密级状态：绝密(　　)　秘密(　　)　内部资料(　　)　公开(√)

RKISPV11_Camera_Hal_Introduction

文件状态： ［　］草稿 ［　］正式发布 ［√］正在修改	文件标识：	
	当前版本：	1.0
	作　　者：	钟以崇
	完成日期：	2016-9-22

历 史 版 本

版本	日 期	描 述	作 者	审核
V1.0	2016-9-22	建立文档，主要介绍 RK1108 CVR CameraHal 框架及使用	钟以崇	

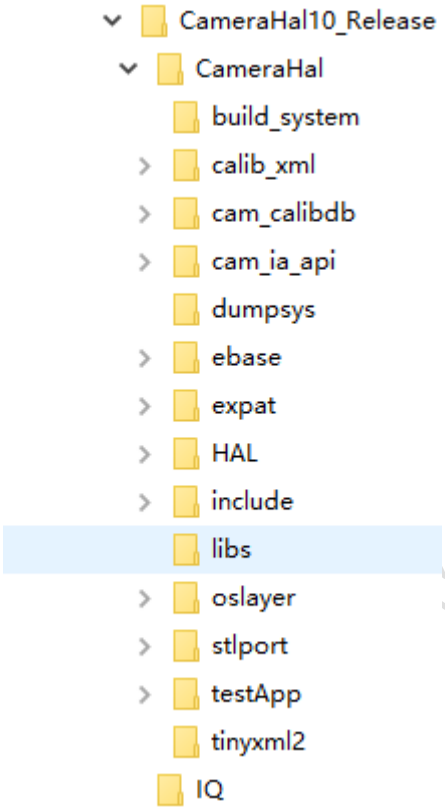
目录

1. 文档适用平台

文档适用于 RockChips 公司 RK1108 CVR 平台。

2. HAL 目录文件

Camera HAL 层为硬件抽象层，用于连接应用层和驱动层，使得应用程序不需关心具体的硬件驱动实现，各种硬件实现（ISP，CIF 以及 USB camera）可以使用同一套 API 接口，简化应用开发



build_system: 简易的 HAL 编译系统，兼容 Android。
calib_xml: 解析 tunning xml 文件接口。
cam_calibdb: tunning xml 解析数据管理。
cam_ia_api: 3A,ISP 子模块控制。
dumpsys: 用于 dump RAW 数据，主要用于 tunning。
ebase：基础数据类型，考虑到跨平台时使用。
expat：xml 解析器，移植于 Android。
HAL: Camera 硬件抽象，给应用提供统一接口。
include：HAL 中共用的头文件，其中许多头文件从其他文件夹拷贝而来。
libs：打包好的 3A 算法库。
oslayer：os 抽象，考虑到跨平台时使用。
stlport: c++ stlPort 移植，HAL 主要要 C++编写，某些平台编译器可能需要此支持。

