目录

[环境 1](#_Toc71882063)

[环境搭建 1](#_Toc71882064)

[Tftp32使用方式 1](#_Toc71882065)

[服务器搭建 1](#_Toc71882066)

[海思 3](#_Toc71882067)

[代码参考 3](#_Toc71882068)

[MPP（Media Process Platform，简称 MPP） 4](#_Toc71882069)

[VI（视频输入） 6](#_Toc71882070)

[VENC（视频编码器） 7](#_Toc71882071)

[VPSS（Video Process Sub-System是视频前处理单元） 8](#_Toc71882072)

[lVB 9](#_Toc71882073)

[网络功能 12](#_Toc71882074)

[参考 13](#_Toc71882075)

[TUTK 13](#_Toc71882076)

[编译 13](#_Toc71882077)

[运行 15](#_Toc71882078)

[基础概念 15](#_Toc71882079)

[H264 15](#_Toc71882080)

[码率 18](#_Toc71882081)

[移植 18](#_Toc71882082)

[安装编译链 18](#_Toc71882083)

[编译joylink2.0 19](#_Toc71882084)

[TUTK移植 20](#_Toc71882085)

[编译media\_sdk 20](#_Toc71882086)

[libcurl & libopenssl 20](#_Toc71882087)

[zxing-cpp（安装失败） 21](#_Toc71882088)

[zbar安装（qr码识别） 21](#_Toc71882089)

[安装libjpeg 21](#_Toc71882090)

[RUN 22](#_Toc71882091)

[源码修改 22](#_Toc71882092)

[思考 22](#_Toc71882093)

[加解密 23](#_Toc71882094)

[AES在线解密工具 23](#_Toc71882095)

[源码分析 23](#_Toc71882096)

[media\_sdk 23](#_Toc71882097)

[FFmpeg 25](#_Toc71882098)

[问题分析 25](#_Toc71882099)

[计划 25](#_Toc71882100)

[操作 26](#_Toc71882101)

[未操作 27](#_Toc71882102)

[问题 27](#_Toc71882103)

## 环境

### 环境搭建

内网主机设置192.168.1.101

设置有线IP同一网段，192.168.1.100

共享目录，远程使用\\192.168.1.100进行登录

安装虚拟机

安装tftpd32工具

使用驱动精灵安装串口驱动

安装vlc

### Tftp32使用方式

cd /tmp

tftp -g -r sample\_venc 192.168.1.100

chmod +x sample\_venc

./sample\_venc 1

tftp -p -l stream\_chn1.h264 192.168.1.100

### 服务器搭建

安装14.04 64位ubuntu server版

apt-get install libc6-dev-i386 64位机器运行32位库

apt-get install lib32z1 lib32ncurses5

apt-get install lib32stdc++6

sudo apt-get install cmake-qt-gui

sudo apt-get install qtbase5-dev

apt-get install libqt5multimedia5-plugins

apt-get install m4

**cmake安装**

<https://blog.csdn.net/qq_35398033/article/details/106457777>

sudo apt-get install build-essential

cmake -D CMAKE\_BUILD\_TYPE=Release -G "Unix Makefiles" ..

**gcc安装**

下载地址：<http://mirrors.ustc.edu.cn/gnu/gcc/>

./configure --prefix=/usr/local/gcc-11.1.0/ --enable-checking=release --enable-languages=c,c++ --disable-multilib --with-gmp=/usr/local/gmp-5.0.1 --with-mpfr=/usr/local/mpfr-3.1.5 --with-mpc=/usr/local/mpc-0.9

make

make install

gcc11安装依赖如下软件包

编译时中途报错，找不到libmpc.so.2，添加到/etc/profile中无效，只能在/usr/lib/下创建软连接

**gcc 11.1.0依赖库**

**gmp安装**

wget <ftp://ftp.gnu.org/gnu/gmp/gmp-5.0.1.tar.bz2>

tar -vxf gmp-5.0.1.tar.bz2

cd gmp-5.0.1/

./configure --prefix=/usr/local/gmp-5.0.1

make

sudo make install

**mpfr安装**

wget <https://ftp.gnu.org/gnu/mpfr/mpfr-3.1.5.tar.xz>

tar -vxf mpfr-3.1.5.tar.gz

cd mpfr-3.1.5/

./configure --prefix=/usr/local/mpfr-3.1.5 --with-gmp=/usr/local/gmp-5.0.1

make

sudo make install

**mpc安装**

wget <http://www.multiprecision.org/downloads/mpc-0.9.tar.gz>

tar -vxf mpc-0.9.tar.gz

cd mpc-0.9/

./configure --prefix=/usr/local/mpc-0.9 --with-gmp=/usr/local/gmp-5.0.1 --with-mpfr=/usr/local/mpfr-3.1.5

make

sudo make install

为gcc创建软连接和添加环境变量

cmake -D CMAKE\_BUILD\_TYPE=Release -D CMAKE\_INSTALL\_PREFIX=/usr/local/zxing -G "Unix Makefiles" ..

## 海思

### 代码参考

mpp包里面视频编码的例子测试方法：

cd mpp/sample/venc

make clean

make

编译正常后，会生成sample\_venc 这个拷贝到nfs目录

挂载nfs命令测试：

mount -t nfs -o nolock 192.168.1.199:/root/hi3516/nfs /home;cd /home

./sample\_venc 0

运行就行

会在当前目录生成1个h264的码流文件，1个h265的码流文件

h265码流文件使用资料里面电h265码流文件播放工具

h264码流文件使用VLC播放

#### 代码流程

step 0: Initialize related parameters

step 1: init sys alloc common vb

step 2: init and start vi

step 3: init and start vpss: One group with two channel(0:BigStream 1:SmallStream)

step 4: Bind VI and VPSS

step 5: start VENC (0: H265 BigStream; 1: JPEG snap with BigStream; 2: H264 SmallStream)

step 6: Bind VPSS and VENC( BigStream attach Vpss chan[0], SmallStream attch Vpss chan[1])

step 7: start get video stream (this thread just Get video channel stream(venc chan[0] and chan[2]))

step 8: snap picture process

初始化属性

SAMPLE\_VPSS\_CHN\_ATTR\_S stParam;

SAMPLE\_VB\_ATTR\_S commVbAttr;

初始化mpi系统接口

HI\_MPI\_SYS\_Exit();

HI\_MPI\_VB\_Exit();

HI\_MPI\_VB\_SetConfig(pstVbConfig);

HI\_MPI\_VB\_Init();

HI\_MPI\_SYS\_Init();

获取底层流数据

mpi\_venc.h

使用类型，获取图像分辨率

SAMPLE\_COMM\_SYS\_GetPicSize(enSize[i], &stSize[i]);

初始化VI配置文件

SAMPLE\_COMM\_VI\_GetSensorInfo(&stViConfig);

编码器初始化

SAMPLE\_VENC\_SYS\_Init

绑定VI到VENC

SAMPLE\_VENC\_VI\_Init(&stViConfig, stParam.ViVpssMode);

设置设置Vpss属性

SAMPLE\_VENC\_GetDefaultVpssAttr

使用Vpss 初始化commVbAttr

SAMPLE\_VENC\_GetCommVbAttr

使用commVbAttr进行venc系统初始化

通道号是固定写死的

获取通道属性

HI\_MPI\_VENC\_GetChnAttr(VencChn, &stVencChnAttr); //判断类型，用于设置流后缀（.h264）和fopen

设置VencFd

VencFd[i] = HI\_MPI\_VENC\_GetFd(VencChn);

两个VencFd放入fd\_set中，使用select进行监控

HI\_MPI\_VENC\_QueryStatus(VencChn, &stStat); //VencChn通道中获取状态，主要获取u32CurPacks

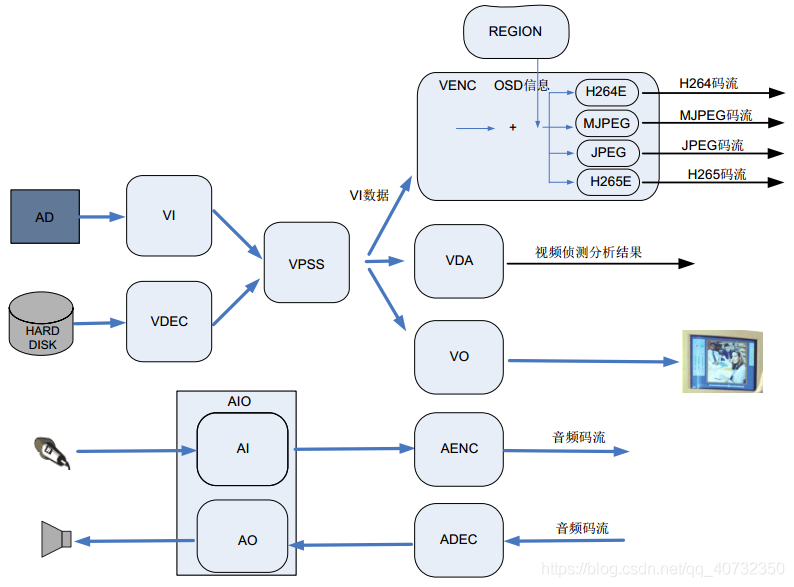
HI\_MPI\_VENC\_GetStream(i, &stStream, HI\_TRUE);

### MPP（Media Process Platform，简称 MPP）

海思提供的媒体处理软件平台(Media Process Platform,简称 MPP)，可支持应用软件快速开发。该平台对应用软件屏蔽了芯片相关的复杂的底层处理，并对应用软件直接提供MPI（MPP Programe Interface）接口完成相应功能。该平台支持应用软件快速开发以下功能：输入视频捕获、 H.265/H.264/MJPEG/JPEG/MPEG4 编码、 H.264/MPEG4/MPEG2解码、视频输出显示、视频图像前处理（包括去噪、增强、锐化、 Deinterlace）、编码码流叠加 OSD、视频侦测分析、智能分析、音频捕获及输出、音频编解码等功能。

**海思媒体处理平台架构**

海思媒体处理平台的主要内部处理流程如图 1-2 所示，主要分为视频输入（VI）、视频处理（VPSS）、视频编码（VENC）、视频解码（VDEC）、视频输出(VO)、视频侦测分析(VDA)、音频输入(AI)、音频输出(AO)、音频编码（AENC）、音频解码（ADEC）、区域管理（REGION）等模块。主要的处理流程介绍如下：



VI 模块捕获视频图像，可对其做剪切、缩放等处理，并输出多路不同分辨率的图像数据。解码模块对编码后的视频码流进行解码，并将解析后的图像数据送 VPSS 进行图像处理或直接送 VO 显示。可对 H264/MPEG4/MPEG2 格式的视频码流进行解码。

VPSS 模块接收 VI 和解码模块发送过来的图像，可对图像进行去噪、图像增强、锐化等处理，并实现同源输出多路不同分辨率的图像数据用于编码、预览或抓拍。

编码模块接收 VI 捕获并经 VPSS 处理后输出的图像数据，可叠加用户通过 Region模块设置的 OSD 图像，然后按不同协议进行编码并输出相应码流。

VDA 模块接收 VI 的输出图像，并进行移动侦测和遮挡侦测，最后输出侦测分析结果。

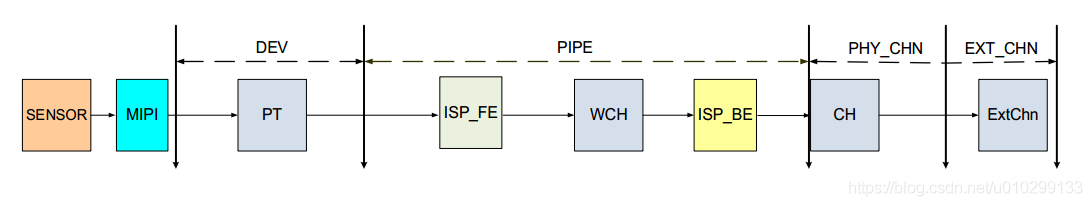
VO 模块接收 VPSS 处理后的输出图像，可进行播放控制等处理，最后按用户配置的输出协议输出给外围视频设备。

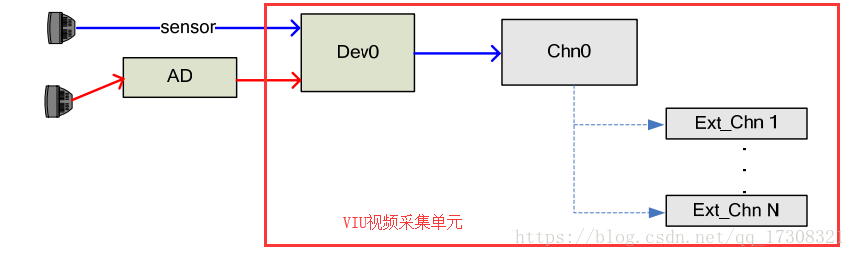
AI 模块捕获音频数据，然后 AENC 模块支持按多种音频协议对其进行编码，最后输出音频码流。

用户从网络或外围存储设备获取的音频码流可直接送给 ADEC 模块， ADEC 支持解码多种不同的音频格式码流，解码后数据送给 A O 模块即可播放声音。

### VI（视频输入）

视频输入（VI）模块实现的功能：通过 MIPI Rx(含 MIPI 接口、LVDS 接口和 HISPI 接口)，SLVS-EC，BT.1120，BT.656，BT.601，DC 等接口接收视频数据。VI 将接收到的数据存入到指定的内存区域，在此过程中，VI 可以对接收到的原始视频图像数据进行处理，实现视频数据的采集。





VI可通过ITU-R BT656/601/1120 接口或 Digital Camera接口、 MIPI Rx(含 MIPI 接口、 LVDS 接口和 HISPI 接口)接收视频数据。

Vi还可以对接收到的原始视频图像数据进行裁剪(Crop)等处理,并实现一路原始视频图像输入,输出一路视频图像功能.

