Linux ISP
开发指南

版本号: 1.0
发布日期: 2021.04.14

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.04.14	AWA1636	建立初版

目　录

1 前言

1.1 文档简介

ISP 模块简介，3A 开发指南，API 接口，数据结构。

1.2 目标读者

ISP 模块维护，开发人员

1.3 适用范围

ISP600

2 模块介绍

2.1 模块功能介绍

ISP(Image Signal Processor)，即图像处理，主要作用是对前端图像传感器输出的信号做后期处理，主要功能黑电平校正、色差校正、颜色空间校正、颜色增强、色彩降噪、串音校正、二/三维降噪、去马赛克、坏点校正、动态范围压缩、镜头阴影校正、周边去噪、色调映射、宽动态范围、自动曝光、自动白平衡、等，依赖于 ISP 才能在不同的光学条件下都能较好的还原现场细节，ISP 技术在很大程度上决定了摄像机的成像质量。它可以分为独立与集成两种形式。

其所处位置如下：

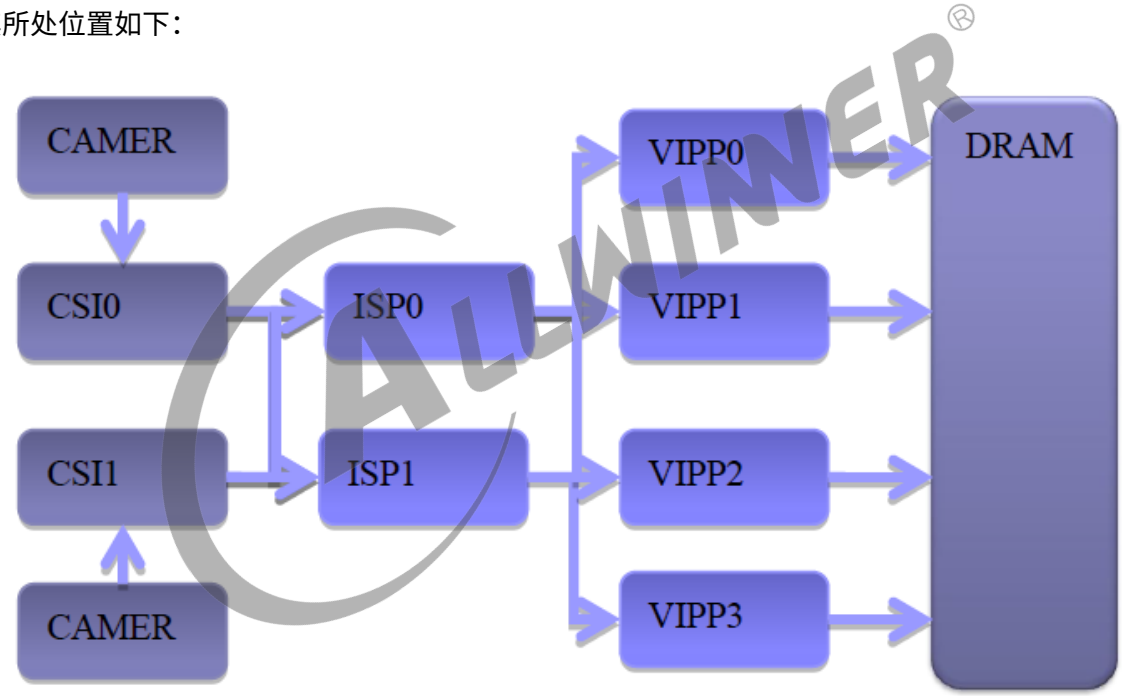

图 2-1

2.2 ISP Server 简介

ISP Server 模块主要包括 ISP 算法库和 ISP 中间件部分：

- ISP 算法库部分，其主要用于在 ISP 运行时图像效果的处理，包括 3A 算法以及一系列 ISP 正常运行所需的基本算法；

- ISP 中间件部分，其主要用于控制 ISP 以及 Sensor 驱动、响应 Camera 应用以及 Tuning 工具命令、调度 ISP 相关算法等，包括事件管理、Pipeline 管理、Buffer 管理以及算法调度等模块。

ISP 算法库、中间件、驱动以及 Camera 应用相互关系如下图所示：

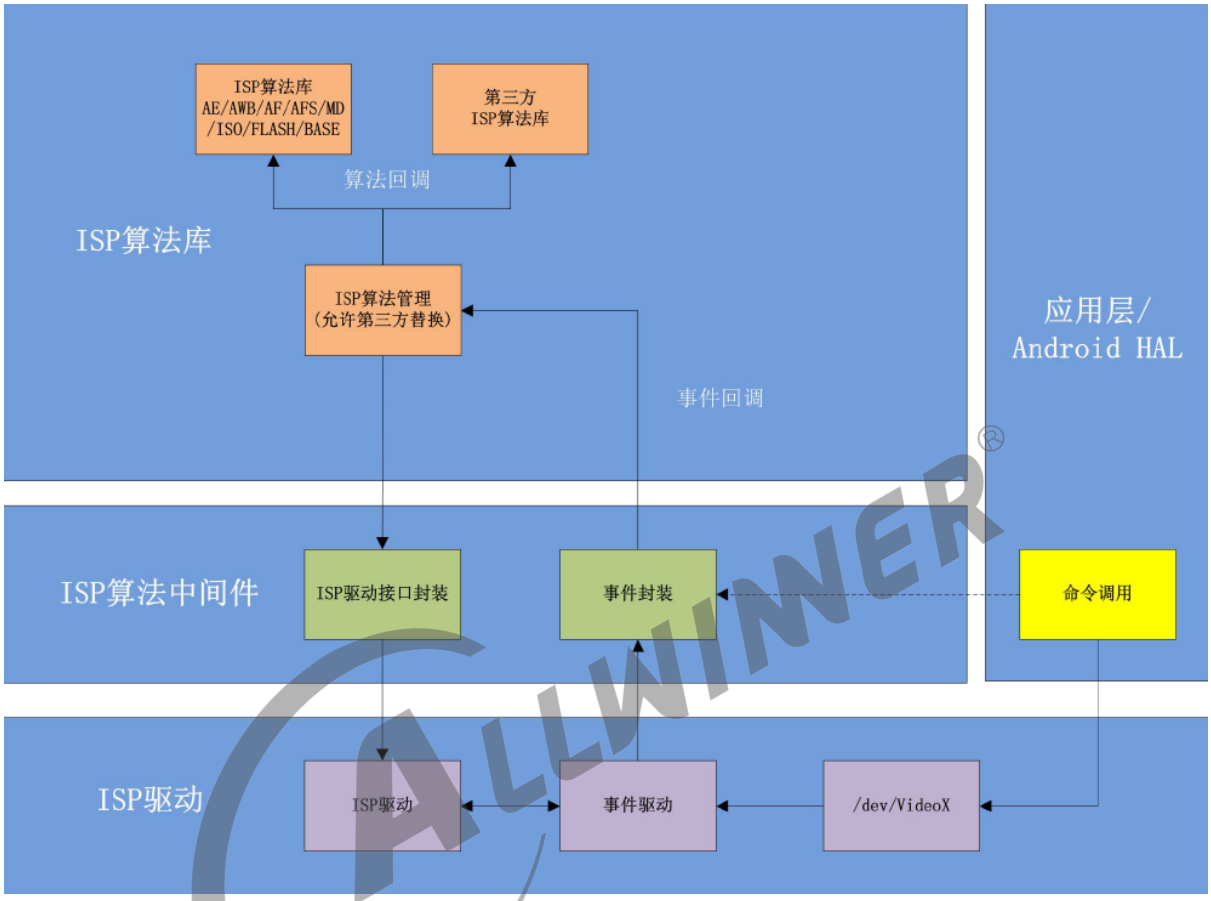

图 2-2: ISP Server 基本框架

2.3 ISP Server 代码结构

ISP Server 开源部分代码结构如下：

```
Isp_server:
| isp.c ;ISP 对外接口实现（对接调试工具、Camera 应用）
| isp.h ;ISP 对外接口头文件
| Makefile
├─include
|    isp_3a_ae.h ; 自动曝光算法头文件
|    isp_3a_af.h ; 自动对焦算法头文件
|    isp_3a_awb.h ; 自动白平衡算法头文件
|    isp_tuning.h ; ISP 效果tuning 接口头文件
|    isp_base.h ; 基本算法头文件
|    isp_ini_parse.h ; ini 文件解析头文件
|    isp_iso_config.h ; 动态参数设置头文件
```

```
|    isp_cmd_intf.h ; 命令处理头文件
|    isp_comm.h ; 公共头文件
|    isp_debug.h ; Debug 头文件
|    isp_manage.h ; 算法管理模块头文件
|    isp_module_cfg.h ; 硬件模块头文件
|    isp_reg.h ; 寄存器平台选择头文件
|    isp_rolloff.h ; 镜头阴影矫正算法头文件
|    isp_tone_mapping.h ; 色调映射算法头文件
|    isp_type.h ; 类型定义头文件
|
├─iniparser
|  ├─src
|      iniparser.c
|      iniparser.h
|      dictionary.c
|      dictionary.h
|
├─isp_cfg
|    isp_ini_parse.c ; Sensor 模组ISP 配置文件解析
|  ├──SENSOR_H ; Sensor 模组ISP 配置头文件
|       imx290_default_ini.h ; imx290 ISP 配置
|       imx317_default_ini.h ; imx317 ISP 配置
|       ar0238_default_ini.h ; ar0238 ISP 配置
|       ov2718_wdr_ini.h ; ov2718 wdr ISP 配置
|       Makefile
|
├─isp_dev
|  ├──video
|      video_priv.h ; 视频设备私有头文件
|      video.c ; 用于管理标准v4l2 视频设备
|
|     isp_dev.c ; ISP 设备封装，用于管理绑定相关设备
|     isp_dev_priv.h ; ISP 设备私有头文件
|     isp_stats.h ; ISP 统计值管理头文件
|     isp_stats.c ; ISP 统计值管理模块
|     isp_stats_priv.h ; ISP 统计值管理私有头文件
|     isp_subdev.c ;ISP 子设备模块
|     isp_subdev.h ;ISP 子设备头文件
|     isp_v4l2_helper.c ;V4l2 帮助函数
|     isp_v4l2_helper.h ;V4l2 帮助函数头文件
|     media.c ;media 框架帮助函数
|     media.h ;media 框架帮助函数头文件
|     tools.h ;工具文件
|     Makefile
|
├─isp_events
|     events.c ;事件管理模块，用于监听分发驱动事件
|     events.h ;事件管理模块头文件
|
├─isp_manage
|     isp_manage.c ;ISP 算法管理模块
|
└─isp_tuning
|     isp_tuning.c ;效果调试接口
|     isp_tuning_priv.h ;效果调试私有头文件
|
└─out
    libisp_ae.a ;自动曝光算法库
    libisp_af.a ;自动对焦算法库
    libisp_afs.a ;自动去工频算法库
```

```
libisp_awb.a ;自动白平衡算法库
libisp_base.a ;基础算法库
libisp_ini.a ;ISP 配置参数获取库
libisp_iso.a ;动态参数设置算法库
libisp_md.a ;运动检测算法库
libisp_gtm.a ;全局色调映射算法库
libisp_pltm.a ;局部色调映射算法库
libisp_rolloff.a ;自动color shading 矫正算法库
libisp_math.a ;自定义数学运算库
libmatrix.a ;矩阵运算库
```

ISP Server 开源代码可以概括为 3 部分：

- 1. 设备管理部分，主要包括 isp_dev、isp_events 目录下的代码，其中 isp_dev 部分负责统一管理视频设备，CSI 设备，Sensor 设备，统计设备；

a) 在初始化时，其会建立好相应的 Sensor->CSI->ISP ->Video 通路，对只需要图像数据的设备来说，仅需操作 Video 设备即可获取想要的数据。
b) 在设备运行时，isp_events 模块会根据 CSI 或者 ISP 返回来的事件来通知不同模块进行必要的事件处理，主要事件一般有 Frame Sync 事件 Vsync 事件，Vidoe 命令事件以及统计值 Ready 事件等。

- 2.Sensor 配置管理部分，主要包括 iniparser、isp_cfg 以及 isp_tuning 部分；其中 iniparser 为标准 ini 文件解析库；isp_cfg 为 sensor 配置文件匹配部分，有两种方法获取配置文件：a. 读取 ini 文件，b. 头文件预定义；isp_tuning 为外部调整 ISP 效果提供接口。
- 3. 算法管理部分，主要包括 isp_manage 部分，其负责各个子算法的初始化、运行、退出等动作。