testApp：HAL 的示例程序。
tinyxml2：xml 解析器，移植于 Android。
IQ: tunning xml 文件。

3. HAL 基本框架

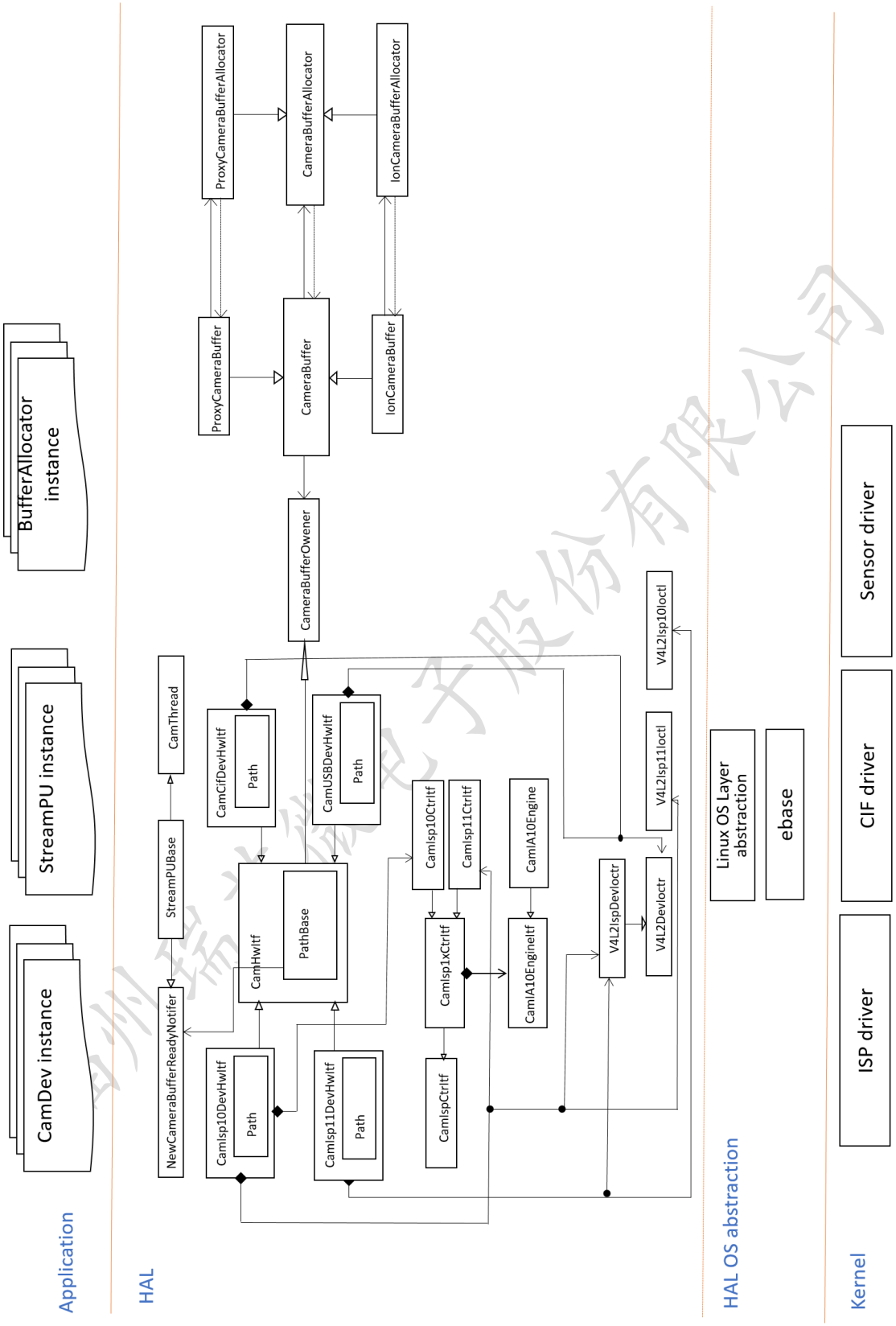


图 1

Application layer:

CamDev instance：为 Camera 设备对象实例，目前可为以下四种对象：CamIsp10DevHwItf(ISP10,RK3288)、CamIsp11DevHwItf(ISP11,RK1108)、CamCifDevHwItf(CIF 控制器)、CamUSBDevHwItf(USB Camera)。

StreamPU instance：继承 StreamPUBase，将该对象加入到 CamDev instance，可接收来自 CamDev instance 的帧 Buffer，该对象并非必需，只是为了方便后续的数据流处理而提供。用户也可直接继 NewCameraBufferReadyNotifier 类，然后将该类对象指针加入到 CamDev instance 来接收帧 Buffer。

BufferAllocator Instance：帧 buffer 分配器，CamDev 对象会使用该分配器来分配帧 buffer。

HAL:

HAL OS abstraction：用于抽象基本的系统接口，便于跨平台移植，目前只有 Linux 接口。

HAL：抽象不同类型的 Camera 设备，为 Application 层提供统一接口，详细内容请参考第四章节。

Kernel:

