全志科技
Allwinner Technology

音视频解码库项目

视频解码库模块概要设计报告

文档履历

版本号	日期	制/修订人	内容描述
V0.1	2013-12-3		建立初稿
V0.2	2014-06-17		1. 修改 4.1.3 InitializeVideoDecoder 函数，该函数增加设置 Config 选项的参数； 2. 删除设置 Config 选项的函数，包括： ConfigHorizonScaleDownRatio() ConfigVerticalScaleDownRatio() ConfigRotation() ConfigThumbnailMode() ConfigOutputPicturePixelFormat() ConfigNoBFrames() ConfigDisable3D() 3. 修改 RotatePicture()函数的说明； 4. 附录部分增加对视频初始化信息的描述； 5. 附录部分增加对视频图像像素格式的说明； 6. 取出 NV12 格式，改为 NV21； 7. 修改 2.3 节模块配置介绍； 8. 删除 2.4 节源码结构介绍；
V0.3	2014-11-18		1. 添加设置 Config 选项的函数： ConfigHorizonScaleDownRatio() ConfigVerticalScaleDownRatio() ConfigSecHorizonScaleDownRatio() ConfigSecVerticalScaleDownRatio() 2. 添加 VConfig 数据结构的成员： nSecHorizonScaleDownRatio nSecVerticalScaleDownRatio
V1.0	2015-05-28		Release 版

目 录

1. 概述

1.1. 编写目的

设计视频解码库的基本框架、内/外部接口、主要数据结构和流程。指导视频解码库的开发、使用和后续维护。

1.2. 适用范围

A40/T3/A80/A83/H3/H8 等各个芯片平台的 Android 系统 SDK 和 Linux SDK。

1.3. 相关人员

开发和维护视频解码库的相关人员。

2. 模块介绍

2.1. 功能介绍

视频解码库是一个提供视频解码功能的库，编译输出的库文件未 libvdecoder.so。基于视频解码库，应用程序可以在全志公司的各个 IC 平台上实现高效的、多格式的视频解码功能。

2.2. 相关术语介绍

Stream：码流，被压缩编码过的视频数据；

Stream Buffer Manager：负责管理 Stream 的程序模块；

SBM：Stream Buffer Manager；

Frame Buffer：存放一帧图像的数据结构，包括存放图像数据的内存空间；

Frame Buffer Manager：负责管理 Frame Buffer 的程序模块；

FBM：Frame Buffer Manager；

VE：芯片中负责解码视频的硬件模块，也叫 VPU；

Video Engine：通过控制 VE 或通过软件代码解码视频码流的程序模块；

PTS：时间戳，表示图像的显示时间，presentation time stamp 的缩写；

Aspect Ratio：图像中一个像素的宽、高比值，和图像的分辨率一起控制显示时图像的宽高比例；

2.3. 模块配置介绍

见文档《LIBRARY 配置指南》。

目前该指南所列配置项中，对本模块造成影响的配置项如下：

配置项	影响
CONFIG_DRAM_INTERFACE	配置目标平台的 DRAM 接口类型，影响 VE 从 DRAM 读写数据的效率。
CONFIG_LOG_LEVEL	配置打印信息输出级别。
CONFIG_ENABLE_VIDEO_DECODER_MJPEG	是否支持 MJPEG 解码
CONFIG_ENABLE_VIDEO_DECODER_MPEG1	是否支持 MPEG1 解码
CONFIG_ENABLE_VIDEO_DECODER_MPEG2	是否支持 MPEG2 解码
CONFIG_ENABLE_VIDEO_DECODER_MPEG4	是否支持 MPEG4 解码
CONFIG_ENABLE_VIDEO_DECODER_MSMPEG4V1	是否支持 MSMPEG4V1 解码

CONFIG_ENABLE_VIDEO_DECODER_MSMPEG4V2	是否支持 MSMPEG4V2 解码
CONFIG_ENABLE_VIDEO_DECODER_DIVX3	是否支持 DIVX3 解码
CONFIG_ENABLE_VIDEO_DECODER_DIVX4	是否支持 DIVX4 解码
CONFIG_ENABLE_VIDEO_DECODER_DIVX5	是否支持 DIVX5 解码
CONFIG_ENABLE_VIDEO_DECODER_XVID	是否支持 XVID 解码
CONFIG_ENABLE_VIDEO_DECODER_H263	是否支持 H263 解码
CONFIG_ENABLE_VIDEO_DECODER_SORENSSON_H263	是否支持 Sorensson H263 解码
CONFIG_ENABLE_VIDEO_DECODER_RXG2	是否支持 RXG2 解码
CONFIG_ENABLE_VIDEO_DECODER_WMV1	是否支持 WMV1 解码
CONFIG_ENABLE_VIDEO_DECODER_WMV2	是否支持 WMV2 解码
CONFIG_ENABLE_VIDEO_DECODER_WMV3	是否支持 WMV3 解码
CONFIG_ENABLE_VIDEO_DECODER_VP6	是否支持 VP6 解码
CONFIG_ENABLE_VIDEO_DECODER_VP8	是否支持 VP8 解码
CONFIG_ENABLE_VIDEO_DECODER_VP9	是否支持 VP9 解码
CONFIG_ENABLE_VIDEO_DECODER_H264	是否支持 H264 解码
CONFIG_ENABLE_VIDEO_DECODER_H265	是否支持 H265 解码

3. 模块体系结构设计

视频解码库由码流管理模块（Stream Buffer Manager）、视频解码引擎（Video Engine）、帧缓冲管理模块（Frame Buffer Manager）以及解码控制模块（vdecoder）组成，如图 2 所示。其中:

Stream Buffer Manager 负责视频码流 Buffer 的管理以及码流数据的管理；

Frame Buffer Manager 负责解码图像 Buffer 的管理；

Video Engine 负责将码流解码成视频图像；

vdecoder 控制解码流程，对外提供解码库的接口函数。

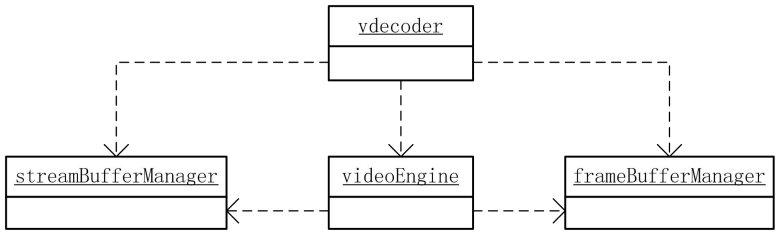


图 2. 视频解码库模块结构图

vdecoder 调用 Video Engine 的解码函数时，Video Engine 从 Stream Buffer Manager 取出码流，从 Frame Buffer Manager 获取空的图像 Buffer，解码码流后输出图像到图像 Buffer。

解码后，Video Engine 将 Stream Buffer 归还 Stream Buffer Manager 管理，将 Frame Buffer 归还 Frame Buffer Manager 管理。

外部程序输入码流时，vdecoder 将码流数据存储在 Stream Buffer Manager 中。外部程序获取输出图像时，vdecoder 从 Frame Buffer Manager 获取。

3.1. Stream Buffer 管理

Stream Buffer Manager 负责管理码流 Buffer 和码流数据。初始化时，该模块申请一片物理连续的内存（4MB~12MB 不等），用于存储视频码流。

Stream Buffer Manager 按照帧结构管理码流，模块内部的 frameFifo 结构体记录了每一笔码流的内存地址、长度、时间戳等信息。各种视频编码标准都定义了数据单元的概念，例如 H264 标准定义的 NALU。Stream Buffer Manager 中的一帧码流，是指包含整数个数据单元的一笔数据。

由于需要被硬件解码模块访问，码流 Buffer 是物理连续的，它被当做循环 Buffer 使用。frameFifo 结构体内部使用一个循环数组记录每笔码流信息，按照先进先出的规则向解码器提供码流数据。

Stream Buffer Manager 的接口和流程设计，在 4.3.1 节描述。

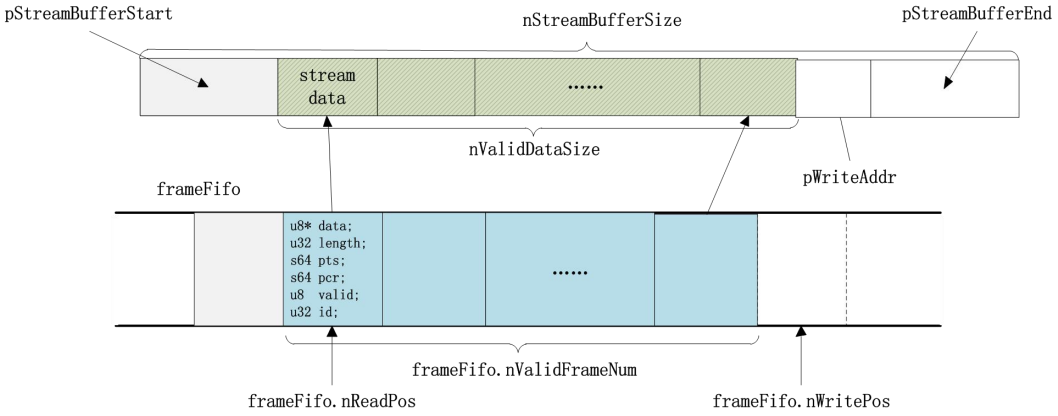


图 3.码流管理模块数据结构设计

3.2. Frame Buffer 管理

Frame Buffer Manager 负责管理图像 Buffer。初始化时，该模块申请指定个数的图像 Buffer，每个图像 Buffer 的信息放在其内部数组 frames[]中。当解码器需要图像 Buffer 时，Frame Buffer Manager 从空 Buffer 队列取出一个 Buffer 给解码器，解码器完成解码后，图像 Buffer 被放入显示队列 pValidPictureQueue 等待显示。当 Render 需要获取图像用于显示时，Frame Buffer Manager 从 pValidPictureQueue 取出一个图像给 Render，Render 完成显示归还 Buffer 时，如果解码器没有在使用该 Buffer，图像 Buffer 被放入 pEmptyBufferQueue。

