function） ax_lookat 0x02301018 -s 0x00020003,其他配置同 mclk0	同 mclk0
mclk7	同 mclk1	同 mclk1	需要将 PINMUX 的 uart1_rxd 配置成 MCLK7（function）	同 mclk1

			ax_lookat 0x02304060 -s 0x00010083 其他配置同 mclk1	

2.3.3 Sensor Stream on 开流

完成 pfn_sensor_streaming_ctrl 回调函数的赋值，该函数用于控制 sensor 开关流，需要在 VIN 和 ISP 使能完成之后调用，支持动态配置。

2.3.4 调试 Sensor 输出

如何确定 sensor 有没有输出数据？可以通过查看 VIN Proc 状态信息判断，如果帧计数一直在增加，说明 sensor 出流正常。

2.4 调试 AE 控制接口

自动曝光功能主要包含三个部分，图像亮度统计分析、AE 算法计算、控制 sensor，sensor 驱动作为最后一环，负责将算法输出的曝光参数配置给 sensor，这个过程包含两个重要的任务：寄存器配置时序和曝光参数到寄存器的转换。

用户需要采用合适的方法来确定 sensor 驱动中 AE 控制的准确性。

2.4.1 开发流程

AE 相关的接口，请按照如下的顺序实现：

pfn_sensor_get_sns_reg_info

pfn_sensor_get_hw_exposure_params

pfn_sensor_set_again

pfn_sensor_set_dgain

pfn_sensor_set_integration_time

pfn_sensor_get_integration_time_range

pfn_sensor_hcglcg_ctrl

2.4.2 注意事项

SDK 提供的 AE 控制回调函数具备兼容性，支持用户将 gain 参数解析、曝光时间计算（或者长短帧曝光限制的处理）等逻辑放在驱动中实现，此时用户需要实现 pfn_sensor_get_gain_table 接口；也支持用户将 gain 和 shutter 相关的计算放在算法内部实现，此时用户不需要实现 pfn_sensor_get_gain_table 接口。

以 SDK 提供的源码为例，将 gain 和 shutter 相关的计算放在 sensor 驱动中，开发流程如下：

步骤1　首先实现参数获取接口 pfn_sensor_get_params，将参数传递到 AE 算法库中；

步骤2　然后实现 pfn_sensor_get_sns_reg_info，更新寄存器配置表。除了实现该接口之外，还需要在 sensor settings 配置完成之后，完成寄存器配置表的初始化赋值（如 VMAX、HMAX 等寄存器赋值）；另外还需要用户实现一个更新寄存器配置表的接口，用户在需要配置曝光相关的寄存器时，将寄存器更新到配置表中，代替真正的写寄存器。详见 sensor 驱动的开源代码实例。

步骤3　然后查阅 sensor datasheet，查找 HCG/LCG mode、gain table、曝光时间的计算方法、HDR 模式下长短帧的曝光时间限制等内容，并实现 pfn_sensor_set_again、pfn_sensor_set_dgain、pfn_sensor_set_integration_time、pfn_sensor_hcglcg_ctrl 四个接口；

步骤4　以上接口完成之后，用户需要验证寄存器配置和参数配置是否生效，进一步地，用户可以确认曝光时间的配置与寄存器的值是否对应（可通过 sensor 厂商提供的计算表确认）。

2.5 调试图像默认参数接口

默认参数包含两个接口：获取模块默认参数接口和获取 sensor black level 接口。用户可以将不同 HDR 模式下的参数存放在不同的头文件中，完成参数赋值，然后将函数赋值给 pfn_sensor_get_default_params 回调接口。

sensor 固有的 black level 值可以赋值给 pfn_sensor_get_black_level，ISP Firmware 中会调用该接口，以实现参数的配置生效。

2.6 图像质量调优

请参阅相关的文档：

《AX 图像在线调试指南.docx》和《AXIQ 工具使用指南.docx》

2.7 Sample_vin 新增 sensor 步骤

下面以新增 OS04A10 为例，用户需要完成如下四个步骤，这四个步骤对应的详细代码修改见 4.7.1、4.7.2、4.7.3 章节

（1） 配置 Sensor 对应的 common pool

（2） 配置 Sensor、VIN、CHN 的参数

（3） 注册 Sensor 库

（4） common 子函数中添加对应的 Sensor Case

2.7.1 sample_vin.c 修改

步骤1 新增 SYS_CASE_SINGLE_OS04A10

```
typedef enum {
    SAMPLE_VIN_NONE = -1,
    SAMPLE_VIN_SINGLE_DUMMY = 0,
    SAMPLE_VIN_SINGLE_OS04A10 = 1,
    SYS_CASE_SINGLE_DVP = 20,
    SYS_CASE_SINGLE_BT601 = 21,
    SYS_CASE_SINGLE_BT656= 22,
    SYS_CASE_SINGLE_BT1120 = 23,
    SYS_CASE_SINGLE_LVDS = 24,
    SAMPLE_VIN_BUTT
} SAMPLE_VIN_CASE_E;
```

步骤2　新增 sensor 对应的 buff pool 并配置 gtSysCommPoolSingleOs04a10

```
COMMON_SYS_POOL_CFG_T gtSysCommPoolSingleOs04a10Sdr[] = {
    [0]={.nWidth=2688, .nHeight=1520, .nWidthStride=2688, .nFmt=AX_FORMAT_YUV420_SEMIPLANAR, .nBlkCnt=3},    /* vin nv21/nv21 use */
};

COMMON_SYS_POOL_CFG_T gtPrivatePoolSingleOs04a10Sdr[] = {
    [0]={.nWidth=2688, .nHeight=1520, .nWidthStride=2688, .nFmt=AX_FORMAT_BAYER_RAW_10BPP_PACKED, .nBlkCnt=8},    /* vin raw10 use */
};
```

```
case SAMPLE_VIN_SINGLE_OS04A10:
    eSnsType = OMNIVISION_OS04A10;
    /* comm pool config */
    __cal_dump_pool(pool: gtSysCommPoolSingleOs04a10Sdr, eHdrMode: pVinParam->eHdrMode, nFrameNum: pVinParam->nDumpFrameNum);
    pCommonArgs->nPoolCfgCnt = sizeof(gtSysCommPoolSingleOs04a10Sdr) / sizeof(gtSysCommPoolSingleOs04a10Sdr[0]);
    pCommonArgs->pPoolCfg = gtSysCommPoolSingleOs04a10Sdr;

    /* private pool config */
    __cal_dump_pool(pool: gtPrivatePoolSingleOs04a10Sdr, eHdrMode: pVinParam->eHdrMode, nFrameNum: pVinParam->nDumpFrameNum);
    pPrivArgs->nPoolCfgCnt = sizeof(gtPrivatePoolSingleOs04a10Sdr) / sizeof(gtPrivatePoolSingleOs04a10Sdr[0]);
    pPrivArgs->pPoolCfg = gtPrivatePoolSingleOs04a10Sdr;

    /* cams config */
    __sample_case_single_os04a10(pCamList, eSnsType, pVinParam, pCommonArgs);
    break;
```

步骤3　配置 Sensor、VIN、Chn 的属性

```
static AX_U32 __sample_case_single_os04a10(AX_CAMERA_T *pCamList, SAMPLE_SNS_TYPE_E eSnsType,
        SAMPLE_VIN_PARAM_T *pVinParam, COMMON_SYS_ARGS_T *pCommonArgs)
{
    AX_CAMERA_T *pCam = NULL;
    COMMON_VIN_MODE_E eSysMode = pVinParam->eSysMode;
    AX_SNS_HDR_MODE_E eHdrMode = pVinParam->eHdrMode;
    AX_S32 j = 0;
    SAMPLE_LOAD_RAW_NODE_E eLoadRawNode = pVinParam->eLoadRawNode;
    pCommonArgs->nCamCnt = 1;
    pCam = &pCamList[0];
    COMMON_VIN_GetSnsConfig(eSnsType, ptMipiAttr: &pCam->tMipiAttr, ptSnsAttr: &pCam->tSnsAttr,
                            ptSnsClkAttr: &pCam->tSnsClkAttr, pDevAttr: &pCam->tDevAttr,
                            pPipeAttr: &pCam->tPipeAttr, pChnAttr: pCam->tChnAttr);
    pCam->nDevId = 0;
    pCam->nRxDev = AX_MIPI_RX_DEV_0;
    pCam->nPipeId = 0;
    pCam->tSnsClkAttr.nSnsClkIdx = 0;
    pCam->tDevBindPipe.nNum = 1;
    pCam->tDevBindPipe.nPipeId[0] = pCam->nPipeId;
    pCam->eLoadRawNode = eLoadRawNode;
    __set_pipe_hdr_mode(pHdrSel: &pCam->tDevBindPipe.nHDRSel[0], eHdrMode);
    __set_vin_attr(pCam, eSnsType, eHdrMode, eSysMode, bAiispEnable: pVinParam->bAiispEnable);
```