主要为 ISP、CIF、Camera Sensor 驱动，遵循 V4L2 标准。ISP V4L2 接口详细内容请参考第五章节。

4. HAL 中主要的接口类

主要的类关系如前述图 1 所示。主要分为如下三个部分：CamDev 相关类 ISP 控制相关类，V4L2 设备对象接口类。

4.1. CamDev 抽象类

主要包括基类 CamHwItf 以及四种设备继承类，CamIsp10DevHwItf(ISP10,RK3288)、CamIsp11DevHwItf(ISP11,RK1108)、CamCifDevHwItf(CIF 控制器)、CamUSBDevHwItf(USB Camera)。
CamHwItf 类分为两部分，用于数据流管理的 Path 内部类，以及用于控制设备 Controls 相关接口。

Path 类主要接口如下：

(1) Prepare：设置输出帧格式，主要为分辨率、帧率、数据格式等。

```
virtual bool prepare(frm_info_t &frmFmt,unsigned int numBuffers,CameraBufferAllocator
&allocator,bool cached,unsigned int minNumBuffersQueued = 1) = 0;
virtual bool prepare(frm_info_t &frmFmt,list<shared_ptr<CameraBuffer> >
&bufPool,unsigned int numBuffers,unsigned int minNumBuffersQueued = 1) = 0;
```

(2) Notifier：添加帧消费者，path 产生帧 Buffer 后会送入到 notifier。

```
virtual void addBufferNotifier(NewCameraBufferReadyNotifier *bufferReadyNotifier) = 0;
virtual bool removeBufferNotifer(NewCameraBufferReadyNotifier *bufferReadyNotifier) = 0;
```

(3) Path 控制：开启关闭数据流。

```
virtual bool start(void) = 0;
virtual void stop(void) = 0;
```