**VI内部单元工作流程**

摄像机通过镜头(lens)聚焦光线,将光信号投射到 sensor 的感光区域,sensor经过光电转换，将 Bayer 格式的原始图像送给 ISP(Image Signal Processing 图像信号处理) 经过算法处理，输出 RGB 空间域的图像给后端的视频采集单元。在这个过程中，ISP 通过运行在其上的 firmware 对 lens 和 sensor 进行相应控制，进而完成自动光圈、自动曝光、自动白平衡等功能。其中，firmware 的运转靠视频采集单元的中断驱动。

**bayer格式**

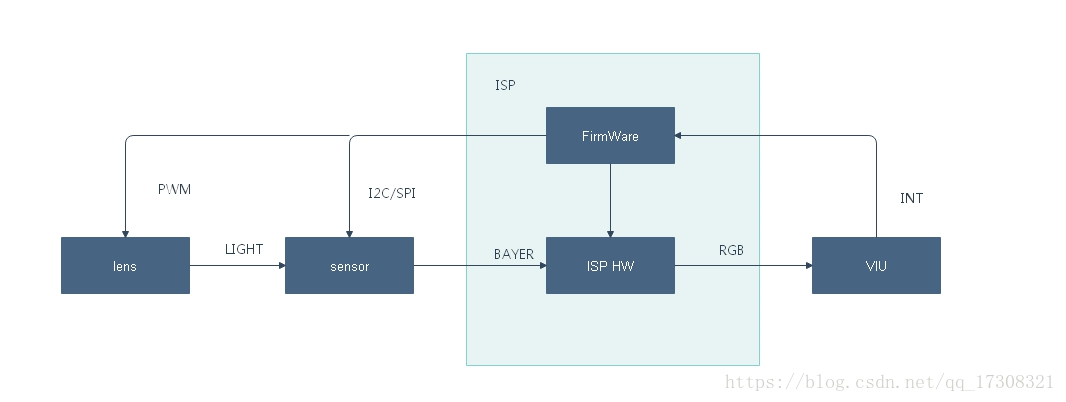
bayer格式图片是伊士曼·柯达公司科学家Bryce Bayer发明的，Bryce Bayer所发明的拜耳阵列被广泛运用数字图像。

　　对于彩色图像，需要采集多种最基本的颜色，如rgb三种颜色，最简单的方法就是用滤镜的方法，红色的滤镜透过红色的波长，绿色的滤镜透过绿色的波长，蓝色的滤镜透过蓝色的波长。如果要采集rgb三个基本色，则需要三块滤镜，这样价格昂贵，且不好制造，因为三块滤镜都必须保证每一个像素点都对齐。当用bayer格式的时候，很好的解决了这个问题。bayer 格式图片在一块滤镜上设置的不同的颜色，通过分析人眼对颜色的感知发现，人眼对绿色比较敏感，所以一般bayer格式的图片绿色格式的像素是是r和g像素的和。

Bayer格式是相机内部的原始图片, 一般后缀名为.raw。

参考：https://zhuanlan.zhihu.com/p/72581663

**ISP控制结构**



[初始化Vi后直接绑定Vi到Venc编码](https://blog.csdn.net/qq_17308321/article/details/80004319)

### VENC（视频编码器）

VENC 模块，即视频编码模块。本模块支持多路实时编码，且每路编码彼此独立，编

码协议和编码profile 可以不同。本模块支持视频编码同时，调度 Region 模块对编码图

像内容进行叠加和遮挡。本模块的输入源包括三类：第一类是用户态读取图像文件向

编码模块发送数据；第二类是视频输入（VIU ）模块采集的图像经视频处理子系统

（VPSS）发送到编码模块；第三类是视频输入（VIU ）模块采集的图像直接发送到编码

模块

#### 编码方式

**CBR编码**

　　在流式播放方案中使用CBR编码最为有效。使用CBR编码时，比特率在流的进行过程中基本保持恒定并且接近目标比特率，始终处于由缓冲区大小确定的时间窗内。CBR编码的缺点在于编码内容的质量不稳定。因为内容的某些片段要比其他片段更难压缩，所以CBR流的某些部分质量就比其他部分差。此外，CBR编码会导致相邻流的质量不同。通常在较低比特率下，质量的变化会更加明显。采用CBR编码方式时节目合成时间会短一些，但文件的长度会大一些(即相同时长的节目会占用更多的空间或者相同的空间只能容纳更短的节目)。

**VBR编码**

　　当计划提供内容供用户下载、 将内容在本地播放或者在读取速度有限的设备（如CD或DVD播放机）上播放时，请使用VBR编码。（计划流式播放内容时也可以采用峰值VBR编码模式）当 编码内容中混有简单数据和复杂数据（例如，在快动作和慢动作间切换的视频）时，VBR 编码是很有优势的。使用VBR编码时，系统将自动为内容的简单部分分配较少的比特，从而留出足量的比特用于生成高质量的复杂部分。这意味着复杂性恒定的内 容（例如新闻播音）不会受益于VBR编码。对混合内容使用 VBR 编码时，在文件大小相同的条件下，VBR编码的输出结果要比CBR编码的输出结果质量好得多。在某些情况下，与CBR编码文件质量相同的VBR编码文件， 其大小可能只有前者的一半。

**ABR**

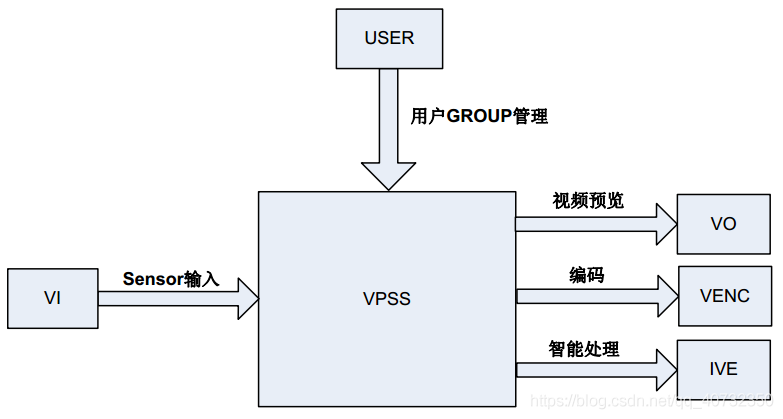
平均比特率，是VBR的一种插值参数。Lame针对CBR不佳的文件体积比和VBR生成文件大小不定的特点独创了这种编码模式。ABR也 被称为“Safe VBR”，它是在指定的平均Bitrate内，以每50帧（30帧约1秒）为一段，低频和不敏感频率使用相对低的流量，高频和大动态表现时使用高流量。举例来说，当指定用192kbps ABR对一段wav文件进行编码时，Lame会将该文件的85%用192kbps固定编码，然后对剩余15%进行动态优化：复杂部分用高于192kbps 来编码、简单部分用低于192kbps来编码。与192kbps CBR相比，192kbps ABR在文件大小上相差不多，音质却提高不少。ABR编码在速度上是VBR编码的2到3倍，在128-256kbps范围内质量要好于CBR。可以做为VBR和CBR的一种折衷选择。

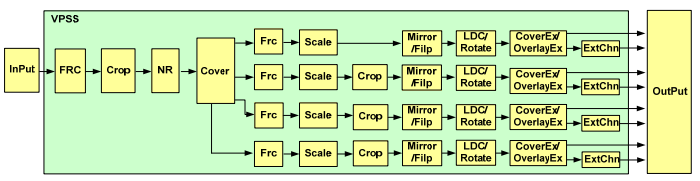
### VPSS（Video Process Sub-System是视频前处理单元）

VPSS属于硬件，支持对一幅输入图像进行  
统一预处理，如去噪、去隔行等，然后再对各通道分别进行缩放、锐化等处理，最后  
输出多种不同分辨率的图像。

VPSS单元支持的具体图像处理功能包括PreScale、De-ring/De-block、NR、IE 、DIE、

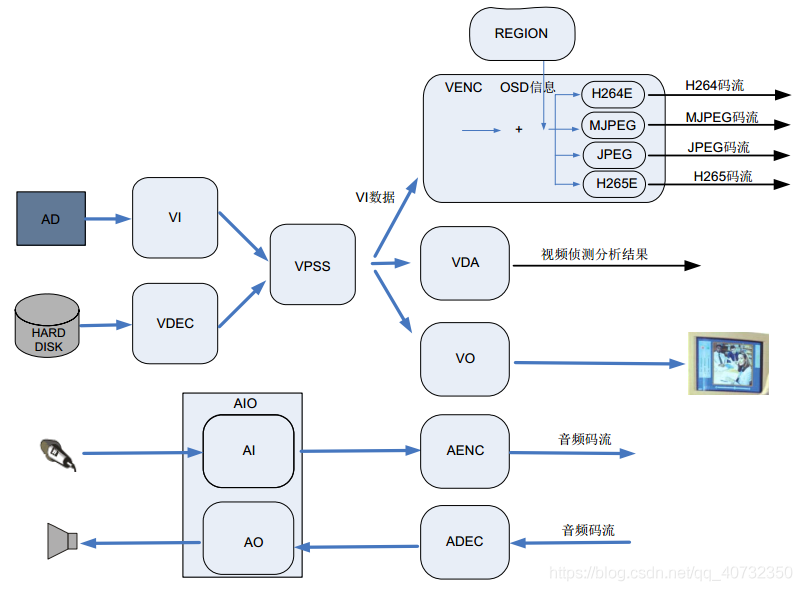
Sharpen等





参考：https://blog.csdn.net/qq\_40732350/article/details/88097957

### lVB



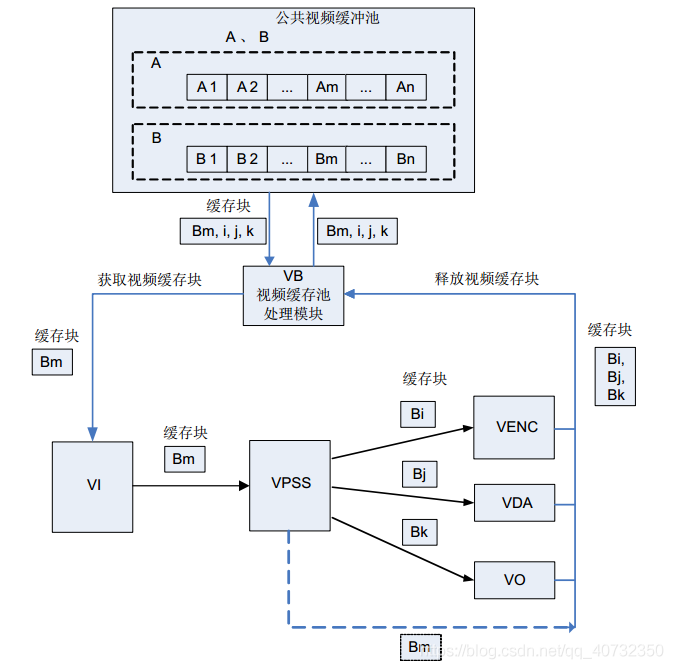
**视频缓存池**

视频缓存池主要向媒体业务提供大块物理内存管理功能，负责内存的分配和回收，充分发挥内存缓存池的作用，让物理内存资源在各个媒体处理模块中合理使用。

一组大小相同、物理地址连续的缓存块组成一个视频缓存池。

视频输入通道需要使用公共视频缓存池。所有的视频输入通道都可以从公共视频缓存池中获取视频缓存块用于保存采集的图像（如图中所示从公共视频缓存池 B 中获取视频缓存块 Bm）。由于视频输入通道不提供创建和销毁公共视频缓存池功能，因此，在系统初始化之前，必须为视频输入通道配置公共视频缓存池。