ISP Server 的构成结构如下图所示：

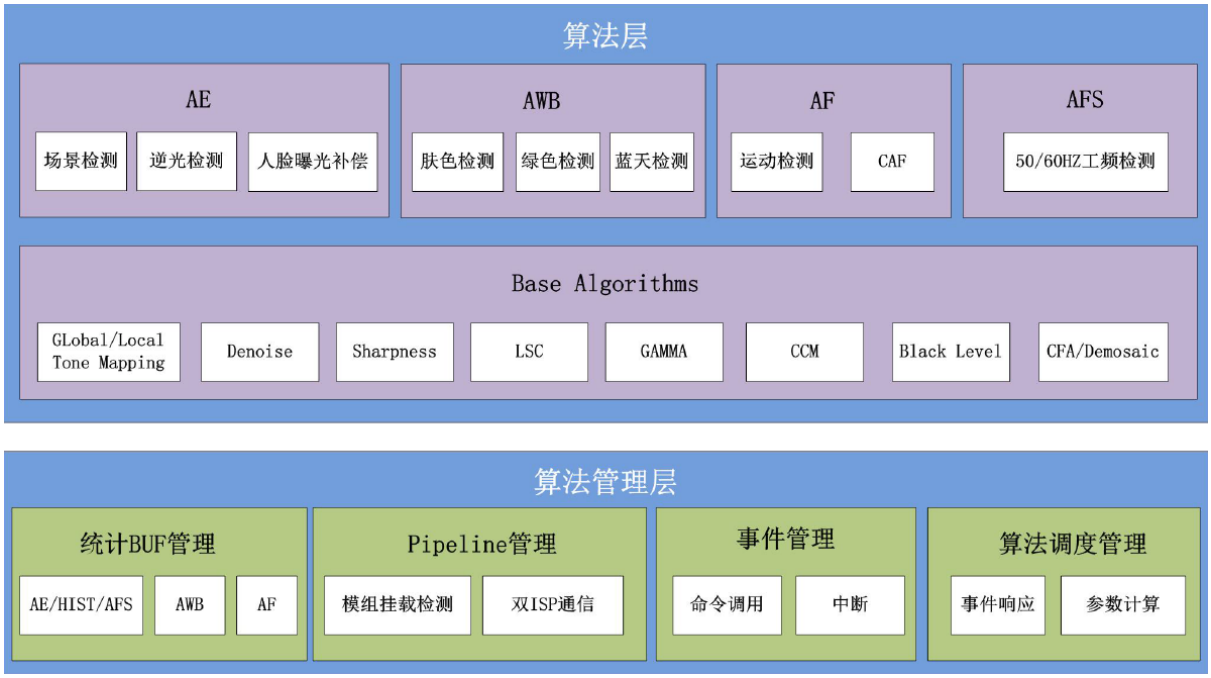

图 2-3: ISP 算法库的构成

3 使用指南

3.1 ISP 使用流程

- ISP Server 需要与 VI 采集协同工作，使用时应当先初始化 VI 采集相关配置，在初始化 ISP 相关配置，打开 VI 采集流之后运行 ISP run 即可，ISP Server 此时会动态调整 Sensor、Lens 以及 ISP 相关配置。
- ISP Server 使用方法非常简单，示例代码如下：

```
int main(int argc __attribute__((__unused__)), char *argv[] __attribute__((__unused__)))
{
    media_dev_init(); //初始化多媒体设备
    isp_init(0); //初始化isp0
    isp_run(0); //运行isp0 线程
    isp_pthread_join(0); //等待isp0 线程结束
    isp_exit(0); //退出isp0
    media_dev_exit(); //退出多媒体设备
    return 0;
}
```

3.2 Sensor 设备操作

- Sensor 与 ISP 的对应关系在一般情况下可以由内核中的 Device tree 配置，执行 media_dev_init() 时，ISP Server 中的设备管理模块便可获取其相对应关系，应用操作 Sensor 无需直接找到对应 Sensor 设备，只需要操作 ISP 接口即可，如：

```
/*isp_dev.h 中定义的Sensor 相关的操作集*/
int isp_sensor_get_configs(struct hw_isp_device *isp, struct sensor_config *cfg);
int isp_sensor_set_exp_gain(struct hw_isp_device *isp, struct sensor_exp_gain *exp_gain);
```

3.3 ISP 3A 开发方法

ISP Server 框架可以为客户提供三种开发模式：

- 1. 极简模式，使用全志提供的全套算法库，ISP Server 部分对客户不可见，客户只需要通过 MPI 操作相应视频设备节点即可获取图像数据，图像效果完全由 Tuning 工具给出的配置控制。

- 2. 一般模式，使用全志提供的全套算法库，ISP Server 部分对客户不可见，客户可以通过 MPI 操作相应视频设备节点获取图像数据，同时可以操作 ISP、Sensor 设备，通过禁用全志 ISP 内部某些特定算法，然后通过 MPI 接口获取统计信息，再通过 MPI 接口设置给 ISP，可达到替换某些特定算法的目的。
- 3. 高级模式，对于开发能力强的核心客户，可以部分替换 ISP Server 中的算法库，ISP Server 开源部分可以开放给这类客户。

3.4 3A 算法库注册

- 所有 ISP 软件算法都有一组统一的注册接口，如 AE 算法：

```
/*isp_3a_ae.h 中定义的AE 算法相关的操作集*/
typedef struct isp_ae_core_ops {
     HW_S32 (*isp_ae_set_params)(void * ae_core_obj, ae_param_t *param, ae_result_t *result
    );
     HW_S32 (*isp_ae_get_params)(void *ae_core_obj, ae_param_t *param);
     HW_S32 (*isp_ae_run)(void *ae_core_obj, ae_stats_t *stats, ae_result_t *result);
} isp_ae_core_ops_t;
void* ae_init(isp_ae_core_ops_t **ae_core_ops);
void ae_exit(void *ae_core_obj);
```

运行算法的通用基本流程为：

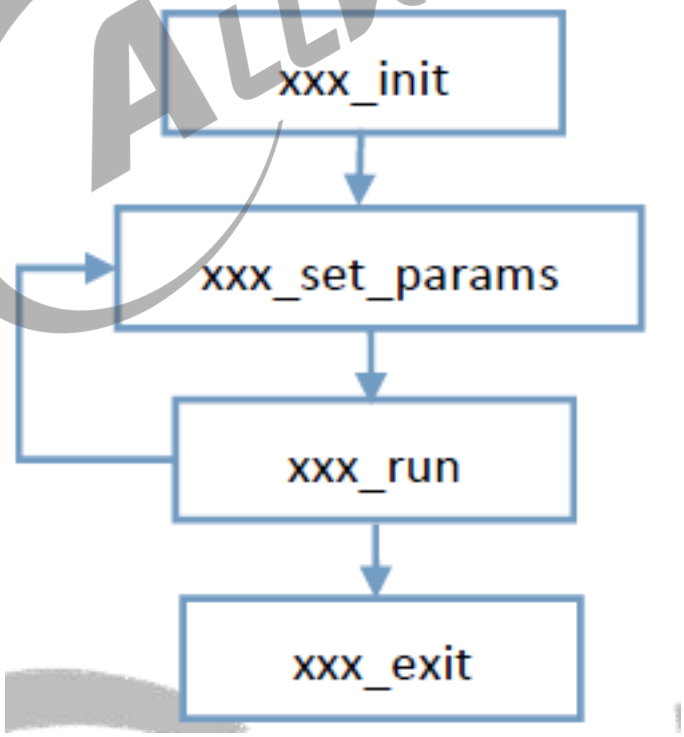

图 3-1

例如 AE 算法：

```
/****************AE init**************/
void __isp_ae_init(struct isp_lib_context *isp_gen)
{
    isp_gen->ae_entity_ctx.ae_entity = ae_init(&isp_gen->ae_entity_ctx.ops);
    if (isp_gen->ae_entity_ctx.ae_entity == NULL || NULL == isp_gen->ae_entity_ctx.ops) {
        ISP_ERR("AE Entity is BUSY or NULL!\\n");
        return -1;
    } else {
        clear(isp_gen->ae_entity_ctx.ae_param);
        isp_gen->ae_entity_ctx.ae_param.u.isp_platform_id = isp_gen->module_cfg.
    isp_platform_id;
        isp_ae_set_params_helper(ISP_AE_PLATFORM_ID);
    }
}
/****************AE set_params**************/
void __isp_ae_set_params(struct isp_lib_context *isp_gen)
{
    clear(isp_gen->ae_entity_ctx.ae_param);
    isp_gen->ae_entity_ctx.ae_param.u.ae_frame_id = isp_gen->ae_frame_cnt;
    isp_ae_set_params_helper(ISP_AE_FRAME_ID);
    clear(isp_gen->ae_entity_ctx.ae_param);
    isp_gen->ae_entity_ctx.ae_param.u.ae_setting = isp_gen->ae_settings;
    isp_ae_set_params_helper(ISP_AE_SETTINGS);
//ae_sensor_info.
    clear(isp_gen->ae_entity_ctx.ae_param);
    isp_gen->ae_entity_ctx.ae_param.u.ae_sensor_info = isp_gen->sensor_info;
    isp_ae_set_params_helper(ISP_AE_SENSOR_INFO);
}
/****************AE exit**************/
void __isp_ae_exit(struct isp_lib_context *isp_gen)
{
    ae_exit(isp_gen->ae_entity_ctx.ae_entity);
}
```

4 3A 开发指南

4.1 概述

对于高阶开发者，可以使用自己开发的算法替换 ISP out 目录下相应的算法模块，主要涉及 AE、AWB 以及 AF 部分，ISP Server 中每个独立算法的形式基本一致，因此可以提出一个固定模板来对接具体的算法，通用模板格式如下：

```
/*
******************************************************************************
*/
#include "../include/isp_manage.h"
#include "../include/isp_3a_xxx.h"
#include "../isp_math/isp_math_util.h"
typedef struct isp_xxx_config_entity
{
}xxx_config_t;
typedef struct isp_xxx_core_entity
{
}xxx_core_entity_t;
int __IspxxxIsr(xxx_core_entity_t *entity, xxx_stats_t *stats, xxx_result_t *result)
{
    return 0;
}
xxx_core_entity_t *__IspAllocxxxEntity(void)
{
    xxx_core_entity_t *entity = NULL;
    entity = malloc(sizeof(*entity));
    if(entity == NULL) {
        ISP_ERR("xxx Entity No memory!");
        return NULL;
    }
    memset(entity, 0, sizeof(*entity));
    entity->busy_flag = 1;
    return entity;
}
void __IspFreexxxEntity(xxx_core_entity_t *xxx_core_obj)
{
    if(xxx_core_obj) {
        xxx_core_obj->busy_flag = 0;
        free(xxx_core_obj);
    }
}
#define ISP_xxx_SET_PARAMS(key)\\
{\\
    entity->config.key = param->u.key;\\
}
#define ISP_xxx_GET_PARAMS(key)\\
{\\
    param->u.key = entity->config.key;\\
```

```
}
int __AwxxxSetParams(void * xxx_core_obj, xxx_param_t *param, xxx_result_t *result)
{
    int ret = 0;
    xxx_core_entity_t *entity;
    if(xxx_core_obj && param)
    {
        entity = (xxx_core_entity_t *)xxx_core_obj;
    }
    else
    {
        ret = -1;
        goto set_param_end;
    }
    switch (param->type)
    {
        case ISP_xxx_PLATFORM_ID:
          ISP_xxx_SET_PARAMS(isp_platform_id);
          break;
        case ISP_xxx_FRAME_ID:
          ISP_xxx_SET_PARAMS(xxx_frame_id);
          break;
        default:
          ret = -1;
    }
set_param_end:
    return ret;
}
int __AwxxxGetParams(void *xxx_core_obj, xxx_param_t *param)
{
    int ret = 0;
    xxx_core_entity_t *entity;
    if(xxx_core_obj && param)
    {
        entity = (xxx_core_entity_t *)xxx_core_obj;
    }
    else
    {
        ret = -1;
        goto get_param_end;
    }
    ISP_LIB_LOG(ISP_LOG_AF, "aw_af_get_params param->type = %d\\n", param->type);
    switch (param->type)
    {
        case ISP_xxx_PLATFORM_ID:
          ISP_xxx_GET_PARAMS(isp_platform_id);
          break;
        case ISP_xxx_FRAME_ID:
          ISP_xxx_GET_PARAMS(xxx_frame_id);
          break;
        default:
          ret = -1;
    }
get_param_end:
    return ret;
}
int __AwxxxRun(void *xxx_core_obj, xxx_stats_t *stats, xxx_result_t *result)
{
    int ret = 0;
    xxx_core_entity_t *entity;
```

```
    if(xxx_core_obj &&stats && result)
    {
        entity = (xxx_core_entity_t *)xxx_core_obj;
    }
    ret = __IspxxxIsr(entity, stats, result);
    return ret;
}
static isp_xxx_core_ops_t AwxxxOps =
{
    .isp_xxx_set_params = __AwxxxSetParams,
    .isp_xxx_get_params = __AwxxxGetParams,
    .isp_xxx_run = __AwxxxRun,
};
void __xxxInitEntity(xxx_core_entity_t *entity)
{
    entity->xxx_detect_flicker_type = 0xff;
    entity->xxx_stat_cnt = 0;
    entity->xxx_weight[0] = 1;
    entity->xxx_weight[1] = -1;
    entity->xxx_weight[2] = 0;
    return;
}
void* xxx_init(isp_xxx_core_ops_t **xxx_core_ops)
{
    xxx_core_entity_t *entity;
    entity = __IspAllocxxxEntity();
    if(entity)
    {
       __xxxInitEntity(entity);
       *xxx_core_ops = &AwxxxOps;
       return (void *)entity;
    }
    ISP_ERR("xxx_init Error!\\n");
    return NULL;
}
void xxx_exit(void *xxx_core_obj)
{
    __IspFreexxxEntity((xxx_core_entity_t *)xxx_core_obj);
}
```

4.2 AE 算法开发

算法管理单元通过 ae_init 函数初始化 AE 算法结构体，并返回 isp_ae_core_ops_t 操作集合：

```
/*isp_3a_ae.h 中定义的AE 算法相关的操作集*/
typedef struct isp_ae_core_ops {
    HW_S32 (*isp_ae_set_params)(void * ae_core_obj, ae_param_t *param, ae_result_t *result)
    ;
    HW_S32 (*isp_ae_get_params)(void *ae_core_obj, ae_param_t *param);
    HW_S32 (*isp_ae_run)(void *ae_core_obj, ae_stats_t *stats, ae_result_t *result);
} isp_ae_core_ops_t;
void* ae_init(isp_ae_core_ops_t **ae_core_ops);
void ae_exit(void *ae_core_obj);
```

通过该操作集合，可以控制 AE 参数，调度 AE 算法，具体流程详见 3A 算法库注册部分，接口中

用到主要结构体描述如下：

4.2.1 isp_ae_stats_s

- PROTOTYPE

```
struct isp_ae_stats_s {
    HW_U32 win_pix_n;
    HW_U32 avg[ISP_AE_ROW*ISP_AE_COL];
    HW_U32 hist[ISP_HIST_NUM];
    HW_U32 hist1[ISP_HIST_NUM];
};
```