(4) 释放 buffer。

```
virtual void releaseBuffers(void) = 0;
```

CamHwItf 类中主要的控制接口如下：

```
//设备初始化及反初始化，参数 inputId 为该控制器上的输入设备 id，可先设置成 0，
//initHw 执行后可通过 enumInputDev 得到所有连接的设备，再通过 setInputDev 来设置。
virtual bool initHw(int inputId) = 0;
virtual void deInitHw() = 0;
// AE 控制
virtual int getSupportedAeModes(vector<enum HAL_AE_OPERATION_MODE> &aeModes);
virtual int setAeMode(enum HAL_AE_OPERATION_MODE aeMode);
virtual int getAeMode(enum HAL_AE_OPERATION_MODE &aeMode);
virtual int setAeBias(int aeBias);
virtual int getAeBias(int &curAeBias);
virtual int getSupportedBiasRange(HAL_RANGES_t &range);
virtual int setAbsExp(int exposure);
virtual int getAbsExp(int &curExposure);
virtual int getSupportedExpMeterModes(vector<enum HAL_AE_METERING_MODE> modes);
virtual int setExposureMeterMode(enum HAL_AE_METERING_MODE aeMeterMode);
virtual int getExposureMeterMode(enum HAL_AE_METERING_MODE &aeMeterMode);
//fps,anti banding contols,related to ae control
virtual int setFps(HAL_FPS_INFO_t fps);
//for mode HAL_AE_OPERATION_MODE_AUTO,
virtual int enableAe(bool aeEnable);
virtual int setAntiBandMode(enum HAL_AE_FLK_MODE flkMode);
virtual int getSupportedAntiBandModes(vector<enum HAL_AE_FLK_MODE> &flkModes);
virtual int setExposure(unsigned int exposure, unsigned int gain, unsigned int gain_percent);
//AWB
virtual int getSupportedWbModes(vector<HAL_WB_MODE> &modes);
virtual int setWhiteBalance(HAL_WB_MODE wbMode);
virtual int getWhiteBalanceMode(HAL_WB_MODE &wbMode);
//for mode HAL_WB_AUTO,
virtual int enableAwb(bool awbEnable);
//AF
virtual int setFocusPos(int position);
virtual int getFocusPos(int &position);
virtual int getSupportedFocusModes(vector<enum HAL_AF_MODE> fcModes);
virtual int setFocusMode(enum HAL_AF_MODE fcMode);
virtual int getFocusMode(enum HAL_AF_MODE &fcMode);
virtual int getAfStatus(enum HAL_AF_STATUS &afStatus);
//for af algorithm ?
virtual int enableAf(bool afEnable);
//single AF
virtual int trigggerAf(bool trigger);
//3A lock
virtual int getSupported3ALocks(vector<enum HAL_3A_LOCKS> &locks);
virtual int set3ALocks(int locks);
virtual int get3ALocks(int &curLocks);
//fmts:format , size , fps
virtual bool enumSensorFmts(vector<frm_info_t> &frmInfos);
virtual bool enumSupportedFmts(vector<RK_FRMAE_FORMAT> &frmFmts);
virtual bool enumSupportedSizes(RK_FRMAE_FORMAT frmFmt,vector<frm_size_t>
```

```
&frmSizes);
    virtual bool enumSupportedFps(RK_FRMAE_FORMAT frmFmt,frm_size_t
frmSize,vector<HAL_FPS_INFO_t> &fpsVec);
    virtual int    tryFormat(frm_info_t inFmt, frm_info_t &outFmt);
    //flash control
    virtual int getSupportedFlashModes(vector<enum HAL_FLASH_MODE> &flModes);
    virtual int setFlashLightMode(enum HAL_FLASH_MODE flMode,int intensity,int timeout);
    virtual int getFlashLightMode(enum HAL_FLASH_MODE &flMode);
    //virtual int getFlashStrobeStatus(bool &flStatus);
    //virtual int enableFlashCharge(bool enable);
    //miscellaneous controls:
    //color effect,brightness,contrast,hue,saturation,ISO,scene mode,zoom
    //color effect
    virtual int getSupportedImgEffects(vector<enum HAL_IAMGE_EFFECT> &imgEffs);
    virtual int setImageEffect(enum HAL_IAMGE_EFFECT image_effect);
    virtual int getImageEffect(enum HAL_IAMGE_EFFECT &image_effect);
    //scene
    virtual int getSupportedSceneModes(vector<enum HAL_SCENE_MODE> &sceneModes);
    virtual int setSceneMode(enum HAL_SCENE_MODE sceneMode);
    virtual int getSceneMode(enum HAL_SCENE_MODE &sceneMode);
    //ISO
    virtual int getSupportedISOModes(vector<enum HAL_ISO_MODE> &isoModes);
    virtual int setISOMode(enum HAL_ISO_MODE isoMode, int sens);
    virtual int getISOMode(enum HAL_ISO_MODE &isoMode );
    //zoom
    virtual int getSupportedZoomRange(HAL_RANGES_t &zoomRange);
    virtual int setZoom(int zoomVal);
    virtual int getZoom(int &zoomVal);
    //brightness
    virtual int getSupportedBtRange(HAL_RANGES_t &brightRange);
    virtual int setBrightness(int brightVal);
    virtual int getBrithtness(int &brightVal);
    //contrast
    virtual int getSupportedCtRange(HAL_RANGES_t &contrastRange);
    virtual int setContrast(int contrast);
    virtual int getContrast(int &contrast);
    //saturation
    virtual int getSupportedStRange(HAL_RANGES_t &saturationRange);
    virtual int setSaturation(int sat);
    virtual int getSaturation(int &sat);
    //hue
    virtual int getSupportedHueRange(HAL_RANGES_t &hueRange);
    virtual int setHue(int hue);
    virtual int getHue(int &hue);
    virtual int setJpegQuality(int jpeg_quality){UNUSED_PARAM(jpeg_quality);return -1;}
    //flip
    virtual int setFlip(int flip);
    int queryBusInfo(unsigned char *busInfo);
```

4.2. ISP 控制类

包括以下类：基类 CamIspCtrItf，继承类 CamIsp1xCtrItf、CamIsp10CtrItf、CamIsp11CtrItf；基类 CamIA10EngineItf，继承类 CamIA10Engine。

CamIspCtrItf 类：

该类主要功能是：从驱动得到 3A 统计数据，调用 CamIA10EngineItf 中 3A 算法接口，得到新的 ISP 设置，此外还提供 ISP 子模块的外部控制接口。