根据业务的不同，公共缓存池的数量、缓存块的大小和数量不同。 图中所示缓存块的生存期是指经过 VPSS 通道传给后续模块的情形（图 实线路径）。如果该缓存块完全没有经过 VPSS 通道传给其他模块，则将在 VPSS 模块处理后被放回公共缓存池（图虚线路径）。



参考

https://blog.csdn.net/qq\_40732350/article/details/87970605

### 网络功能

板子启动后建立ap热点命令：

ifconfig wlan0 up

hostapd /etc/hostapd.conf -B

ifconfig wlan0 192.168.2.1

udhcpd -fS /etc/udhcpd.conf &

默认的热点名字和密码配置文件在/etc/hostapd.conf 文件里面

默认的热点名字：qddytt

密码：qddytt11

板子连接到无线路由器命令

ifconfig wlan0 up

iwlist wlan0 scanning (扫描附近的无线路由器)

wpa\_supplicant -B -Dwext -iwlan0 -c/etc/wpa.conf

动态获取ip

udhcpc -i wlan0

手动设置ip

ifconfig wlan0 192.168.1.67

ping 192.168.1.1 //测试网络是否通，请ping 客户wifi路由器的ip，连接上来就可以ping通路由器

连接到无线路由器的ssid和密码请在/etc/wpa.conf 里面修改。用户可以直接使用vi编辑这个文件，也可以修改rootfs里面的，然后重新制作烧写rootfs

命令：

ifconfig wlan0 up

wpa\_supplicant -B -Dwext -iwlan0 -c/etc/wpa.conf

udhcpc -i wlan0

修改/etc/wpa.conf SSID password

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ctrl\_interface=/var/run/wpa\_supplicant

update\_config=1

network={

ssid="HUAWEI-i8"

key\_mgmt=WPA-PSK

proto=RSN WPA WPA2

pairwise=TKIP CCMP

group=TKIP CCMP

psk="jd654321"

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### 参考

https://blog.csdn.net/u010299133/article/details/100993946?utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-3.control&dist\_request\_id=1330147.33897.16182172534293061&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-3.control

海思Hi3518E的视频处理

https://blog.csdn.net/qq\_40732350/category\_8693414.html

## TUTK

### 编译

**Makefile编译**

静态

Makefile添加红色部分

CFLAGS += -g -O2 -Wall -std=gnu99 -static

make CC=arm-himix100-linux-gcc ARCH=Arm\_Hi3516EV200\_6.3.0 LINK=static

动态

make CC=arm-himix100-linux-gcc ARCH=Arm\_Hi3516EV200\_6.3.0

UID

//博冠，志伟提供

FZYAA93CPHFMUNPGU1YJ

FLKAAX1MRHFMAM6GU1MJ  
DLPAAX1WR9ZM9M6GY1P1  
DRKU8X1CKTFCUNPGU1Y1  
FFYUAT14Y1Z4VMPGY1W1  
FBUAAXJMVXZCBM6GU1W1  
FFYA9T1MKDF4BM6GU1EJ  
F3YA811MKNZMVMPGY14J  
F7UU8114K9FM8NPGY1WJ  
DFPUB51WZXFWTN6GU1E1  
DVPA95JCUDZW9M6GU1EJ

**手动编译**

静态

/home/suxiaowen/gitsource/joylink\_camera\_sdk/joylink\_stream\_media\_sdk/toolchain/hisi-linux/x86-arm/arm-himix100-linux/bin/arm-himix100-linux-gcc -I../../../Include -g -O2 -Wall -static -std=gnu99 -I. -I./extension/ -I./extension/TUTK\_CLI -I./extension/json-c AVAPIs\_Server.c extension/emulater/emulater.c extension/ability/ability.c extension/speaker/speaker.c extension/token/token.c extension/playback/playback.c -lAVAPIsT -lIOTCAPIsT -lTUTKGlobalAPIsT -L../../../Lib/Linux/Arm\_Hi3516EV200\_6.3.0 -Wl,-rpath=../../../Lib/Linux/Arm\_Hi3516EV200\_6.3.0 -lpthread -o AVAPIs\_Server

动态

/home/suxiaowen/gitsource/joylink\_camera\_sdk/joylink\_stream\_media\_sdk/toolchain/hisi-linux/x86-arm/arm-himix100-linux/bin/arm-himix100-linux-gcc -I../../../Include -g -O2 -Wall -std=gnu99 -I. -I./extension/ -I./extension/TUTK\_CLI -I./extension/json-c AVAPIs\_Server.c extension/emulater/emulater.c extension/ability/ability.c extension/speaker/speaker.c extension/token/token.c extension/playback/playback.c -lAVAPIsT -lIOTCAPIsT -lTUTKGlobalAPIsT -L../../../Lib/Linux/Arm\_Hi3516EV200\_6.3.0 -Wl,-rpath=../../../Lib/Linux/Arm\_Hi3516EV200\_6.3.0 -lpthread -o AVAPIs\_Server

设备端(linux)：SDK\Sample\Linux\Sample\_AVAPIs\AVAPIs\_Server.c

### 运行

./AVAPIs\_Server FZYAA93CPHFMUNPGU1YJ 1.txt 2.txt

./AVAPIs\_Client -u FZYAA93CPHFMUNPGU1YJ

thread\_ForAVServerStart

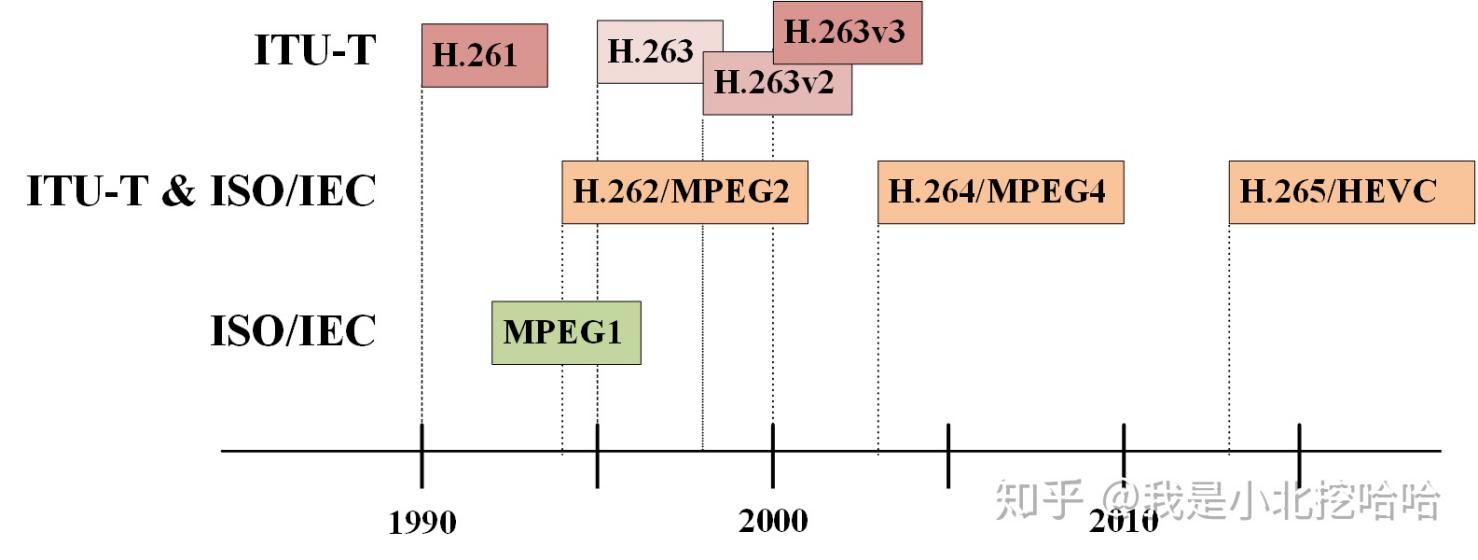
## 基础概念

### H264

**百度百科ISO/IEC会发现，其实这是2个机构：**

**ISO**并不陌生 - 国际标准化组织，下文同样摘自百度百科（会发现，ISO就是一个制定国际化标准）：  
国际标准化组织（International Organization for Standardization，ISO）简称ISO，是一个全球性的非政府组织，是国际标准化领域中一个十分重要的组织。ISO国际标准组织成立于1947年。ISO负责目前绝大部分领域（包括军工、石油、船舶等垄断行业）的标准化活动。ISO的宗旨是“在世界上促进标准化及其相关活动的发展，以便于商品和服务的国际交换，在智力、科学、技术和经济领域开展合作。”

**IEC**是国际电工委员会，其介绍和历史摘自百度百科：  
国际电工委员会（International Electrotechnical Commission，IEC）成立于1906年，它是世界上成立最早的国际性电工标准化机构，负责有关电气工程和电子工程电子工程/5448766)领域中的国际标准化工作。  
1947年作为一个电工部门并入国际标准化组织（ISO），1976年又从ISO中分立出来。宗旨促进电气、电子工程领域中标准化及有关问题的国际合作，增进国际间的相互了解。



两个组织也在一起共同制定了一些标准：H.262标准等同于MPEG-2的视频编码标准；H.264标准等同于MPEG-4，并且被收录到第10部分。之前，笔者之前一直不明白，对于『H.264』，为什么有的地方会称呼为『MPEG4 AVC』，整理到这里终于明白了：这个协议是两个组织机构共同指定的，每个机构收录到各自的标准手册中，为了保持命名统一，所以才衍生了H.264和MPEG4 AVC两个名词。而最新的H.264标准则被纳入MPEG-4的第10部分，所以又称MPEG4 Part 10 AVC。

MPEG3并不存在！从维基百科摘录原因如下：  
MPEG-3是在制定MPEG-2标准之后准备推出的适用于HDTV（高清晰度电视）的视频、音频压缩标准，但是由于MPEG-2标准已经可以满足要求，故MPEG-3标准并未正式推出。

H264能同时做到高压缩比和良好的网络亲和性；

**H264可以分为2层：视频编码层『VCL』和网络提取层『NAL』。**

#### VCL

VCL   video coding layer    视频编码层；  
NAL   network abstraction layer   网络提取层；

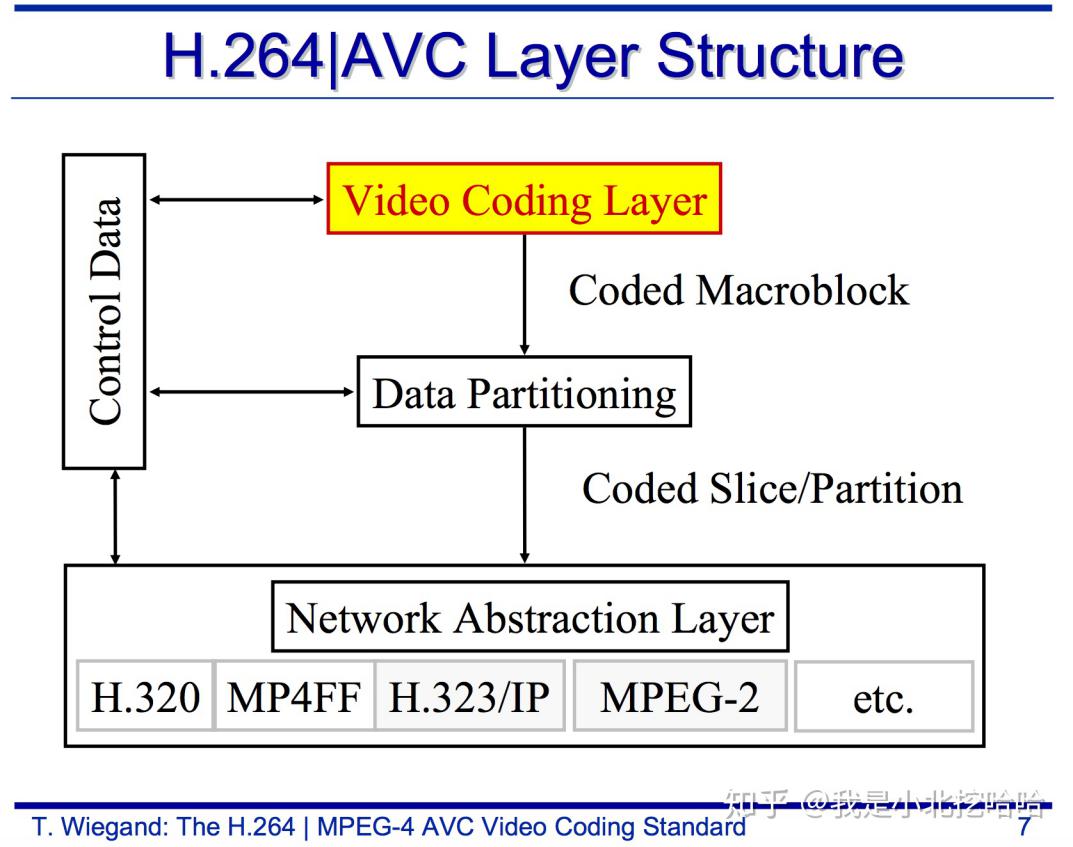
其中，VCL层是对核心算法引擎，块，宏块及片的语法级别的定义，他最终输出编码完的数据 SODB；