- MEMBERS

```
win_pix_n ： 每个窗口像素数。
avg ： 分区域亮度平均值。
hist ： 直方图统计值。
hist1 ： 直方图1统计值。
```

- DESCRIPTION

```
isp_ae_stats_s 为用于描述AE 统计值的一个结构体。
```

4.2.2 ae_param_t

- PROTOTYPE

```
typedef struct isp_ae_param {
    ae_param_type_t type;
    HW_U32 current_frame_cnt;
    union {
        HW_S32 isp_platform_id;
        HW_S32 ae_frame_id;
        ae_ini_cfg_t ae_ini;
        isp_ae_settings_t ae_setting;
        HW_S32 ae_pline_index;
        HW_S32 sensor_update_done;
        struct isp_h3a_coor_win ae_coor;
        isp_sensor_info_t ae_sensor_info;
        ae_test_config_t test_cfg;
    } u;
} ae_param_t;
```

- MEMBERS

```
type ：ISP AE 命令类型，其每种类型与联合体u 中参数一一对应，定义如下：
typedef enum isp_ae_param_type {
    ISP_AE_PLATFORM_ID,
    ISP_AE_FRAME_ID,
    ISP_AE_INI_DATA,
    ISP_AE_SETTINGS,
    ISP_AE_HDR_SETTINGS,
    ISP_AE_UPDATE_AE_TABLE,
    ISP_AE_SET_EXP_IDX,
    ISP_AE_BUILD_TOUCH_WEIGHT,
    ISP_AE_SENSOR_INFO,
    ISP_AE_TEST_CONFIG,
    ISP_AE_PARAM_TYPE_MAX,
} ae_param_type_t;
isp_platform_id ：ISP 平台ID。
ae_frame_id ：AE 算法执行计数。
ae_ini ：AE 算法初始化INI 配置。
ae_setting ：AE 算法运行时配置，包括曝光补偿、手动曝光，场景模式等。
ae_pline_index ：手动设置Pline 索引。
sensor_update_done ：Sensor 更新曝光设定完成。
ae_coor ：AE ROI 窗口坐标。
ae_sensor_info ：Sensor 信息，包括VTS，HTS 以及输出宽高等。
test_cfg ：AE 测试配置。
```

- DESCRIPTION

```
ae_param_t 为用于描述AE 命令参数的一个结构体。
```

4.2.3 isp_ae_result

- PROTOTYPE

```
typedef struct isp_ae_result {
    enum ae_status ae_status;
    sensor_setting_t sensor_set;
    sensor_setting_t sensor_set_short;
    HW_S32 BrightPixellValue;
    HW_S32 DarkPixelValue;
    HW_U32 ae_gain;
    HW_S32 ae_target;
    HW_S32 ae_avg_lum;
    HW_S32 ae_weight_lum;
    HW_S32 ae_wdr_ratio;
    HW_S32 wdr_hi_th;
    HW_S32 wdr_low_th;
    HW_U8 backlight;
} ae_result_t;
```

- MEMBERS

```
ae_status ：AE 状态。
sensor_set ：Sensor 曝光设置。
sensor_set_short ：Sensor 短曝光设置。
BrightPixellValue ：AE 亮像素参考值。
DarkPixelValue ：AE 暗像素参考值。
ae_gain ：AE 调整变化率。
ae_target ：AE 目标亮度。
ae_avg_lum ：AE 平均亮度。
ae_weight_lum ：AE 加权亮度。
ae_wdr_ratio ：AE WDR 比率。
wdr_hi_th ：AE WDR 高阈值。
wdr_low_th ：AE WDR 低阈值。
backlight ：AE 背光程度。
```

- DESCRIPTION

```
ae_result_t 为用于描述AE 输出结果的一个结构体。
```

4.3 AWB 算法开发

算法管理单元通过 aw_init 函数初始化 AWB 算法结构体，并返回 isp_awb_core_ops_t 操作集合：

```
/*isp_3a_awb.h 中定义的AWB 算法相关的操作集*/
typedef struct isp_awb_core_ops {
    HW_S32 (*isp_awb_set_params)(void * awb_core_obj, awb_param_t *param, awb_result_t *
    result);
    HW_S32 (*isp_awb_get_params)(void *awb_core_obj, awb_param_t *param);
    HW_S32 (*isp_awb_run)(void *awb_core_obj, awb_stats_t *stats, awb_result_t *result);
} isp_awb_core_ops_t;
void* awb_init(isp_awb_core_ops_t **awb_core_ops);
void awb_exit(void *awb_core_obj);
```

通过该操作集合，可以控制 AWB 参数，调度 AWB 算法，具体流程详见 3A 算法库注册部分，接口中用到主要结构体描述如下：

4.3.1 isp_awb_stats_s

- PROTOTYPE

```
struct isp_awb_stats_s {
    HW_U32 awb_avg_r[ISP_AWB_ROW][ISP_AWB_COL];
    HW_U32 awb_avg_g[ISP_AWB_ROW][ISP_AWB_COL];
    HW_U32 awb_avg_b[ISP_AWB_ROW][ISP_AWB_COL];
    HW_U32 avg[ISP_AWB_ROW][ISP_AWB_COL];
};
```

- MEMBERS

```
awb_avg_r ：分区域R 像素平均值（归一化为0~255）。
awb_avg_g ：分区域G 像素平均值（归一化为0~255）。
awb_avg_b ：分区域B 像素平均值（归一化为0~255）。
avg ：分区域加权平均值。
```

- DESCRIPTION

```
isp_awb_stats_s 为用于描述AWB 统计值的一个结构体。
```

4.3.2 awb_param_t

- PROTOTYPE

```
typedef struct isp_awb_param {
    awb_param_type_t type;
    HW_U32 current_frame_cnt;
    union {
        HW_S32 isp_platform_id;
        HW_S32 awb_frame_id;
        isp_awb_setting_t awb_ctrl;
        awb_ini_cfg_t awb_ini;
        isp_sensor_info_t awb_sensor_info;
        awb_test_config_t test_cfg;
    } u;
} awb_param_t;
```

- MEMBERS

```
type ：ISP AWB 命令类型，其每种类型与联合体u 中参数一一对应，定义如下：
typedef enum isp_awb_param_type {
    ISP_AWB_PLATFORM_ID,
    ISP_AWB_FRAME_ID,
    ISP_AWB_CTRL_CFG,
    ISP_AWB_INI_DATA,
    ISP_AWB_SENSOR_INFO,
    ISP_AWB_TEST_CONFIG,
    ISP_AWB_PARAM_TYPE_MAX,
} awb_param_type_t;
isp_platform_id ：ISP 平台ID。
awb_frame_id ：AWB 算法执行计数。
awb_ctrl ：AWB 算法运行时配置。
awb_ini ：AWB 算法初始化INI 配置。
awb_sensor_info ：Sensor 信息，包括VTS，HTS 以及输出宽高等。
test_cfg ：AWB 测试配置。
```

- DESCRIPTION

```
awb_param_t 为用于描述AWB 命令参数的一个结构体。
```

4.3.3 awb_result_t

- PROTOTYPE

```
typedef struct isp_awb_result {
    struct isp_wb_gain wb_gain_output;
    HW_S32 color_temp_output;
} awb_result_t;
```

- MEMBERS

```
wb_gain_output ： 白平衡输出增益。
color_temp_output ： 色温输出参考值。
```

- DESCRIPTION

```
awb_result_t 为用于描述AWB 输出结果的一个结构体。
```

4.4 AF 算法开发

算法管理单元通过 af_init 函数初始化 AF 算法结构体，并返回 isp_af_core_ops_t 操作集合：

```
/*isp_3a_af.h 中定义的AF 算法相关的操作集*/
typedef struct isp_af_core_ops {
    HW_S32 (*isp_af_set_params)(void * af_core_obj, af_param_t *param, af_result_t *result)
    ;
    HW_S32 (*isp_af_get_params)(void *af_core_obj, af_param_t *param);
    HW_S32 (*isp_af_run)(void *af_core_obj, af_stats_t *stats, af_result_t *result);
} isp_af_core_ops_t;
void* af_init(isp_af_core_ops_t **af_core_ops);
void af_exit(void *af_core_obj);
```

通过该操作集合，可以控制 AF 参数，调度 AF 算法，具体流程详见 3A 算法库注册部分，接口中用到主要结构体描述如下：

4.4.1 isp_af_stats_s

- PROTOTYPE

```
struct isp_af_stats_s {
    HW_U64 af_iir[ISP_AF_ROW][ISP_AF_COL];
    HW_U64 af_fir[ISP_AF_ROW][ISP_AF_COL];
    HW_U64 af_iir_cnt[ISP_AF_ROW][ISP_AF_COL];
    HW_U64 af_fir_cnt[ISP_AF_ROW][ISP_AF_COL];
    HW_U64 af_hlt_cnt[ISP_AF_ROW][ISP_AF_COL];

    HW_U32 af_count[ISP_AF_ROW][ISP_AF_COL];
    HW_U32 af_h_d1[ISP_AF_ROW][ISP_AF_COL];
    HW_U32 af_h_d2[ISP_AF_ROW][ISP_AF_COL];
    HW_U32 af_v_d1[ISP_AF_ROW][ISP_AF_COL];
    HW_U32 af_v_d2[ISP_AF_ROW][ISP_AF_COL];
};
```

- MEMBERS

```
af_count ： 锐像素计数。
af_h_d1 ： 水平AF 统计值1。
af_h_d2 ： 水平AF 统计值2。
af_v_d1 ： 垂直AF 统计值1。
af_v_d2 ： 垂直AF 统计值2。
```

- DESCRIPTION

```
isp_af_stats_s 为用于描述AF 统计值的一个结构体。
```

4.4.2 af_param_t

- PROTOTYPE

```
typedef struct isp_af_param {
    af_param_type_t type;
    HW_U32 current_frame_cnt;
    union{
        HW_S32 isp_platform_id;
        HW_S32 af_frame_id;
        af_ini_cfg_t af_ini;
        HW_S32 focus_absolute;
        HW_S32 focus_relative;
        enum auto_focus_run_mode af_run_mode;
        enum auto_focus_metering_mode af_metering_mode;
        enum auto_focus_range_new af_range;
        struct vcm_para vcm;
        bool focus_lock;
        isp_sensor_info_t sensor_info;
        af_test_config_t test_cfg;
        HW_S32 auto_focus_trigger;
    }u;
} af_param_t;
```

- MEMBERS

```
type ：ISP AF 命令类型，其每种类型与联合体u 中参数一一对应，定义如下：
typedef enum isp_af_param_type {
    ISP_AF_PLATFORM_ID,
    ISP_AF_FRAME_ID,
    ISP_AF_INI_DATA,
    ISP_AF_FOCUS_ABSOLUTE,
    ISP_AF_FOCUS_RELATIVE,
    ISP_AF_RUN_MODE,
    ISP_AF_METERING_MODE,
    ISP_AF_RANGE,
    ISP_AF_VCM_PARAM,
    ISP_AF_LOCK,
    ISP_AF_SENSOR_INFO,
    ISP_AF_TEST_CONFIG,
    ISP_AF_TRIGGER,
    ISP_AF_PARAM_TYPE_MAX,
} af_param_type_t;
isp_platform_id ：ISP 平台ID。
af_frame_id ：AF 算法执行计数。
af_ini ：AF 算法初始化INI 配置。
focus_absolute ：对焦位置绝对值。
focus_relative ：对焦位置相对值。
af_run_mode ：AF 运行模式。
af_metering_mode ：AF 测量模式。
af_range ：AF 范围。
vcm ：AF VCM 配置。
focus_lock ：焦距锁定。
sensor_info ：Sensor 信息，包括VTS，HTS 以及输出宽高等。
test_cfg ：AF 测试配置。
auto_focus_trigger ：自动对焦触发。
```

- DESCRIPTION

```
af_param_t 为用于描述AF 命令参数的一个结构体。
```

4.4.3 af_result_t

- PROTOTYPE

```
typedef struct isp_af_result {
    enum auto_focus_status af_status_output;
    HW_U32 last_code_output;
    HW_U32 real_code_output;
    HW_U32 std_code_output;
    HW_U16 af_sap_lim_output;
    HW_U32 af_sharp_output;
} af_result_t;
```

- MEMBERS

```
af_status_output ： AF 状态输出。
last_code_output ： VCM 上一次Code 值输出（归一化到0－1224）。
real_code_output ： VCM 实际Code 值输出（对应配置文件）。
std_code_output ： VCM 标准Code 值输出（归一化到0－1224）。
af_sap_lim_output ： AF 锐度限制输出。
af_sharp_output ： 清晰度参考值。
```

- DESCRIPTION

```
af_result_t 为用于描述AF 输出结果的一个结构体。
```

5 API 接口

5.1 media_dev_init

【目的】

初始化 Media 相关设备, 如 ISP, Sensor, CSI 等。

【语法】

int media_dev_init(void);

【参数】

参数	描述

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

头文件：isp.h 库文件：libisp_dev.a

【注意】

在进行 ISP 相关操作前应该首先调用此接口，保证 ISP 设备已经初始化。

【举例】

```
/*declaration*/
int ret = 0;
/* init VI device*/
ret = media_dev_init();
if (ret)
{
    return -1;
}
```

5.2 media_dev_exit

【目的】

去初始化 Media 设备。

【语法】

void media_dev_exit(void);

【参数】

参数	描述

【返回值】

返回值	描述

【需求】

头文件：isp.h 库文件：libisp_dev.a

【注意】

调用 media_dev_exit 之前要确保设备已经初始化，与 media_dev_exit 调用次数对应。

【举例】

无。

5.3 isp_init

【目的】

初始化 ISP 设备。

【语法】

int isp_init(int dev_id);