CamIA10EngineItf 类：

该类主要功能是：3A 算法库的封装类，提供 ISP 子模块控制实现接口，3A 控制流程请参考第六章节。

4.3. V4L2 接口类

V4L2DevIoctr 类，封装标准 v4l2 ioctrl 调用，具体可参考 V4L2 相关标准。

5. ISP V4L2 接口

包括 V4l2Isp10Ioctl 及 V4l2Isp11Ioctl 类，封装了 ISP 子模块的 ioctrl 调用。主要接口如下：

```
/* dpcc */
bool getDpccCfg(struct cifisp_dpcc_config &dpccCfg);
bool setDpccCfg(struct cifisp_dpcc_config &dpccCfg,bool enable);
/* bls */
bool getBlsCfg(struct cifisp_bls_config &blsCfg);
bool setBlsCfg(struct cifisp_bls_config &blsCfg,bool enable);
/* degamma */
bool getSdgCfg(struct cifisp_sdg_config &sdgCfg);
bool setSdgCfg(struct cifisp_sdg_config &sdgCfg,bool enable);
/* lsc */
bool getLscCfg(struct cifisp_lsc_config &lscCfg);
bool setLscCfg(struct cifisp_lsc_config &lscCfg,bool enable);
/* awb measure */
bool getAwbMeasCfg(struct cifisp_awb_meas_config &awbCfg);
bool setAwbMeasCfg(struct cifisp_awb_meas_config &awbCfg,bool enable);
/* filter */
bool getFltCfg(struct cifisp_flt_config &fltCfg);
bool setFltCfg(struct cifisp_flt_config &fltCfg,bool enable);
/* demosaic */
bool getBdmCfg(struct cifisp_bdm_config &bdmCfg);
bool setBdmCfg(struct cifisp_bdm_config &bdmCfg, bool enable);
/* cross talk */
bool getCtkCfg(struct cifisp_ctk_config &ctkCfg);
bool setCtkCfg(struct cifisp_ctk_config &ctkCfg,bool enable);
/* gamma out */
bool getGocCfg(struct cifisp_goc_config &gocCfg);
bool setGocCfg(struct cifisp_goc_config &gocCfg,bool enable);
/* Histogram Measurement */
bool getHstCfg(struct cifisp_hst_config &hstCfg);
bool setHstCfg(struct cifisp_hst_config &hstCfg,bool enable);
/* Auto Exposure Measurements */
bool getAecCfg(struct cifisp_aec_config &aecCfg);
bool setAecCfg(struct cifisp_aec_config &aecCfg,bool enable);
/* awb gain */
bool getAwbGainCfg(struct cifisp_awb_gain_config &awbgCfg);
bool setAwbGainCfg(struct cifisp_awb_gain_config &awbgCfg,bool enable);
```

```
        /* color process */
        bool getCprocCfg(struct cifisp_cproc_config &cprocCfg);
        bool setCprocCfg(struct cifisp_cproc_config &cprocCfg,bool enable);
        /* afc */
        bool getAfcCfg(struct cifisp_afc_config &afcCfg);
        bool setAfcCfg(struct cifisp_afc_config &afcCfg,bool enable);
        /* ie */
        bool getIeCfg(struct cifisp_ie_config &ieCfg);
        bool setIeCfg(struct cifisp_ie_config &ieCfg,bool enable);
        /* dpf */
        bool getDpfCfg(struct cifisp_dpf_config &dpfCfg);
        bool setDpfCfg(struct cifisp_dpf_config &dpfCfg, bool enable);
        bool getDpfStrengthCfg(struct cifisp_dpf_strength_config &dpfStrCfg);
        bool setDpfStrengthCfg(struct cifisp_dpf_strength_config &dpfStrCfg, bool enable);
/* wdr */
        bool getWdrCfg(struct cifisp_wdr_config &wdrCfg);
                bool setWdrCfg(struct cifisp_wdr_config &wdrCfg, bool enable);
```