NAL层定义片级以上的语法级别（如序列参数集和图像参数集，针对网络传输），

同时支持以下功能：独立片解码，起始码唯一保证，SEI以及流格式编码数据传送，NAL层将SODB打包成RBSP然后加上NAL头，组成一个NALU（NAL单元）；

H264网络传输的结构  
  
H264在网络传输的是NALU，NALU的结构是：NAL头+RBSP，实际传输中的数据流如图所示：





**抛开H264压缩算法细节来看就3步：**

1、压缩：预测（帧内预测和帧间预测）-> DCT变化和量化 -> 比特流编码；

2、切分数据，主要为了第三步。这里一点，网上看到的“切片（slice）”、“宏块（macroblock）”是在VCL中的概念，一方面提高编码效率和降低误码率、另一方面提高网络传输的灵活性。（这块本文将不展开）

3、包装成『NAL』。

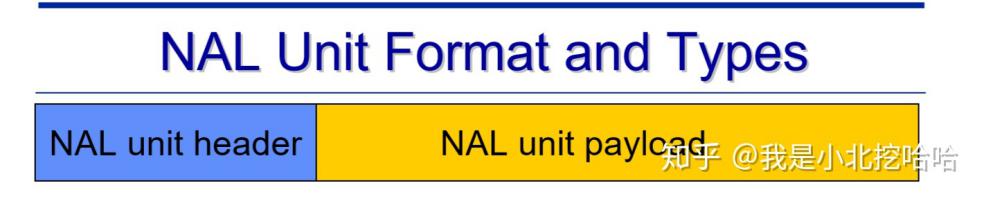
--------------------------------------------------------------------------------------------------

The NAL is designed in order to provide "network friendliness" to enable simple and effective customization of the use of VCL for a broad variety of systems. The NAL facilitates the ability to map VCL data to transport layers such as:  
· RTP/IP for any kind of real-time wire-line and wireless Internet services.  
· File formats, e.g., ISO MP4 for storage and MMS.  
· H.32X for wireline and wireless conversational services.  
· MPEG-2 systems for broadcasting services, etc.  
  
NAL旨在提供“网络友好性”，以便为各种系统简单有效地定制VCL的使用。 NAL有助于将VCL数据映射到传输层，例如：  
· RTP / IP适用于任何类型的实时有线和无线互联网服务。  
· 文件格式，例如用于存储和MMS的ISO MP4。  
· H.32X用于有线和无线会话服务。  
· 用于广播服务等的MPEG-2系统。

--------------------------------------------------------------------------------------------------

**直白翻译过来就是：『NAL』就是为了包装『VCL』以达到更好网络传输效果**

#### NAL



**『NAL』的组成单元 - 『NALU』**

参考

https://zhuanlan.zhihu.com/p/71928833

#### 码流格式

https://winddoing.github.io/post/35564.html

### 码率

视频码率就是数据传输时单位时间传送的数据位数，一般我们用的单位是kbps即千位每秒。通俗一点的理解就是取样率，单位时间内取样率越大，精度就越高，处理出来的文件就越接近原始文件。

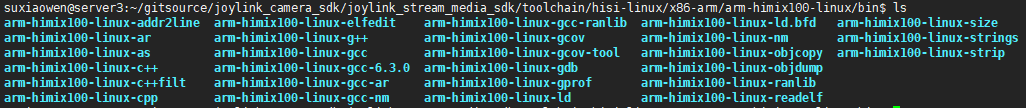
比特率又称“二进制位速率

## 移植

### 安装编译链

安装编译链

hisi-linux/x86-arm/arm-himix100-linux/bin



设置编译链可以使用全路径

CROSS\_COMPLE=/home/suxiaowen/gitsource/joylink\_camera\_sdk/joylink\_stream\_media\_sdk/toolchain/hisi-linux/x86-arm/arm-himix100-linux/bin/arm-himix100-linux-

### 编译joylink2.0

joylink\_stream\_media\_sdk/extern\_lib/joylink/src/scripts/中添加配置文件Arm\_Hi3516EV200.mk

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#CROSS\_COMPLE=/opt/hisi-linux/x86-arm/arm-himix100-linux/bin/arm-himix100-linux-

CROSS\_COMPLE=/home/suxiaowen/gitsource/joylink\_camera\_sdk/joylink\_stream\_media\_sdk/toolchain/hisi-linux/x86-arm/arm-himix100-linux/bin/arm-himix100-linux-

CC=${CROSS\_COMPLE}gcc

AR=${CROSS\_COMPLE}ar

#RANLIB=${CROSS\_COMPLE}gcc-ranlib

RANLIB=${CROSS\_COMPLE}ranlib

LDFLAGS=-fPIC -shared

CFLAGS +=-fPIC

CFLAGS += -D\_\_LINUX\_UB2\_\_

CFLAGS += -D\_\_LINUX\_\_

CFLAGS += -g

CFLAGS += -DJOYLINK\_FOR\_LINUX

CFLAGS += -DJOYLINK\_SDK\_EXAMPLE\_TEST

CFLAGS += -D\_\_LINUX\_PAL\_\_

#CFLAGS += -D\_\_CPU\_64\_BIT\_\_

CFLAGS += -D\_\_CPU\_32\_BIT\_\_

LIBS+= -lpthread

#CFLAGS +=-Wextra

#CFLAGS += -Wall -Werror

CFLAGS +=-Wshadow -Wpointer-arith -Waggregate-return -Winline -Wunreachable-code -Wcast-align -Wredundant-decls -std=gnu99

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

./once.sh

make distclean

make

在target/lib/下生成静态库libjoylinksdk.a -> libjoylink.a

手动将静态库拷贝到extern\_lib/joylink/lib/Arm\_Hi3516EV200/下面

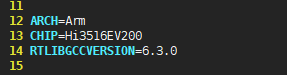


joylink\_stream\_media\_sdk在编译是使用的静态库名为libjoylink.a，手动拷贝出一份

### TUTK移植

将TUTK库拷贝到extern\_lib下面，TUTK库中默认支持Arm\_Hi3516EV200\_6.3.0

设置joylink\_stream\_media\_sdk中Makefile.rule中板载信息，对应TUTK库



### 编译media\_sdk

在scripts中创建Arm\_Hi3516EV200.mk，参考[编译joylink2.0](#_编译joylink2.0)

./once.sh

make distclean

make //生成sdk库

make jt //生成执行文件

由于TUTK\_3.3更新后链接库名有变化，修改example中Makefile 链接库



应用使用target/bin/jt，不要使用release/xxx/target/bin/jt

### libcurl & libopenssl

TUTK库中包含了libcurl & libopenssl，可以直接使用，如果未提供，可参考《libcurl、libopenssl编译方式》

**以下是简单示例**

编译openssl

./config CC=arm-himix100-linux-gcc no-asm shared no-async --prefix=/home/suxiaowen/gitsource/joylink\_camera\_sdk/openssl/build --openssldir=/home/suxiaowen/gitsource/joylink\_camera\_sdk/openssl/openssl-1.1.1k/ssl

编译curl

vi ~/.bashrc

export PATH=/home/suxiaowen/gitsource/joylink\_camera\_sdk/joylink\_stream\_media\_sdk/toolchain/hisi-linux/x86-arm/arm-himix100-linux/bin:$PATH

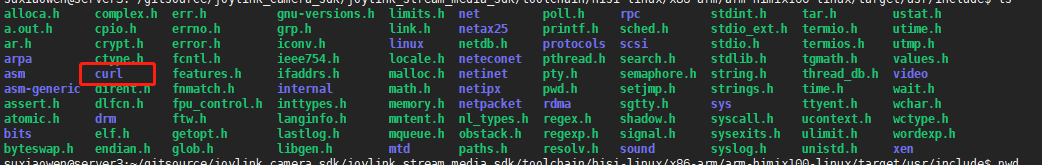
./configure --with-ssl=/home/suxiaowen/gitsource/joylink\_camera\_sdk/openssl/build --prefix=/home/suxiaowen/gitsource/joylink\_camera\_sdk/curl/curl-7.76.1/build --host=arm-linux CC=arm-himix100-linux-gcc CXX=arm-himix100-linux-g++

make

make install

编译链中需要添加curl头文件

hisi-linux/x86-arm/arm-himix100-linux/target/usr/include



**参考：**

https://blog.csdn.net/swallow\_he/article/details/98478466

<https://blog.csdn.net/wuzhiwuweisun/article/details/79129838>

<https://www.cnblogs.com/dakewei/p/11365179.html>

<https://www.cnblogs.com/dakewei/p/11365258.html>

<http://eleaction01.spaces.eepw.com.cn/articles/article/item/177269>

### zxing-cpp（安装失败）

<https://github.com/nu-book/zxing-cpp>

<https://blog.csdn.net/pyt1234567890/article/details/105529640>

### zbar安装（qr码识别）

下载地址：http://zbar.sourceforge.net/download.html

./configure --enable-shared --without-python --without-gtk --without-qt --without-imagemagick CFLAGS="" --disable-video

https://blog.csdn.net/lwjian147/article/details/80481237

<https://www.freesion.com/article/805754910/>

交叉编译

./configure --host=arm-himix100-linux --enable-shared --without-gtk --without-python --without-qt --without-imagemagick --disable-video CFLAGS=""

### 安装libjpeg

下载地址：<http://libjpeg.sourceforge.net>

tar zxf jpegsrc.v9c.tar.gz

cd jpeg-9c/

./configure --prefix=/usr/local/jpeg

make

sudo make install

**libjpeg内存方式解码**

<https://www.cnblogs.com/superbi/p/3447350.html>

<https://blog.csdn.net/zjf82031913/article/details/5826388>

**jpeg解码yuv**

<https://blog.csdn.net/dawudayudaxue/article/details/113139047>

交叉编译

./configure CC=arm-himix100-linux-gcc --host=arm-linux --prefix=/usr/local/jpeg

### RUN

[编译media\_sdk](#_编译media_sdk)会生成执行文件，将文件推送到小机，同时推送TUTK动态库，joylink、joylink\_media\_sdk使用静态方式链接，无需推送。

### 源码修改

对于TUTK3.3接口，一些结构体内部字段重新定义，添加两个宏用于编译

TUTK\_SDK\_VER\_3\_1

TUTK\_SDK\_VER\_3\_3

Arm\_Hi3516EV200.mk中添加

CFLAGS += -DTUTK\_SDK\_VER\_3\_3

### 思考

对于多个编译链如何管理，repo ?