Frame Buffer Manager 的接口和流程设计，在 4.3.2 节描述。

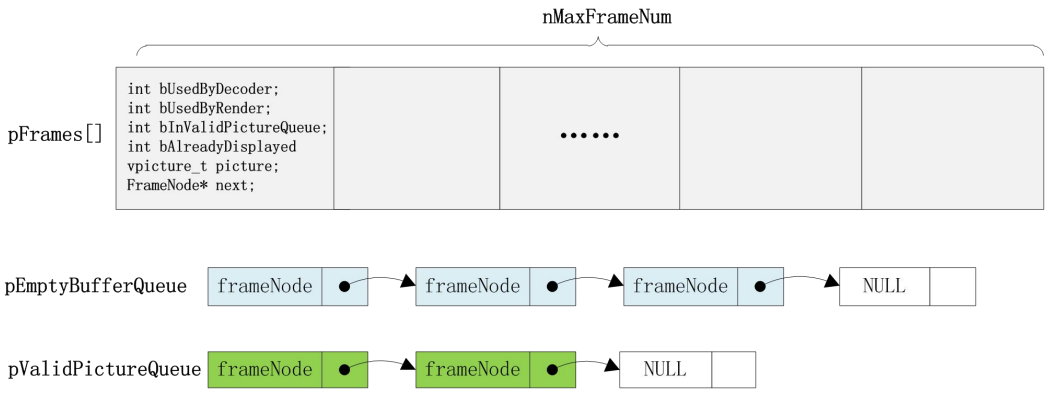


图 4. Frame Buffer 管理模块数据结构设计

3.3. Video Engine

对于多数视频格式，Video Engine 模块使用硬件解码。

如图 5 所示，Video Engine 的上层代码对外提供解码接口，这一层代码负责初始化硬件环境，调用底层 H264 等解码模块处理解码工作。

H264、MPEG4 等解码模块处理码流解码任务，这些模块通过外部提供的 Bitstream 管理模块的接口获取码流，通过 Frame Buffer 管理模块的接口获取图像 Buffer 和输出图像。

Video Engine 模块的适配层负责实现内存申请、调试信息打印、VE 硬件设备驱动调用等功能。Bitstream、Frame Buffer 管理模块跟 Video Engine 对接的代码，也在适配层实现。

Video Engine 模块的接口设计，在 4.3.3 节描述。

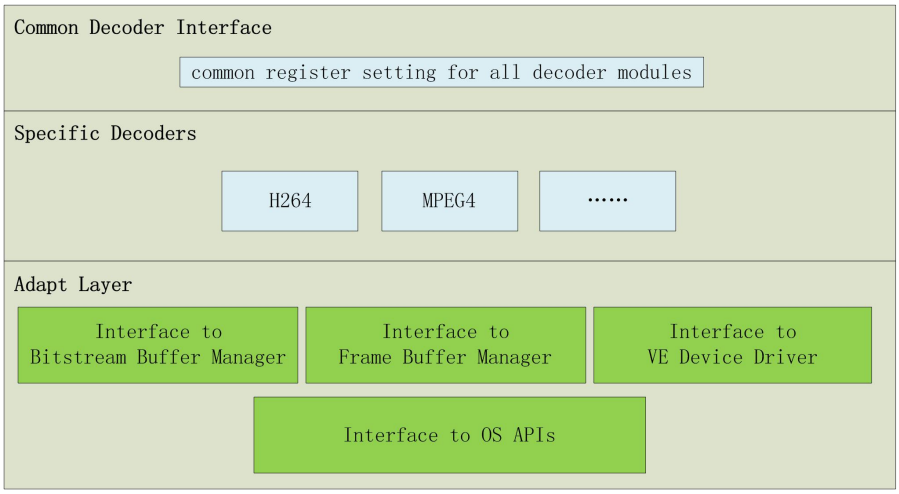


图 5. Video Engine 模块逻辑层次

4. 接口和流程设计

4.1. 视频解码库 API

视频解码库的 API 接口如下表所示。

视频解码库 APIs		
1	CreateVideoDecoder	创建一个视频解码器
2	DestroyVideoDecoder	销毁一个视频解码器，释放资源
3	InitializeVideoDecoder	根据视频码流信息（编码格式等）初始化视频解码器
4	ResetVideoDecoder	重置视频解码器，重置后视频宽高等基本信息仍保留，用于播放器跳播等操作
5	DecodeVideoStream	解码一笔视频码流
6	GetVideoStreamInfo	从解码器获取视频信息
7	RequestVideoStreamBuffer	获取码流 Buffer，用于填充码流数据
8	SubmitVideoStreamData	填充完码流数据后，将数据提交给解码器
9	VideoStreamBufferSize	获取码流缓冲区的大小，以字节为单位
10	VideoStreamDataSize	码流缓冲区内有效（未解码）数据的大小，以字节为单位
11	VideoStreamFrameNum	码流缓冲区内有多少笔有效（未解码）数据
12	RequestPicture	获取视频图像
13	ReturnPicture	归还视频图像
14	NextPictureInfo	获取下一帧视频的信息，如时间戳等信息
15	TotalPictureBufferNum	解码器内总共有多少个视频图像 Buffer
16	EmptyPictureBufferNum	目前空闲的图像 Buffer 数量
17	ValidPictureNum	等待显示的视频图像数量
18	ConfigHorizonScaleDownRatio	设置视频输出图像水平方向缩放倍数，支持 2 倍和 4 倍缩放
19	ConfigVerticalScaleDownRatio	设置视频输出图像垂直方向缩放倍数，支持 2 倍和 4 倍缩放
20	ConfigSecHorizonScaleDownRatio	设置从路通道图像水平方向缩放倍数，支持 2/4/8/16/32 倍缩放
21	ConfigSecVerticalScaleDownRatio	设置从路通道图像垂直方向缩放倍数，支持 2/4/8/16/32 倍缩放
22	ReopenVideoEngine	重新打开 Video Engine 模块，用于支持多分辨率视频流的播放
23	AllocatePictureBuffer	申请一个独立的，指定大小的图像 Buffer
24	FreePictureBuffer	释放由 AllocatePictureBuffer 函数申请的图像 Buffer
25	RotatePicture	旋转图像

开始解码前，应用程序首先调用 CreateVideoDecoder 函数创建一个解码器，然后调用 InitializeVideoDecoder 函数，将视频基本信息作为参数，初始化解码器。

初始化后，解码器可以开始解码视频流。

应用程序通过 RequestVideoStreamBuffer 函数从解码器获取码流 Buffer，将数据填入后，通过 SubmitVideoStreamData 函数将码流提交给解码器。

应用程序通过调用 DecodeVideoStream 函数解码视频码流。

应用程序通过调用 RequestPicture 函数获取视频图像，视频图像显示完毕后，应用程序调用 ReturnPicture 将图像 Buffer 归还解码器。

视频解码库支持多线程操作，码流的传送、解码和视频图像的输出工作可以在不同的线程中进行。一般来说，播放器会有 Demux 线程、视频解码线程和视频渲染（Render）等三个线程处理视频相关的工作。Demux 线程不断调用 RequestVideoStreamBuffer 函数和 SubmitVideoStreamData 函数传送数据；视频解码线程通过调用 DecodeVideoStream 函数解码视频流；视频渲染线程不断调用 RequestPicture 函数获取图像用于显示，调用 ReturnPicture 归还已经显示的图像。

视频解码库还支持图像缩放、旋转等功能，这些功能需要调用其他 API 进行配置。下文详细介绍各个 API 函数。

4.1.1. CreateVideoDecoder

函数原型	VideoDecoder* CreateVideoDecoder(void)
功能	创建一个视频解码器
参数	无
返回值	成功：视频解码器指针； 失败：返回 NULL；
调用说明	视频解码库支持创建多个解码器，同时解码多路视频。

4.1.2. DestroyVideoDecoder

函数原型	void DestroyVideoDecoder(VideoDecoder* pDecoder)
功能	销毁一个视频解码器，释放相关软硬件资源
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针
返回值	无
调用说明	无