【参数】

参数	描述
dev_id	ISP 设备号

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

头文件：isp.h

库文件：libisp_dev.a

【注意】

初始化 ISP 设备，分配 ISP 所需内存资源，初始化 ISP 配置参数。

【举例】

无。

5.4 isp_exit

【目的】

退出 ISP 设备。

【语法】

int isp_exit(int dev_id);

【参数】

参数	描述
dev_id	ISP 设备号

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

头文件：isp.h

库文件：libisp_dev.a

【注意】

该操作用于退出 ISP 设备工作，并释放相关资源。

【举例】

无。

5.5 isp_run

【目的】

运行 ISP 设备。

【语法】

int isp_run(int dev_id);

【参数】

参数	描述
dev_id	ISP 设备号

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

头文件：isp.h

库文件：libisp_dev.a

【注意】

该操作用于启动 ISP 处理线程，该线程用于监听 ISP 设备中断、调度 ISP 算法、配置 ISP 参数等工作。

【举例】

无。

5.6 isp_pthread_join

【目的】

等待 ISP 处理线程。

【语法】

int isp_pthread_join(int dev_id);

【参数】

参数	描述
dev_id	ISP 设备号

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

头文件：isp.h

库文件：libisp_dev.a

【注意】

该操作用于等待 ISP 处理线程处理结束。

【举例】

无。

5.7 isp_stats_req

【目的】

获取 ISP 统计值。

【语法】

int isp_stats_reqs(int dev_id, struct isp_stats_context *stats_ctx);

【参数】

参数	描述
dev_id	ISP 设备号
stats_ctx	VI 通道号

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

头文件：isp.h

库文件：libisp_dev.a

【注意】

该操作用来获取 ISP 统计值信息，包括 AE、AWB、AF、PLTM 等模块统计值。

【举例】

无。

5.8 isp_get_cfg

【目的】

获取 ISP 配置参数。

【语法】

int isp_get_cfg(int dev_id, unsigned char group_id, unsigned int cfg_ids, void *cfg_data);

【参数】

参数	描述
dev_id	ISP 设备号
group_id	命令组 ID
cfg_ids	配置 ID
cfg_data	配置数据结构

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

头文件：isp.h

库文件：libisp_dev.a

【注意】

该操作用来获取 ISP 模块参数，通过命令组 + 配置方式定位具体模块参数，关于命令组与配置命令后面数据结构部分会有具体介绍。

【举例】

无。

5.9 isp_set_cfg

【目的】

设置 ISP 配置参数。

【语法】

int isp_set_cfg(int dev_id, unsigned char group_id, unsigned int cfg_ids, void *cfg_data);

【参数】

参数	描述
dev_id	ISP 设备号
group_id	命令组 ID
cfg_ids	配置 ID
cfg_data	配置数据结构

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

头文件：isp.h

库文件：libisp_dev.a

【注意】

该操作用来设置 ISP 模块参数，通过命令组 + 配置方式定位具体模块参数，关于命令组与配置命令后面数据结构部分会有具体介绍。

【举例】

无。

6 数据结构

6.1 命令组

6.1.1 hw_isp_cfg_groups

- PROTOTYPE

```
typedef enum {
    HW_ISP_CFG_TEST = 0x01,
    HW_ISP_CFG_3A = 0x02,
    HW_ISP_CFG_TUNING = 0x03,
    HW_ISP_CFG_DYNAMIC = 0x04,
    HW_ISP_CFG_GROUP_COUNT
} hw_isp_cfg_groups;
```

- MEMBERS

```
HW_ISP_CFG_TEST : 测试参数配置组
HW_ISP_CFG_3A : 3A 配置参数组
HW_ISP_CFG_TUNING : Tuning 参数组
HW_ISP_CFG_DYNAMIC : 动态参数配置组
HW_ISP_CFG_GROUP_COUNT : 组计数
```

- DESCRIPTION

```
hw_isp_cfg_groups 用于枚举ISP 参数命令组。
```

6.2 Test 命令组

6.2.1 hw_isp_cfg_test_ids

- PROTOTYPE

```
typedef enum {
    /* isp_test_param_cfg */
    HW_ISP_CFG_TEST_PUB = 0x00000001,
    HW_ISP_CFG_TEST_EXPTIME = 0x00000002,
    HW_ISP_CFG_TEST_GAIN = 0x00000004,
    HW_ISP_CFG_TEST_FOCUS = 0x00000008,
    HW_ISP_CFG_TEST_FORCED = 0x00000010,
    HW_ISP_CFG_TEST_ENABLE = 0x00000020,
} hw_isp_cfg_test_ids;
```

- MEMBERS

```
HW_ISP_CFG_TEST_PUB : 测试参数公共属性
HW_ISP_CFG_TEST_EXPTIME : 曝光时间测试配置
HW_ISP_CFG_TEST_GAIN : 模拟增益测试配置
HW_ISP_CFG_TEST_FOCUS : 对焦测试配置
HW_ISP_CFG_TEST_FORCED : 强制AE 测试配置
HW_ISP_CFG_TEST_ENABLE : 模块使能配置
```

- DESCRIPTION

```
hw_isp_cfg_test_ids 用于枚举测试命令组中的相关测试命令
```

6.2.2 isp_test_param_cfg

- PROTOTYPE

```
/* isp_test_param cfg */
struct isp_test_param_cfg {
    struct isp_test_pub_cfg test_pub; /* id: 0x0100000001 */
    struct isp_test_item_cfg test_exptime; /* id: 0x0100000002 */
    struct isp_test_item_cfg test_gain; /* id: 0x0100000004 */
    struct isp_test_item_cfg test_focus; /* id: 0x0100000008 */
    struct isp_test_forced_cfg test_forced; /* id: 0x0100000010 */
    struct isp_test_enable_cfg test_enable; /* id: 0x0100000020 */
};
```

- MEMBERS

```
test_pub : 测试配置公共属性
test_exptime : 测试曝光时间配置
test_gain : 测试模拟增益配置
test_focus : 测试对焦马达配置
test_forced : 强制自动曝光配置
test_enable : 模块使能配置
```

- DESCRIPTION

isp_test_param_cfg 用于设置测试模式相关配置，其与hw_isp_cfg_test_ids 中所定义命令相对应。

6.2.3 isp_test_pub_cfg

- PROTOTYPE

```
struct isp_test_pub_cfg {
    HW_S32 test_mode;
    HW_S32 gain;
    HW_S32 exp_line;
    HW_S32 color_temp;
    HW_S32 log_param;
};
```

- MEMBERS

```
test_mode ： 测试模式，无效
gain ： 模拟增益，gain=16 代表1 倍增益，范围16~65535
exp_line ： 曝光行数，exp_line=16 代表1 行曝光时间，范围16~65535
color_temp ： 当前测试光源色温，1900K~13000K
log_param ： ISP Debug 掩码，其每个bit 意义如下:
#define ISP_LOG_AE              (1 << 0)    //0x1
#define ISP_LOG_AWB             (1 << 1)    //0x2
#define ISP_LOG_AF              (1 << 2)    //0x4
#define ISP_LOG_ISO             (1 << 3)    //0x8
#define ISP_LOG_GAMMA               (1 << 4)    //0x10
#define ISP_LOG_COLOR_MATRIX            (1 << 5)    //0x20
#define ISP_LOG_AFS             (1 << 6)    //0x40
#define ISP_LOG_MOTION_DETECT           (1 << 7)    //0x80
#define ISP_LOG_GAIN_OFFSET         (1 << 8)    //0x100
#define ISP_LOG_DEFOG               (1 << 9)    //0x200
#define ISP_LOG_LSC             (1 << 10)   //0x400
#define ISP_LOG_GTM             (1 << 11)   //0x800
#define ISP_LOG_PLTM                (1 << 12)   //0x1000
#define ISP_LOG_SUBDEV              (1 << 13)   //0x2000
#define ISP_LOG_CFG             (1 << 14)   //0x4000
#define ISP_LOG_VIDEO               (1 << 15)   //0x8000
#define ISP_LOG_ISP             (1 << 16)   //0x10000
#define ISP_LOG_FLASH               (1 << 17)   //0x20000
```

- DESCRIPTION

isp_test_pub_cfg 用于描述测试参数公共属性，与HW_ISP_CFG_TEST_PUB 命令相对应。

6.2.4 isp_test_item_cfg

- PROTOTYPE

```
struct isp_test_item_cfg {
    HW_S32 enable;
    HW_S32 start;
    HW_S32 step;
    HW_S32 end;
    HW_S32 change_interval;
};
```

- MEMBERS

```
enable ： 使能测试项，范围0~1
start ： 测试起始值，范围0~65535
step ： 测试步进值，范围0~65535
end ： 测试结束值，范围0~65535
change_interval ： 测试间隔帧数，范围0~1024
```

- DESCRIPTION

```
isp_test_item_cfg 用于描述测试项目具体参数，与HW_ISP_CFG_TEST_EXPTIME、
HW_ISP_CFG_TEST_GAIN 以及HW_ISP_CFG_TEST_FOCUS 配置命令相对应。
```

6.2.5 isp_test_forced_cfg

- PROTOTYPE

```
struct isp_test_forced_cfg {
    HW_S32 ae_enable;
    HW_S32 lum;
};
```

- MEMBERS

```
ae_enable ： 强制AE 使能，范围0~1
lum ： 强制AE 目标亮度，范围0~255
```

- DESCRIPTION

```
isp_test_forced_cfg 用于配置强制AE 测试模式，与HW_ISP_CFG_TEST_FORCED 配置命令相对应。
```

6.2.6 isp_test_enable_cfg

- PROTOTYPE

```
struct isp_test_enable_cfg {
    HW_S32 manual;
    HW_S32 afs;
    HW_S32 sharp;
    HW_S32 contrast;
    HW_S32 denoise;
    HW_S32 drc;
    HW_S32 cem;
    HW_S32 lsc;
    HW_S32 gamma;
    HW_S32 cm;
    HW_S32 ae;
    HW_S32 af;
    HW_S32 awb;
    HW_S32 hist;
    HW_S32 blc;
    HW_S32 so;
    HW_S32 wb;
    HW_S32 otf_dpc;
    HW_S32 cfa;
    HW_S32 tdf;
    HW_S32 cnr;
    HW_S32 saturation;
    HW_S32 defog;
    HW_S32 linearity;
    HW_S32 hdr_gamma;
    HW_S32 gtm;
    HW_S32 dig_gain;
    HW_S32 pltm;
};
```

- MEMBERS

```
manual ： 手动模块开关使能
afs ： 自动工频检测使能
sharp ： 锐化使能
contrast ： 对比度使能
denoise ： 2D 降噪使能
drc ： 动态范围压缩使能
cem ： 颜色增强使能
lsc ： 镜头阴影矫正使能
gamma ： Gamma 矫正使能
cm ： Color Matrix 使能
ae ： 自动曝光使能
af ： 自动对焦使能
awb ： 自动白平衡使能
hist ： 自动直方图统计使能
blc ： 黑电平矫正使能
so ： Sensor Offset 使能
wb ： 白平衡校正使能
otf_dpc ： 坏点补偿使能
cfa ： CFA 插值使能
tdf ： 3D 降噪使能
cnr ： 色度降噪使能
saturation ： 饱和度调整使能
defog ： 去雾使能
linearity ： 线性度校正使能
hdr_gamma ： 无效
```

```
gtm : 全局色调映射使能
dig_gain : 数字增益使能
pltm : 局部色调映射使能
```

- DESCRIPTION

```
isp_test_enable_cfg 用于使能具体模块功能的开关，与HW_ISP_CFG_TEST_ENABLE 配置命令相对应。
```

6.3 3A 命令组

6.3.1 hw_isp_cfg_3a_ids

- PROTOTYPE

```
typedef enum {
    /* isp_3a_param_cfg */
    /* ae */
    HW_ISP_CFG_AE_PUB = 0x00000001,
    HW_ISP_CFG_AE_PREVIEW_TBL = 0x00000002,
    HW_ISP_CFG_AE_CAPTURE_TBL = 0x00000004,
    HW_ISP_CFG_AE_VIDEO_TBL = 0x00000008,
    HW_ISP_CFG_AE_WIN_WEIGHT = 0x00000010,
    HW_ISP_CFG_AE_DELAY = 0x00000020,
    /* awb */
    HW_ISP_CFG_AWB_SPEED = 0x00000040,
    HW_ISP_CFG_AWB_TEMP_RANGE = 0x00000080,
    HW_ISP_CFG_AWB_DIST = 0x00000100,
    HW_ISP_CFG_AWB_LIGHT_INFO = 0x00000200,
    HW_ISP_CFG_AWB_EXT_LIGHT_INFO= 0x00000400,
    HW_ISP_CFG_AWB_SKIN_INFO = 0x00000800,
    HW_ISP_CFG_AWB_SPECIAL_INFO = 0x00001000,
    HW_ISP_CFG_AWB_PRESET_GAIN = 0x00002000,
    HW_ISP_CFG_AWB_FAVOR = 0x00004000,
    /* af */
    HW_ISP_CFG_AF_VCM_CODE = 0x00008000,
    HW_ISP_CFG_AF_OTP = 0x00010000,
    HW_ISP_CFG_AF_SPEED = 0x00020000,
    HW_ISP_CFG_AF_FINE_SEARCH = 0x00040000,
    HW_ISP_CFG_AF_REFOCUS = 0x00080000,
    HW_ISP_CFG_AF_TOLERANCE = 0x00100000,
    HW_ISP_CFG_AF_SCENE = 0x00200000,
} hw_isp_cfg_3a_ids
```