编译链是一定要上传到服务器的，并且openssl 和 curl也需要进行编译

打包文件不仅要包含日期，还要包含模组信息和芯片信息，如Arm\_His3516EV200

TUTK代码库也要做合理保存

## 加解密

Base64

<https://blog.csdn.net/qq_26093511/article/details/78836087>

### AES在线解密工具

<https://www.mklab.cn/utils/aes>

**摄像头AES加密方式**

模式cbc 填充Pkcs7 位数256 密钥UUID右补0至32位 偏移量空

## 源码分析

### media\_sdk

使用mailbox进行指令发送，涉及三个模块：

cloud iot rt

没激活之前g\_media\_sdk\_storage\_param为空

启动四个task

jl\_media\_sdk\_sm\_start();

jl\_media\_sdk\_rt\_app\_start();

jl\_media\_sdk\_iot\_app\_start();

jl\_media\_sdk\_cloud\_app\_start();

导入token到/tmp/token.txt

往通道送数据帧接口

thread\_VideoFrameData

avSendFrameData

#define TUTK\_ER\_NO\_LICENSE\_KEY -1005

gUID[21] = "CZYAB114ZHB4TG6GUHDJ";

jl\_media\_tutk\_frame\_data\_get

问题：

ability\_file 需要设置吗

需要license

看iot授权

media\_cloud模块是做什么用的

向云端请求设备激活绑定

joylink\_dev\_active\_post

jl\_media\_iot\_loop

在视频流接收的地方，接收长度大于send\_buf



## FFmpeg

1）libavcodec：音视频编解码库；

2）libavdevice：音视频输出输入设备库；

3）Libavfilter：音视频滤镜库，滤镜主要是用来实现图像的各种特殊效果；

4）libavformat：音视频格式库，用于各种音视频封装格式的生成和解析；

5）libavresample：采样率库；？？？

6）libavutil：音视频工具库，包含一些公共的工具函数；

7）libpostproc：音视频后期效果处理库；

8）libswresample：采样率库；？？？

9）libswscale：视频场景比例缩放、色彩映射转换库；

avcodec：编解码（最重要的库）。

avformat：封装格式处理。

avfilter：滤镜特效处理。

avdevice：各种设备的输入输出。

avutil：工具库（大部分库都需要这个库的支持）。

postproc：后加工。

swresample：音频采样数据格式转换。

swscale：视频像素数据格式转换

https://blog.csdn.net/guyuealian/article/details/79493019

### 图像反交错

ffmpeg使用deinterlace流程分析

命令参数：

ffmpeg -i test.mpg -vcodec libx264 -s 1024x768 -b:v 700k -r 25 -deinterlace -acodec libmp3lame -ar 22050 -f flv -y test.flv

ffmpeg -i test.mpg -vcodec libx264  -s 1280x768 -b:v 700k -r 25 -vf yadif -acodec libmp3lame -ar 22050 -f flv -y test.flv

ffmpeg\_opt.c中定义参数

### 图像缩放

parse\_sws\_flags 解析缩放算法

命令使用方式：

ffmpeg -s 480x272 -pix\_fmt yuv420p -i src01\_480x272.yuv -s 1280x720 -sws\_flags bilinear -pix\_fmt yuv420p src01\_bilinear\_1280x720.yuv

ffmpeg -s 480x272 -pix\_fmt yuv420p -i src01\_480x272.yuv -s 1280x720 -sws\_flags bicubic -pix\_fmt yuv420p src01\_bicubic\_1280x720.yuv

ffmpeg -s 480x272 -pix\_fmt yuv420p -i src01\_480x272.yuv -s 1280x720 -sws\_flags neighbor -pix\_fmt yuv420p src01\_neighbor\_1280x720.yuv

### 函数解析

avfilter\_graph\_parse2

和 avfilter\_graph\_parse\_ptr 函数类似，不同的是，inputs 和 outputs 函数不作为输入参数，

仅作为输出参数，返回字符串描述的新的被解析的graph。在这个parse函数后，仍然处于open状态的inputs和outputs。

返回的 inout 应该使用 avfilter\_inout\_free() 释放掉。

猜测：根据字符串，创建filter并添加到链中

configure\_input\_filter

创建输入源

configure\_output\_filter

创建输出源

视频滤镜-去掉logo

vf\_delogo.c

### ffmpeg应用程序分析

**获取原始数据流送入avfilter.c**

static int transcode\_step(void)

int configure\_filtergraph(FilterGraph \*fg)

字幕

void sub2video\_update(InputStream \*ist, AVSubtitle \*sub)

static void sub2video\_push\_ref(InputStream \*ist, int64\_t pts)

int attribute\_align\_arg av\_buffersrc\_add\_frame\_flags(AVFilterContext \*ctx, AVFrame \*frame, int flags)

int attribute\_align\_arg av\_buffersrc\_add\_frame(AVFilterContext \*ctx, AVFrame \*frame)

int attribute\_align\_arg av\_buffersrc\_add\_frame\_flags(AVFilterContext \*ctx, AVFrame \*frame, int flags)

static int av\_buffersrc\_add\_frame\_internal(AVFilterContext \*ctx, AVFrame \*frame, int flags)

static int request\_frame(AVFilterLink \*link)

int ff\_filter\_frame(AVFilterLink \*link, AVFrame \*frame)

**filter处理**

static int transcode\_step(void)

static int transcode\_from\_filter(FilterGraph \*graph, InputStream \*\*best\_ist)

int avfilter\_graph\_request\_oldest(AVFilterGraph \*graph)

int ff\_filter\_graph\_run\_once(AVFilterGraph \*graph)

int ff\_filter\_activate(AVFilterContext \*filter)

static int ff\_filter\_activate\_default(AVFilterContext \*filter)

static int ff\_filter\_frame\_to\_filter(AVFilterLink \*link)

static int ff\_filter\_frame\_framed(AVFilterLink \*link, AVFrame \*frame)

int (\*filter\_frame)(AVFilterLink \*, AVFrame \*); //滤镜处理

**buffersrc获取数据**

static int transcode\_step(void)

static int process\_input(int file\_index)

static int process\_input\_packet(InputStream \*ist, const AVPacket \*pkt, int no\_eof)

static int decode\_video(InputStream \*ist, AVPacket \*pkt, int \*got\_output, int64\_t \*duration\_pts, int eof,

int \*decode\_failed)

static int send\_frame\_to\_filters(InputStream \*ist, AVFrame \*decoded\_frame)

static int ifilter\_send\_frame(InputFilter \*ifilter, AVFrame \*frame)

int attribute\_align\_arg av\_buffersrc\_add\_frame\_flags(AVFilterContext \*ctx, AVFrame \*frame, int flags)

**合成最终数据**

static int reap\_filters(int flush)

static void do\_video\_out(OutputFile \*of,

OutputStream \*ost,

AVFrame \*next\_picture,

double sync\_ipts)

int attribute\_align\_arg avcodec\_send\_frame(AVCodecContext \*avctx, const AVFrame \*frame)

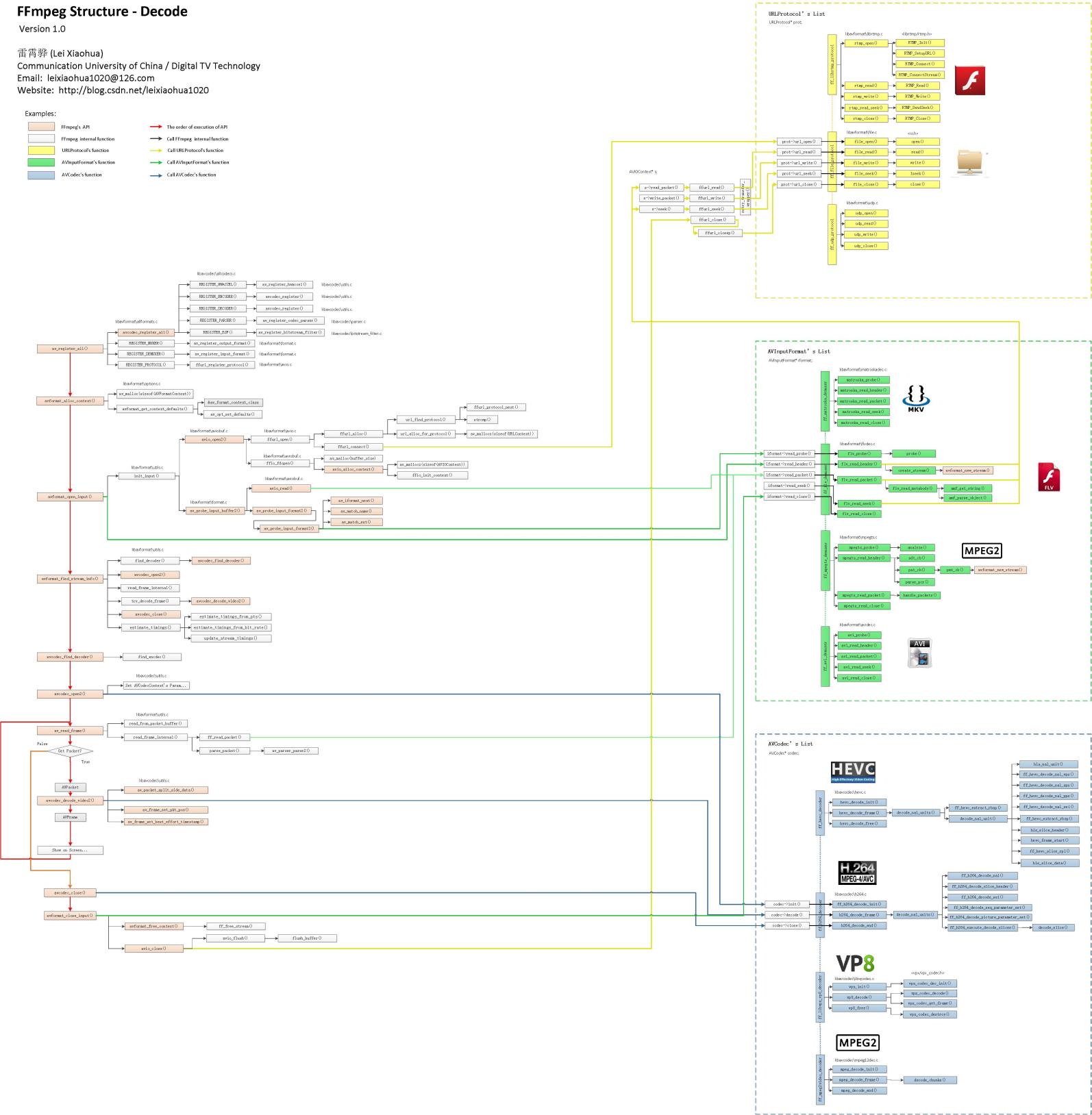
int attribute\_align\_arg avcodec\_encode\_video2(AVCodecContext \*avctx,

AVPacket \*avpkt,

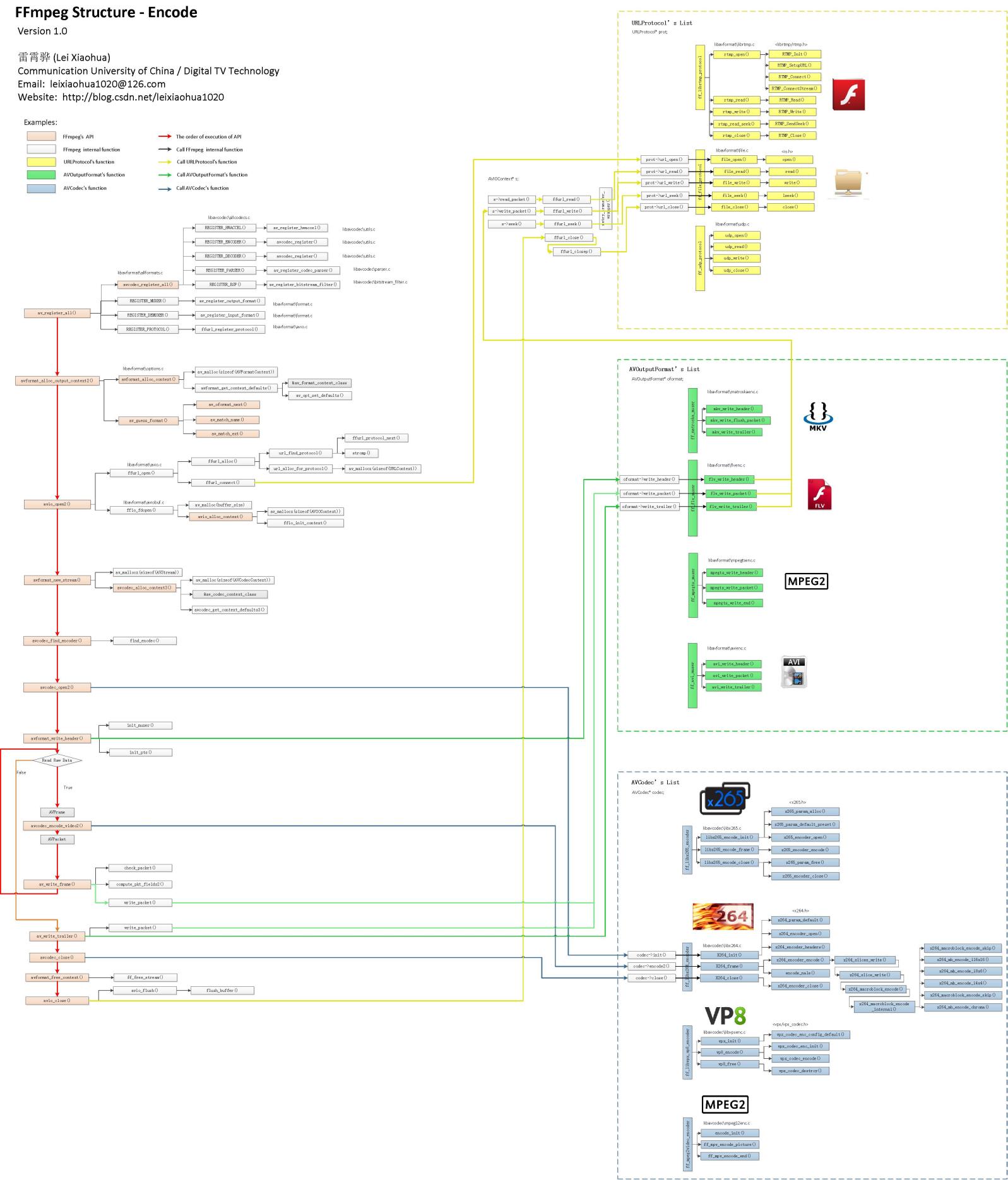
const AVFrame \*frame, int \*got\_packet\_ptr)