4.1.3. InitializeVideoDecoder

函数原型	int InitializeVideoDecoder(VideoDecoder* pDecoder VideoStreamInfo* pVideoInfo, VConfig* pVConfig)
功能	初始化视频解码器
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针 pVideoInfo: 视频码流的基本信息，如编码格式、分辨率、帧率等 pVConfig: 视频解码器配置选项，用于配置旋转、图像缩小、缩略图模式等
返回值	0: 表示成功; -1: 失败，不支持的编码格式或内存资源不足;
调用说明	VideoStreamInfo 中，不是所有信息都是必须的。 对于 H264 和 MPEG2，可以只填写编码格式信息（format），其他视频一般还需要正确填写视频分辨率信息（width 和 height，由于码流中没有该信息）。某些视频解码前需要初始化数据（initData）。不同格式视频对应的初始化信息如何填写，可以参考本文档第 6 节附录部分的“视频初始化信息设置” pVConfig：配置解码器的旋转，缩小等信息 1． bScaleDownEn: 配置缩小输出，取值为 0 或 1， 默认值为 0; 2． bRotationEn：配置旋转输出，取值为 0 或 1， 默认值为 0; 3． nHorizonScaleDownRatio: 视频输出图像水平方向缩小比例，0 表示不缩放，1 表示缩放为 1/2 大小，2 表示缩放为 1/4 大小，3 表示缩放为 1/8 大小; 4． nVerticalScaleDownRatio: 视频输出图像竖直方向缩小比例，0 表示不缩放，1 表示缩放为 1/2 大小，2 表示缩放为 1/4 大小，3 表示缩放为 1/8 大小;（注：缩放比例是针对原始图像设置， 同时进行旋转时需要注意；VP6、WMV1、WMV2 格式的视频不支持 ScaleDown 功能）; 5． nSecHorizonScaleDownRatio: 从通道视频输出图像水平方向缩小比例，0 表示不缩放，1 表示缩放为 1/2 大小，2 表示缩放为 1/4 大小，3 表示缩放为 1/8 大小，4 表示缩放 1/16，5 表示缩放 1/32; 6． nSecVerticalScaleDownRatio: 从通道视频输出图像垂直方向缩小比例，0 表示不缩放，1 表示缩放为 1/2 大小，2 表示缩放为 1/4 大小，3 表示缩放为 1/8 大小，4 表示缩放 1/16，5 表示缩放 1/32; 7． nRotateDegree：视频输出图像的旋转角度，以顺时针方向计算,0 表示不旋转，1 表示 90 度，2 表示 180 度，3 表示 270 度; 8． bThumbnailMode:解码器以缩略图模式工作,当应用程序只是希望解码视频文件的一幅图像作为缩略图显示时，解码器可以只申请一个图像 Buffer，解码输出图像后应用程序关闭解码器，不再继续解码, 取值为 0 或 1， 默认值为 0; 9． eOutputPixelFormat:解码器输出图像的像素格式,像素格式对应的数据存放方式，请参考本文档附录 6.3 节; 10．bNoBFrames:视频源是否没有 B 帧， 取值为 0 或 1(默认值为 0); 11．bDisable3D:不支持 3D 模式，取值 0 或 1(默认值为 0); 12．bSupportedMaf:是否支持 de_interlace 功能，只用于 A20 或 A23 平台;

4.1.4. ResetVideoDecoder

函数原型	void ResetVideoDecoder(VideoDecoder* pDecoder)
功能	重置视频解码器
参数	pDecoder：通过 CreateVideoDecoder 函数创建的视频解码器指针
返回值	无
调用说明	调用本函数后，Video Engine 模块被重置，但视频初始化信息被保留，码流 Buffer 中的数据被清空，图像数据也被清空。 本函数一般用于跳播清除视频解码器数据。

4.1.5. DecodeVideoStream

函数原型	int DecodeVideostream(VideoDecoder* pDecoder, int bEndOfStream, int bDropBFrameIfDelay, int64_t nCurrentTimeUs)
功能	解码一帧图像，解码库会对码流 Buffer 中的码流进行解码
参数	pDecoder：通过 CreateVideoDecoder 函数创建的视频解码器指针； bEndOfStream：是否码流结束，等于 1 表示所有码流数据都已经传送到 Sbm； bDropBFrameIfDelay：码流的时间戳（PTS）比当前时间大时，是否丢弃 B 帧，等于 0 表示不丢弃，等于 1 表示丢弃 B 帧； nCurrentTimeUs：当前时间，用于比较码流是否过时；
返回值	VDECODE_RESULT_FRAME_DECODED(1)：解码成功，输出了一帧图像； VDECODE_RESULT_CONTINUE(2)：码流被解码，但没有图像输出，需继续解码； VDECODE_RESULT_KEYFRAME_DECODED(3)：解码成功，输出了一帧关键帧图像； VDECODE_RESULT_NO_FRAME_BUFFER(4)：当前无法获取到图像 Buffer； VDECODE_RESULT_NO_BITSTREAM(5)：当前无法获取到码流数据； VDECODE_RESULT_RESOLUTION_CHANGE(6)：视频分辨率发生变化，无法继续； VDECODE_RESULT_UNSUPPORTED(-1)：不能支持的格式或申请内存失败，无法继续解码；
调用说明	在解码性能不足的情况下，通过丢弃过时的 B 帧码流，解码器可以追赶视频播放的进度，避免音视频不同步的问题；

4.1.6. GetVideoStreamInfo

函数原型	int GetVideoStreamInfo(VideoDecoder* pDecoder, VideoStreamInfo* pVideoInfo)
功能	获取视频信息
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针 pVideoInfo: 输出参数，存放视频信息；
返回值	0: 表示成功；-1: 失败；
调用说明	除了初始化时设置的信息，解码器会将解码后获得的信息一起输出。

4.1.7. RequestVideoStreamBuffer

函数原型	int RequestVideoStreamBuffer(VideoDecoder* pDecoder, int nRequireSize, char** ppBuf, int* pBufSize, char** ppRingBuf, int* pRingBufSize, int nStreamBufIndex)
功能	向解码器请求存放码流 Buffer，用于存放数据
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针 nRequireSize: 请求 Buffer 的大小，以字节为单位； ppBuf: 输出参数，码流 Buffer 起始地址，等于 NULL 表示失败； pBufSize: 输出参数，码流 Buffer ppBuf 的大小； ppRingBuf: 输出参数，第二块 Buffer 的起始地址，等于 NULL 表示没有； pRingBufSize: 第二块 Buffer ppRingBuf 的大小； nStreamBufIndex: 对于蓝光 MVC 等 3D 视频，解码器需要处理两路码流，nStreamBufIndex 指定从第几路视频码流 Buffer 获取 Buffer，0 表示第 0 路（MVC 主码流）、1 表示第 1 路（MVC 从码流）。
返回值	0: 表示成功；-1: 失败；
调用说明	码流 Buffer 是一块循环 Buffer，当 Buffer 回头时，外部请求的 Buffer 被分成两段，ppBuf 和 pBufSize 返回第一段 Buffer 的地址和大小，ppRingBuf 和 pRingBufSize 返回第二段 Buffer 的地址和大小。 Buffer 没有回头时，ppRingBuf 和 pRingBufSize 返回 NULL。

4.1.8. SubmitVideoStreamData

函数原型	int SubmitVideoStreamData(VideoDecoder* pDecoder, VideoStreamDataInfo* pDataInfo, int nStreamBufIndex)
功能	向解码器提交码流数据
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针; pDataInfo: 码流数据信息，包括地址、长度、时间戳、边界信息等; nStreamBufIndex: 对于蓝光 MVC 等 3D 视频，解码器需要处理两路码流，nStreamBufIndex 指定从第几路视频码流 Buffer 获取 Buffer，0 表示第 0 路（MVC 主码流）、1 表示第 1 路（MVC 从码流）。
返回值	0: 表示成功; -1: 失败;
调用说明	提交数据时，数据可以是一笔包含整数个数据单元的完整码流帧，也可以只包含一个数据单元的部分数据，只需将 VideoStreamDataInfo 结构体中的两个表示数据边界信息的变量正确填写即可。两个边界信息变量为 bIsFirstPart: 表示该笔数据第一个字节是否是一个数据单元的开始; bIsLastPart: 表示该笔数据最后一个有效字节是否是一个数据单元的结束;

4.1.9. VideoStreamBufferSize

函数原型	int VideoStreamBufferSize(VideoDecoder* pDecoder, int nStreamBufIndex)
功能	查询码流 Buffer 的总大小，单位为字节
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针; nStreamBufIndex: 对于蓝光 MVC 等 3D 视频，解码器需要处理两路码流，nStreamBufIndex 指定从第几路视频码流 Buffer 获取 Buffer，0 表示第 0 路（MVC 主码流）、1 表示第 1 路（MVC 从码流）。
返回值	码流 Buffer 的大小，单位为字节。
调用说明	在解码器初始化后才能正确返回码流 Buffer 的大小，否则返回 0.