2.7.2 common_vin.c 修改

步骤1　注册对应的 Sensor 库，确认已经有对应的 Sensor 库

```
const static AX_SENSOR_LIB_TAB s_libSensorTab[] = {
    [0]={.eSnsType=SAMPLE_SNS_DUMMY,          .libSnsName="libsns_dummy.so",      .pSnsObjName="gSnsdummyObj"},
    [1]={.eSnsType=OMNIVISION_OS04A10,        .libSnsName="libsns_os04a10.so",    .pSnsObjName="gSnsos04a10Obj"},
    [2]={.eSnsType=SAMPLE_SNS_DVP,            .libSnsName="libsns_sc4210.so",     .pSnsObjName="gSnssc4210Obj"},
    [3]={.eSnsType=SAMPLE_SNS_TYPE_BUTT,      .libSnsName="NULL",                 .pSnsObjName="NULL"},
};
```

步骤2　配置 Sensor 的硬件接口

```
static AX_SNS_CONNECT_TYPE_E COMMON_ISP_GetSnsBusType(SAMPLE_SNS_TYPE_E eSnsType)
{
    AX_SNS_CONNECT_TYPE_E enBusType;

    switch (eSnsType) {
    case SAMPLE_SNS_DUMMY:
    case OMNIVISION_OS04A10:
    case SAMPLE_SNS_TYPE_BUTT:
        enBusType = ISP_SNS_CONNECT_I2C_TYPE;
        break;
    default:
        enBusType = ISP_SNS_CONNECT_I2C_TYPE;
        break;
    }

    return enBusType;
}
```

步骤3　配置 mipi 的属性，注意这里面的全局变量定义参照 common_config.h

```
case OMNIVISION_OS04A10:
    memcpy(dest: ptMipiAttr, src: &gOs04a10MipiAttr, n: sizeof(AX_MIPI_RX_ATTR_T));
    memcpy(dest: ptSnsAttr, src: &gOs04a10SnsAttr, n: sizeof(AX_SNS_ATTR_T));
    memcpy(dest: ptSnsClkAttr, src: &gOs04a10SnsClkAttr, n: sizeof(AX_SNS_CLK_ATTR_T));
    memcpy(dest: pDevAttr, src: &gOs04a10DevAttr, n: sizeof(AX_VIN_DEV_ATTR_T));
    memcpy(dest: pPipeAttr, src: &gOs04a10PipeAttr, n: sizeof(AX_VIN_PIPE_ATTR_T));
    memcpy(dest: &pChnAttr[0], src: &gOs04a10Chn0Attr, n: sizeof(AX_VIN_CHN_ATTR_T));
    break;
```

步骤4　获取 Sensor、Dev、Pipe、Chn 的属性，注意这里面的全局变量定义参照

common_config.h

```
case OMNIVISION_OS04A10:
    memcpy(dest: ptMipiAttr, src: &gOs04a10MipiAttr, n: sizeof(AX_MIPI_RX_ATTR_T));
    memcpy(dest: ptSnsAttr, src: &gOs04a10SnsAttr, n: sizeof(AX_SNS_ATTR_T));
    memcpy(dest: ptSnsClkAttr, src: &gOs04a10SnsClkAttr, n: sizeof(AX_SNS_CLK_ATTR_T));
    memcpy(dest: pDevAttr, src: &gOs04a10DevAttr, n: sizeof(AX_VIN_DEV_ATTR_T));
    memcpy(dest: pPipeAttr, src: &gOs04a10PipeAttr, n: sizeof(AX_VIN_PIPE_ATTR_T));
    memcpy(dest: &pChnAttr[0], src: &gOs04a10Chn0Attr, n: sizeof(AX_VIN_CHN_ATTR_T));
    break;
```

步骤5　完善通道属性获取，注意这里面的全局变量定义参照 common_config.h

```
AX_VIN_CHN_ATTR_T gOs04a10Chn0Attr = {
    .nWidth = 2688,
    .nHeight = 1520,
    .nWidthStride = 2688,
    .eImgFormat = AX_FORMAT_YUV420_SEMIPLANAR,
    .nDepth = 1,
    .tCompressInfo = {.enCompressMode=AX_COMPRESS_MODE_NONE, .u32CompressLevel=0},
    .tFrameRateCtrl = {.fSrcFrameRate=AX_INVALID_FRMRATE, .fDstFrameRate=AX_INVALID_FRMRATE},
};
```

2.7.3 common_vin.h 修改

```
typedef enum {
    SAMPLE_SNS_TYPE_NONE = -1,
    /* ov sensor */
    OMNIVISION_OS04A10 = 1,
    /* dvp sensor */
    SAMPLE_SNS_DVP = 50,
    /* bt sensor */
    SAMPLE_SNS_BT601 = 55,
    SAMPLE_SNS_BT656 = 56,
    SAMPLE_SNS_BT1120 = 57,
    /* lvds sensor */
    SAMPLE_SNS_LVDS = 60,

    /* dummy sensor */
    SAMPLE_SNS_DUMMY = 100,
    SAMPLE_SNS_TYPE_BUTT,
} SAMPLE_SNS_TYPE_E;
```

3 Sensor Callback API 参考

3.1 功能描述

本章节主要介绍 sensor 驱动对外提供的一系列回调函数，这些回调函数集统一整理在 AX_SENSOR_REGISTER_FUNC_T 结构体中，sensor 驱动头文件中以函数指针的形式定义了这些回调函数。为方便用户快速理解，下文将一一展开介绍。

3.2 Sensor 控制 callback 函数

pfn_sensor_chipid

【描述】

获取 sensor 的 chip id。

【语法】

```
AX_S32(*pfn_sensor_chipid)(ISP_PIPE_ID nPipeId, int *pSnsChipId);
```

【参数】

参数名称	描述	输入/输出
ISP_PIPE_ID nPipeId	PIPE Id	输入
AX_S32 *pSnsChipId	Sensor Id	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 该接口可作为 sensor 的识别校验，方便用户判断 sensor 类型是否匹配。

【举例】

无

【相关主题】

无

pfn_sensor_reset

【描述】

Sensor 复位。

【语法】

```
AX_VOID(*pfn_sensor_reset)(ISP_PIPE_ID nPipeId, AX_U32 nResetGpio)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
nResetGpio	Sensor 硬复位管脚 ID	输入

【返回值】

返回值	描述
无	空

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 该接口配合 MIPI 的复位和初始化使用

【举例】

无

【相关主题】

无

pfn_sensor_init

【描述】

sensor 初始化。

【语法】

```
AX_VOID(*pfn_sensor_init)(ISP_PIPE_ID nPipeId)
```

【参数】

参数名称	描述	输入/输出
ISP_PIPE_ID nPipeId	Pipe Id	输入

【返回值】

返回值	描述
无	空

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 该接口没有返回值，ISP FW 内部不做返回值的判断。
- 用户如果自己实现 sensor 控制部分的驱动，可以不使用 AX 提供的 sensor 控制部分的 callback，但是也请实现该接口，以完成 sensor 驱动代码中 context 结构的初始化，这样做能保证 AE 控制相关的 callback 接口能够正常工作。

【举例】

无

【相关主题】

无

pfn_sensor_exit

【描述】

sensor 退出。

【语法】

```
AX_VOID(*pfn_sensor_exit)(ISP_PIPE_ID nPipeId)
```

【参数】

参数名称	描述	输入/输出
ISP_PIPE_ID nPipeId	PIPE Id	输入

【返回值】

返回值	描述
无	空

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

无

【举例】

无

【相关主题】

无

pfn_sensor_streaming_ctrl

【描述】

sensor 的开关流控制。

【语法】

```
AX_S32(*pfn_sensor_streaming_ctrl)(ISP_PIPE_ID nPipeId, AX_U32 on)
```

【参数】

参数名称	描述	输入/输出
ISP_PIPE_ID nPipeId	PIPE Id	输入
AX_U32 on	0-关流，1-开流	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 需要在 sensor init 完成之后调用，否则该配置可能会被 sensor 初始化序列覆盖。

【举例】

无

【相关主题】

无

pfn_sensor_set_slaveaddr

【描述】

设置 sensor i2c slave addr

【语法】

```
AX_S32(*pfn_sensor_set_slaveaddr)(ISP_PIPE_ID nPipeId, AX_U8 nSlaveAddr)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
nSlaveAddr	设置 sensor i2c slave addr	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

无

【举例】

无

pfn_sensor_testpattern

【描述】

sensor 的测试模式开关控制。

【语法】

```
AX_S32(*pfn_sensor_testpattern)(ISP_PIPE_ID nPipeId, AX_U32 nTestpatternEn)
```

【参数】

参数名称	描述	输入/输出
ISP_PIPE_ID nPipeId	PIPE Id	输入
AX_U32 nTestpatternEn	是否使能测试模式，0-不使能，1-使能	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 通过 AX_ISP_SetSnsAttr 配置是否使能 test pattern 模式，ISP FW 内部会隐式调用该函数完成配置。
- 支持用户在自己的应用层程序中直接调用该函数。

【举例】

无

【相关主题】

无

pfn_sensor_set_mode

【描述】

设置 sensor 的运行模式。

【语法】

```
AX_S32(*pfn_sensor_set_mode)(ISP_PIPE_ID nPipeId, AX_SNS_ATTR_T *ptSnsMode)
```

【参数】

参数名称	描述	输入/输出
nPipeId	pipe Id	输入
ptSnsMode	Sensor 参数配置	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- Sensor 参数通过 AX_ISP_SetSnsAttr 配置，在 AX_ISP_Create()之前调用。