### 结构及流程框图

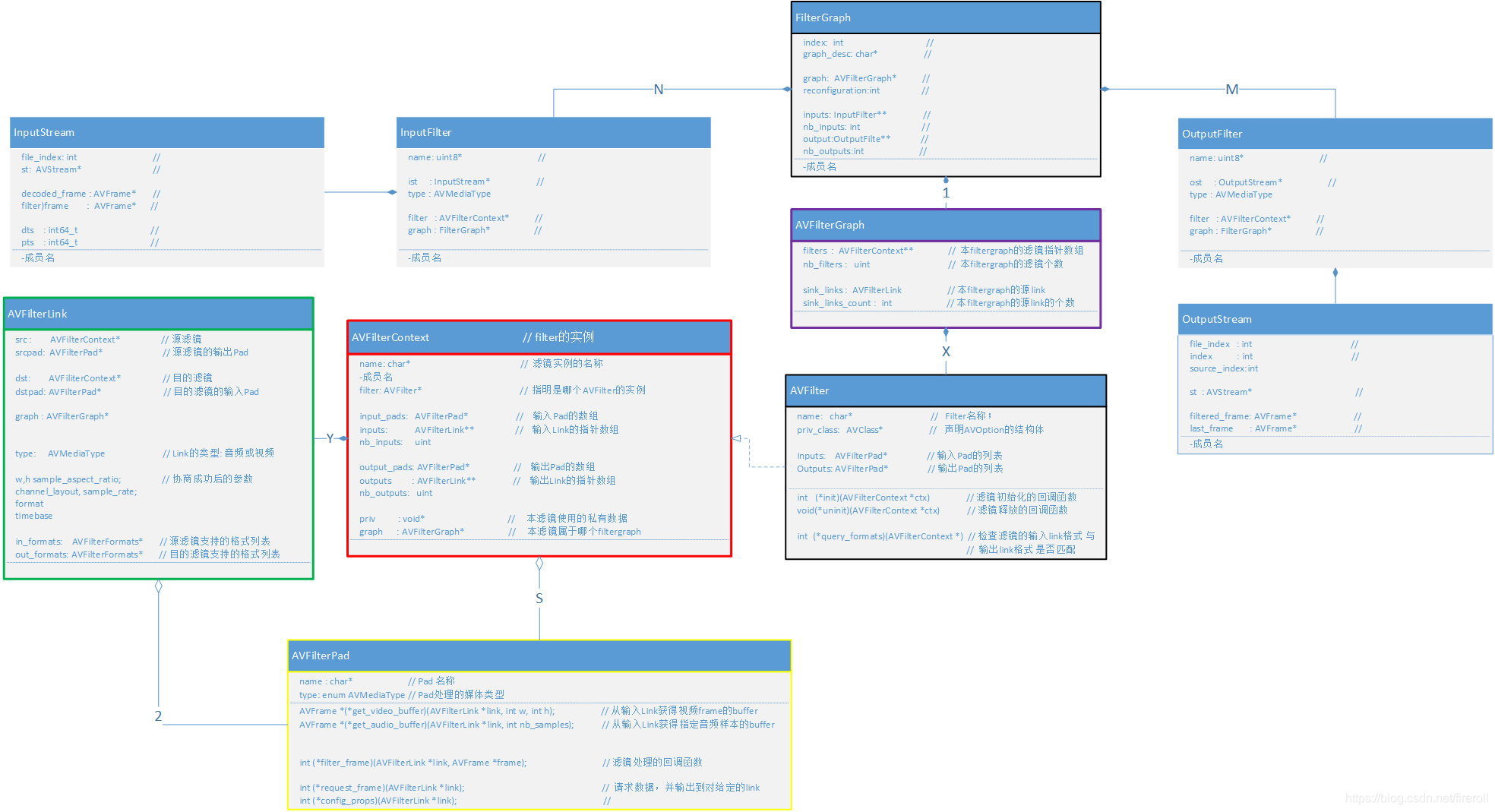
#### 解码流程



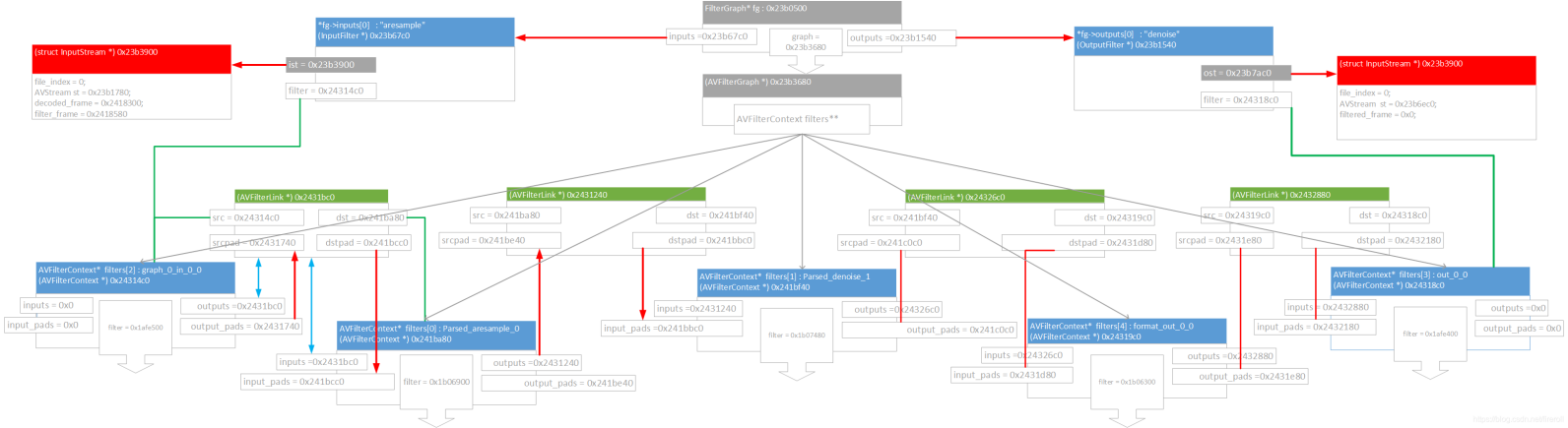
#### 编码流程



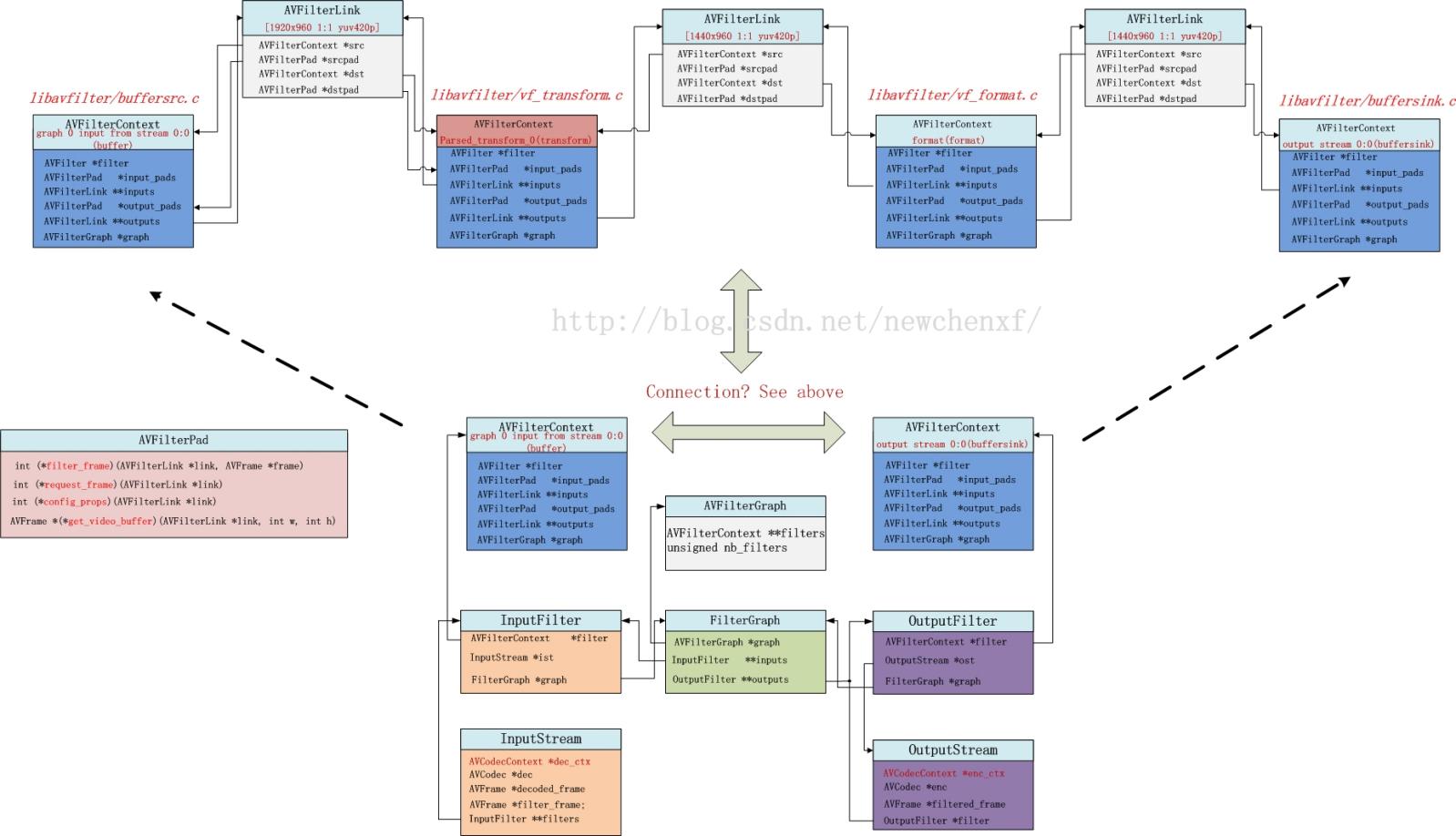
#### FFmpeg数据结构关系图



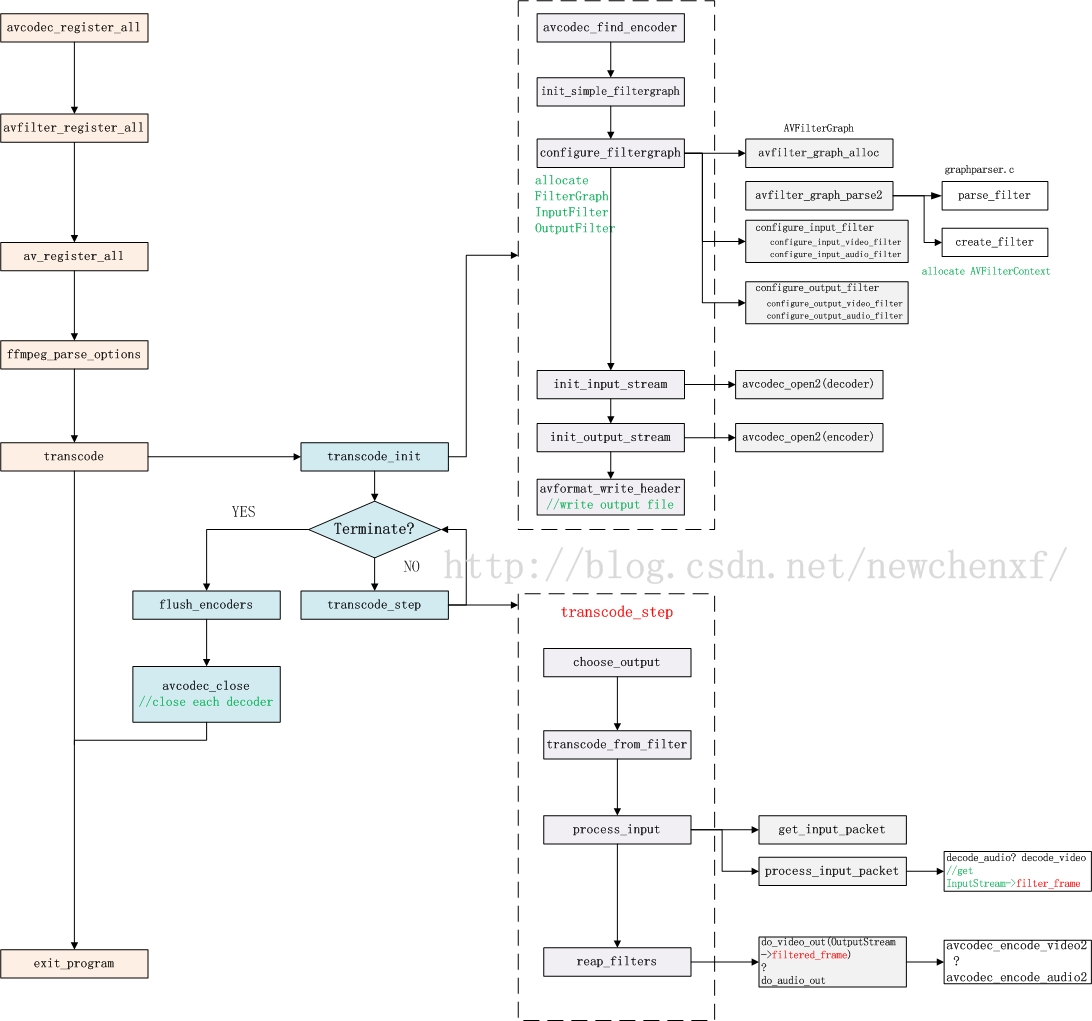
#### ffmpeg filter graph



#### ffmpeg filter graph 2



#### filter函数流程图



## Ijkplayer

### 下载及安装

下载地址：

<https://github.com/Bilibili/ijkplayer>

按照安装步骤进行安装，多次执行./init-android.sh下载代码

You must define ANDROID\_NDK before starting.

They must point to your NDK directories.

<https://blog.csdn.net/vnanyesheshou/article/details/53814951>

ubuntu安装NDK

wget -c <http://dl.google.com/android/ndk/android-ndk64-r10b-linux-x86_64.tar.bz2>

build on Linux x86\_64

ANDROID\_NDK=/home/suxiaowen/gitsource/Android-NDK/android-ndk-r10b/

IJK\_NDK\_REL=

You need the NDKr10e or later

编译时版本不支持

下载r10e版本

wget -c <http://dl.google.com/android/ndk/android-ndk-r10e-linux-x86_64.bin>

./android-ndk-r10e-linux-x86\_64.bin

export PATH=$PATH:/home/suxiaowen/gitsource/Android-NDK/android-ndk-r10e/

ANDROID\_NDK=/home/suxiaowen/gitsource/Android-NDK/android-ndk-r10e/

export ANDROID\_NDK

nasm/yasm not found or too old. Use --disable-x86asm for a crippled build.

安装yasm

curl http://www.tortall.net/projects/yasm/releases/yasm-1.3.0.tar.gz >yasm.tar.gz

tar zxvf yasm-1.3.0.tar.gz

cd yasm-1.3.0/

./configure --prefix=/usr/local/yasm

make

make install

/etc/profile 文件末尾添加export PATH="$PATH:/usr/local/yasm/bin"

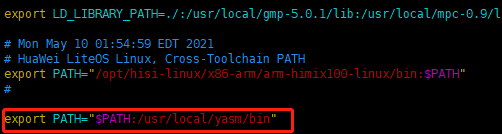
ijk环境搭建完成，接口在ijkplayer/android/ijkplayer/ijkplayer-example

源码添加

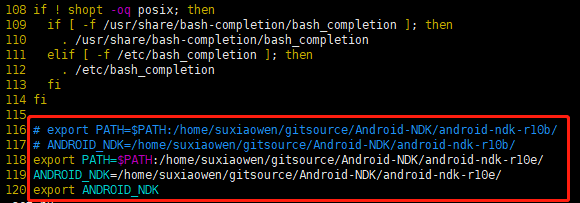
ijkplayer-example ijkplayer-java

<https://www.jianshu.com/p/32e2045df7ed>

/etc/profile



.bashrc



### 编译

cd config

rm module.sh

ln -s module-default.sh module.sh

cd android/contrib

# cd ios

sh compile-ffmpeg.sh clean

./compile-ffmpeg.sh clean

./compile-ffmpeg.sh all

./compile-openssl.sh clean

./compile-openssl.sh all

./compile-ijk.sh all

添加export COMMON\_FF\_CFG\_FLAGS="$COMMON\_FF\_CFG\_FLAGS --disable-linux-perf"到module-default.sh

单独编译

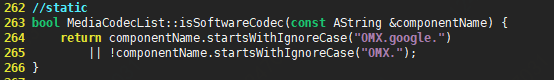
./compile-ffmpeg.sh armv7a

./compile-ijk.sh armv7a

### 硬解处理

https://blog.csdn.net/xmc281141947/article/details/53219537

frameworks/av/media/libstagefright/MediaCodecList.cpp