4.1.10. VideoStreamDataSize

函数原型	int VideoStreamDataSize(VideoDecoder* pDecoder, int nStreamBufIndex)
功能	查询码流 Buffer 中有效数据（未解码的数据）大小，单位为字节
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针; nStreamBufIndex: 对于蓝光 MVC 等 3D 视频，解码器需要处理两路码流，nStreamBufIndex 指定从第几路视频码流 Buffer 获取 Buffer。
返回值	码流 Buffer 中未解码的数据量，单位为字节。
调用说明	

4.1.11. VideoStreamFrameNum

函数原型	int VideoStreamFrameNum (VideoDecoder* pDecoder, int nStreamBufIndex)
功能	查询码流 Buffer 中有效数据（未解码的数据）有多少帧；
参数	pDecoder：通过 CreateVideoDecoder 函数创建的视频解码器指针； nStreamBufIndex：对于蓝光 MVC 等 3D 视频，解码器需要处理两路码流，nStreamBufIndex 指定从第几路视频码流 Buffer 获取 Buffer，0 表示第 0 路（MVC 主码流）、1 表示第 1 路（MVC 从码流）。
返回值	码流 Buffer 中未解码的数据有多少帧。
调用说明	

4.1.12. RequestPicture

函数原型	VideoPicture* RequestPicture(VideoDecoder* pDecoder, int nStreamIndex)
功能	获取一帧图像
参数	pDecoder：通过 CreateVideoDecoder 函数创建的视频解码器指针； nStreamIndex：对于蓝光 MVC 等 3D 视频，解码器有两路视频可供显示，nStreamIndex 指定从获取第几路视频的图像。
返回值	成功：返回图像 Buffer 指针； 失败：返回 NULL；
调用说明	

4.1.13. ReturnPicture

函数原型	int ReturnPicture(VideoDecoder* pDecoder, VideoPicture* pPicture)
功能	归还通过 RequestPicture 获取的视频图像给解码器
参数	pDecoder：通过 CreateVideoDecoder 函数创建的视频解码器指针； pPicture：通过 RequestPicture 函数获取的图像 Buffer；
返回值	0： 表示成功；-1：失败；
调用说明	

4.1.14. NextPictureInfo

函数原型	VideoPicture* NextPictureInfo(VideoDecoder* pDecoder,

	int nStreamIndex)
功能	获取下一帧输出图像的信息
参数	pDecoder：通过 CreateVideoDecoder 函数创建的视频解码器指针； nStreamIndex：对于蓝光 MVC 等 3D 视频，解码器有两路视频可供显示，nStreamIndex 指定从获取第几路视频的图像。
返回值	成功：返回下一帧待显示图像 Buffer 的指针； 失败：返回 NULL；
调用说明	与 RequestPicture 函数相比，本函数只是获取视频图像的信息，不会使该图像从解码器的显示队列中删除。

4.1.15. TotalPictureBufferNum

函数原型	int TotalPictureBufferNum(VideoDecoder* pDecoder, int nStreamIndex)
功能	询问解码器内总共有多少个图像 Buffer
参数	pDecoder：通过 CreateVideoDecoder 函数创建的视频解码器指针； nStreamIndex：对于蓝光 MVC 等 3D 视频，解码器有两路视频可供显示，nStreamIndex 指定从获取第几路视频的图像。
返回值	图像 Buffer 个数
调用说明	某些视频格式(H264 和 MPEG2)需要解码部分码流(SPS/PPS、Sequence Header)信息后才申请图像 Buffer，在此之前，本函数返回 0。

4.1.16. EmptyPictureBufferNum

函数原型	int EmptyPictureBufferNum(VideoDecoder* pDecoder, int nStreamIndex)
功能	询问解码器内有多少个空闲的图像 Buffer，即未被解码器和显示占用的图像 Buffer 个数。 nStreamIndex：对于蓝光 MVC 等 3D 视频，解码器有两路视频可供显示，nStreamIndex 指定从获取第几路视频的图像。
参数	pDecoder：通过 CreateVideoDecoder 函数创建的视频解码器指针；
返回值	空闲图像 Buffer 个数
调用说明	

4.1.17. ValidPictureNum

函数原型	int ValidPictureNum(VideoDecoder* pDecoder, int nStreamIndex)

功能	询问解码器内显示队列中有多少个待显示的图像。
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针; nStreamIndex: 对于蓝光 MVC 等 3D 视频，解码器有两路视频可供显示，nStreamIndex 指定从获取第几路视频的图像。
返回值	待显示的图像个数
调用说明	

4.1.18. ConfigHorizonScaleDownRatio

函数原型	int ConfigHorizonScaleDownRatio(VideoDecoder* pDecoder, int nScaleDownRatio)
功能	设置图像水平方向缩放比例因子。
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针; nScaleDownRatio: 图像缩放比例因子。
返回值	0: 表示成功; -1: 失败;
调用说明	

4.1.19. ConfigVerticalScaleDownRatio

函数原型	int ConfigVerticalScaleDownRatio(VideoDecoder* pDecoder, int nScaleDownRatio)
功能	设置图像垂直方向缩放比例因子。
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针; nScaleDownRatio: 图像缩放比例因子。
返回值	0: 表示成功; -1: 失败;
调用说明	

4.1.20. ConfigSecHorizonScaleDownRatio

函数原型	int ConfigSecHorizonScaleDownRatio(VideoDecoder* pDecoder, int nScaleDownRatio)
功能	设置从路通道图像水平方向缩放比例因子。
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针; nScaleDownRatio: 图像缩放比例因子。
返回值	0: 表示成功; -1: 失败;
调用说明	当从路通道缩放比例需求与主路通道不同时，可通过调用此接口进行设置;

4.1.21. ConfigSecVerticalScaleDownRatio

函数原型	int ConfigSecVerticalScaleDownRatio(VideoDecoder* pDecoder, int nScaleDownRatio)
功能	设置从路通道图像垂直方向缩放比例因子。
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针; nScaleDownRatio: 图像缩放比例因子。
返回值	0: 表示成功; -1: 失败;
调用说明	当从路通道缩放比例需求与主路通道不同时，可通过调用此接口进行设置;

4.1.22. ReopenVideoEngine

函数原型	int ReopenVideoEngine (VideoDecoder* pDecoder)
功能	重新打开解码器内 Video Engine 模块
参数	pDecoder: 通过 CreateVideoDecoder 函数创建的视频解码器指针;
返回值	0: 表示成功; -1: 失败;
调用说明	如果视频分辨率发生改变，DecodeVideoStream 函数会返回对应的值 VDECODE_RESULT_RESOLUTION_CHANGE，并将码流归还到码流 Buffer; 此时应用应该调用本函数重新打开 Video Engine 模块; 重新打开 Video Engine 模块会导致图像 Buffer 被释放，图像 Buffer 管理模块重新初始化。这一点显示控制需要注意。

4.1.23. AllocatePictureBuffer

函数原型	VideoPicture* AllocatePictureBuffer (int nWidth, int nHeight, int nLineStride, int ePixelFormat)
功能	申请一个图像 Buffer。
参数	nWidth: 图像像素宽度; nHeight: 图像像素高度; nLineStride: 图像在内存中存放的行宽，以像素为单位; ePixelFormat: 图像像素格式;
返回值	0: 表示成功; -1: 失败;
调用说明	本函数独立于 VideoDecoder 模块，是全局函数。

4.1.24. FreePictureBuffer

函数原型	int FreePictureBuffer (VideoPicture* pPicture)
功能	释放一个由 AllocatePictureBuffer 函数申请的图像 Buffer。
参数	pPicture: 通过 AllocatePictureBuffer 函数申请的图像 Buffer;

返回值	成功：返回图像 Buffer 指针； 失败：返回 NULL
调用说明	本函数独立于 VideoDecoder 模块，是全局函数。

4.1.25. RotatePicture

函数原型	int RotatePicture (VideoDecoder* pDecoder, VideoPicture* pPictureIn, VideoPicture* pPictureOut, int nRotateDegree)
功能	把图像 pPictureIn 旋转输出到图像 Buffer pPictureOut
参数	pDecoder：通过 CreateVideoDecoder 函数创建的视频解码器指针； pPictureIn：输入图像； pPictureOut：输出图像； nRotateDegree：顺时针旋转角度，0 表示不旋转、1 表示旋转 90 度、2 表示 180 度、3 表示 270 度；
返回值	0：表示成功； -1：表示失败；
调用说明	除旋转图像外，本函数还支持像素格式的转换，输出像素格式由 pPictureOut 图像的 ePixelFormat 参数指定； 目前已经支持的像素格式转换有： (1) PIXEL_FORMAT_YUV_MB32_420 转为 PIXEL_FORMAT_YV12 (2) PIXEL_FORMAT_YUV_MB32_422 转为 PIXEL_FORMAT_YV12 (3) PIXEL_FORMAT_YUV_MB32_420 转为 PIXEL_FORMAT_NV21 (4) PIXEL_FORMAT_YUV_MB32_422 转为 PIXEL_FORMAT_NV21 (5) PIXEL_FORMAT_YV12 转为 PIXEL_FORMAT_NV21 (6) PIXEL_FORMAT_NV21 转为 PIXEL_FORMAT_YV12

4.2. 流程设计

4.2.1. 码流数据传输流程

码流数据的传输通过 RequestVideoStreamBuffer 和 SubmitVideoStreamData 两个 API 函数完成。

RequestVideoStreamBuffer 函数从 Stream Buffer Manager 获取 Buffer 给外部程序，SubmitVideoStreamData 函数将外部提交的数据送入 Stream Buffer Manager。