【举例】

无

【相关主题】

无

pfn_sensor_get_mode

【描述】

获取 sensor 的参数。

【语法】

```
AX_S32(*pfn_sensor_get_mode)(ISP_PIPE_ID nPipeId, AX_SNS_ATTR_T *ptSnsMode)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
ptSnsMode	Sensor 参数信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

无

【举例】

无

【相关主题】

无

pfn_sensor_mirror_flip

【描述】

配置 sensor 端的镜像翻转属性。

【语法】

```
AX_S32(*pfn_sensor_mirror_flip)(ISP_PIPE_ID nPipeId, AX_SNS_MIRRORFLIP_TYPE_E eSnsMirrorFlip)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
eSnsMirrorFlip	Sensor 端的镜像翻转属性	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 镜像旋转后，图像颜色可能会异常，原因可能是图像格式发生了变化，可以通过 x y 的像素偏移来实现矫正（有些 sensor 旋转后 bayer partten 不发生变化）。

【举例】

无

【相关主题】

无

pfn_sensor_sleep_wakeup

【描述】

配置 sensor 睡眠唤醒。

【语法】

```
AX_S32(*pfn_sensor_sleep_wakeup)(ISP_PIPE_ID nPipeId, AX_SNS_SLEEP_WAKEUP_E
eSleepWakeup)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
eSleepWakeup	Sensor 睡眠唤醒状态	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 睡眠状态不同 sensor 实现的方式不同，默认采用寄存器不清除的情况，使 sensor 进入睡眠低功耗状态，唤醒时无需重新写入 setting。

【举例】

无

pfn_sensor_set_fps

【描述】

初始化配置 sensor 的帧率或 P/N 制切换时使用。

【语法】

```
AX_S32(*pfn_sensor_set_fps)(ISP_PIPE_ID nPipeId, AX_F32 fFps)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
fFps	Sensor 帧率的写入数值	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 此接口为内部调用，其他情况不建议使用。

【举例】

无

【相关主题】

无

pfn_sensor_get_fps

【描述】

获取 sensor 当前的帧率。

【语法】

```
AX_S32(*pfn_sensor_get_fps)(ISP_PIPE_ID nPipeId, AX_F32 *pFps)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
*pFps	Sensor 帧率的读取数值	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 此接口为内部调用，其他情况不建议使用。

【举例】

无

【相关主题】

无

pfn_sensor_set_bus_info

【描述】

设置和 sensor 通信的方式。

【语法】

```
AX_S32(*pfn_sensor_set_bus_info)(ISP_PIPE_ID nPipeId, AX_SNS_COMMBUS_T tSnsBusInfo)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
tSnsBusInfo	设置和 sensor 通信的方式 i2c 或者 spi	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

无

【举例】

- 必须在 `pfn_sensor_init` 函数之前配置

【相关主题】

无

3.3 Sensor 默认参数 callback 函数

pfn_sensor_get_default_params

【描述】

获取 sensor 各个算法模块的默认参数。

【语法】

```
AX_S32(*pfn_sensor_get_default_params)(ISP_PIPE_ID nPipeId,
AX_SENSOR_DEFAULT_PARAM_T *ptDftParam)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
ptDftParam	Sensor 各模块默认参数配置	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- Sensor 驱动中，将 SDR 模式和 HDR 模式的默认参数分开存储在两个头文件中，变量命名添加 static 字段，以限定变量的使用范围，默认参数可通过 ispTuning 工具导出，用户可自定义文件的名称。
- 默认参数中也包含自研 3A 算法模块的参数。

【举例】

无

【相关主题】

无

pfn_sensor_get_black_level

【描述】

获取 sensor 的黑电平值。

【语法】

```
AX_S32(*pfn_sensor_get_black_level)(ISP_PIPE_ID nPipeId, AX_SNS_BLACK_LEVEL_T *ptBlackLevel)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
ptBlackLevel	Sensor 黑电平值	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 该接口用于填写 sensor 自己的 black level 值，以方便 ISP 各个图像效果模块使用。
- Black level 值的精度为 U8.6。

【举例】

无

【相关主题】

无

3.4 AE 控制 callback 函数

pfn_sensor_get_hw_exposure_params

【描述】

获取 sensor 曝光控制相关的参数。

【语法】

```
AX_S32(*pfn_sensor_get_hw_exposure_params)(ISP_PIPE_ID nPipeId,
AX_SNS_EXP_CTRL_PARAM_T *ptAeCtrlParam)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
ptSnsParam	Sensor 曝光控制相关的参数	输出

【返回值】

返回值	描述
0	成功

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 用户可以使用该接口，将 sensor 的 gain、shutter 相关的最大最小值传递给 AE 算法。

【举例】

无

【相关主题】

无

pfn_sensor_get_gain_table

【描述】

获取 sensor 的 gain 值表。

【语法】

```
AX_S32(*pfn_sensor_get_gain_table)(ISP_PIPE_ID nPipeId,
AX_SNS_AE_GAIN_TABLE_T *ptSnsGainTbl)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
ptSnsGainTbl	Gain table	输出

【返回值】

返回值	描述
0	成功

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 提供更为灵活的使用方法，用户的 AE 算法可通过该接口获取 gain 配置表，也可以在 sensor 驱动中使用 gain 表。

【举例】

无

【相关主题】

无

pfn_sensor_set_again

【描述】

设置 sensor 的模拟增益寄存器。

【语法】

```
AX_S32(*pfn_sensor_set_again)(ISP_PIPE_ID nPipeId, AX_SNS_AE_GAIN_CFG_T *
ptAGain)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
ptAGain	Sensor again 值	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- HDR 模式下，支持独立配置长短帧的 gain value。
- 增益的单位是倍数。
- 增益有三种表示方法：倍数、分贝（db）、寄存器值，增益与分贝之间的转换关系有标准公式，倍数与寄存器之间的转换公式一般在 sensor datasheet 文档中有说明。多数 sensor 厂家会给出一个 gain table 表，可以将 table 表维护在 AX 驱动源码中，提高代码的可读性。

【举例】

无

【相关主题】

无

pfn_sensor_set_dgain

【描述】

设置 sensor 的数字增益寄存器。

【语法】

```
AX_S32(*pfn_sensor_set_dgain)(ISP_PIPE_ID nPipeId, AX_SNS_AE_GAIN_CFG_T *
ptDGain)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
ptDGain	Sensor dgain 值	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- HDR 模式下，支持独立配置长短帧的 digital gain value。

- 增益的单位是倍数。
- 增益有三种表示方法：倍数、分贝（db）、寄存器值，增益与分贝之间的转换关系有标准公式，倍数与寄存器之间的转换公式一般在 sensor datasheet 文档中有说明。多数 sensor 厂家会给出一个 gain table 表，可以将 table 表维护在 AX 驱动源码中，提高代码的可读性。
- 如果 sensor 不支持 dgain，该回调函数可以注册为 NULL。

【举例】

无

【相关主题】

无

pfn_sensor_hcglcg_ctrl

【描述】

控制 sensor 的 hcg/lcg 切换。

【语法】

```
AX_S32(*pfn_sensor_hcglcg_ctrl)(ISP_PIPE_ID nPipeId, AX_U32 nSnsHcgLcg)
```

【参数】

参数名称	描述	输入/输出
PipeId	PIPE Id	输入
nSnsHcgLcg	模式，0 表示 HCG，1 表示 LCG，详见 AX_SNS_HCGLCG_MODE_E	输入

【返回值】

返回值	描述
0	成功

返回值	描述
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 用户需要查阅手册，确定 sensor 是否支持 HCG/LCG 两种模式，不支持时，用户需要配置 eSnsHcgLcgMode 属性，将模式设置为不支持的模式。
- HCG/LCG 切换的寄存器，其生效时机可能与 gain/shutter 寄存器不同，如 HCG 寄存器可能当帧生效，但是 gain/shutter 寄存器可能下一帧生效，此时需要用户延时一帧配置 HCG 寄存器，以保证所有寄存器都在下一帧生效。推荐用户使用 AX 的 AX_SNS_REGS_CFG_TABLE_T 寄存器配置表，以实现延时配置的功能。用户也可以利用 sensor 的 group hold 机制来实现 HCG 寄存器的配置。详细实现请参考 SDK 提供的驱动代码。

【举例】

无

【相关主题】

无

pfn_sensor_set_integration_time

【描述】

设置 sensor 的曝光时间，支持一次性配置长帧、短帧的曝光时间和曝光比。

【语法】

```
AX_S32(*pfn_sensor_set_integration_time)(ISP_PIPE_ID nPipeId,
AX_SNS_AE_SHUTTER_CFG_T *ptIntTime)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
ptIntTimeTbl	Sensor 的曝光时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 曝光时间的单位是行，使用浮点型表示，曝光比使用浮点型表示。
- 如果是 HDR 3DOL 模式，支持配置长短帧的曝光比、短帧和超短帧的曝光比。

【举例】

无

【相关主题】

无

pfn_sensor_get_integration_time_range

【描述】

获取 sensor 的曝光范围参数，曝光最大最小曝光行。

【语法】

```
AX_S32(*pfn_sensor_get_integration_time_range)(ISP_PIPE_ID nPipeId, AX_F32
fHdrRatio, AX_SNS_AE_INT_TIME_RANGE_T *ptIntTimeRange)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
fHdrRatio	Hdr Ratio	输入
ptIntTimeRange	曝光范围参数	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- HDR 模式下，曝光时间的范围与曝光比参数有关，SDR 模式下，曝光时间的范围与用户初始化序列中配置的 VTS 寄存器有关。