6. 3A control loop

以 ISP11 为例，3A loop 基本流程如下：

```
bool CamIsp11CtrItf::threadLoop()
{
        ……
// 1）从 isp 驱动得到 3A 统计数据
        if (!getMeasurement(v4l2_buf)) {
                ALOGE("%s: getMeasurement failed", __func__);
                return false;
        }
        // 2）将数据结构转换为 HAL 层结构
        convertIspStats(mIspStats[v4l2_buf.index], &ia_stat);
        ……
// 3) 运行 3A 算法，得到新的 ISP 设置
        runIA(&ia_dcfg, &ia_stat, &ia_results);
        // 4）外部直接控制 ISP 子模块，会覆盖掉 3A 算法得到的结果
        //run isp manual config?will override the 3A results
        if (!runISPManual(&ia_results,BOOL_TRUE))
                ALOGE("%s:run ISP manual failed!",__func__);
        // 5）将算法结果转化成 ISP 驱动层数据结构
        convertIAResults(&isp_cfg, &ia_results);
        //6) 将新的 ISP 配置到驱动
        applyIspConfig(&isp_cfg);
}
```

上述流程中 runIA 函数主要流程如下：

```
bool CamIsp1xCtrItf::runIA(struct CamIA10_DyCfg *ia_dcfg,
        struct CamIA10_Stats *ia_stats,
        struct CamIA10_Results *ia_results)
{
        // 1）设置新的 3A 参数，如模式，窗口等。
        if (ia_dcfg)
                mCamIAEngine->initDynamic(ia_dcfg);
        // 2）3A 算法
```

```
        if (ia_stats) {
            mCamIAEngine->setStatistics(ia_stats);
            if (ia_stats->meas_type & CAMIA10_AEC_MASK) {
                mCamIAEngine->runAEC();
                mCamIAEngine->runADPF();
            }
            if (ia_stats->meas_type & CAMIA10_AWB_MEAS_MASK) {
                mCamIAEngine->runAWB();
            }
            if (ia_stats->meas_type & CAMIA10_AFC_MASK) {
                mCamIAEngine->runAF();
            }
        }
        // 3） 获取 3A 结果
        if (ia_results) {
            ia_results->active = 0;
            if (mCamIAEngine->getAECResults(&ia_results->aec) == RET_SUCCESS) {
            }
            if (mCamIAEngine->getAWBResults(&ia_results->awb) == RET_SUCCESS) {
            }
            if (mCamIAEngine->getADPFResults(&ia_results->adpf) == RET_SUCCESS) {
            }
        }
}
```

上述流程中的 3A 算法基本流程如下：

（1）runAec

该算法实现位于 CameraHal/libs/libisp_aaa_aec.a 中，基本流程如下：

1. 根据 AE 5x5 窗口的统计亮度值来估计当前场景，如是否背光，然后根据当前场景设置目标亮度值。根据当前帧直方图得到当前帧的亮度，将该亮度和目标亮度进行比较，如果差距在范围之外，则根据步长来预调整曝光量，然后预估调整后的帧亮度，再将该亮度同目标亮度进行比较，如此循环迭代，直到预调整曝光后的估计帧亮度与目标亮度的差距在容许范围之内时，停止迭代。
2. 将新计算出来的曝光量分解成曝光时长和增益。该过程需要考虑帧率以及 fliker 影响，还有要考虑具体 sensor 的曝光时间增益表。

（2）runAwb

该算法实现位于 CameraHal/libs/libisp_aaa_awb.a 中，基本流程如下：

1. 根据 sensor 当前的曝光量估计是室内还是室外
2. 根据当前帧统计的 wb 增益逆推得出实际的增益值，由于 ISP 的 wb 统计是在
3. CCM 之后的，所以需要该过程。
4. 进行光源估计，得到主光源几其他光源比重。
5. 计算新的 wb 增益
6. 计算新的 CCM 矩阵
7. 计算新的 LSC 参数。

（3）runAF

还未实现，后续补充。

7. dump RAW 图像

sensor tunning 时需要抓取各种曝光配置的 RAW 图片，HAL 编译后会生成 build/bin/dumpsys 可执行程序。该程序需要 capcmd.xml 文件，capcmd.xml 文件