- MEMBERS

```
/* ae */
HW_ISP_CFG_AE_PUB : AE 公共属性
HW_ISP_CFG_AE_PREVIEW_TBL : AE 预览曝光表
```

```
HW_ISP_CFG_AE_CAPTURE_TBL : AE 拍照曝光表
HW_ISP_CFG_AE_VIDEO_TBL : AE 视频曝光表
HW_ISP_CFG_AE_WIN_WEIGHT : AE 窗口权重
HW_ISP_CFG_AE_DELAY : AE 曝光和增益延时配置
/* awb */
HW_ISP_CFG_AWB_SPEED : AWB 速度控制
HW_ISP_CFG_AWB_TEMP_RANGE : AWB 色温矫正范围
HW_ISP_CFG_AWB_DIST : AWB 特殊场景距离配置
HW_ISP_CFG_AWB_LIGHT_INFO : AWB 参考光源配置
HW_ISP_CFG_AWB_EXT_LIGHT_INFO : AWB 扩展光源配置
HW_ISP_CFG_AWB_SKIN_INFO : AWB 肤色配置
HW_ISP_CFG_AWB_SPECIAL_INFO : AWB 特殊光源配置
HW_ISP_CFG_AWB_PRESET_GAIN : AWB 预设增益配置
HW_ISP_CFG_AWB_FAVOR : AWB 整体偏色配置
/* af */
HW_ISP_CFG_AF_VCM_CODE : AF 马达近端远端配置
HW_ISP_CFG_AF_OTP : AF OTP 使能配置
HW_ISP_CFG_AF_SPEED : AF 速度配置
HW_ISP_CFG_AF_FINE_SEARCH : AF 细搜索配置
HW_ISP_CFG_AF_REFOCUS : AF 重新对焦判定配置
HW_ISP_CFG_AF_TOLERANCE : AF 重对焦容忍度配置
HW_ISP_CFG_AF_SCENE : AF 场景配置
```

- DESCRIPTION

```
hw_isp_cfg_3a_ids 列出了3A 相关的配置命令，这些配置一般为不随曝光变化的配置。
```

6.3.2 isp_3a_param_cfg

- PROTOTYPE

```
/* isp_3a_param cfg*/
struct isp_3a_param_cfg {
    /* ae */
    struct isp_ae_pub_cfg ae_pub; /* id: 0x0200000001 */
    struct isp_ae_table_cfg ae_preview_tbl; /* id: 0x0200000002 */
    struct isp_ae_table_cfg ae_capture_tbl; /* id: 0x0200000004 */
    struct isp_ae_table_cfg ae_video_tbl; /* id: 0x0200000008 */
    struct isp_ae_weight_cfg ae_win_weight; /* id: 0x0200000010 */
    struct isp_ae_delay_cfg ae_delay; /* id: 0x0200000020 */
    /* awb */
    struct isp_awb_speed_cfg awb_speed; /* id: 0x0200000040 */
    struct isp_awb_temp_range_cfg awb_temp_range; /* id: 0x0200000080 */
    struct isp_awb_dist_cfg awb_dist; /* id: 0x0200000100 */
    struct isp_awb_temp_info_cfg awb_light_info; /* id: 0x0200000200 */
    struct isp_awb_temp_info_cfg awb_ext_light_info; /* id: 0x0200000400 */
    struct isp_awb_temp_info_cfg awb_skin_info; /* id: 0x0200000800 */
    struct isp_awb_temp_info_cfg awb_special_info; /* id: 0x0200001000 */
    struct isp_awb_preset_gain_cfg awb_preset_gain; /* id: 0x0200002000 */
    struct isp_awb_favor_cfg awb_favor; /* id: 0x0200004000 */
    /* af */
    struct isp_af_vcm_code_cfg af_vcm_code; /* id: 0x0200008000 */
    struct isp_af_otp_cfg af_otp; /* id: 0x0200010000 */
```

```
    struct isp_af_speed_cfg af_speed; /* id: 0x0200020000 */
    struct isp_af_fine_search_cfg af_fine_search; /* id: 0x0200040000 */
    struct isp_af_refocus_cfg af_refocus; /* id: 0x0200080000 */
    struct isp_af_tolerance_cfg af_tolerance; /* id: 0x0200100000 */
    struct isp_af_scene_cfg af_scene; /* id: 0x0200200000 */
};
```

- MEMBERS

```
ae_pub ：AE 公共属性
ae_preview_tbl ：AE 预览曝光表
ae_capture_tbl ：AE 拍照曝光表
ae_video_tbl ：AE 视频曝光表
ae_win_weight ：AE 窗口权重
ae_delay ：AE 曝光和增益延时配置
awb_speed ：AWB 速度控制
awb_temp_range ：AWB 色温矫正范围
awb_dist ：AWB 特殊场景距离配置
awb_light_info ：AWB 参考光源配置
awb_ext_light_info ：AWB 扩展光源配置
awb_skin_info ：AWB 肤色配置
awb_special_info ：AWB 特殊光源配置
awb_preset_gain ：AWB 预设增益配置
awb_favor ：AWB 整体偏色配置
af_vcm_code ：AF 马达近端远端配置
af_otp ：AF OTP 使能配置
af_speed ：AF 速度配置
af_fine_search ：AF 细搜索配置
af_refocus ：AF 重新对焦判定配置
af_tolerance ：AF 重对焦容忍度配置
af_scene ：AF 场景配置
```

- DESCRIPTION

```
isp_3a_param_cfg 用于描述3A 相关配置参数，其与hw_isp_cfg_3a_ids 中定义的命令相对应。
```

6.3.3 AE 控制命令

6.3.3.1 isp_ae_pub_cfg

- PROTOTYPE

```
struct isp_ae_pub_cfg {
    HW_S32 define_table;
    HW_S32 max_lv;
    HW_S32 hist_mode_en;
    HW_S32 compensation_step;
    HW_S32 touch_dist_index;
```

```
    HW_S32 iso2gain_ratio;
    HW_S32 fno_table[16];
};
```

- MEMBERS

```
define_table ： 是否使用自定义AE table
max_lv ： AE Table 第一项所对应的亮度值LV，range：0~2500，default：1600
hist_mode_en ： 是否启用直方图统计来计算AE，range：0~1，default：1
compensation_step ： AE 曝光补偿所跳表的步数，25 为1 倍，range：0~32，default：7
touch_dist_index ： 点对焦模式，相关点大小，range：0~5，default：2
iso2gain_ratio; ： ISO 到Gain 的转化系数，如果一倍增益对应ISO 50，则该值为32，
iso2gain_ratio = 1(倍)*16*100/50
fno_table ： 光圈查找表
```

- DESCRIPTION

```
isp_ae_pub_cfg 用于描述AE 的公共属性
```

6.3.3.2 isp_ae_table_cfg

- PROTOTYPE

```
struct isp_ae_table_cfg {
    HW_S32 length;
    struct ae_table value[7];
};
```

- MEMBERS

```
length ： 每个表中的节点个数，范围为1~7。
value ： Ae table 中每个节点的描述，包括最大最小曝光，最大最小增益，最大最小光圈，range：0~65535，
    default：无
```

- DESCRIPTION

```
isp_ae_table_cfg 为用于描述AE 调整表的一个结构体。
```

6.3.3.3 isp_ae_weight_cfg

- PROTOTYPE

```
struct isp_ae_weight_cfg {
    HW_S32 weight[64];
};
```

- MEMBERS

```
weight ： AE 窗口权重，分8*8 个窗口，内部插值到48*32。range：0~128，default：无
```

- DESCRIPTION

```
isp_ae_weight_cfg 为用于描述AE 窗口权重的一个结构体。
```

6.3.3.4 isp_ae_delay_cfg

- PROTOTYPE

```
struct isp_ae_delay_cfg {
    HW_S32 ae_frame;
    HW_S32 exp_frame;
    HW_S32 gain_frame;
};
```

- MEMBERS

```
ae_frame ： AE 调整延时帧数。range：0~64，default：0
exp_frame ： 曝光生效延时帧数。range：1~2，default：2
gain_frame ： 增益生效延时帧数。range：1~2，default：2
```

- DESCRIPTION

```
isp_ae_delay_cfg 为用于描述AE 延时设置的一个结构体。
```

6.3.4 AWB 控制命令

6.3.4.1 isp_awb_speed_cfg

- PROTOTYPE

```
struct isp_awb_speed_cfg {
    HW_S32 interval;
    HW_S32 value;
};
```

- MEMBERS

```
interval ：AWB 调整间隔帧数。range: 1~64, default: 1
value ：AWB 调整速度，数字越大速度越慢。range: 1~47, default: 16
```

- DESCRIPTION

```
isp_awb_speed_cfg 为用于描述AWB 速度设置的一个结构体。
```

6.3.4.2 isp_awb_temp_range_cfg

- PROTOTYPE

```
struct isp_awb_temp_range_cfg {
    HW_S32 low;
    HW_S32 high;
    HW_S32 base;
};
```

- MEMBERS

```
low ：AWB 校正最低色温。range: 1000~13000, default: 1900
high ：AWB 校正最高色温。range: 1000~13000, default: 8500
base ：AWB 校正基准色温。range: 1000~13000, default: 5500
```

- DESCRIPTION

```
isp_awb_temp_range_cfg 为用于描述AWB 色温限制的一个结构体。
```

6.3.4.3 isp_awb_dist_cfg

- PROTOTYPE

```
struct isp_awb_dist_cfg {
    HW_S32 green_zone;
    HW_S32 blue_sky;
};
```

- MEMBERS

```
green_zone ： 绿区距离限定，一般设置60~90 之间。range: 0~256，default: 75
blue_sky ： 蓝天距离限定，一般设置60~90 之间。range: 0~256，default: 75
```

- DESCRIPTION

```
isp_awb_dist_cfg 为用于描述AWB 特殊场景配置的一个结构体。
```

6.3.4.4 isp_awb_temp_info_cfg

- PROTOTYPE

```
struct isp_awb_temp_info_cfg {
    HW_S32 number;
    HW_S32 value[320];
};
```

- MEMBERS

```
number ： 参考光源数量，不超过32 个。
value ： 参考光源描述，具体意义如下图:
```

图 6-1

- DESCRIPTION

```
isp_awb_temp_info_cfg 为用于描述AWB 参考光源信息的一个结构体。
```

6.3.4.5 isp_awb_preset_gain_cfg

- PROTOTYPE

```
struct isp_awb_preset_gain_cfg{
    HW_S32 value[22];
};
```

- MEMBERS

```
value ：对应11 种白平衡模式，每个模式需要两个值，前两种可填为256,256。
enum white_balance_mode
{
    WB_MANUAL = 0,
    WB_AUTO = 1,
    WB_INCANDESCENT = 2,
    WB_FLUORESCENT = 3,
    WB_FLUORESCENT_H = 4,
    WB_HORIZON = 5,
    WB_DAYLIGHT = 6,
    WB_FLASH = 7,
    WB_CLOUDY = 8,
    WB_SHADE = 9,
    WB_TUNGSTEN = 10,
};
```

- DESCRIPTION

```
isp_awb_preset_gain_cfg 为用于描述AWB 配置手动白平衡的的一个结构体。
```

6.3.4.6 isp_awb_favor_cfg

- PROTOTYPE

```
struct isp_awb_favor_cfg {
    HW_S32 rgain;
    HW_S32 bgain;
};
```

- MEMBERS

```
rgain : 红色通道增益。range: 0~4095, default: 256
bgain : 蓝色通道增益。range: 0~4095, default: 256
```

- DESCRIPTION

```
isp_awb_favor_cfg 为用于描述AWB 偏好配置的一个结构体。
```

6.3.5 AF 控制命令

6.3.5.1 isp_af_vcm_code_cfg

- PROTOTYPE

```
struct isp_af_vcm_code_cfg {
    HW_S32 min;
    HW_S32 max;
};
```

- MEMBERS

```
min : 无限远端VCM code 值，一般对准5m 即可。range: 0~1023, default: 100
max : 微距端VCM code 值，一般对准7~10 cm。range: 0~1023, default: 600
```

- DESCRIPTION

```
isp_af_vcm_code_cfg 为用于描述AF VCM 马达最小最大配置的一个结构体。
```

6.3.5.2 isp_af_otp_cfg

- PROTOTYPE

```
struct isp_af_otp_cfg {
    HW_S32 use_otp;
};
```

- MEMBERS

```
use_otp ：是否使用OTP 配置，目前该选项无效。range：0~1
```

- DESCRIPTION

```
isp_af_otp_cfg 为用于配置AF 算法是否使用OTP 校准。
```

6.3.5.3 isp_af_speed_cfg

- PROTOTYPE

```
struct isp_af_speed_cfg {
    HW_S32 interval_time;
    HW_S32 index;
};
```

- MEMBERS

```
interval_time ：AF 调整每步间隔时间单位为ms。range：0~1000，default：100
index ：微距端VCM code 值，一般对准7~10 cm。range：0~1023，default：600
```

- DESCRIPTION

```
isp_af_speed_cfg 为用于控制AF 速度的一个结构体。
```

6.3.5.4 isp_af_fine_search_cfg

- PROTOTYPE

```
struct isp_af_fine_search_cfg {
    HW_S32 auto_en;
    HW_S32 single_en;
    HW_S32 step;
};
```

- MEMBERS

```
auto_en ：连续自动对焦细搜索开关。range：0~1，default：0
single_en ：单次对焦细搜索开关。range：0~1，default：1
step ：细搜索步长，该值越小对焦精度越高，但对焦失败概率会上升。range：0~64，default：10
```

- DESCRIPTION

```
isp_af_fine_search_cfg 为用于控制AF 细搜索配置的一个结构体。
```

6.3.5.5 isp_af_refocus_cfg

- PROTOTYPE

```
struct isp_af_refocus_cfg {
    HW_S32 move_cnt;
    HW_S32 still_cnt;
    HW_S32 move_monitor_cnt;
    HW_S32 still_monitor_cnt;
};
```

- MEMBERS

```
move_cnt ：运动帧计数阈值。range：0~64，default：4
still_cnt ：静止帧计数阈值。range：0~64，default：2
move_monitor_cnt ：运动帧检测数。range：0~64，default：6
still_monitor_cnt ：静止帧检测数。range：0~64，default：3
```