4.2.2. 解码流程

解码工作通过 DecodeVideoStream 这个 API 函数完成。

4.2.3. 视频图像输出流程

视频图像的传送通过 RequestPicture 和 ReturnPicture 两个 API 函数完成。

RequestPicture 函数从 Frame Buffer Manager 模块取出一帧待显示的图像，送给外部程序。

ReturnPicture 函数将图像归还给 Frame Buffer Manager，使解码器可以继续使用该图像 Buffer 解码新的图像。

4.3. 内部模块接口设计

4.3.1. Stream Buffer Manager 模块接口设计

码流管理模块接口如下表所示。

Stream Buffer Manager 模块接口		
1	SbmCreate	创建码流管理模块
2	SbmDesdroy	销毁码流管理模块，释放内存
3	SbmReset	重置码流管理模块，恢复到刚创建时的状态
4	SbmBufferAddress	获取码流 Buffer 的起始地址（基地址）
5	SbmBufferSize	获取码流 Buffer 的大小（整个 Buffer 的大小）
6	SbmStreamFrameNum	查询码流 Buffer 中未解码的数据有多少帧；
7	SbmStreamDataSize	查询码流 Buffer 中未解码的数据有多少，以字节为单位
8	SbmRequestBuffer	获取一块指定大小的 Buffer
9	SbmAddStream	提交一帧码流给码流管理模块
10	SbmRequestStream	获取一帧码流，解码模块调用
11	SbmReturnStream	把码流归还给码流管理模块，解码模块把未解码的码流归还
12	SbmFlushStream	刷除一帧码流，解码模块把已经解码的码流刷除

4.3.1.1. SbmCreate 函数

函数原型	Sbm* SbmCreate(int nBufferSize)
功能	创建码流 Buffer 管理模块，指定 Buffer 大小为 nBufferSize 字节
参数	nBufferSize：码流 Buffer 大小；
返回值	成功：返回码流管理模块指针； 失败：返回 NULL；
调用说明	指定码流 Buffer 的大小一般在 4MB 至 12MB 之间

4.3.1.2. SbmDestroy 函数

函数原型	void SbmDestroy(Sbm* pSBM)
功能	销毁码流管理模块，释放相关软内存资源
参数	pSBM：通过 SbmCreate 函数创建的码流管理模块指针
返回值	无
调用说明	

4.3.1.3. SbmReset 函数

函数原型	void SbmReset(Sbm* pSBM)
功能	重置码流管理模块
参数	pSBM: 通过 SbmCreate 函数创建的码流管理模块指针;
返回值	无
调用说明	重置后，码流管理模块恢复刚创建时的状态，码流数据被丢弃

4.3.1.4. SbmBufferAddress 函数

函数原型	void* SbmBufferAddress (Sbm* pSBM)
功能	获取码流 Buffer 的基地址
参数	pSBM: 通过 SbmCreate 函数创建的码流管理模块指针;
返回值	码流 Buffer 的起始地址
调用说明	

4.3.1.5. SbmBufferSize 函数

函数原型	int SbmBufferSize (Sbm* pSBM)
功能	获取码流 Buffer 的大小，以字节为单位
参数	pSBM: 通过 SbmCreate 函数创建的码流管理模块指针;
返回值	返回码流 Buffer 的大小，以字节为单位
调用说明	码流 Buffer 的大小在模块创建时指定

4.3.1.6. SbmStreamFrameNum 函数

函数原型	int SbmStreamFrameNum(Sbm* pSBM)
功能	获取码流 Buffer 中有多少帧未解码的数据
参数	pSBM: 通过 SbmCreate 函数创建的码流管理模块指针;
返回值	返回未解码的码流帧数
调用说明	

4.3.1.7. SbmStreamDataSize 函数

函数原型	int SbmStreamDataSize(Sbm* pSBM)
功能	获取码流 Buffer 中未解码数据的总量，以字节为单位
参数	pSBM: 通过 SbmCreate 函数创建的码流管理模块指针；
返回值	返回未解码数据的总量
调用说明	

4.3.1.8. SbmRequestBuffer 函数

函数原型	int SbmRequestBuffer(Sbm* pSBM, int nRequireSize, char** ppBuf, int* pBufSize)
功能	请求一块码流 Buffer
参数	pSBM: 通过 SbmCreate 函数创建的码流管理模块指针； nRequireSize: 需要 Buffer 的大小，以字节为单位； ppBuf: 输出参数，保存 Buffer 的地址； pBufSize: 输出参数，保存 Buffer 的大小，以字节为单位；
返回值	0: 表示成功；-1: 失败；
调用说明	由于码流 Buffer 是一块循环 Buffer，所以 ppBuf 返回的 Buffer 有可能会跨越 Buffer 的末尾，外部需根据码流 Buffer 的基地址、Buffer 大小和本函数返回的 Buffer 地址和大小来判断 Buffer 是否回头。如果 Buffer 回头，外部程序使用时应该把该 buffer 分成两块使用。 例如，码流 Buffer 创建时指定大小为 4MB（0x400000），码流 Buffer 的基地址（通过 SbmBufferAddress 函数得到）为 0x80000000，调用本函数时，ppBuf 返回的地址(*ppBuf)为 0x803FF000，pBufSize 返回的 Buffer 大小（*pBufSize）为 32KB(0x2000)，此时本函数给出的 Buffer 是回头的，被分成两段，第一段起始地址为 0x803FF000，大小为 16KB(0x1000)，第二段起始地址为 0x80000000，大小为 16KB（0x1000），外部程序应该注意这一点。

4.3.1.9. SbmAddStream 函数

函数原型	int SbmAddStream(Sbm* pSBM, VideoStreamDataInfo* pDataInfo)
功能	提交一帧码流给码流管理模块
参数	pSBM: 通过 SbmCreate 函数创建的码流管理模块指针； pDataInfo: 码流数据信息，包括时间戳、地址、长度等；
返回值	0: 表示成功；-1: 失败；
调用说明	pDataInfo 包含的数据应该是完整的一帧数据，包含整数个数据单元。 也就是说 pDataInfo 中的 bIsFirstPart 和 bIsLastPart 都应该等于 1。

4.3.1.10. SbmRequestStream 函数

函数原型	VideoStreamDataInfo* SbmRequestStream(Sbm* pSBM)
功能	从码流管理模块过去一帧码流
参数	pSBM：通过 SbmCreate 函数创建的码流管理模块指针；
返回值	成功：返回一帧码流信息的指针； 失败：返回 NULL
调用说明	码流按照先进先出的规则给出； 本函数一般是 Video Engine 获取码流时调用。

4.3.1.11. SbmReturnStream 函数

函数原型	int SbmReturnStream(Sbm* pSBM, VideoStreamDataInfo* pDataInfo)
功能	归还未解码的一帧码流给码流管理
参数	pSBM：　　通过 SbmCreate 函数创建的码流管理模块指针； pDataInfo：通过 SbmRequestStream 函数获取的一帧码流
返回值	0：　表示成功；-1：失败；
调用说明	本函数一般是 Video Engine 归还未解码的码流时调用； 调用本函数后，码流会按照原来的顺序重新放入管理模块。

4.3.1.12. SbmFlushStream 函数

函数原型	int SbmFlushStream(Sbm* pSBM, VideoStreamDataInfo* pDataInfo)
功能	刷除一帧码流
参数	pSBM：　　通过 SbmCreate 函数创建的码流管理模块指针； pDataInfo：通过 SbmRequestStream 函数获取的一帧码流
返回值	0：　表示成功；-1：失败；
调用说明	本函数一般是 Video Engine 解码完一帧码流后，刷除该码流时调用； 调用本函数后，码流会从管理模块中刷除，内存空间被腾出，用于存放新的码流数据。

4.3.2. Frame Buffer Manager 模块接口设计

图像 Buffer 管理模块接口如下表所示。

Frame Buffer Manager 模块接口		
1	FbmCreate	创建图像 Buffer 管理模块
2	FbmDestroy	销毁图像 Buffer 管理模块
3	FbmFlush	将待显示队列中的帧都刷除，放入空帧队列
4	FbmGetBufferInfo	获取图像 Buffer 的信息，包括分辨率、像素格式、Line Stride 等
5	FbmTotalBufferNum	询问总共有多少个图像 Buffer
6	FbmEmptyBufferNum	询问目前有多少个空闲的图像 Buffer
7	FbmValidPictureNum	询问有多少帧图像待显示
8	FbmRequestBuffer	获取一个图像 Buffer
9	FbmReturnBuffer	归还一个图像 Buffer，解码器解码完后输出图像时调用
10	FbmShareBuffer	解码器与图像管理模块共享一个图像 Buffer，解码器解码完后输出图像，但该图像同时需要保留在解码器内作为参考帧时使用
11	FbmRequestPicture	获取一个输出图像
12	FbmReturnPicture	归还一个图像，图像显示完后被归还，图像 Buffer 被回收用于解码新图像
13	FbmNextPictureInfo	获取下一帧图像的信息，与 FbmRequestPicture 类似，但不会使图像从显示队列中删除。
14	FbmAllocatePictureBuffer	为一个图像结构体对象申请数据 Buffer 空间。
15	FbmFreePictureBuffer	释放一个图像对象的数据 Buffer 空间。