中定义了具体的抓取任务。capcmd.xml 文件配置说明及如何抓取 RAW 文件请参考 tunning 文档。

8. 集成编译 HAL

HAL 可适用于多种平台，如 Ubuntu，Android 等，由于使用的编译器各异，编译前需要根据系统情况，配置 productConfigs.mk。

```
 //是否为 Android 系统
IS_ANDROID_OS = false
//是否需要 HAL 中自定义的 C++ 智能指针支持
IS_NEED_SHARED_PTR = false
//是否需要编译 STLPORT 库
IS_NEED_COMPILE_STLPORT = false
//是否需要链接 STLPORT 库
IS_NEED_LINK_STLPORT = false
//是否需要编译 TINYXML 解析器
IS_NEED_COMPILE_TINYXML2 = true
//是否需要编译 EXPAT 解析器
IS_NEED_COMPILE_EXPAT = true
//是否支持 ION
IS_SUPPORT_ION = true
//是否需要编译 test 测试程序
IS_BUILD_TEST_APP = true
//是否为 ISP10(rk1108 为 isp11，该项设置为 false)
IS_CAM_IA10_API = false
IS_RK_ISP10 = false
//是否使用 RK 的 V4L2 头文件，RK 可能有修改一些标准的系统 V4L2 头文件，为了和 kernel
//中的头文件统一，需要定义此项。
IS_USE_RK_V4L2_HEAD = true
//需要链接的外部库路径，如 IONd 额库
BUILD_OUPUT_EXTERNAL_LIBS:=$(CURDIR)/../../out/system/lib/
//编译器路径
CROSS_COMPILE ?= $(CURDIR)/../../prebuilts/toolschain/usr/bin/arm-linux-
```

配置好 productConfigs.mk 后，执行 make 命令进行编译，编译成功后会生成 build/lib/libcam_hal.so 库文件。注意，如果修改的文件位于 HAL 以外的文件夹，那么需要 make clean 后再 make。

9. test 测试程序

test 测试程序演示了使用 HAL 的基本流程。ISP 预览基本流程如下所示：

```
  //获取 ISP camDev
 //shared_ptr<CamHwItf> testIspDev =    shared_ptr<CamHwItf>(new CamIsp11DevHwItf());
 shared_ptr<CamHwItf> testIspDev =    getCamHwItf();
  //初始化
  testIspDev->initHw(0)
  //获取数据流 path，rk1108 ISP 有 MP(main path)和 SP(self path)，两路 path 可同时输出，这里以 MP
为例
  shared_ptr<CamHwItf::PathBase> mpath = testIspDev->getPath(CamHwItf::MP);
  //prepare 设置 format 以及 buffer 分配器，这里 buffer 由 ION 分配器分配
 shared_ptr<IonCameraBufferAllocator> bufAlloc(new IonCameraBufferAllocator());
 mpath->prepare(frmFmt,4,*(bufAlloc.get()),false,0))
```

```
//添加 notifier，notifier 用于接收帧 buffer
NewCameraBufferReadyNotifier * mBufNotifer= new CambufMpNotifierImp();
mpath->addBufferNotifier(mBufNotifer);
//开启数据流
mpath->start()
//数据流开启后，testIspDev 会将 buffer 数据分发给注册的 notifier，notifier 的基本流程如下：
class CambufNotifierImp : public NewCameraBufferReadyNotifier
{
    virtual bool bufferReady(weak_ptr<CameraBuffer> buffer, int status)
    {
        //获取 buffer
        shared_ptr<CameraBuffer> spCamBuf = buffer.lock();
        if (spCamBuf.get())
        {
            //使用前增加 buffer 引用计数
            /* increase used count */
            spCamBuf->incUsedCnt();
            //可根据需要获取该帧的 metaData，metaData 中包含了该帧的曝光配置信息，当前的
            //ISP 模块配置信息等。
            //test get metadata
            struct HAL_Buffer_MetaData* metaData= spCamBuf->getMetaData();
            //对 buffer 进行处理
            ……
            //使用完后减小引用计数，当引用计数为 0 时，会将该 buffer 还给所有者。
            spCamBuf->decUsedCnt();
        }
    }
}
//反初始化流程如下
//移除 notifier
mpath->removeBufferNotifer(mMpBufNotifer);
delete mMpBufNotifer;
//停止 path 数据流
mpath->stop();
//释放 buffer
mpath->releaseBuffers();
//反初始化
testIspDev->deInitHw();
```