【举例】

无

【相关主题】

无

pfn_sensor_set_slow_fps

【描述】

设置 sensor 的慢快门模式下的帧率。

【语法】

```
AX_S32(*pfn_sensor_set_slow_fps)(ISP_PIPE_ID nPipeId, AX_F32 fFps)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
fFps	Sensor 的慢快门参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

此接口为 AE 算法内部调用，其他情况不建议使用。

【举例】

无

【相关主题】

无

pfn_sensor_get_slow_shutter_param

【描述】

AE 算法获取 sensor 慢快门挡位参数的接口。

【语法】

```
AX_S32(*pfn_sensor_get_slow_shutter_param)(ISP_PIPE_ID nPipeId,
AX_SNS_AE_SLOW_SHUTTER_PARAM_T *ptSlowShutterParam)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
ptSlowShutterParam	Sensor 的慢快门参数	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 该接口支持用户自定义挡位，挡位范围最大值为当前初始默认配置的帧率，数值整数小数部分都可以设置。

【举例】

此接口用户可以自定义设置帧率挡位，具体可以根据实际情况去设置，示例如下：

```
static AX_F32 gFpsGear[] = { 5.00, 10.00, 15.00, 20.00, 25.00 };
```

挡位设置大小顺序推荐从小到大排列，此处示例 config 可以在代码中找到，对应进行配置即可。

【相关主题】

无

pfn_sensor_get_sns_reg_info

【描述】

isp firmware 获取 sensor 驱动中曝光相关的寄存器信息，然后将寄存器包传递到内核态，由内核态完成时序控制，保证寄存器的配置在同一帧内完成。

【语法】

```
AX_S32(*pfn_sensor_get_sns_reg_info)(ISP_PIPE_ID nPipeId,
AX_SNS_REGS_CFG_TABLE_T *ptSnsRegsInfo);
```

【参数】

参数名称	描述	输入/输出
PipeId	PIPE Id	输入
ptSnsRegsInfo	Sensor 曝光相关的寄存器信息	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

- 该函数接口用于配置需要确保同步性的 sensor 寄存器，如曝光时间、增益及总行数等。
- 如果寄存器配置时序由用户自己保证，用户可以将该回调接口置为 NULL。
- 曝光、增益、帧长等寄存器等也可以在用户态 sensor drv 程序中直接配置，但无法保证同步性，可能出现闪烁。

- nDataByteNum 是寄存器配置延时帧数。很多 sensor 的增益是下一帧生效，但曝光时间是下下帧生效，所以需要增益晚一帧配置，以使增益和曝光时间同时生效，这时就需要将 gain 寄存器延时一帧配置，以保证同步性。

【举例】

无

【相关主题】

无

pfn_sensor_set_lfhdr_mode

【描述】

设置 sensor 配置长帧模式。

【语法】

```
AX_S32(*pfn_sensor_set_lfhdr_mode)(ISP_PIPE_ID nPipeId,
AX_SNS_AE_LFHDR_ATTR_T *ptLFHdrAttr)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
ptLFHdrAttr	配置 sensor 长帧模式	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

无

【举例】

无

【相关主题】

无

pfn_sensor_get_temperature_info

【描述】

获取 sensor 温度。

【语法】

```
AX_S32(*pfn_sensor_get_temperature_info)(ISP_PIPE_ID nPipeId, AX_S32
*nTemperature)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
nTemperature	获取 sensor 温度	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

无

【举例】

无

【相关主题】

无

pfn_sensor_set_wbgain

【描述】

设置 sensor wbgain。

【语法】

```
AX_S32(*pfn_sensor_set_wbgain)(ISP_PIPE_ID nPipeId, AX_SNS_AWB_PARAM_T
*ptAWBresult)
```

【参数】

参数名称	描述	输入/输出
nPipeId	PIPE Id	输入
ptAWBresult	设置 sensor awb 参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

- 头文件：ax_sensor_struct.h
- 库文件：libsns_xxx.so（xxx 代表用户 sensor 名称）

【注意】

无

【举例】

无

【相关主题】

无

4 数据结构

HDR_MAX_FRAME_NUM

说明】

Sensor hdr 支持最大的 frame num。

【定义】

AX650A/N

#define HDR_MAX_FRAME_NUM (4)

AX630C

#define HDR_MAX_FRAME_NUM (4)

AX620Q

#define HDR_MAX_FRAME_NUM (4)

【芯片差异】

无

【注意】

无

ISP_MAX_SNS_REGISTER_NUM

【说明】

Sensor 内核写寄存器最大数量。

【定义】

AX650A/N

#define ISP_MAX_SNS_REGISTER_NUM (32)

AX630C

#define ISP_MAX_SNS_REGISTER_NUM (32)

AX620Q

#define ISP_MAX_SNS_REGISTER_NUM (32)

【芯片差异】

无

【注意】

无

SNS_MAX_FRAME_RATE

【说明】

Sensor 可以设置的最大帧率。

【定义】

AX650A/N

#define SNS_MAX_FRAME_RATE (60)

AX630C

#define SNS_MAX_FRAME_RATE (60)

AX620Q

#define SNS_MAX_FRAME_RATE (60)

【芯片差异】

无

【注意】

无

AX_SNS_HCGLCG_MODE_E

【说明】

Camera Sensor HCG/LCG 模式。

【定义】

```
typedef enum _AX_SNS_HCGLCG_MODE_E_ {

    AX_HCG_MODE             = 0,

    AX_LCG_MODE             = 1,

    AX_LCG_NOTSUPPORT_MODE  = 2,

} AX_SNS_HCGLCG_MODE_E;
```

【成员】

成员名称	描述
AX_HCG_MODE	High conversion gain 模式
AX_LCG_MODE	low conversion gain 模式
AX_LCG_NOTSUPPORT_MODE	不支持 HCG LCG

【注意】

有些 Sensor 不区分 HCG/LCG 模式，此时用户可以将该参数设置为 AX_LCG_NOTSUPPORT_MODE。

【相关数据类型及接口】

结构体：

- AX_VIN_DEV_T
- AX_SNS_ATTR_T

AX_SNS_MASTER_SLAVE_E

【说明】

sensor 的控制模式，分为主模式和从模式，从模式的 sensor，需要外部提供同步信号。

【定义】

```
typedef enum {

    AX_SNS_MASTER   = 0,

    AX_SNS_SLAVE    = 1,

    AX_SNS_SYNC_MASTER   = 2,

    AX_SNS_SYNC_SLAVE    = 3,

} AX_SNS_MASTER_SLAVE_E
```

【成员】

成员名称	描述
AX_SNS_MASTER	主模式
AX_SNS_SLAVE	从模式
AX_SNS_SYNC_MASTER	硬同步主机
AX_SNS_SYNC_SLAVE	硬同步从机

【注意】

无

【相关数据类型及接口】

AX_SNS_ATTR_T

AX_SNS_CLK_RATE_E

【说明】

Sensor 时钟频率值。

【定义】

```
typedef enum {

    AX_SNS_CLK_24M      = 0,

    AX_SNS_CLK_27M      = 1,

    AX_SNS_CLK_37_125M  = 2,

    AX_SNS_CLK_74_25M   = 3,

} AX_SNS_CLK_RATE_E
```

【成员】

成员名称	描述
AX_SNS_CLK_24M	24MHz，默认值
AX_SNS_CLK_27M	27MHz
AX_SNS_CLK_37_125M	37.125MHz
AX_SNS_CLK_74_25M	74.25MHz

【注意】

无

【相关数据类型及接口】

AX_SNS_ATTR_T

AX_SNS_ATTR_T

【说明】

Camera Sensor 属性信息。

【定义】

```
typedef struct {
    AX_U32                  nWidth;
    AX_U32                  nHeight;
    AX_F32                  fFrameRate;
    AX_SNS_HDR_MODE_E       eSnsMode;
    AX_RAW_TYPE_E           eRawType;
    AX_BAYER_PATTERN_E      eBayerPattern;
    AX_BOOL                 bTestPatternEnable;
    AX_SNS_MASTER_SLAVE_E   eMasterSlaveSel;
    AX_U32                  nSettingIndex;
    AX_SNS_CONNECT_TYPE_E   eBusType;
    AX_SNS_OUTPUT_MODE_E    eSnsOutputMode;
} AX_SNS_ATTR_T;
```

【成员】

成员名称	描述
nWidth	Sensor 输出数据的宽度
nHeight	Sensor 输出数据的高度
fFrameRate	配置帧率 fps（常见帧率如 25/30/50/60）
eSnsMode	Sensor HDR 模式选择（Linear/HDR_2X/HDR_3X），enum 类型

成员名称	描述
eRawType	Raw 数据 bit 属性，enum 类型
eBayerPattern	Bayer 模板类型，enum 类型
bTestPatternEnable	sensor 出 test pattern 数据的开关
eMasterSlaveSel	主从模式
nSettingIndex	用于进行 Sensor 初始化序列的选择，在分辨率和帧率相同时，配置不同的 nSettingIndex 对应不同的初始化序列；其他情况，nSettingIndex 默认配置为 0，可通过 nWidth、nHeight 和 nFrameRate 进行初始化序列的选择。
eBusType	sensor 通信类型，支持 I2C/SPI 两种类型
eSnsOutputMode	sensor 的输出模式