4.3.2.1. FbmCreate 函数

函数原型	Fbm* FbmCreate(int nFrameNum, int nWidth, int nHeight, int nLineStride, int ePixelFormat, int bThumbnailMode)
功能	创建图像 Buffer 管理模块，指定图像大小、像素格式和 Buffer 个数
参数	nFrameNum：图像 Buffer 个数； nWidth：图像像素宽度； nHeight：图像像素高度； nLineStride：图像在内存中存放的行宽，以像素为单位； ePixelFormat：图像像素格式； bThumbnailMode：是否是解码缩略图模式；

返回值	成功：返回图像 Buffer 管理模块指针； 失败：返回 NULL；
调用说明	本函数由 Video Engine 模块调用。 bThumbnailMode 等于 1 表示解码缩略图模式，此时 FBM 模块申请 nFrameNum 个图像 Buffer 对象，但每个图像 Buffer 对象的数据指针都指向一个内存地址，实际上 FBM 只申请了一个图像大小的内存。这种做法可以节省缩略图解码时的内存使用，使应用程序可以同时解码多个视频文件的缩略图，加快缩略图生成的速度。 ePixelFormat 指定不同的图像像素格式，解码器目前支持 YUVPlaner、YV12、以及全志自定义的 32x32 宏块排列方式。ePixelFormat 可以是下面几个值中的一个： PIXEL_FORMAT_YUV_PLANER_420 PIXEL_FORMAT_YUV_PLANER_422 PIXEL_FORMAT_YUV_PLANER_444 PIXEL_FORMAT_YV12 PIXEL_FORMAT_NV21 PIXEL_FORMAT_YUV_MB32_420 PIXEL_FORMAT_YUV_MB32_422 PIXEL_FORMAT_YUV_MB32_444 每种像素格式对应的数据存放方式，请参考本文档附录 6.3 节。

4.3.2.2. FbmDestroy 函数

函数原型	void FbmDestroy(Fbm* pFbm)
功能	销毁图像 Buffer 管理模块，释放内存资源
参数	pFBM: 通过 FbmCreate 函数创建的图像 Buffer 管理模块指针；
返回值	无
调用说明	本函数由 Video Engine 模块调用。

4.3.2.3. FbmFlush 函数

函数原型	void FbmFlush(Fbm* pFbm)
功能	将待显示队列中的帧都刷除
参数	pFBM: 通过 FbmCreate 函数创建的图像 Buffer 管理模块指针；
返回值	无
调用说明	一般在播放器跳播时使用

4.3.2.4. FbmGetBufferInfo 函数

函数原型	int FbmGetBufferInfo(Fbm* pFbm , VideoPicture* pVPicture)
功能	获取图像 Buffer 的信息，包括分辨率、像素格式、Line Stride 等
参数	pFBM：通过 FbmCreate 函数创建的图像 Buffer 管理模块指针； pVPicture：输出参数，存放图像信息；
返回值	0：表示成功； -1：表示失败；
调用说明	指定码流 Buffer 的大小一般在 4MB 至 12MB 之间

4.3.2.5. FbmTotalBufferNum 函数

函数原型	int FbmTotalBufferNum(Fbm* pFbm)
功能	查询总共有多少个图像 Buffer
参数	pFBM：通过 FbmCreate 函数创建的图像 Buffer 管理模块指针；
返回值	图像 Buffer 的数量。
调用说明	

4.3.2.6. FbmEmptyBufferNum 函数

函数原型	int FbmEmptyBufferNum(Fbm* pFbm)
功能	查询目前有多少个空闲的图像 Buffer
参数	pFBM：通过 FbmCreate 函数创建的图像 Buffer 管理模块指针；
返回值	空闲图像 Buffer 的数量
调用说明	空闲图像 Buffer 是指未被解码器占用，不在待显示队列，也不是被外部取出用于显示的图像 Buffer。

4.3.2.7. FbmValidPictureNum 函数

函数原型	int FbmValidPictureNum(Fbm* pFbm)
功能	询问有多少帧图像待显示
参数	pFBM：通过 FbmCreate 函数创建的图像 Buffer 管理模块指针；
返回值	等待显示的图像 Buffer 数量
调用说明	本函数返回 FBM 模块内待显示队列中图像 Buffer 的数量。

4.3.2.8. FbmRequestBuffer 函数

函数原型	VideoPicture* FbmRequestBuffer(Fbm* pFbm)
功能	请求一个空闲的图像 Buffer
参数	pFBM：通过 FbmCreate 函数创建的图像 Buffer 管理模块指针；
返回值	成功：返回图像 Buffer 指针； 失败：返回 NULL；
调用说明	本函数一般由 Video Engine 模块调用。

4.3.2.9. FbmReturnBuffer 函数

函数原型	void FbmReturnBuffer(Fbm* pFbm, VideoPicture* pVPicture, int bValidPicture)
功能	归还一个空闲的图像 Buffer 或一帧有效的图像
参数	pFBM：通过 FbmCreate 函数创建的图像 Buffer 管理模块指针； pVPicture：通过 FbmRequestBuffer 函数请求的图像 Buffer； bValidPicture：指明 pVPicture 是否是一帧有效图像，等于 0 表示该图像无效，作为空闲图像 Buffer 归还，等于 1 表示该图像有效，应该放入待显示队列。
返回值	无
调用说明	本函数一般由 Video Engine 模块调用。

4.3.2.10. FbmShareBuffer 函数

函数原型	void FbmShareBuffer(Fbm* pFbm, VideoPicture* pVPicture)
功能	解码器与显示队列分享一个图像。
参数	pFBM：通过 FbmCreate 函数创建的图像 Buffer 管理模块指针； pVPicture：通过 FbmRequestBuffer 函数请求的图像 Buffer；
返回值	无
调用说明	某些图像解码后已经可以输出显示，但解码器仍然需要使用该图像作为后续图像的参考帧。本函数一般由 Video Engine 模块调用。

4.3.2.11. FbmRequestPicture 函数

函数原型	VideoPicture* FbmRequestPicture(Fbm* pFbm)
功能	请求一个有效的图像 Buffer
参数	pFBM：通过 FbmCreate 函数创建的图像 Buffer 管理模块指针；
返回值	成功：返回图像 Buffer 指针； 失败：返回 NULL；
调用说明	本函数一般用于取出图像显示。

4.3.2.12. FbmReturnPicture 函数

函数原型	int FbmReturnPicture(Fbm* pFbm,VideoPicture* pPicture)
功能	归还一帧图像
参数	pFbm：通过 FbmCreate 函数创建的图像 Buffer 管理模块指针； pPicture：通过 FbmRequestPicture 函数获取的图像 Buffer；
返回值	0：表示成功； -1：表示失败；
调用说明	图像归还后，将被放入空闲 Buffer 队列，用于解码。

4.3.2.13. FbmNextPictureInfo 函数

函数原型	VideoPicture* FbmNextPictureInfo(Fbm* pFbm)
功能	获取下一帧待显示图像的信息
参数	pFBM：通过 FbmCreate 函数创建的图像 Buffer 管理模块指针；
返回值	成功：返回图像 Buffer 指针； 失败：返回 NULL；
调用说明	本函数返回待显示队列中第一帧图像的信息，但不会把该图像从队列中取出。

4.3.2.14. FbmAllocatePictureBuffer 函数

函数原型	int FbmAllocatePictureBuffer (VideoPicture* pVPicture)
功能	为 pVPicture 申请图像数据的 Buffer 空间；
参数	pVPicture：需要申请数据 Buffer 空间的图像，pVPicture 内的 nWidth、nHeight 等变量指明图像的分辨率和像素格式；
返回值	0：表示成功； -1：表示失败；
调用说明	

4.3.2.15. FbmFreePictureBuffer 函数

函数原型	int FbmFreePictureBuffer(VideoPicture* pVPicture)
功能	释放图像 pVPicture 的数据 Buffer 空间
参数	pVPicture：需要释放数据 Buffer 空间的图像
返回值	0：表示成功； -1：表示失败；
调用说明	

4.3.3. Video Engine 模块接口

Video Engine 模块接口如下表所示。

Video Engine 模块接口		
1	VideoEngineCreate	创建一个指定格式、指定配置的视频解码器
2	VideoEngineDestroy	销毁一个视频解码器
3	VideoEngineReset	重置视频解码器，重置后，视频宽度、高度等信息仍保留
4	VideoEngineSetSbm	设置一个码流管理模块给视频解码器
5	VideoEngineGetFbmNum	查询有几个图像 Buffer 管理模块可输出图像，MVC 双流视频的情况下有两路图像输出，一般情况下等于 1
6	VideoEngineGetFbm	获取一个图像管理模块；
7	VideoEngineDecode	对码流进行解码

4.3.3.1. VideoEngineCreate

函数原型	VideoEngine* VideoEngineCreate(VConfig* pVConfig, VideoStreamInfo* pVideoInfo)
功能	创建一个视频解码器，用于后续的视频解码
参数	pVConfig：视频解码器配置选项，用于配置旋转、图像缩小等； pVideoInfo：视频码流的基本信息，如编码格式、分辨率、帧率等
返回值	成功：返回视频解码器指针； 失败：返回 NULL；
调用说明	

4.3.3.2. VideoEngineDestroy

函数原型	void VideoEngineDestroy(VideoEngine* pVideoEngine)
功能	销毁一个视频解码器
参数	pVideoEngine: 通过 VideoEngineCreate 函数创建的视频解码器指针;
返回值	无
调用说明	