系统中注册的解码器在如下文件中配置

device/mediatek/mt8167/media\_codecs.xml

需要确认传入的参数

ijkmedia/ijkplayer/android/ijkplayer\_jni.c操作ffmpeg底层接口

音视频学习笔记-附有FFmpeg讲解

<https://blog.csdn.net/leixiaohua1020>

FFmpeg讲解

<https://blog.csdn.net/leixiaohua1020/article/details/12677265>

### example流程

VideoActivity.java

protected void onCreate(Bundle savedInstanceState)

[ijkplayer框架简析 -- 从构造到 onPrepared - 极思路 (joyk.com)](https://www.joyk.com/dig/detail/1552728663367182)

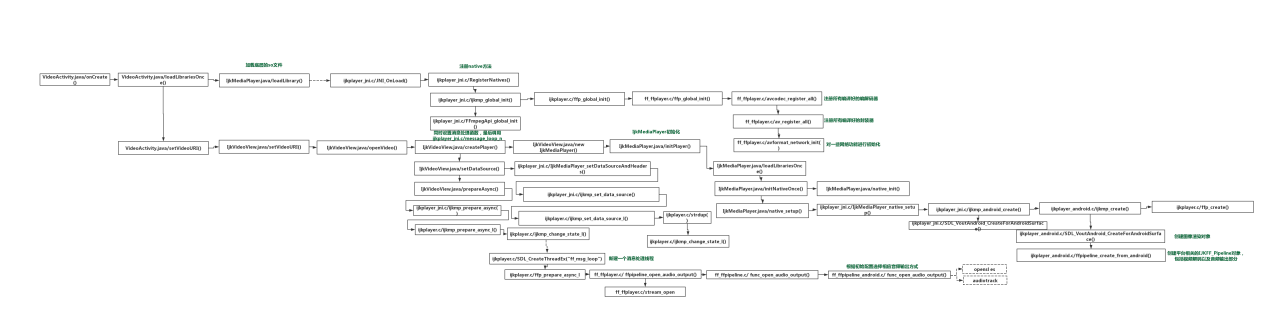
https://www.joyk.com/dig/detail/1552728663367182

[ijkplayer初始化流程\_Michael’s Blog-CSDN博客](https://blog.csdn.net/liuxiaoheng1992/article/details/80502903)

https://blog.csdn.net/liuxiaoheng1992/article/details/80502903

[理解ijkplayer（三）从Java层开始初始化 - 简书 (jianshu.com)](https://www.jianshu.com/p/0501be9cf4bf)

https://www.jianshu.com/p/0501be9cf4bf



### FFmpeg阶段

http://ffmpeg.org/

ffmpeg获取硬解码

ff\_ffplay.c

avcodec\_find\_decoder\_by\_name(forced\_codec\_name);

Decode\_video.c

avcodec\_register\_all();

注册了四种插件

#define REGISTER\_HWACCEL(X, x)硬解码器

#define REGISTER\_ENCODER(X, x)编码器

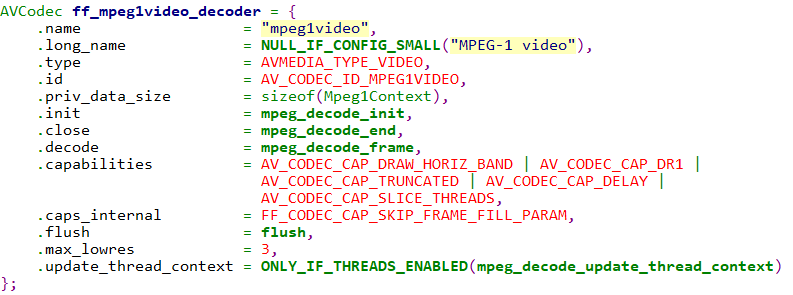
#define REGISTER\_DECODER(X, x)解码器

#define REGISTER\_PARSER(X, x) 解析器

所有编码器、解码器、插件加载到first\_avcodec链表中

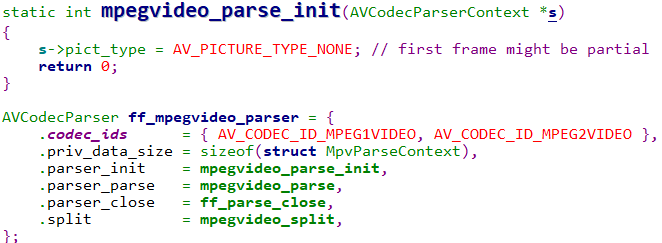
通过宏定义方式加载，ff\_##x##\_decoder -> ff\_mpeg1video\_decoder

ff\_mpeg1video\_decoder



找到解析器

av\_parser\_init(codec->id);



static int stream\_component\_open(FFPlayer \*ffp, int stream\_index)

avcodec\_alloc\_context3(NULL);

avcodec\_register\_all();

av\_packet\_alloc(); //传输介质

avcodec\_alloc\_context3(codec); //获取context，用context进行交互 AVCodecContext

avcodec\_open2(c, codec, NULL)

av\_frame\_alloc();

av\_parser\_parse2(parser, c, &pkt->data, &pkt->size,

data, data\_size, AV\_NOPTS\_VALUE, AV\_NOPTS\_VALUE, 0);

decode()

avcodec\_send\_packet(dec\_ctx, pkt);

While()

avcodec\_receive\_frame(dec\_ctx, frame);

问题：

1. 默认开始播放就使用软解？拖动才会进入软解？
2. 分水岭是在哪个位置？

ffpipenode\_android\_mediacodec\_vdec.c

ffpipenode\_create\_video\_decoder\_from\_android\_mediacodec: MediaCodec: (-99) unknown profile: disabled

ijk封装成pipeline方式，内部切换成硬解还是软解

1. 拖动后音视频同步问题还是拖动后软硬解码问题
2. 如果使用硬解码，为什么要使用ijk，是因为支持的格式多吗
3. 两个日志的版本不同，给我抓的TV版是k0.8.8，JD版k0.7.5，两个版本时间差了两年