- DESCRIPTION

```
isp_af_refocus_cfg 为用于控制CAF 重新对焦策略配置的一个结构体。当视频连续move_monitor_cnt帧出现多于
    move_cnt 帧的运动帧，且之后连续still_monitor_cnt 帧出现多于still_cnt 静止帧时，则判定重新执行自动
    对焦。
```

6.3.5.6 isp_af_tolerance_cfg

- PROTOTYPE

```
struct isp_af_tolerance_cfg {
    HW_S32 near_distance;
    HW_S32 far_distance;
    HW_S32 offset;
    HW_S32 table_length;
    HW_S32 std_code_table[20];
```

```
    HW_S32 value[20];
};
```

- MEMBERS

```
near_distance ： 无限远端对焦检测灵敏度，range: 0~32，default: 8
far_distance ： 微距端灵敏度，range: 0~32，default: 10
offset ： 该值一般设置为0。range: 0~32，default: 0
table_length ： 灵敏度表长度。range: 0~20，default: 0
std_code_table ： VCM code 表。
value ： 以上VCM code 表对应的期望灵敏度。range: 0~32，default: 8
```

- DESCRIPTION

```
isp_af_tolerance_cfg 为用于控制CAF 重新对焦容忍度配置的一个结构体，table_length/std_code_table/
    value 属于细调节模式，当table_length==0 时，使用near_distance/far_distance/offset 所描述容忍度
    配置，当table_length>0，则需要配置std_code_table/value，这时会使用容忍度细调模式。
```

6.3.5.7 isp_af_scene_cfg

- PROTOTYPE

```
struct isp_af_scene_cfg {
    HW_S32 stable_min;
    HW_S32 stable_max;
    HW_S32 low_light_lv;
    HW_S32 peak_thres;
    HW_S32 direction_thres;
    HW_S32 change_ratio;
    HW_S32 move_minus;
    HW_S32 still_minus;
    HW_S32 scene_motion_thres;
};
```

- MEMBERS

```
stable_min ： 曝光稳定判定范围最小值，曝光不变为256，曝光减小1 倍则为128。range: 0~512，default: 245
stable_max ： 曝光稳定判定范围最大值，曝光不变为256，曝光增加1 倍则为512。range: 0~512，default: 265
low_light_lv ： 低照度判定条件，LV<low_light_lv 则判定为低照度。range: 0~1000，default: 200
peak_thres ： 峰值判定阈值。range: 0~1023，default: 100
direction_thres ： 方向判定阈值。range: 0~1023，default: 10
change_ratio ： FV 变化阈值，当FV 变化比例超过该值，则判定为运动；range: 0~100，default: 20
move_minus ： 该值一般设置为0。range: 0~10，default: 0
still_minus ： 该值一般设置为0。range: 0~10，default: 0
scene_motion_thres ： 运动检测阈值。range: 0~100，default: 10
```

- DESCRIPTION

isp_af_scene_cfg 为AF 判定场景运动所需的参数结构体。

6.4 Tuning 命令组

6.4.1 hw_isp_cfg_tuning_ids

- PROTOTYPE

```
typedef enum {
    /* isp_tuning_param_cfg */
    HW_ISP_CFG_TUNING_FLASH = 0x00000001,
    HW_ISP_CFG_TUNING_FLICKER = 0x00000002,
    HW_ISP_CFG_TUNING_DEFOG = 0x00000004,
    HW_ISP_CFG_TUNING_VISUAL_ANGLE = 0x00000008,
    HW_ISP_CFG_TUNING_GTM = 0x00000010,
    HW_ISP_CFG_TUNING_DPC_OTF= 0x00000020,
    HW_ISP_CFG_TUNING_BLACK_LV = 0x00000040,
    HW_ISP_CFG_TUNING_BAYER_GAIN = 0x00000080,
    HW_ISP_CFG_TUNING_CCM_LOW = 0x00000100,
    HW_ISP_CFG_TUNING_CCM_MID = 0x00000200,
    HW_ISP_CFG_TUNING_CCM_HIGH = 0x00000400,
    HW_ISP_CFG_TUNING_LSC = 0x00000800,
    HW_ISP_CFG_TUNING_GAMMA = 0x00001000,
    HW_ISP_CFG_TUNING_LINEARITY = 0x00002000,
    HW_ISP_CFG_TUNING_DISTORTION = 0x00004000,
    HW_ISP_CFG_TUNING_BDNF = 0x00008000,
    HW_ISP_CFG_TUNING_TDNF = 0x00010000,
    HW_ISP_CFG_TUNING_CONTRAST = 0x00020000,
    HW_ISP_CFG_TUNING_SHARP = 0x00040000,
    HW_ISP_CFG_TUNING_CEM = 0x00080000,
    HW_ISP_CFG_TUNING_PLTM = 0x00100000,
} hw_isp_cfg_tuning_ids;
```

- MEMBERS

```
HW_ISP_CFG_TUNING_FLASH ： 闪光灯配置。
HW_ISP_CFG_TUNING_FLICKER ： 工频检测配置。
HW_ISP_CFG_TUNING_DEFOG ： 去雾配置。
HW_ISP_CFG_TUNING_VISUAL_ANGLE ： 视场角配置。
HW_ISP_CFG_TUNING_GTM ： 全局色调映射配置（全局对比度调整）。
HW_ISP_CFG_TUNING_DPC_OTF ： 去坏点配置。
HW_ISP_CFG_TUNING_BLACK_LV ： 黑电平配置。
HW_ISP_CFG_TUNING_BAYER_GAIN ： Bayer 增益配置。
HW_ISP_CFG_TUNING_CCM_LOW ： 高色温下CCM 配置。
HW_ISP_CFG_TUNING_CCM_MID ： 中色温下CCM 配置。
HW_ISP_CFG_TUNING_CCM_HIGH ： 低色温下CCM 配置。
HW_ISP_CFG_TUNING_LSC ： 镜头阴影矫正配置。
HW_ISP_CFG_TUNING_GAMMA ： 伽马配置。
HW_ISP_CFG_TUNING_LINEARITY ： 线性度矫正配置。
HW_ISP_CFG_TUNING_DISTORTION ： 畸变矫正配置（V5 不支持）。
HW_ISP_CFG_TUNING_BDNF ： 2D 降噪配置。
```

```
HW_ISP_CFG_TUNING_TDNF : 3D 降噪配置。
HW_ISP_CFG_TUNING_CONTRAST : 对比度配置。
HW_ISP_CFG_TUNING_SHARP : 锐化配置。
HW_ISP_CFG_TUNING_CEM : 颜色增强配置。
HW_ISP_CFG_TUNING_PLTM : PLTM 配置。
```

- DESCRIPTION

```
hw_isp_cfg_tuning_ids 列出了一些基本模块的配置命令。
```

6.4.2 isp_tuning_param_cfg

- PROTOTYPE

```
/* isp_tuning_param cfg */
struct isp_tuning_param_cfg {
    struct isp_tuning_flash_cfg flash; /* id: 0x0300000001 */
    struct isp_tuning_flicker_cfg flicker; /* id: 0x0300000002 */
    struct isp_tuning_defog_cfg defog; /* id: 0x0300000004 */
    struct isp_tuning_visual_angle_cfg visual_angle; /* id: 0x0300000008 */
    struct isp_tuning_gtm_cfg gtm; /* id: 0x0300000010 */
    struct isp_tuning_dpc_otf_cfg dpc_otf; /* id: 0x0300000020 */
    struct isp_tuning_blc_gain_cfg black_level; /* id: 0x0300000040 */
    struct isp_tuning_blc_gain_cfg bayer_gain; /* id: 0x0300000080 */
    struct isp_tuning_ccm_cfg ccm_low; /* id: 0x0300000100 */
    struct isp_tuning_ccm_cfg ccm_mid; /* id: 0x0300000200 */
    struct isp_tuning_ccm_cfg ccm_high; /* id: 0x0300000400 */
    struct isp_tuning_lens_shading_cfg lsc; /* id: 0x0300000800 */
    struct isp_tuning_gamma_table_cfg gamma; /* id: 0x0300001000 */
    struct isp_tuning_linearity_cfg linearity; /* id: 0x0300002000 */
    struct isp_tuning_distortion_cfg distortion; /* id: 0x0300004000 */
    struct isp_tuning_bdnf_cfg bdnf; /* id: 0x0300008000 */
    struct isp_tuning_tdnf_cfg tdnf; /* id: 0x0300010000 */
    struct isp_tuning_contrast_cfg contrast; /* id: 0x0300020000 */
    struct isp_tuning_sharp_cfg sharp; /* id: 0x0300040000 */
    struct isp_tuning_cem_cfg cem; /* id: 0x0300080000 */
    struct isp_tuning_pltm_cfg pltm; /* id: 0x0300100000 */
};
```

- MEMBERS

```
flash : 闪光灯配置。
flicker : 工频检测配置。
defog : 去雾配置。
visual_angle : 视场角配置。
gtm : 全局色调映射配置（全局对比度调整）。
dpc_otf : 去坏点配置。
black_level : 黑电平配置。
bayer_gain : Bayer 增益配置。
ccm_low : 高色温下CCM 配置。
```

```
ccm_mid ：中色温下CCM 配置。
ccm_high ：低色温下CCM 配置。
lsc ：镜头阴影矫正配置。
gamma ：伽马配置。
linearity ：线性度矫正配置。
distortion ：畸变矫正配置（V5 不支持）。
bdnf ：2D 降噪配置。
tdnf ：3D 降噪配置。
contrast ：对比度配置。
sharp ：锐化配置。
cem ：颜色增强配置。
pltm ：PLTM 配置。
```

- DESCRIPTION

```
isp_tuning_param_cfg 用于描述ISP 基本模块的配置参数。
```

6.4.3 isp_tuning_flash_cfg

- PROTOTYPE

```
struct isp_tuning_flash_cfg {
    HW_S32 gain;
    HW_S32 delay_frame;
};
```

- MEMBERS

```
gain ：闪光比例，256*（预闪亮度/主闪亮度）。range: 0~1023, default: 64
delay_frame ：预闪帧数。range: 0~63, default: 16
```

- DESCRIPTION

```
isp_tuning_flash_cfg 为描述闪光灯模块配置的结构体。
```

6.4.4 isp_tuning_flicker_cfg

- PROTOTYPE

```
struct isp_tuning_flicker_cfg {
    HW_S32 type;
```

```
    HW_S32 ratio;
};
```

- MEMBERS

```
type ： 工频检测类型；0 为自动检测，1 为50HZ，2 为60HZ。range：0~3，default：0
ratio ： 检测变化比例，一般取10~30，该值越低检测越灵敏，误检率也会提高。range：0~100，default：16
```

- DESCRIPTION

```
isp_tuning_flicker_cfg 为描述工频检测模块配置的结构体。
```

6.4.5 isp_tuning_defog_cfg

- PROTOTYPE

```
struct isp_tuning_defog_cfg {
    HW_S32 strength;
};
```

- MEMBERS

```
strength ： 去雾强度。range：0~1023，default：20
```

- DESCRIPTION

```
isp_tuning_defog_cfg 为描述去雾模块配置的结构体。
```

6.4.6 isp_tuning_visual_angle_cfg

- PROTOTYPE

```
struct isp_tuning_visual_angle_cfg {
    HW_S32 horizontal;
    HW_S32 vertical;
    HW_S32 focus_length;
};
```

- MEMBERS

```
horizontal ：水平可视角度。range：0~360，default：60
vertical ：垂直可视角度。range：0~360，default：40
focus_length ：焦距，例如焦距为28mm，则该值为28*100。range：0~65535，default：300
```

- DESCRIPTION

```
isp_tuning_visual_angle_cfg 为描述可视角和焦距等镜头规格的结构体。
```

6.4.7 isp_tuning_gtm_cfg

- PROTOTYPE

```
struct isp_tuning_gtm_cfg {
    HW_S32 type;
    HW_S32 gamma_type;
    HW_S32 auto_alpha_en;
};
```

- MEMBERS

```
type ：对比度调整类型，0：按照预设亮度对比度来调整DRC 模块，1：根据直方图自适应调整DRC 模块。
gamma_type ：gamma 类型，0：使用标准gamma，1：使用动态gamma。
auto_alpha_en ：是否根据图像内容自动调整亮暗阈值，0：否，1：是。
```

- DESCRIPTION

```
isp_tuning_gtm_cfg 为描述全局色调映射（对比度调整）模块配置的结构体。
```

6.4.8 isp_tuning_dpc_otf_cfg

- PROTOTYPE

```
struct isp_tuning_dpc_otf_cfg {
    HW_S32 thres_slop;
    HW_S32 min_thres;
    HW_S32 max_thres;
    HW_S32 mode;
    HW_S32 cfa_dir_thres;
};
```

- MEMBERS

```
thres_slop ：阈值斜率。range：0~16，default：4
min_thres ：最小阈值。range：0~64，default：16
max_thres ：最大阈值。range：0~4095，default：2048
cfa_dir_thres ：CFA 插值所需方向阈值。range：0~2048，default：600
```

- DESCRIPTION

```
isp_tuning_dpc_otf_cfg 为描述坏点检测模块配置的结构体。
```

6.4.9 isp_tuning_blc_gain_cfg

- PROTOTYPE

```
struct isp_tuning_blc_gain_cfg {
    HW_S32 value[ISP_RAW_CH_MAX];
};
```

- MEMBERS

```
value ：所要设置的各个通道值，各通道意义如下：
enum isp_raw_ch {
    ISP_RAW_CH_R = 0,
    ISP_RAW_CH_GR,
    ISP_RAW_CH_GB,
    ISP_RAW_CH_G,
    ISP_RAW_CH_MAX,
};
```