4.3.3.3. VideoEngineReset

函数原型	void VideoEngineReset(VideoEngine* pVideoEngine)
功能	重置视频解码器
参数	pVideoEngine: 通过 VideoEngineCreate 函数创建的视频解码器指针;
返回值	无
调用说明	Reset 后，视频基本信息仍保留; Reset 后，VideoEngine 应该将内部缓冲的视频图像输出到 FBM 模块;

4.3.3.4. VideoEngineSetSbm

函数原型	int VideoEngineSetSbm(VideoEngine* pVideoEngine, Sbm* pSbm, int nIndex)
功能	设置一个码流管理模块给解码器，解码器从该模块获取视频码流
参数	pVideoEngine: 通过 VideoEngineCreate 函数创建的视频解码器指针; pSbm*: 码流管理模块指针; nIndex: 在蓝光 MVC 等 3D 视频中，解码器处理 2 路码流，nIndex 指定 pSbm 模块是第几路码流，取值为 0 或者 1。
返回值	0: 表示成功; -1: 表示失败;
调用说明	

4.3.3.5. VideoEngineGetFbmNum

函数原型	int VideoEngineGetFbmNum(VideoEngine* pVideoEngine)
功能	查询图像管理模块的个数; 在蓝光 MVC 等 3D 视频中，解码器有 2 路图像数据，通过 2 个 FBM 模块对图像 Buffer 进行管理。
参数	pVideoEngine: 通过 VideoEngineCreate 函数创建的视频解码器指针;

返回值	图像 Buffer 管理模块的个数
调用说明	

4.3.3.6. VideoEngineGetFbm

函数原型	Fbm* VideoEngineGetFbm(VideoEngine* pVideoEngine, int nIndex)
功能	获取图像管理模块的指针；
参数	pVideoEngine：通过 VideoEngineCreate 函数创建的视频解码器指针； nIndex：在蓝光 MVC 等 3D 视频中，解码器输出 2 路图像，nIndex 指定获取第几路视频的 FBM 模块指针，取值为 0 或者 1。
返回值	成功：返回图像 Buffer 管理模块的指针； 失败：返回 NULL；
调用说明	

4.3.3.7. VideoEngineDecode

函数原型	int VideoEngineDecode(VideoEngine* pVideoEngine, int bEndOfStream, int bDropBFrameIfDelay, int64_t nCurrentTimeUs)
功能	解码一帧图像
参数	pVideoEngine：通过 VideoEngineCreate 函数创建的视频解码器指针； bEndOfStream：是否码流结束，等于 1 表示所有码流数据都已经传送到 Sbm； bDropBFrameIfDelay：码流的时间戳（PTS）比当前时间大时，是否丢弃 B 帧，等于 0 表示不丢弃，等于 1 表示丢弃 B 帧； nCurrentTimeUs：当前时间，用于比较码流是否过时；
返回值	VDECODE_RESULT_FRAME_DECODED(1)：解码成功，输出了一帧图像； VDECODE_RESULT_CONTINUE(2)：码流被解码，但没有图像输出，需继续解码； VDECODE_RESULT_KEYFRAME_DECODED(3)：解码成功，输出了一帧关键帧图像； VDECODE_RESULT_NO_FRAME_BUFFER(4)：当前无法获取到图像 Buffer； VDECODE_RESULT_NO_BITSTREAM(5)：当前无法获取到码流数据； VDECODE_RESULT_RESOLUTION_CHANGE(6)：视频分辨率发生变化，无法继续； VDECODE_RESULT_UNSUPPORTED(-1)：不能支持的格式或申请内存失败，无法继续解码；
调用说明	一般情况下，解码器在本函数中解码完一帧图像后返回，如果码流数据用完但仍没有图像输出，解码器应返回 VDECODE_RESULT_NO_BITSTREAM； bEndOfStream 等于 1 时，解码器在码流 Buffer 没有码流数据时应该将所有已解码图像送出。

5. 数据结构设计

5.1. VideoDecoderContext

名称	VideoDecoder	
功能描述	表示一个视频解码器，包含视频解码器的所有信息。	
属性	类型	描述
vconfig	VConfig	保存视频解码器的配置信息，如旋转、缩放、输出像素格式选择等；
videoStreamInfo	VideoStreamInfo	保存视频码流的信息，如编码格式、分辨率等；
pVideoEngine	VideoEngine*	VideoEngine 模块的指针，负责解码视频码流；
nFbmNum	int	解码器内有多少个 FBM 模块用于输出，对于蓝光 3D 等视频，解码器输出两路视频图像；
pFbm[2]	Fbm*	FBM 模块指针,用于管理图像 Buffer；
nSbmNum	int	解码器内有多少个 SBM 模块，对于蓝光 3D 等视频，解码器需要管理和解码两路视频码流；
pSbm[2]	Sbm*	SBM 模块指针，用于管理码流数据；
partialStreamDataInfo[2]	VideoStreamDataInfo	用于暂存外部提交的码流信息；

5.2. VideoStreamInfo

名称	VideoStreamInfo	
功能描述	描述视频码流的基本信息，包括编码格式、分辨率、帧率等等。	
属性	类型	描述
eCodecFormat	int	视频编码格式
nWidth	int	视频画面宽度，以像素为单位
nHeight	int	视频画面高度，以像素为单位
nFrameRate	int	视频帧率，表示每 1000 秒有多少帧画面被播放，例如 25000、29970、30000 等；
nFrameDuration	int	两帧画面之间的间隔时间，是帧率的倒数，单位为微妙；
nAspectRatio	int	像素宽高比，指定视频画面内一个像素的宽高比值，用于调整视频画面显示比例
bIs3DStream	int	是否是 3D 码流，3D 码流是指带有两路视频码流的视频码流，如蓝光 MVC 3D 视频； 左右半幅、上下半幅等 3D 视频不是 3D 码流，应设为 0；
nCodecSpecificDataLen	int	解码器初始化数据的长度
pCodecSpecificData	char*	解码器初始化数据的地址， 各解码格式所需的初始化数据及其格式，请参考本文档附录 6.1 节

5.3. VideoStreamDataInfo

名称	VideoStreamDataInfo	
功能描述	描述一笔码流数据的信息，包括数据地址、大小、时间戳、边界信息等	
属性	类型	描述
pData	char*	码流数据的地址
nLength	int	码流数据的长度
nPts	int64_t	第一个数据单元的时间戳，等于-1 表示无效
nPcr	int64_t	码流数据对应的 PCR 时钟，数字电视应用中使用
bIsFirstPart	int	第一个字节是否是一个数据单元的开始
bIsLastPart	int	最后一个有效字节是否是一个数据单元的结束
nID	int	一笔码流数据的 ID 号，由 SBM 模块赋值
nStreamIndex	int	由外部使用，解码器控制模块 vdecoder 用于标记该笔数据属于第几路码流；
bValid	int	表示这笔码流是否有效，由 SBM 模块赋值

5.4. VideoPicture

名称	VideoPicture	
功能描述	描述一帧图像的信息，包括数据地址、分辨率、时间戳、像素格式等等	
属性	类型	描述
nID	int	图像 Buffer 的 ID 号，由 FBM 模块赋值
nStreamIndex	int	由外部使用，解码器控制模块 vdecoder 用于标记该图像 Buffer 属于第几路视频；
ePixelFormat	int	图像像素格式，可以是下面几个值中的一个： PIXEL_FORMAT_YUV_PLANER_420 PIXEL_FORMAT_YUV_PLANER_422 PIXEL_FORMAT_YUV_PLANER_444 PIXEL_FORMAT_YV12 PIXEL_FORMAT_NV21 PIXEL_FORMAT_YUV_MB32_420 PIXEL_FORMAT_YUV_MB32_422 PIXEL_FORMAT_YUV_MB32_444 每种像素格式对应的数据存放方式，请参考本文档附录 6.3 节。
nWidth	int	图像的宽度，以像素为单位
nHeight	int	图像的高度，以像素为单位
nLineStride	int	图像在内存存储时的行宽，以像素为单位
nTopOffset	int	图像的有效内容从第几行开始
nLeftOffset	int	图像的有效内容从第几列开始
nFrameRate	int	视频帧率，表示每 1000 秒有多少帧画面被播放，例如

		25000、29970、30000 等；
nAspectRatio	int	像素宽高比，指定视频画面内一个像素的宽高比值，用于调整视频画面显示比例
bIsProgressive	int	是否是逐行扫描的源图像
bTopFieldFirst	int	是否是顶场数据先显示
bRepeatTopField	int	显示时重复一次顶场
bRepeatBottomField	int	显示时重复一次底场
nPts	int64_t	画面的显示时间，等于-1 表示无效
nPcr	int64_t	画面对应的 PCR 时钟，数字电视应用中使用
pData0	char*	Y 分量数据的起始地址
pData1	char*	U 分量或 UV 混合分量数据的起始地址
pData2	char*	V 分量数据的起始地址
pData3	char*	暂时没有使用