【注意】

nSettingIndex 参数，如描述所介绍，默认情况为 0，当需要配置 2lane setting.或者一些特殊的 setting 配置，就需要在 setting 的枚举列表选择对应 setting 的 index,当前的特殊 setting index 用例如“e_OS04A10_setting_special_idx = 0x20, /* index = 32 */”作为分隔，在配置加载相应 special setting,需要注意 index 数值从 32 开始计数。

【相关数据类型及接口】

枚举：

- AX_SNS_MASTER_SLAVE_E

AX_SNS_CONNECT_TYPE_E

【说明】

Sensor 通信类型，支持 I2C/SPI 两种类型。

【定义】

```
typedef enum _AX_SNS_CONNECT_TYPE_E_ {
    ISP_SNS_CONNECT_I2C_TYPE = 0,
    ISP_SNS_CONNECT_SPI_TYPE,
    ISP_SNS_CONNECT_TYPE_BUTT,
} AX_SNS_CONNECT_TYPE_E
```

【成员】

成员名称	描述
ISP_SNS_CONNECT_I2C_TYPE	通信方式为 I2C
ISP_SNS_CONNECT_SPI_TYPE	通信方式为 SPI

【注意】

无

【相关数据类型及接口】

AX_SNS_AE_GAIN_TABLE_T

【说明】

AE 增益配置表。

【定义】

```
typedef struct _AX_SNS_AE_GAIN_TABLE_T_ {
    AX_S32  nAgainTableSize;
    AX_F32 *pAgainTable;
    AX_S32  nDgainTableSize;
    AX_F32 *pDgainTable;
} AX_SNS_AE_GAIN_TABLE_T;
```

【成员】

成员名称	描述
nAgainTableSize	模拟增益表的大小
pAgainTable	模拟增益表
nDgainTableSize	数字增益表的大小
pDgainTable	数字增益表

【注意】

无

【相关数据类型及接口】

AX_SNS_AE_LIMIT_T

【说明】

AE 增益和曝光时间的最大最小值限制。

【定义】

```
typedef struct _AX_SNS_AE_LIMIT_T_ {
        AX_F32 fMinAgain[HDR_MAX_FRAME_NUM];
        AX_F32 fMaxAgain[HDR_MAX_FRAME_NUM];
        AX_F32 fMinDgain[HDR_MAX_FRAME_NUM];
        AX_F32 fMaxDgain[HDR_MAX_FRAME_NUM];
        AX_F32 fMinratio;
        AX_F32 fMaxratio;
        AX_SNS_AE_INT_TIME_RANGE_T  tIntTimeRange;
} AX_SNS_AE_LIMIT_T;
```

【成员】

成员名称	描述
fMinAgain[HDR_MAX_FRAME_NUM]	各个通道的模拟增益最小值，单位：倍数
fMaxAgain[HDR_MAX_FRAME_NUM]	各个通道的模拟增益最大值，单位：倍数
fMinDgain [HDR_MAX_FRAME_NUM]	各个通道的数字增益最小值，单位：倍数
fMaxDgain [HDR_MAX_FRAME_NUM]	各个通道的数字增益最大值，单位：倍数
fMinratio	最小曝光比，单位：倍数
fMaxratio	最大曝光比，单位：倍数
tIntTimeRange	曝光时间范围参数，单位：行

【注意】

在 HDR 模式下，支持用户配置差异化的参数，最大支持 4 个通道的参数配置，数组下标的含义分别是：

HDR 2X 时：0-long frame, 1- short frame

HDR 3X 时：0-long frame, 1- medium frame, 2- short frame

【相关数据类型及接口】

无

AX_SNS_AE_PARAM_T

【说明】

曝光控制参数的结构体，包括 gain、曝光时间、补偿和 ratio。

【定义】

```
typedef struct _AX_SNS_AE_PARAM_T_ {
        AX_F32 fCurAGain[HDR_MAX_FRAME_NUM];
        AX_F32 fCurDGain[HDR_MAX_FRAME_NUM];
        AX_F32 fCurIspDGain[HDR_MAX_FRAME_NUM];
        AX_U32 nCurIntegrationTime[HDR_MAX_FRAME_NUM];
        AX_F32 fAGainIncrement[HDR_MAX_FRAME_NUM];
        AX_F32 fDGainIncrement[HDR_MAX_FRAME_NUM];
        AX_F32 fIspDGainIncrement[HDR_MAX_FRAME_NUM];
        AX_U32 fIntegrationTimeIncrement[HDR_MAX_FRAME_NUM];
        AX_F32 fCurFps;
        AX_F32 fIntegrationTimeOffset[HDR_MAX_FRAME_NUM];
} AX_SNS_AE_PARAM_T;
```

【成员】

成员名称	描述
fCurAGain[HDR_MAX_FRAME_NUM]	当前模拟增益值
fCurDGain[HDR_MAX_FRAME_NUM]	当前数字增益值
fCurIspDGain[HDR_MAX_FRAME_NUM]	当前 ISP gain 值
nCurIntegrationTime[HDR_MAX_FRAME_NUM]	当前积分时间，单位：行
fAGainIncrement[HDR_MAX_FRAME_NUM]	模拟增益的调节步长

成员名称	描述
fDGainIncrement[HDR_MAX_FRAME_NUM]	数字增益的调节步长
fIspDGainIncrement[HDR_MAX_FRAME_NUM]	ISP gain 的调节步长
fIntegrationTimeIncrement[HDR_MAX_FRAME_NUM]	积分时间的调节步长
fCurFps	Sensor HCG/LCG ratio
fIntegrationTimeOffset[HDR_MAX_FRAME_NUM]	Sensor 固有的曝光延迟时间

【注意】

在 HDR 模式下，支持用户配置差异化的参数，最大支持 4 个通道的参数配置，数组下标的含义分别是：

HDR 2X 时：0-long frame, 1- short frame

HDR 3X 时：0-long frame, 1- medium frame, 2- short frame, 3-无用

【相关数据类型及接口】

AX_SNS_EXP_CTRL_PARAM_T

【说明】

曝光控制参数的结构体，包括 gain、曝光时间、补偿和 ratio。

【定义】

```
typedef struct _AX_SNS_EXP_CTRL_PARAM_T_ {
    AX_SNS_ATTR_T           sns_dev_attr;
    AX_SNS_AE_LIMIT_T       sns_ae_limit;
    AX_SNS_AE_PARAM_T       sns_ae_param;
    AX_SNS_HCGLCG_MODE_E    eSnsHcgLcgMode;
    AX_F32                  fSnsHcgLcgRatio;
    AX_F32                  fTimePerLine;
    AX_U32                  nInitIntegrationTime;
    AX_F32                  fInitAGain;
    AX_F32                  fInitDGain;
} AX_SNS_EXP_CTRL_PARAM_T;
```

【成员】

成员名称	描述
sns_dev_attr	Sensor 属性参数
sns_ae_limit	曝光范围相关的参数
sns_ae_param	曝光参数的当前值和曝光调节步长的参数
nCurIspDGain[HDR_MAX_FRAME_NUM]	当前 ISP gain 值
nCurIntegrationTime[HDR_MAX_FRAME_NUM]	当前积分时间，单位：行

成员名称	描述
nAGainIncrement[HDR_MAX_FRAME_NUM]	模拟增益的调节步长
nDGainIncrement[HDR_MAX_FRAME_NUM]	数字增益的调节步长
nIspDGainIncrement[HDR_MAX_FRAME_NUM]	ISP gain 的调节步长
nIntegrationTimeIncrement[HDR_MAX_FRAME_NUM]	积分时间的调节步长
eSnsHcgLcgRatio	Sensor HCG/LCG ratio
eSnsHcgLcgMode	HCG/LCG 模式
fTimePerLine	一行的曝光时间（单位：微妙）
nInitIntegrationTime	曝光时间初始值
fInitAGain	again 初始值
fInitDGain	dgain 初始值

【注意】

- AE limit 限制参数，建议在 sensor 初始化的时候进行赋值。

【相关数据类型及接口】

AX_SNS_AE_GAIN_CFG_T

【说明】

AE 算法配置的曝光增益值和 HDR 比值，或者 AE 算法通过 get 接口获取的增益值和 HDR 比。

【定义】

```
typedef struct _AX_SNS_AE_GAIN_CFG_T_ {
    AX_F32 fGain[HDR_MAX_FRAME_NUM];
} AX_SNS_AE_GAIN_CFG_T
```

【成员】

成员名称	描述
nGain[HDR_MAX_FRAME_NUM]	增益值，单位：倍数

【注意】

无

【相关数据类型及接口】

AX_SNS_AE_SHUTTER_CFG_T

【说明】

AE 算法配置的曝光时间和曝光比，或者 AE 算法通过 get 接口获取的曝光时间和曝光比。

【定义】

```
typedef struct _AX_SNS_AE_SHUTTER_CFG_T_ {
    AX_F32 fIntTime[HDR_MAX_FRAME_NUM];
    AX_F32 fHdrRatio[HDR_MAX_FRAME_NUM];
} AX_SNS_AE_SHUTTER_CFG_T
```

【成员】

成员名称	描述
fIntTime [HDR_MAX_FRAME_NUM]	曝光时间，单位：行
fHdrRatio[HDR_MAX_FRAME_NUM]	HDR 曝光比