[ijkplayer框架深入剖析\_风平浪静的专栏-CSDN博客](https://blog.csdn.net/xipiaoyouzi/article/details/74280170)

https://blog.csdn.net/xipiaoyouzi/article/details/74280170

[ijkplayer 源码分析（上） - 简书 (jianshu.com)](https://www.jianshu.com/p/5345ab4cf979)

https://blog.csdn.net/xipiaoyouzi/article/details/74280170

[BiliBili-IJKPlayer播放器源码分析（一）\_veilling的专栏-CSDN博客\_ijkplayer源码解析](https://blog.csdn.net/veilling/article/details/52711159)

https://blog.csdn.net/veilling/article/details/52711159

猜调用顺序，可以通过日志获取流程

Ijkplayer\_jni.c

IjkMediaPlayer\_native\_setup

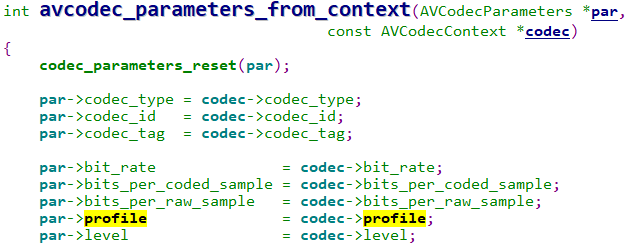
IjkMediaPlayer\_prepareAsync

IjkMediaPlayer\_start

调试方法：

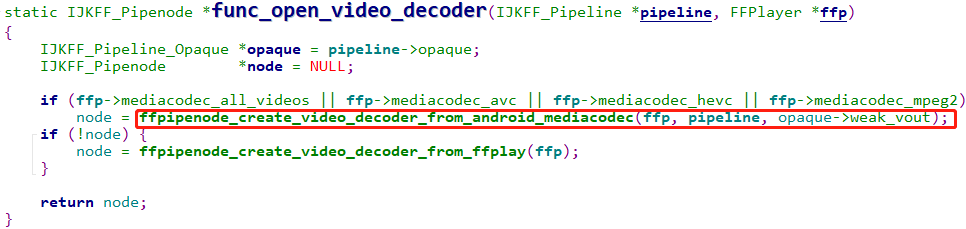
在jni中添加打印，查看调用流程

为了进入硬解码，第一种修改方法，修改codec->profile（可能）



ffpipeline\_create\_from\_android 选择硬解还是软解





### java环境配置

jdk下载地址

账号：[1447149954@qq.com](mailto:1447149954@qq.com)

密码：Sxx@1xxxxx

<https://www.oracle.com/cn/java/technologies/javase/javase-jdk8-downloads.html>

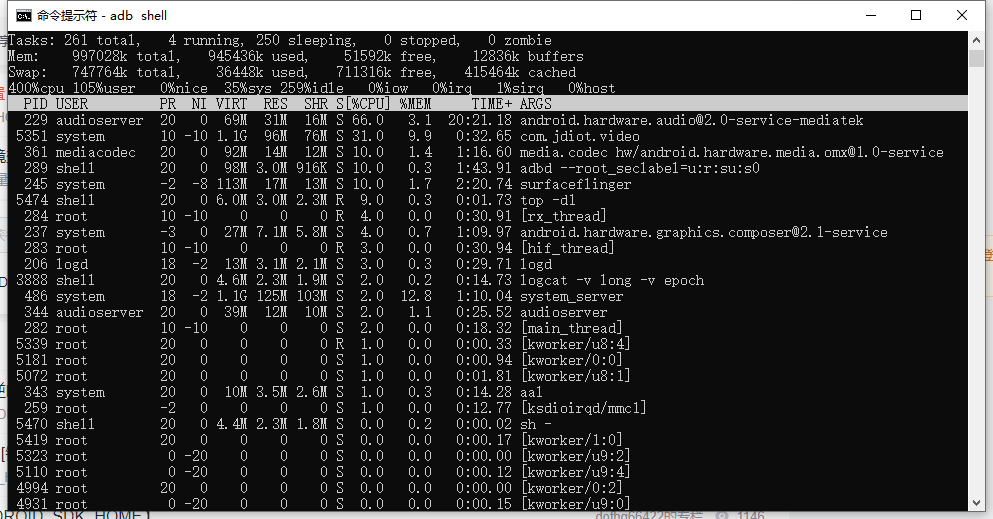
配置jdk jre环境

<https://www.runoob.com/w3cnote/windows10-java-setup.html>

下载android3.5版本

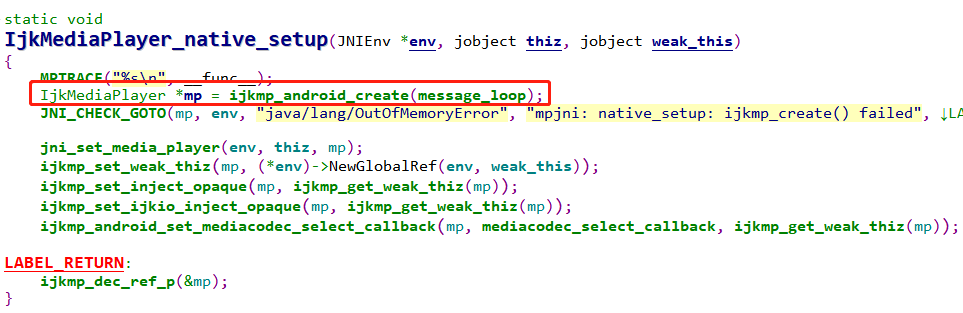
<http://www.android-studio.org/index.php/84-study/>

下载压缩版，启动android-studio\bin\studio64.exe（依赖JAVA\_HOME）



### 调用流程

1.



创建ijkMediaPlayer结构，内部包含ffplayer，并创建结构，初始化msg\_queue链表，用于接受ffmpeg运行中通知的时间。创建IjkMediaMeta，暂不清楚作用。创建SDL（渲染模块，封装跨平台渲染接口），pipeline，并将SDL设置到pipeline中。

2.



接受到app传入和URL，设置到ijkmp中，更新ijkmp中mp\_state状态，并通知ffp

3.

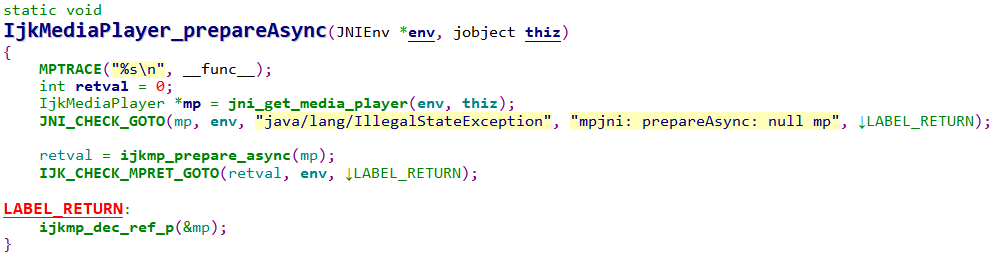


把android surface设置到SDL和ffpipeline中

SDL\_VoutAndroid\_SetAndroidSurface(env, mp->ffplayer->vout, android\_surface);

ffpipeline\_set\_surface(env, mp->ffplayer->pipeline, android\_surface);

4.



设置mp状态MP\_STATE\_ASYNC\_PREPARING

设置audio pipeline ffp->aout = ffpipeline\_open\_audio\_output(ffp->pipeline, ffp);

调用：VideoState \*is = stream\_open(ffp, file\_name, NULL);

创建VideoState结构，创建三个帧队列，分别用于视频、音频、字幕

创建两个线程video\_refresh\_thread、read\_thread

video\_refresh\_thread 视频渲染线程

read\_thread /\* this thread gets the stream from the disk or the network \*/

数据的读取过程都是由ffmpeg内部完成的。



创建AVFormatContext上下文，这里面就存放了probsize（存放探针大小）

初始化VideoState

在AVFormatContext中设置打断回调（异常），回调中设置Videostate->abort\_request。

设置部分format\_opts

avformat\_open\_input()

该函数用于打开多媒体数据并且获得一些相关的信息。它的声明位于libavformat\avformat.h

解析

avformat\_find\_stream\_info

该函数主要用于给每个媒体流（音频/视频）的AVStream结构体赋值

选择编码器

ffpipenode\_create\_video\_decoder\_from\_android\_mediacodec

ffpipenode\_create\_video\_decoder\_from\_android\_mediacodec: MediaCodec: H264\_MAIN: enabled

ff\_hevc\_nvenc\_encoder

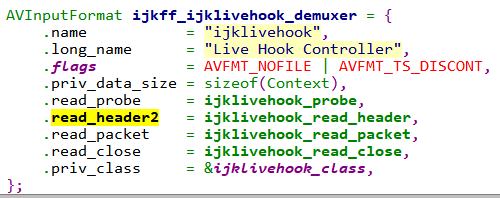
hevc\_nvenc

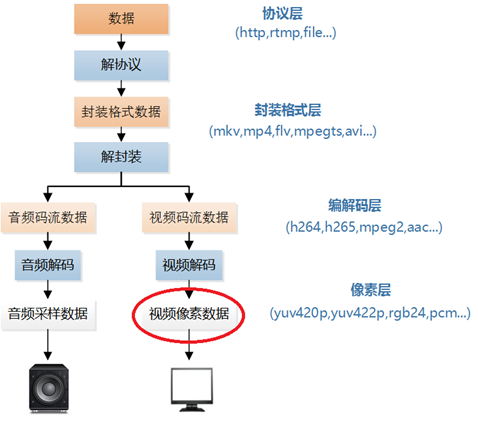
AMediaFormat: video/avc, 1920x1080

probsize宏定义

libavformat/options\_table.h

解码后的数据在ffplay\_video\_thread中送入pictq。





### 参考

ijk播放问题汇总

<https://www.jianshu.com/p/496257563f69>

option解析

<https://www.jianshu.com/p/ab405e5bc1de/>

[ijkplayer 开源项目分析](https://www.jianshu.com/p/026d6071514f" \t "https://www.jianshu.com/p/_blank)

<https://www.jianshu.com/p/23fc56fe23d3>

关键接口分析-重点-详细

<https://blog.csdn.net/leixiaohua1020/article/details/44084321>

TS流

<https://blog.csdn.net/yangguoyu8023/article/details/98451866>

[FFmpeg 开发之 AVFilter 使用流程总结](https://www.cnblogs.com/renhui/p/14663071.html)

https://www.cnblogs.com/renhui/p/14663071.html

流媒体技术基础-流媒体编码与协议

https://www.jianshu.com/p/984f42a71355

FFMPEG 3.4.2 - ffmpeg源代码分析 （一）

https://www.jianshu.com/p/e9ec10350bad

视频：FFmpeg命令源码分析

https://www.zhihu.com/zvideo/1282047820925394944

源码分析 - 多文章

https://blog.csdn.net/fireroll/category\_565707.html

FFmpeg filter 详细解析

https://blog.csdn.net/yue\_huang/article/details/78124370

FFPlayer \*ffp\_create()

msg\_queue\_init(&ffp->msg\_queue);

ffp\_notify\_msg1 各个阶段通过msg上报事件

ffp\_notify\_msg1(ffp, FFP\_MSG\_OPEN\_INPUT);

## Android开发记录点

Bundle

容器，键值对形式，两个activity之间相互传递

setVisibility(View.GONE)

不可见，不赞用资源（类似强制播放？）

接触新项目开发问题

问题：

对于宏定义、注释一定要准确。如 \_\_M11\_\_ 为什么看不到 \_\_M12\_\_的宏定义

定期会议

做项目前要同意规则

404

02-24 11:22:21.721

02-24 11:22:42.329

Yife

Python go

## 问题分析

视频卡顿问题排查：

数据流过大导致收发阻塞，可降低码率，从而减少流量

排查是否有帧或数据丢失

## 产品创建

IOT控制平台：https://smartdev.jd.com/

PIN：never\_deset

**华为摄像头001**

uuid：4C35EF

SecretKey：KRTRZPAZJICTKF9E

PublicKey：0322CAFA6AC1C83FCF02F2E56C3B6861F1A59EBF50AF712DFE399EA903442C5ADB

PrivateKey：D03DDD6C7CD37FFBC4526C79036498E5DEF64DD91C5037D85A2CEEEC7AA1B973

MAC：68B9D3165A3F

## 计划

1、编译链移植完成

2、编译joylinksdk

3、更新tutk lib

4、编译报错，对照demo修改编译错误

5、两个源码环境搭好了，开始比对

6、media\_sdk编译好了 //0415 done

7、先运行一下demo //done

8、开发板环境网络功能调通； //done

9、设备激活注册

10、使用假数据

11、流移植；

将升级失败原因也都梳理出来

包括之前协议升级不同步的问题；

同时包括OTA的整体优化点

### 操作

使用vlc打开h264文件，可以看到图像

发送延时会导致视频流卡顿，可能内部有丢帧处理

如何使用时间戳来确定问题

当前hi3516ev200图像信息

h264 640\*480 FPS=21 size=50K

当前状态，server端可以发送信息数据到client端，局域网链接的状态下显示效果基本无卡顿

当网络断开后，server如何收到事件，如果关闭

二维码可以从真实设备中识别出可用信息

### 未操作

onvif测试 PC安装软件电脑控制

p2p软件测试 设备运行p2p软件，手机安装测试软件，使用ap方式进行直连

rtmp推流测试

### 问题

Rtsp rtmp 区别

cbr

vbr

avbr

cvbr

qvbr

fixQp

这些属于编码格式

Vpss

海思视频处理子系统

高斯噪声去除

贪心算法