- DESCRIPTION

```
isp_tuning_blc_gain_cfg 为描述预校正增益以及黑电平模块配置的结构体。
```

6.4.10 isp_tuning_lens_shading_cfg

- PROTOTYPE

```
struct isp_tuning_lens_shading_cfg {
    HW_S32 ff_mod;
    HW_S32 center_x;
    HW_S32 center_y;
    HW_S32 rolloff_ratio;
```

```
    HW_U16 value[12][768];
    HW_U16 color_temp_triggers[6];
};
```

- MEMBERS

```
ff_mod ：对焦模式，0：自动对焦，1：固定对焦。
center_x ：水平中心点坐标，归一化到0~4095。range：0~4095，default：2048
center_y ：垂直中心点坐标，归一化到0~4095。range：0~4095，default：2048
rolloff_ratio ：自动颜色阴影矫正比例一般为85。range：0~100，default：85
value ：LSC 校正表。range：0~4096
color_temp_triggers ：LSC 校正色温表。range：1000~13000
```

- DESCRIPTION

```
isp_tuning_lens_shading_cfg 为描述镜头阴影校正模块配置的结构体。
```

6.4.11 isp_tuning_gamma_table_cfg

- PROTOTYPE

```
struct isp_tuning_gamma_table_cfg {
    HW_S32 number;
    HW_U16 value[5][ISP_GAMMA_TBL_LENGTH];
    HW_U16 lv_triggers[5];
};
```

- MEMBERS

```
number ：gamma 表数量，不超过5。
value ：gamma 表。
lv_triggers ：亮度LV 触发节点。range：0~2000
```

- DESCRIPTION

```
isp_tuning_gamma_table_cfg 为描述gamma 模块配置的结构体。
```

6.4.12 isp_tuning_linearity_cfg

- PROTOTYPE

```
struct isp_tuning_linearity_cfg {
    HW_U16 value[768];
};
```

- MEMBERS

```
value ： 线性度矫正表
```

- DESCRIPTION

```
isp_tuning_linearity_cfg 为描述线性度矫正模块配置的结构体。
```

6.4.13 isp_tuning_distortion_cfg

- PROTOTYPE

```
struct isp_tuning_distortion_cfg {
    HW_U16 value[512];
};
```

- MEMBERS

```
value ： 畸变矫正表（V5 不支持）。
```

- DESCRIPTION

```
isp_tuning_distortion_cfg 为描述畸变矫正模块配置的结构体。
```

6.4.14 isp_tuning_bdnf_cfg

- PROTOTYPE

```
struct isp_tuning_bdnf_cfg {
    HW_U16 thres[ISP_REG_TBL_LENGTH];
};
```

- MEMBERS

```
thres : 2D 降噪配置表，用来配置不同像素亮度下的降噪阈值。range: 0~4095 #define ISP_REG_TBL_LENGTH 33
```

- DESCRIPTION

```
isp_tuning_bdnf_cfg 为描述2D 降噪模块配置的结构体。
```

6.4.15 isp_tuning_tdnf_cfg

- PROTOTYPE

```
struct isp_tuning_tdnf_cfg {
    HW_U16 thres[ISP_REG_TBL_LENGTH];
    HW_U16 ref_noise[ISP_REG_TBL_LENGTH];
    HW_U8 k_val[ISP_REG_TBL_LENGTH-1];
    HW_U8 diff[256];
};
```

- MEMBERS

```
thres : 3D 降噪配置表，用来配置不同像素亮度下的降噪阈值。range: 0~4095 #define ISP_REG_TBL_LENGTH 33
ref_noise : 3D 降噪参考噪声配置表。range: 0~4095
k_val : 3D 降噪收敛配置表。range: 0~32
diff : 3D 降噪DIFF 配置表。
```

- DESCRIPTION

```
isp_tuning_tdnf_cfg 为描述3D 降噪模块配置的结构体。
```

6.4.16 isp_tuning_contrast_cfg

- PROTOTYPE

```
struct isp_tuning_contrast_cfg {
    HW_U16 val[ISP_REG_TBL_LENGTH];
    HW_U16 lum[ISP_REG_TBL_LENGTH];
    HW_U16 pe[128];
};
```

- MEMBERS

```
val : 参考datasheet 或者调试指南。range: 0~4095
lum : 参考datasheet 或者调试指南。range: 0~4095
pe : 参考datasheet 或者调试指南。range: 0~4095
```

- DESCRIPTION

```
isp_tuning_contrast_cfg 为描述对比度模块配置的结构体。
```

6.4.17 isp_tuning_sharp_cfg

- PROTOTYPE

```
struct isp_tuning_sharp_cfg {
    HW_U16 value[ISP_REG_TBL_LENGTH];
    HW_U16 lum[ISP_REG_TBL_LENGTH];
};
```

- MEMBERS

```
value : 参考datasheet 或者调试指南。range: 0~4095
lum : 参考datasheet 或者调试指南。range: 0~4095
```

- DESCRIPTION

```
isp_tuning_sharp_cfg 为描述锐化模块配置的结构体。
```

6.4.18 isp_tuning_cem_cfg

- PROTOTYPE

```
struct isp_tuning_cem_cfg {
    HW_U8 value[ISP_CEM_MEM_SIZE];
};
```

- MEMBERS

```
value : 参考datasheet 或者调试指南。range: 0~4095
```

- DESCRIPTION

isp_tuning_sharp_cfg 为描述颜色增强模块配置的结构体。

6.4.19 isp_tuning_pltm_cfg

- PROTOTYPE

```
struct isp_tuning_pltm_cfg {
    HW_U8 value[ISP_PLTM_MEM_SIZE];
};
```

- MEMBERS

value ：参考datasheet 或者调试指南。range：0~4095

- DESCRIPTION

isp_tuning_pltm_cfg 为描述局部色调映射模块配置的结构体。

6.5 Dynamic 命令组

6.5.1 hw_isp_cfg_dynamic_ids

- PROTOTYPE

```
typedef enum {
    /* isp_dynamic_cfg */
    HW_ISP_CFG_DYNAMIC_LUM_POINT = 0x00000001,
    HW_ISP_CFG_DYNAMIC_GAIN_POINT = 0x00000002,
    HW_ISP_CFG_DYNAMIC_SHARP = 0x00000004,
    HW_ISP_CFG_DYNAMIC_CONTRAST = 0x00000008,
    HW_ISP_CFG_DYNAMIC_DENOISE = 0x00000010,
    HW_ISP_CFG_DYNAMIC_BRIGHTNESS = 0x00000020,
    HW_ISP_CFG_DYNAMIC_SATURATION = 0x00000040,
    HW_ISP_CFG_DYNAMIC_TDF = 0x00000080,
    HW_ISP_CFG_DYNAMIC_AE = 0x00000100,
    HW_ISP_CFG_DYNAMIC_GTM = 0x00000200,
    HW_ISP_CFG_DYNAMIC_PLTM = 0x00000400,
} hw_isp_cfg_dynamic_ids;
```

- MEMBERS

```
HW_ISP_CFG_DYNAMIC_LUM_POINT ：设置亮度映射点（一般不用设置）。
HW_ISP_CFG_DYNAMIC_GAIN_POINT ：设置增益映射点（一般不用设置）。
HW_ISP_CFG_DYNAMIC_SHARP ：设置不同ISO 下的锐化。
HW_ISP_CFG_DYNAMIC_CONTRAST ：设置不同ISO 下的对比度。
HW_ISP_CFG_DYNAMIC_DENOISE ：设置不同ISO 下的降噪。
HW_ISP_CFG_DYNAMIC_BRIGHTNESS ：设置不同ISO 下的亮度。
HW_ISP_CFG_DYNAMIC_SATURATION ：设置不同ISO 下的饱和度。
HW_ISP_CFG_DYNAMIC_TDF ：设置不同ISO 下的3D 降噪强度。
HW_ISP_CFG_DYNAMIC_AE ：设置不同ISO 下的AE 参数。
HW_ISP_CFG_DYNAMIC_GTM ：设置不同ISO 下的全局对比度参数。
HW_ISP_CFG_DYNAMIC_PLTM ：设置不同ISO 下的局部对比度参数。
```

- DESCRIPTION

```
hw_isp_cfg_dynamic_ids 用于枚举ISP 动态参数设置命令
```

6.5.2 isp_dynamic_param_cfg

- PROTOTYPE

```
/* isp_dynamic_param cfg */
struct isp_dynamic_param_cfg {
    struct isp_dynamic_single_cfg lum_mapping_point; /* id: 0x0400000001 */
    struct isp_dynamic_single_cfg gain_mapping_point; /* id: 0x0400000002 */
    struct isp_dynamic_sharp_cfg sharp; /* id: 0x0400000004 */
    struct isp_dynamic_contrast_cfg contrast; /* id: 0x0400000008 */
    struct isp_dynamic_denoise_cfg denoise; /* id: 0x0400000010 */
    struct isp_dynamic_brightness_cfg brightness; /* id: 0x0400000020 */
    struct isp_dynamic_saturation_cfg saturation; /* id: 0x0400000040 */
    struct isp_dynamic_tdf_cfg tdf; /* id: 0x0400000080 */
    struct isp_dynamic_ae_cfg ae; /* id: 0x0400000100 */
    struct isp_dynamic_gtm_cfg gtm; /* id: 0x0400000200 */
    struct isp_dynamic_pltm_cfg pltm; /* id: 0x0400000400 */
};
```

- MEMBERS

```
lum_mapping_point ：设置亮度映射点（一般不用设置）。
gain_mapping_point ：设置增益映射点（一般不用设置）。
sharp ：设置不同ISO 下的锐化。
contrast ：设置不同ISO 下的对比度。
denoise ：设置不同ISO 下的降噪。
brightness ：设置不同ISO 下的亮度。range: 0~255, default: 0
saturation ：设置不同ISO 下的饱和度。range: 0~100, default: 0
tdf ：设置不同ISO 下的3D 降噪强度。
ae ：设置不同ISO 下的AE 参数。
gtm ：设置不同ISO 下的全局对比度参数。
pltm ：设置不同ISO 下的局部对比度参数。
```

- DESCRIPTION

```
isp_dynamic_param_cfg 用于描述ISP 动态参数设置的结构体。
```

6.5.3 isp_dynamic_single_cfg

- PROTOTYPE

```
struct isp_dynamic_single_cfg {
    HW_S32 value[ISP_DYNAMIC_GROUP_COUNT];
};
```

- MEMBERS

```
value ： 映射点配置。
```

- DESCRIPTION

```
isp_dynamic_single_cfg 用于描述ISP 映射点配置的结构体。
```

6.5.4 isp_dynamic_sharp_cfg

- PROTOTYPE

```
struct isp_dynamic_sharp_cfg {
    HW_S32 trigger;
    struct isp_dynamic_sharp_item tuning_cfg[ISP_DYNAMIC_GROUP_COUNT];
};
```

- MEMBERS

```
trigger ： 锐化触发选择，0: gain 触发，1: lum 触发。
tuning_cfg ： 锐化参数配置，其中:
struct isp_dynamic_sharp_item {
    HW_S32 value[ISP_SHARP_MAX];
};
enum isp_sharp_cfg {
    ISP_SHARP_MIN_VAL = 0, //参照Datasheet。range: 0~4095
    ISP_SHARP_MAX_VAL = 1, //参照Datasheet。range: 0~4095
    ISP_SHARP_BLACK_LEVEL = 2, //参照Datasheet。range: 0~4095
    ISP_SHARP_WHITE_LEVEL = 3, //参照Datasheet。range: 0~4095
    ISP_SHARP_BLACK_CLIP = 4, //参照Datasheet。range: 0~4095
    ISP_SHARP_WHITE_CLIP = 5, //参照Datasheet。range: 0~4095
```

```
    ISP_SHARP_BLACK_GAIN = 6, //暗部增益，针对tuning 中的表range: 0~4095
    ISP_SHARP_BLACK_OFFSET = 7, //暗部偏移，针对tuning 中的表range: 0~4095
    ISP_SHARP_WHITE_GAIN = 8, //亮部增益，针对tuning 中的表range: 0~4095
    ISP_SHARP_WHITE_OFFSET = 9, //亮部偏移，针对tuning 中的表range: 0~4095
    ISP_SHARP_MAX,
};
```

- DESCRIPTION

```
isp_dynamic_sharp_cfg 用于描述ISP 动态锐化参数配置的结构体。
```

6.5.5 isp_dynamic_contrast_cfg

- PROTOTYPE

```
struct isp_dynamic_contrast_cfg {
    HW_S32 trigger;
    HW_S32 global_trigger;
    struct isp_dynamic_contrast_item tuning_cfg[ISP_DYNAMIC_GROUP_COUNT];
};
```

- MEMBERS

```
trigger : 对比度触发选择，0: gain 触发，1: lum 触发。
global_trigger : 全局对比度触发选择，0: gain 触发，1: lum 触发。
tuning_cfg : 对比度参数配置，其中:
struct isp_dynamic_contrast_item {
    HW_S32 value[ISP_CONTRAST_MAX];
    HW_S32 global;
};
enum isp_contrast_cfg {
    ISP_CONTRAST_MIN_VAL = 0, //参照Datasheet。range: 0~4095
    ISP_CONTRAST_MAX_VAL = 1, //参照Datasheet。range: 0~4095
    ISP_CONTRAST_BLACK_LEVEL = 2, //参照Datasheet。range: 0~4095
    ISP_CONTRAST_WHITE_LEVEL = 3, //参照Datasheet。range: 0~4095
    ISP_CONTRAST_BLACK_CLIP = 4, //参照Datasheet。range: 0~4095
    ISP_CONTRAST_WHITE_CLIP = 5, //参照Datasheet。range: 0~4095
    ISP_CONTRAST_PLAT_TH = 6, //参照Datasheet。range: 0~4095
    ISP_CONTRAST_BLACK_GAIN = 7, //暗部增益，针对tuning 中的表range: 0~4095
    ISP_CONTRAST_BLACK_OFFSET = 8, //暗部偏移，针对tuning 中的表range: 0~4095
    ISP_CONTRAST_WHITE_GAIN = 9, //亮部增益，针对tuning 中的表range: 0~4095
    ISP_CONTRAST_WHITE_OFFSET = 10, //亮部偏移，针对tuning 中的表range: 0~4095
    ISP_CONTRAST_MAX,
};
```