【注意】

在 HDR 模式下，支持用户配置差异化的曝光时间和曝光比参数，最大支持 4 个通道的参数配置，数组下标的含义分别是：

- HDR 2X 时：nHdrRatio[0]代表长帧和短帧的曝光比，即 nIntTime[0]/nIntTime[1]
- HDR 3X 时：nHdrRatio[0]同上，nHdrRatio[1]代表短帧和超短帧的曝光比，即 nIntTime[1]/ nIntTime[2]

【相关数据类型及接口】

AX_SNS_BLACK_LEVEL_T

【说明】

Sensor black level 值。

【定义】

```
typedef struct _AX_SNS_BLACK_LEVEL_T_ {
    AX_U16  nBlackLevel[AX_ISP_BAYER_CHN_NUM];
} AX_SNS_BLACK_LEVEL_T
```

【成员】

成员名称	描述
nBlackLevel	Sensor black level

【注意】

无

【相关数据类型及接口】

无

AX_SNS_COMMBUS_T

【说明】

配置 I2C 设备节点号，或者 SPI 设备节点号和片选信号。

【定义】

```
typedef union _AX_SNS_COMMBUS_T_ {
    AX_S8   I2cDev;
    struct {
        AX_S8  bit4SpiDev      : 4;
        AX_S8  bit4SpiCs       : 4;
    } SpiDev;
} AX_SNS_COMMBUS_T
```

【成员】

成员名称	描述
I2cDev	I2C 设备节点号，占用一个字节
bit4SpiDev	SPI 设备节点号，占低位 4bit
bit4SpiCs	片选信号，占高位 4bit

【注意】

无

【相关数据类型及接口】

无

AX_SENSOR_DEFAULT_PARAM_T

【说明】

ISP 模块默认参数结构体。

【定义】

AX650A/N

```
typedef struct _AX_SENSOR_DEFAULT_PARAM_T_ {
    AX_ISP_IQ_DPC_PARAM_T       *ptDpc;
    AX_ISP_IQ_BLC_PARAM_T       *ptBlc;
    AX_ISP_IQ_DS_PARAM_T        *ptDarkshading;
    AX_ISP_IQ_FPN_PARAM_T       *ptFpn;
    AX_ISP_IQ_GBL_PARAM_T       *ptGbl;

    AX_ISP_IQ_LSC_PARAM_T       *ptLsc;
    AX_ISP_IQ_WB_GAIN_PARAM_T   *ptWbGain;
    AX_ISP_IQ_RLTM_PARAM_T      *ptRltm;
    AX_ISP_IQ_DEMOSAIC_PARAM_T  *ptDemosaic;
    AX_ISP_IQ_GIC_PARAM_T       *ptGic;
    AX_ISP_IQ_CC_PARAM_T        *ptCc;
    AX_ISP_IQ_3DLUT_PARAM_T     *pt3Dlut;
    AX_ISP_IQ_GAMMA_PARAM_T     *ptGamma;
    AX_ISP_IQ_CA_PARAM_T        *ptCa;
    AX_ISP_IQ_CSC_PARAM_T       *ptCsc;
```

```
    AX_ISP_IQ_DEPURPLE_PARAM_T  *ptDepurple;
    AX_ISP_IQ_HDR_PARAM_T       *ptHdr;
    AX_ISP_IQ_SHARPEN_PARAM_T   *ptSharpen;
    AX_ISP_IQ_SCM_PARAM_T       *ptScm;
    AX_ISP_IQ_YNR_PARAM_T       *ptYnr;
    AX_ISP_IQ_YCPROC_PARAM_T    *ptYcproc;
    AX_ISP_IQ_CNR_PARAM_T       *ptCnr;
    AX_ISP_IQ_CCMP_PARAM_T      *ptCcmp;
    AX_ISP_IQ_YCRT_PARAM_T      *ptYcrt;
    AX_ISP_IQ_MDE_PARAM_T       *ptMde;
    AX_ISP_IQ_AYNR_PARAM_T      *ptAYnr;
    AX_ISP_IQ_ACNR_PARAM_T      *ptACnr;
    AX_ISP_IQ_RAW3DNR_PARAM_T   *ptRaw3dnr;
    AX_ISP_IQ_AINR_PARAM_T      *ptAinr;
    AX_ISP_IQ_AICE_PARAM_T      *ptAice;
    AX_ISP_IQ_SCENE_PARAM_T    *ptScene;
    AX_ISP_IQ_DEHAZE_PARAM_T    *ptDehaze;
    AX_ISP_IQ_AE_PARAM_T        *ptAeDftParam;
    AX_ISP_IQ_AWB_PARAM_T       *ptAwbDftParam;
    AX_ISP_IQ_LDC_PARAM_T       *ptLdc;
    AX_ISP_IQ_DIS_PARAM_T       *ptDis;
    AX_ISP_IQ_ME_PARAM_T        *ptMe;
} AX_SENSOR_DEFAULT_PARAM_T
```

AX630C/620Q

```
typedef struct _AX_SENSOR_DEFAULT_PARAM_T_ {
    AX_ISP_IQ_DPC_PARAM_T       *ptDpc;
    AX_ISP_IQ_BLC_PARAM_T       *ptBlc;

    AX_ISP_IQ_LSC_PARAM_T       *ptLsc;
    AX_ISP_IQ_WB_GAIN_PARAM_T   *ptWbGain;
    AX_ISP_IQ_RLTM_PARAM_T      *ptRltm;
    AX_ISP_IQ_DEHAZE_PARAM_T    *ptDehaze;
    AX_ISP_IQ_DEMOSAIC_PARAM_T  *ptDemosaic;
    AX_ISP_IQ_FCC_PARAM_T       *ptFcc;
    AX_ISP_IQ_GIC_PARAM_T       *ptGic;
    AX_ISP_IQ_CC_PARAM_T        *ptCc;
    AX_ISP_IQ_GAMMA_PARAM_T     *ptGamma;
    AX_ISP_IQ_CA_PARAM_T        *ptCa;
    AX_ISP_IQ_CSC_PARAM_T       *ptCsc;
    AX_ISP_IQ_DEPURPLE_PARAM_T  *ptDepurple;
    AX_ISP_IQ_HDR_PARAM_T       *ptHdr;
    AX_ISP_IQ_SHARPEN_PARAM_T   *ptSharpen;
    AX_ISP_IQ_SCM_PARAM_T       *ptScm;
    AX_ISP_IQ_YNR_PARAM_T       *ptYnr;
    AX_ISP_IQ_YCPROC_PARAM_T    *ptYcproc;
    AX_ISP_IQ_CNR_PARAM_T       *ptCnr;
```

```
    AX_ISP_IQ_CCMP_PARAM_T      *ptCcmp;
    AX_ISP_IQ_HS2DLUT_PARAM_T   *ptHs2dlut;
    AX_ISP_IQ_YCRT_PARAM_T      *ptYcrt;
    AX_ISP_IQ_RAW2DNR_PARAM_T   *ptRaw2dnr;
    AX_ISP_IQ_AINR_PARAM_T      *ptAinr;
    AX_ISP_IQ_YUV3DNR_PARAM_T   *ptYuv3dnr;

    AX_ISP_IQ_AE_PARAM_T        *ptAeDftParam;
    AX_ISP_IQ_AWB_PARAM_T       *ptAwbDftParam;
    AX_ISP_IQ_LDC_PARAM_T       *ptLdc;
} AX_SENSOR_DEFAULT_PARAM_T
```

【成员】

- 详细介绍，请参考《AX ISP API 文档.docx》

【注意】

无

【相关数据类型及接口】

无

AX_SNS_ISP_I2C_DATA_T

【说明】

Sensor 寄存器信息结构体。

【定义】

```
typedef struct _AX_SNS_ISP_I2C_DATA_T_ {
    AX_BOOL bUpdate;          /* AX_TRUE: The sensor registers are written,
AX_FALSE: The sensor registers are not written*/
    AX_U8   nDelayFrmNum;      /* Number of frames for register delay
configuration */
    AX_U8   nDevAddr;         /* sensor device address */
    AX_U8   nIntPos;
    AX_U8   reserve;
    AX_U32  nRegAddr;         /* register address */
    AX_U32  nAddrByteNum;     /* Bit width of the register address */
    AX_U32  nData;            /* Sensor register data */
    AX_U32  nDataByteNum;     /* Bit width of sensor register data */
} AX_SNS_ISP_I2C_DATA_T
```

【成员】

成员名称	描述
bUpdate	寄存器更新 flag，TRUE：表示该寄存器需要配置，FALSE：表示该寄存器不需要配置
nDelayFrmNum	该寄存器需要延时几帧配置，为 1 表示延时 1 帧之后配置
nDevAddr	sensor 从设备地址

成员名称	描述
nIntPos	寄存器配置时机选择，当前仅支持在 sensor 的 SOF 之后开始配置曝光寄存器
Reserve	预留字段，字节对齐使用
nRegAddr	寄存器地址
nAddrByteNum	寄存器地址位宽（单位 bytes），最大支持 2 字节的地址位宽
nData	寄存器值
nDataByteNum	寄存器数据的位宽，最大支持 2 字节的数据位宽

【注意】

- bUpdate 字段的目的是减少每次配置寄存器的个数，用户可以比较前后两次的寄存器值，当寄存器值相同时，将 bUpdate 置为 FALSE，即表示本次不需要配置该寄存器。
- 上述结构体用于存放一个寄存器的信息，一路 sensor 最多支持配置 32 个寄存器，使用 ISP_MAX_SNS_REGISTER_NUM 记录该限制。
- I2C 写寄存器应避免耗时太长，因此推荐客户将 I2C 通信的速率配置为 400Kbit/s，高速率可以缩短寄存器的写入时间。