- DESCRIPTION

isp_dynamic_contrast_cfg 用于描述ISP 动态对比度参数配置的结构体。

6.5.6 isp_dynamic_denoise_cfg

- PROTOTYPE

```
struct isp_dynamic_denoise_cfg {
    HW_S32 trigger;
    HW_S32 color_trigger;
    struct isp_dynamic_denoise_item tuning_cfg[ISP_DYNAMIC_GROUP_COUNT];
};
```

- MEMBERS

```
trigger ：降噪触发选择，0：gain 触发，1：lum 触发。
global_trigger ：色度降噪触发选择，0：gain 触发，1：lum 触发。
tuning_cfg ：降噪参数配置，其中：
struct isp_dynamic_denoise_item {
    HW_S32 value[ISP_DENOISE_MAX];
    HW_S32 color_denoise;
};
enum isp_denoise_cfg {
    ISP_DENOISE_BLACK_GAIN = 0, //暗部增益，针对tuning 中的表range：0~4095
    ISP_DENOISE_BLACK_OFFSET = 1, //暗部偏移，针对tuning 中的表range：0~4095
    ISP_DENOISE_WHITE_GAIN = 2, //亮部增益，针对tuning 中的表range：0~4095
    ISP_DENOISE_WHITE_OFFSET = 3, //亮部偏移，针对tuning 中的表range：0~4095
    ISP_DENOISE_MAX,
};
```

- DESCRIPTION

isp_dynamic_denoise_cfg 用于描述ISP 动态降噪参数配置的结构体。

6.5.7 isp_dynamic_brightness_cfg

- PROTOTYPE

```
struct isp_dynamic_brightness_cfg {
    HW_S32 trigger;
    HW_S32 value[ISP_DYNAMIC_GROUP_COUNT];
};
```

- MEMBERS

```
trigger : 亮度设置触发选择，0: gain 触发，1: lum 触发。
value : 亮度值。
```

- DESCRIPTION

```
isp_dynamic_brightness_cfg 用于描述ISP 动态亮度参数配置的结构体。
```

6.5.8 isp_dynamic_saturation_cfg

- PROTOTYPE

```
struct isp_dynamic_saturation_cfg {
    HW_S32 trigger;
    struct isp_dynamic_saturation_item tuning_cfg[ISP_DYNAMIC_GROUP_COUNT];
};
```

- MEMBERS

```
trigger : 饱和度设置触发选择，0: gain 触发，1: lum 触发。
tuning_cfg : 饱和度参数配置，其中: 。
struct isp_dynamic_saturation_item {
    HW_S32 cb; //无效
    HW_S32 cr; //无效参数
    HW_S32 value[ISP_SATURATION_MAX];
};
enum isp_saturation_cfg {
    ISP_SATURATION_SATU_R = 0, //红色饱和度。range: 0~16，default: 4
    ISP_SATURATION_SATU_G = 1, //绿色饱和度。range: 0~16，default: 8
    ISP_SATURATION_SATU_B = 2, //蓝色饱和度。range: 0~16，default: 4
    ISP_SATURATION_SATU_MODE = 3, //饱和度调整模式。range: 0~1
    ISP_SATURATION_SATU_TBL_SG1 = 4, //参照Datasheet。range: 0~4095
    ISP_SATURATION_SATU_TBL_SG2 = 5, //参照Datasheet。range: 0~4095
    ISP_SATURATION_SATU_TBL_TH = 6, //参照Datasheet。range: 0~4095
    ISP_SATURATION_MAX,
};
```

- DESCRIPTION

```
isp_dynamic_saturation_cfg 用于描述ISP 动态饱和度参数配置的结构体。
```

6.5.9 isp_dynamic_tdf_cfg

- PROTOTYPE

```
struct isp_dynamic_tdf_cfg {
    HW_S32 trigger;
    struct isp_dynamic_tdf_item tuning_cfg[ISP_DYNAMIC_GROUP_COUNT];
};
```

- MEMBERS

```
trigger : 3D 降噪触发选择，0: gain 触发，1: lum 触发。
tuning_cfg : 3D 降噪配置，其中: 。
struct isp_dynamic_tdf_item {
    HW_S32 value[ISP_TDF_MAX];
};
enum isp_tdf_cfg {
    ISP_TDF_NOISE_CLIP_RATIO = 0, //参照Datasheet。range: 0~255
    ISP_TDF_DIFF_CLIP_RATIO = 1, //参照Datasheet。range: 0~255
    ISP_TDF_K_3D_S = 2, //参照Datasheet。range: 0~64
    ISP_TDF_DIFF_CAL_MODE = 3, //参照Datasheet。range: 0~1
    ISP_TDF_BLACK_GAIN = 4, //暗部增益，针对tuning 中的3D 降噪强度表range: 0~4095
    ISP_TDF_BLACK_OFFSET = 5, //暗部偏移，针对tuning 中的3D 降噪强度表range: 0~4095
    ISP_TDF_WHITE_GAIN = 6, //亮部增益，针对tuning 中的3D 降噪强度表range: 0~4095
    ISP_TDF_WHITE_OFFSET = 7, //亮部偏移，针对tuning 中的3D 降噪强度表range: 0~4095
    ISP_TDF_REF_BLACK_GAIN = 8, //暗部增益，针对tuning 中的参考噪声表range: 0~4095
    ISP_TDF_REF_BLACK_OFFSET = 9, //暗部偏移，针对tuning 中的参考噪声表range: 0~4095
    ISP_TDF_REF_WHITE_GAIN = 10, //亮部增益，针对tuning 中的参考噪声表range: 0~4095
    ISP_TDF_REF_WHITE_OFFSET = 11, //亮部偏移，针对tuning 中的参考噪声表range: 0~4095
    ISP_TDF_MAX,
};
```

- DESCRIPTION

```
isp_dynamic_tdf_cfg 用于描述ISP 动态3D 降噪参数配置的结构体。
```

6.5.10 isp_dynamic_ae_cfg

- PROTOTYPE

```
struct isp_dynamic_ae_cfg {
    HW_S32 trigger;
    struct isp_dynamic_ae_item tuning_cfg[ISP_DYNAMIC_GROUP_COUNT];
};
```

- MEMBERS

```
trigger : 自动曝光设置触发选择，0: gain 触发，1: lum 触发。
tuning_cfg : 自动曝光配置，其中: 。
struct isp_dynamic_ae_item {
    HW_S32 value[ISP_EXP_CFG_MAX];
```

```
};
enum exposure_cfg_type {
    ANTI_EXP_WIN_OVER = 0, //窗口抗过曝参数配置。range: 0~4096, default: 256
    ANTI_EXP_WIN_UNDER = 1, //窗口抗欠曝参数配置。range: 0~4096, default: 256
    ANTI_EXP_HIST_OVER = 2, //直方图抗过曝参数配置。range: 0~4096, default: 256
    ANTI_EXP_HIST_UNDER = 3, //直方图抗欠曝参数配置。range: 0~4096, default: 256
    AE_PREVIEW_SPEED = 4, //预览模式自动曝光速度。range: 0~31, default: 12
    AE_CAPTURE_SPEED = 5, //拍照模式自动曝光速度。range: 0~31, default: 12
    AE_VIDEO_SPEED = 6, //录像模式自动曝光速度。range: 0~31, default: 12
    AE_TOUCH_SPEED = 7, //触摸曝光模式自动曝光速度。range: 0~31, default: 12
    AE_TOLERANCE = 8, //自动曝光容忍度。range: 0~31, default: 6
    AE_TARGET = 9, //曝光目标值（Gamma 前）。range: 0~255, default: 40
    AE_HIST_DARK_WEIGHT_MIN = 10, //面光暗部最小权重。range: 0~32, default: 2
    AE_HIST_DARK_WEIGHT_MAX = 11, //背光暗部最大权重。range: 0~32, default: 32
    AE_HIST_BRIGHT_WEIGHT_MIN = 12, //背光亮部最小权重。range: 0~32, default: 2
    AE_HIST_BRIGHT_WEIGHT_MAX = 13, //面光亮部最大权重。range: 0~32, default: 32
    ISP_EXP_CFG_MAX,
};
```

- DESCRIPTION

```
isp_dynamic_ae_cfg 用于描述ISP 动态自动曝光参数配置的结构体。
```

6.5.11 isp_dynamic_gtm_cfg

- PROTOTYPE

```
struct isp_dynamic_gtm_cfg {
    HW_S32 trigger;
    struct isp_dynamic_gtm_item tuning_cfg[ISP_DYNAMIC_GROUP_COUNT];
};
```

- MEMBERS

```
trigger : 自动对比度参数触发选择, 0: gain 触发, 1: lum 触发。
tuning_cfg : 自动对比度配置, 其中: 。
struct isp_dynamic_gtm_item {
    HW_S32 value[ISP_GTM_HEQ_MAX];
};
enum isp_gtm_comm_cfg {
    ISP_GTM_GAIN = 0, //自动对比度调整强度。range: 0~2048, default: 1024
    ISP_GTM_EQ_RATIO = 1, //自动对比度调整比例。range: 0~100, default: 30
    ISP_GTM_EQ_SMOOTH = 2, //自动对比度平滑度。range: 0~32, default: 10
    ISP_GTM_BLACK = 3, //自动对比度暗部阈值。range: 0~255, default: 20
    ISP_GTM_WHITE = 4, //自动对比度亮部阈值。range: 0~255, default: 200
    ISP_GTM_BLACK_ALPHA = 5, //自动对比度暗部压缩强度。range: 0~32, default: 4
    ISP_GTM_WHITE_ALPHA = 6, //自动对比度亮部拉伸强度。range: 0~32, default: 4
    ISP_GTM_GAMMA_IND = 7, //自动对比度gamma 选择。range: -4~3, default: 0
    ISP_GTM_GAMMA_PLUS = 8, //无效参数, 一般为0
    ISP_GTM_HEQ_MAX,
```

```
};
```

- DESCRIPTION

```
isp_dynamic_gtm_cfg 用于描述ISP 动态全局对比参数配置的结构体。
```

6.5.12 isp_dynamic_pltm_cfg

- PROTOTYPE

```
struct isp_dynamic_pltm_cfg {
    HW_S32 trigger;
    struct isp_dynamic_pltm_item tuning_cfg[ISP_DYNAMIC_GROUP_COUNT];
};
```

- MEMBERS

```
trigger ： 局部色调映射触发选择，0：gain 触发，1：lum 触发。
tuning_cfg ： 局部色调映射配置，其中：。
struct isp_dynamic_pltm_item {
    HW_S32 value[ISP_PLTM_MAX];
};
enum isp_pltm_comm_cfg {
    ISP_PLTM_LSS_SWITCH = 0, //参照Datasheet。range：0~1
    ISP_PLTM_CAL_EN = 1, //参照Datasheet。range：0~1
    ISP_PLTM_FRM_SM_EN = 2, //参照Datasheet。range：0~1
    ISP_PLTM_LAST_ORDER_RATIO = 3, //参照Datasheet。range：0~15
    ISP_PLTM_TR_ORDER = 4, //参照Datasheet。range：0~15
    ISP_PLTM_ORIPIC_RATIO = 5, //参照Datasheet。range：0~255
    ISP_PLTM_INTENS_ASYM = 6, //参照Datasheet。range：0~255
    ISP_PLTM_SPATIAL_ASM = 7, //参照Datasheet。range：0~255
    ISP_PLTM_WHITE_LEVEL = 8, //参照Datasheet。range：0~255
    ISP_PLTM_LP_HALO_RES = 9, //参照Datasheet。range：0~15
    ISP_PLTM_LUM_RATIO = 10, //参照Datasheet。range：0~15
    ISP_PLTM_BLOCK_HEIGHT = 11, //参照Datasheet。range：0~255
    ISP_PLTM_BLOCK_WIDTH = 12, //参照Datasheet。range：0~255
    ISP_PLTM_BLOCK_V_NUM = 13, //参照Datasheet。range：0~31
    ISP_PLTM_BLOCK_H_NUM = 14, //参照Datasheet。range：0~31
    ISP_PLTM_STATISTIC_DIV= 15, //参照Datasheet。range：0~(2^32-1)
    ISP_PLTM_MAX,
};
```

- DESCRIPTION

```
isp_dynamic_pltm_cfg 用于描述ISP 动态局部色调映射参数配置的结构体。
```

7 错误码

错误码	宏定义	描述
	AW_ERR_VI_INVALID_PARA	无效参数
	AW_ERR_VI_INVALID_DEVID	无效 VI 设备号
	AW_ERR_VI_INVALID_CHNID	无效 VI 通道号
	AW_ERR_VI_INVALID_NULL_PTR	空指针
	AW_ERR_VI_FAILED_NOTCONFIG	模块未配置
	AW_ERR_VI_SYS_NOTREADY	系统未准备好
	AW_ERR_VI_BUF_EMPTY	缓存为空
	AW_ERR_VI_BUF_FULL	缓存为满
	AW_ERR_VI_NOMEM	无可用内存
	AW_ERR_VI_NOT_SUPPORT	设备不支持
	AW_ERR_VI_BUSY	设备忙
	AW_ERR_VI_FAILED_NOTENABLE	未使能
	AW_ERR_VI_FAILED_NOTDISABLE	未去使能
	AW_ERR_VI_CFG_TIMEOUT	配置超时
	AW_ERR_VI_NORM_UNMATCH	视频没有禁止使能
	AW_ERR_VI_INVALID_PHYCHNID	无效物理通道号
	AW_ERR_VI_FAILED_NOTBIND	设备未绑定
	AW_ERR_VI_FAILED_BINDED	设备已绑定

著作权声明

商标声明

免责声明