【相关数据类型及接口】

无

AX_SNS_ISP_EXP_INFO_T

【说明】

Sensor 曝光相关的信息。

【定义】

```
typedef struct _AX_SNS_ISP_EXP_INFO_T_ {
    AX_U32      szExpTime[HDR_MAX_FRAME_NUM];   /* unit:us */
    AX_F32      szCurAGain[HDR_MAX_FRAME_NUM];
    AX_F32      szCurDGain[HDR_MAX_FRAME_NUM];
    AX_U32      szLineGap[HDR_MAX_FRAME_NUM];   /* index0: long frame and
medium frame line gap, exposure in unit of rows */
} AX_SNS_ISP_EXP_INFO_T
```

【成员】

成员名称	描述
szExpTime[HDR_MAX_FRAME_NUM]	曝光时间，单位行，支持配置长帧、短帧和超短帧的曝光时间
szCurAGain[HDR_MAX_FRAME_NUM]	模拟增益值，支持配置长帧、短帧和超短帧的增益
szCurDGain[HDR_MAX_FRAME_NUM]	数字增益值，支持配置长帧、短帧和超短帧的增益
szLineGap[HDR_MAX_FRAME_NUM]	行差，单位是曝光行，数组下标[0]代表长短帧的行差，下标[1]代表短帧和超短帧的行差

【注意】

无

【相关数据类型及接口】

无

AX_SNS_REGS_CFG_TABLE_T

【说明】

曝光寄存器配置表。

【定义】

```
typedef struct _AX_SNS_REGS_CFG_TABLE_T_ {
    AX_BOOL                 bConfig;
    AX_SNS_CONNECT_TYPE_E   eSnsType;
    AX_BOOL                 bQuickEffectEn;
    AX_U32                  nRegNum;
    AX_U32                  nCfg2ValidDelayMax;
    AX_SNS_COMMBUS_T        tComBus;
    AX_SNS_ISP_I2C_DATA_T   sztI2cData[ISP_MAX_SNS_REGISTER_NUM];
    AX_SNS_ISP_EXP_INFO_T   tSnsExpInfo;
} AX_SNS_REGS_CFG_TABLE_T
```

【成员】

成员名称	描述
bConfig	firmware 和 sensor 驱动的同步标志位，firmware 配置完成之后置为 TRUE；sensor 驱动寄存器更新之后，置为 FALSE，表示未配置的状态
eSnsType	Sensor 通信类型，当前仅支持 I2C 通信
bQuickEffectEn	快速生效使能
nRegNum	寄存器总个数，最大支持 32 个

成员名称	描述
nCfg2ValidDelayMax	配置后最大的生效时间
tComBus	如果是 I2C 通信，表示 I2C 的设备节点
sztI2cData[ISP_MAX_SNS_REGISTER_NUM]	I2C 寄存器信息
tSnsExpInfo	当前帧的曝光参数信息

【注意】

- 为方便客户快速理解寄存器延时配置的机制，我司提供的 sensor 驱动中，定义了一个表征寄存器延时配置的结构体 AX_SNS_DRV_DELAY_TABLE_T（详见 isp_sensor_internal.h），用户可以使用该数据结构存放曝光相关的寄存器，以及寄存器的延时配置帧数。
- 用户需要在注册给 pfn_sensor_init 接口的函数中，完成本结构体内的相关参数的初始化。
- 当发生 SDR/HDR 切换或者 sensor 参数变更时，会重新初始化该结构体，以保证参数的完全同步。

【相关数据类型及接口】

无

AX_SNS_AE_SLOW_SHUTTER_PARAM_T

【说明】

定义慢快门模式下的最大最小帧率和各个频率及对应的最大曝光时间。

【定义】

```
typedef struct _AX_SNS_AE_SLOW_SHUTTER_PARAM_T_ {
    AX_SNS_AE_SLOW_SHUTTER_TBL_T    tSlowShutterTbl[SNS_MAX_FRAME_RATE];
    AX_F32                          fMaxFps;
    AX_F32                          fMinFps;
    AX_S32                          nGroupNum;
} AX_SNS_AE_SLOW_SHUTTER_PARAM_T
```

【成员】

成员名称	描述
tSlowShutterTbl[SNS_MAX_FRAME_RATE]	慢快门挡位表
fMaxFps	最大帧率
fMinFps	最小帧率
nGroupNum	帧率挡位数

【注意】

无

【相关数据类型及接口】

无

AX_SNS_AE_SLOW_SHUTTER_TBL_T

【说明】

定义慢快门模式下的各个频率及对应的最大曝光时间。

【定义】

```
typedef struct _AX_SNS_AE_SLOW_SHUTTER_TBL_T_ {
    AX_U32  nMaxIntTime;   /*unit:us*/
    AX_F32 fRealFps;
} AX_SNS_AE_SLOW_SHUTTER_TBL_T
```

【成员】

成员名称	描述
nMaxIntTime	最大曝光时间
fRealFps	实时帧率

【注意】

无

【相关数据类型及接口】

无

AX_SNS_MIRRORFLIP_TYPE_E

【说明】

sensor 图像旋转角度相关信息。

【定义】

```
typedef enum _AX_SNS_MIRRORFLIP_TYPE_E_{
    AX_SNS_MF_NORMAL        = 0,
    AX_SNS_MF_MIRROR        = 1,
    AX_SNS_MF_FLIP          = 2,
    AX_SNS_MF_MIRROR_FLIP   = 3,
    AX_SNS_MF_BUTT
} AX_SNS_MIRRORFLIP_TYPE_E
```

【成员】

成员名称	描述
AX_SNS_MF_NORMAL	图像默认角度
AX_SNS_MF_MIRROR	图像镜像
AX_SNS_MF_FLIP	图像旋转
AX_SNS_MF_MIRROR_FLIP	图像镜像旋转

【注意】

无

【相关数据类型及接口】

无

AX_SNS_AE_INT_TIME_RANGE_T

【说明】

定义 AE 曝光的最大小值范围。

【定义】

```
typedef struct _AX_SNS_AE_INT_TIME_RANGE_T_ {
    AX_F32 fMinIntegrationTime[HDR_MAX_FRAME_NUM];
    AX_F32 fMaxIntegrationTime[HDR_MAX_FRAME_NUM];
} AX_SNS_AE_INT_TIME_RANGE_T;
```

【成员】

成员名称	描述
fMinIntegrationTime	最小曝光
fMaxIntegrationTime	最大曝光

【注意】

无

【相关数据类型及接口】

无

AX_SNS_AE_PARAM_T

【说明】

定义 AE 的参数信息。

【定义】

```
typedef struct _AX_SNS_AE_PARAM_T_ {
    AX_F32 fCurAGain[HDR_MAX_FRAME_NUM];
    AX_F32 fCurDGain[HDR_MAX_FRAME_NUM];
    AX_F32 fCurIspDGain[HDR_MAX_FRAME_NUM];
    AX_U32 nCurIntegrationTime[HDR_MAX_FRAME_NUM];
    AX_F32 fAGainIncrement[HDR_MAX_FRAME_NUM];
    AX_F32 fDGainIncrement[HDR_MAX_FRAME_NUM];
    AX_F32 fIspDGainIncrement[HDR_MAX_FRAME_NUM];
    AX_F32 fIntegrationTimeIncrement[HDR_MAX_FRAME_NUM];
    AX_F32 fCurFps;
    AX_F32 fIntegrationTimeOffset[HDR_MAX_FRAME_NUM];
} AX_SNS_AE_PARAM_T;
```

【成员】

成员名称	描述
fCurAGain	当前的 Again
fCurDGain	当前的 Dgain
fCurIspDGain	当前的 IspDgain
nCurIntegrationTime	当前的曝光时间
fAGainIncrement	Again 步长

成员名称	描述
fDGainIncrement	DGain 步长
fIspDGainIncrement	IspDGain 步长
fIntegrationTimeIncrement	曝光步长
fCurFps	当前帧率
fIntegrationTimeOffset	Sensor 固有曝光延迟时间

【注意】

无

【相关数据类型及接口】

无

AX_SNS_SLEEP_WAKEUP_E

【说明】

睡眠唤醒状态信息

【定义】

```
typedef enum _AX_SNS_SLEEP_WAKEUP_E_ {
    AX_SNS_EVENT_SLEEP          = 0,
    AX_SNS_EVENT_WAKE_UP        = 1,
    AX_SNS_EVENT_BUTT
} AX_SNS_SLEEP_WAKEUP_E;
```

【成员】

成员名称	描述
AX_SNS_EVENT_SLEEP	Sensor 进入睡眠状态
AX_SNS_EVENT_WAKE_UP	把 Sensor 从睡眠状态唤醒

【注意】

无

【相关数据类型及接口】

无

AX_SNS_AE_LFHDR_ATTR_T

【说明】

Sensor 长帧 HDR 模式参数

【定义】

```
typedef struct _AX_SNS_AE_LFHDR_ATTR_T_ {

    AX_ISP_LFHDR_MODE_E     eLFHdrMode;

} AX_SNS_AE_LFHDR_ATTR_T;
```

【成员】

成员名称	描述
eLFHdrMode	长帧 HDR 模式参数

【注意】

无

【相关数据类型及接口】

无

AX_ISP_LFHDR_MODE_E

【说明】

Sensor 长帧 HDR 模式

【定义】

```
typedef enum {

    AX_ISP_LFHDR_NORMAL_MODE            = 0x0,

    AX_ISP_LFHDR_LONG_FRAME_MODE        = 0x1,

    AX_ISP_LFHDR_AUTO_LONG_FRAME_MODE   = 0x2,

    AX_ISP_LFHDR_MODE_MAX

} AX_ISP_LFHDR_MODE_E;
```

【成员】

成员名称	描述
AX_ISP_LFHDR_NORMAL_MODE	默认模式
AX_ISP_LFHDR_LONG_FRAME_MODE	长帧模式
AX_ISP_LFHDR_AUTO_LONG_FRAME_MODE	自动长帧模式（暂不支持）

【注意】

无

【相关数据类型及接口】

无

AX_SNS_AWB_PARAM_T

【说明】

Sensor awb 参数设置

【定义】

```
typedef struct _AX_SNS_AWB_PARAM_T_ {
    AX_U32 nGrGain;
    AX_U32 nGbGain;
    AX_U32 nRgain;
    AX_U32 nBgain;
    AX_U32 nColorTemp;
} AX_SNS_AWB_PARAM_T;
```

【成员】

成员名称	描述
nGrGain	Gr 增益
nGbGain	Gb 增益
nRgain	Rg 增益
nBgain	Bg 增益
nColorTemp	色温

【注意】

无

【相关数据类型及接口】

无

5 FAQ

5.1 Sensor 初始化退出流程

5.1.1 初始化流程

pfn_sensor_set_bus_info

pfn_sensor_set_mode

pfn_sensor_reset

pfn_sensor_init (*)

pfn_sensor_streaming_ctrl (off)

pfn_sensor_set_fps (*)

pfn_sensor_testpattern (*)

pfn_sensor_get_default_params (*)

pfn_sensor_streaming_ctrl (on)

5.1.2 退出流程

pfn_sensor_streaming_ctrl (off)

pfn_sensor_exit (*)

！ 注意：(*) 代表隐式调用

5.2 如何确认 sensor 是否出流

调试 sensor 过程，有时会遇到不出图的情况，可以从以下几方面进行排查

1. 如果有示波器的，可以直接量 sensor 是否有 mipi 信号输出;
2. cat /proc/ax_proc/vin/statistics ,看 frame cnt 是否有增加计数(有多个 cnt 节点);
3. 如果 sensor 内部有帧计数寄存器，也可以通过 i2ctransfer 直接读取

5.3 P/N 制式切换说明

支持 P/N 制式切换过程中，sensor 不断流。

PN 切换调使用 **AX_ISP_SetSnsAttr()** 重新设置帧率即可完成切换。

！ 注意：切换过程无需 sensor 开关流。

5.4 必要接口说明

用户使用 AX 的 sensor drv 框架，将 sensor handle 注册到 ISP FW 和 3A 算法中，如下的 callback 函数必须在 sensor drv 中注册并实现：

Sensor 控制接口	pfn_sensor_set_bus_info pfn_sensor_reset pfn_sensor_init pfn_sensor_exit pfn_sensor_streaming_ctrl pfn_sensor_set_mode pfn_sensor_get_mode pfn_sensor_set_fps pfn_sensor_get_fps
默认参数接口	pfn_sensor_get_default_params pfn_sensor_get_black_level

AE 控制接口	pfn_sensor_get_hw_exposure_params pfn_sensor_get_gain_table pfn_sensor_set_again pfn_sensor_set_dgain（不支持，可以置 NULL） pfn_sensor_hcglcg_ctrl（不支持，可以置 NULL） pfn_sensor_set_integration_time pfn_sensor_get_integration_time_range pfn_sensor_set_slow_fps（不支持慢快门，可以置 NULL） pfn_sensor_get_slow_shutter_param（不支持慢快门，可以置 NULL） pfn_sensor_get_sns_reg_info（不需要参数同步，不需要在 kernel 写寄存器，可以置 NULL） pfn_sensor_set_lfhdr_mode（不支持长帧模式，可以置 NULL） pfn_sensor_get_temperature_info（sensor 不支持获取温度，可以置 NULL）

5.5 Sensor Log

Sensor 调试过程，可以提高 log 打印等级，来排查 sensor 初始化等各个流程运行是否正常

常用的，可以执行以下执行修改 sensor log 打印等级。

echo ulog 34 7 > /proc/ax_proc/logctl – 设置 sensor log 等级为 **debug**

echo ulog 34 4 > /proc/ax_proc/logctl – 设置 sensor log 等级为 **warning**

5.6 Sensor mipi clk 配置

申请 sensor setting 时，要求 sensor mipi 输出 clk 为非连续模式，具体可以参考 mipi 文档。

5.7 获取不到 chip id 怎么办？

首先运行 tuningserver,确保 mclk 已经打开（sensor 需要有 mclk，i2c 才能正常响应），尝试使用 i2cdetect debug,sensor 是否可以正常识别，确认 sensor 挂在哪个 i2c bus 上，

几个常用的 i2cdetect 指令如下：

1)查看当前 i2c bus 注册情况：

i2cdetect –l

这个指令可以查看当前已经注册的 i2c bus

```
/ # i2cdetect -l
i2c-3   i2c             Synopsys DesignWare I2C adapter         I2C adapter
i2c-8   i2c             Synopsys DesignWare I2C Slave adapter   I2C adapter
i2c-4   i2c             Synopsys DesignWare I2C adapter         I2C adapter
i2c-2   i2c             Synopsys DesignWare I2C adapter         I2C adapter
i2c-0   i2c             Synopsys DesignWare I2C adapter         I2C adapter
```

2)查看当前 i2c bus 硬件有哪些可以识别到的 i2c 设备的 slave addr

i2cdetect -r -y 0 [0 为 i2c bus num]

```
/ # i2cdetect -r -y 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- 21 -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- 36 -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

如上图，可以看到 i2c-0 有 slave addr 为 0x36 的设备

如果这个地址和你的 sensor 的一致，那么就代表硬件上可以识别到 sensor

否则就要排查，硬件连接是否 ok,sensor 是否有异常等。

在识别到 i2c 设备的情况下，可以通过如下指令，读取 sensor 具体的寄存器数值

i2c 读取 0x0100 数值:

i2ctransfer -y -f 0 w2@0x36 0x01 0X00 r1

i2c 写入 0x0100 数值 0x01:

i2ctransfer -y -f 0 w3@0x36 0x01 0x00 0x01

相关命令参数解释如下:

-f 0 是 i2c num

@0x36 是 i2c 设备地址

r1 是 读取一个寄存器数值

w2 是读取 [寄存器地址是两个字节情况下]

w3 是写入 [寄存器地址是两个字节情况下]

5.8 BLC 的默认数值怎么转化为定点数

在代码中,我们常可以看到,如下的函数

```
AX_S32 os04a10_get_black_level(ISP_PIPE_ID nPipeId, AX_SNS_BLACK_LEVEL_T *ptBlackLevel)
{
    SNS_STATE_OBJ *sns_obj = AX_NULL;

    SNS_CHECK_PTR_VALID(ptBlackLevel);
    SNS_CHECK_VALUE_RANGE_VALID(nPipeId, 0, AX_VIN_MAX_PIPE_NUM - 1);

    SENSOR_GET_CTX(nPipeId, sns_obj);
    SNS_CHECK_PTR_VALID(sns_obj);

    /* black level of linear mode */
    if (AX_SNS_LINEAR_MODE == sns_obj->sns_mode_obj.eHDRMode) {
        ptBlackLevel->nBlackLevel[0] = 1024;    /* linear mode 10bit sensor default blc 64(U10.0) --> 1024(U8.6) */
        ptBlackLevel->nBlackLevel[1] = 1024;
        ptBlackLevel->nBlackLevel[2] = 1024;
        ptBlackLevel->nBlackLevel[3] = 1024;
    } else {
        ptBlackLevel->nBlackLevel[0] = 1024;
        ptBlackLevel->nBlackLevel[1] = 1024;
        ptBlackLevel->nBlackLevel[2] = 1024;
        ptBlackLevel->nBlackLevel[3] = 1024;
    }

    return AX_SNS_SUCCESS;
}
```

我们看到 blc 各个通道,数值都初始化为 1024,

这个数值是从哪里得来呢?

对于 sensor 来说,自身都会有一个出厂默认的一个 blc 默认的数值,

这个数值一般是存储在 sensor 某个特定的寄存器中

通常直接询问 sensor 厂商的 fae 即可获取对应数值

那么拿到这个数值后,要怎么转换为定点数,填到驱动的 blc 配置函数中呢?

默认读取 sensor blc 寄存器,默认读取的数值是 16 进制的,

例如读取到 0x40,那么十进制就是 64,也就是 U10.0

这里有个默认的规则,就是代码最终的填入的数值的格式是 U8.6

那么如何把 U10.0,转化为 U8.6

这里有个简单的技巧,就是

U8.6 的 8+6=14

U10.0 的 10+0=10

那么 14-10=4 也就是 U10.0 左移 4 位,即可得到 U8.6

64 左移 4 位,就是 1024,代码中直接填入 1024 即可.