5.5. Sbm

名称	Sbm	
功能描述	Stream Buffer Manager，管理码流 Buffer 的模块	
属性	类型	描述
mutex	pthread_mutex_t	互斥锁，SBM 模块支持多线程操作，互斥锁同步多个线程的操作，保护内部变量。
pStreamBuffer	char*	码流 Buffer 起始地址；
pStreamBufferEnd	char*	码流 Buffer 结束地址，指向 Buffer 最后一个字节；
nStreamBufferSize	int	码流 Buffer 的大小，以字节为单位；
pWriteAddr	char*	下一笔码流数据的写入地址；
nValidDataSize	int	Buffer 内有效码流数据的大小，以字节为单位；
frameFifo	StreamFrameFifo	管理码流帧的 FIFO，存放每帧码流的信息；

5.6. StreamFrameFifo

名称	StreamFrameFifo	
功能描述	描述码流帧 FIFO 队列，SBM 模块内部使用	
属性	类型	描述
pFrames	VideoStreamDataInfo*	存放每帧码流信息的循环数组；
nMaxFrameNum	int	pFrames 数组的大小；
nValidFrameNum	int	表示有效码流帧有多少帧；
nReadPos	int	第一笔有效码流在 pFrames 数组中的位置；
nWritePos	int	下一笔码流信息的写入位置；
nFlushPos	int	已经被取出，但未被刷除的有效码流帧的起始位置。

5.7. Fbm

名称	Fbm	
功能描述	Frame Buffer Manager，管理图像 Buffer 的模块	
属性	类型	描述
mutex	pthread_mutex_t	互斥锁，FBM 模块支持多线程操作，互斥锁同步多个线程的操作，保护内部变量；
nMaxFrameNum	int	图像 Buffer 的数量；
bThumbnailMode	int	是否是解码缩略图模式，见 FbmCreate 函数调用说明中关于缩略图模式的描述；
pEmptyBufferQueue	FrameNode*	空闲图像 Buffer 队列；
pValidPictureQueue	FrameNode*	待显示图像队列；
pFrames	FrameNode*	存放每个图像 Buffer 信息的数组；

5.8. FrameNode

名称	FrameNode	
功能描述	图像 Buffer 队列的一个节点，FBM 模块内部使用。	
属性	类型	描述
bUsedByDecoder	int	该图像 Buffer 是否被解码器占用；
bUsedByRender	int	该图像 Buffer 是否被取出用于显示；
bInValidPictureQueue	int	该图像 Buffer 是否在待显示队列中；
bAlreadyDisplayed	int	该图像 Buffer 是否已经被显示后归还
pNext	FrameNode*	队列中下一个节点的地址；
vpicture	VideoPicture	图像 Buffer 信息；

5.9. VConfig

名称	VConfig	
功能描述	Video Engine 模块的配置信息	
属性	类型	描述
bScaleDownEn	int	是否使能输出图像缩小功能
bRotationEn	int	是否使能输出图像旋转功能
nHorizonScaleDownRatio	int	输出图像水平方向缩小比例， 0：不缩放；1：1/2； 2：1/4；3：1/8
nVerticalScaleDownRatio	int	输出图像垂直方向缩小比例， 0：不缩放；1：1/2； 2：1/4；3：1/8
nSecHorizonScaleDownRatio	int	从路通道输出图像水平方向缩小比例， 0：不缩放；1：1/2； 2：1/4；3：1/8

nSecVerticalScaleDownRatio	int	从路通道输出图像垂直方向缩小比例， 0：不缩放；1：1/2； 2：1/4；3：1/8
nRotateDegree	int	输出图像顺时针旋转角度， 0：不旋转；1：90； 2：180；3：270
bThumbnailMode	int	是否已解码缩略图模式使用解码器
eOutputPixelFormat	int	指定输出图像的像素格式，A23、A31、A31S 平台支持设置为以下两种， PIXEL_FORMAT_DEFAULT（=0） PIXEL_FORMAT_YV12 A10、A13、A10S、A20 平台不支持指定格式， A23 平台上 VP6、WMV1、WMV2 不支持指定格式， A31、A31S 平台上 H265 不支持指定格式。

5.10. VideoEngine

名称	VideoEngine	
功能描述	描述视频解码器模块信息	
属性	类型	描述
config	VConfig	解码器配置信息，记录旋转、缩放等配置选项；
videoStreamInfo	VideoStreamInfo	视频信息，包含编码格式、分辨率、帧率等；
pLibHandle	void*	外部解码库句柄；
pDecoderInterface	DecoderInterface*	解码器内部接口，用于调用 H264 等解码功能；

5.11. DecoderInterface

名称	DecoderInterface	
功能描述	解码器内部接口和上下文信息，用于调用 H264、MPEG2 等解码功能，Video Engine 模块内部使用	
方法	描述	
Init	原型	int (*Init)(DecoderInterface* pSelf, VConfig* pConfig, VideoStreamInfo* pVideoInfo);
	参数	pSelf：通过 CreateSpecificDecoder 函数创建的底层解码器指针； pConfig：解码器配置信息； pVideoInfo：视频码流信息；
	返回值	VDECODE_RESULT_FRAME_OK(0)：成功； VDECODE_RESULT_UNSUPPORTED(-1)：不能支持的格式或申请内存失败；
	功能	初始化底层解码器。
Reset	原型	void (*Reset)(DecoderInterface* pSelf);
	参数	pSelf：通过 CreateSpecificDecoder 函数创建的底层解码器指针；

	返回值	无
	功能	重置底层解码器，Reset 后，视频基本信息仍保留； 每解码完一帧视频后，本函数都会被调用；
SetSbm	原型	int (*SetSbm)(DecoderInterface* pSelf, Sbm* pSbm, int nIndex);
	参数	pSelf：通过 CreateSpecificDecoder 函数创建的底层解码器指针； pSbm*：码流管理模块指针； nIndex：在蓝光 MVC 等 3D 视频中，解码器处理 2 路码流，nIndex 指定 pSbm 模块是第几路码流，取值为 0 或者 1。
	返回值	VDECODE_RESULT_FRAME_OK (0)：表示成功； VDECODE_RESULT_UNSUPPORTED(-1)：表示失败；
	功能	设置一个码流管理模块给底层解码器，底层解码器从该模块获取视频码流
GetFbmNum	原型	int (*GetFbmNum)(DecoderInterface* pSelf);
	参数	pSelf：通过 CreateSpecificDecoder 函数创建的底层解码器指针；
	返回值	图像 Buffer 管理模块的个数
	功能	查询图像管理模块的个数； 在蓝光 MVC 等 3D 视频中，解码器有 2 路图像数据，通过 2 个 FBM 模块对图像 Buffer 进行管理。
GetFbm	原型	Fbm* (*GetFbm)(DecoderInterface* pSelf, int nIndex);
	参数	pSelf：通过 CreateSpecificDecoder 函数创建的底层解码器指针； nIndex：在蓝光 MVC 等 3D 视频中，解码器输出 2 路图像，nIndex 指定获取第几路视频的 FBM 模块指针，取值为 0 或者 1。
	返回值	成功：返回图像 Buffer 管理模块的指针； 失败：返回 NULL；
	功能	获取图像管理模块的指针；
Decode	原型	int (*Decode)(DecoderInterface* pSelf, int bEndOfStream, int bSkipBFrameIfDelay, int64_t nCurrentTimeUs);
	参数	pSelf：通过 CreateSpecificDecoder 函数创建的底层解码器指针； bEndOfStream：是否码流结束，等于 1 表示所有码流数据都已经传送到 Sbm； bDropBFrameIfDelay：码流的时间戳（PTS）比当前时间大时，是否丢弃 B 帧，等于 0 表示不丢弃，等于 1 表示丢弃 B 帧； nCurrentTimeUs：当前时间，用于比较码流是否过时；

	返回值	VDECODE_RESULT_FRAME_DECODED(1)：解码成功，输出了一帧图像； VDECODE_RESULT_CONTINUE(2)：码流被解码，但没有图像输出，需继续解码； VDECODE_RESULT_KEYFRAME_DECODED(3)：解码成功，输出了一帧关键帧图像； VDECODE_RESULT_NO_FRAME_BUFFER(4)：当前无法获取到图像 Buffer； VDECODE_RESULT_NO_BITSTREAM(5)：当前无法获取到码流数据； VDECODE_RESULT_RESOLUTION_CHANGE(6)：视频分辨率发生变化，无法继续； VDECODE_RESULT_UNSUPPORTED(-1)：不能支持的格式或申请内存失败，无法继续解码；
	功能	解码一帧画面
Destroy	原型	void (*Destroy)(DecoderInterface* pSelf);
	参数	pSelf：通过 CreateSpecificDecoder 函数创建的底层解码器指针；
	返回值	无
	功能	销毁底层解码器

6. 附录

6.1. 视频初始化信息设置

6.1.1. H264 解码器的 Initial Data

H264 解码器(mp4、mkv 等帧格式封装)的 initData 存储的是 sps 和 pps 的信息，数据格式为:

Lens(bits)	Contents
8	reserved
8	profile
8	reserved
8	level
6	reserved
2	Size of NALU length minus 1
3	reserved
5	Number of sequence parameter sets
?	Sequence parameter sets, each is prefixed with two byte big-endian size field
8	Number of picture parameter sets
?	Picture parameter sets, each is prefixed with two byte big-endian size field

注：Ts 封装的 H264 码流不需要传输 initData。

6.1.2. MPEG2 解码器的 Initial Data

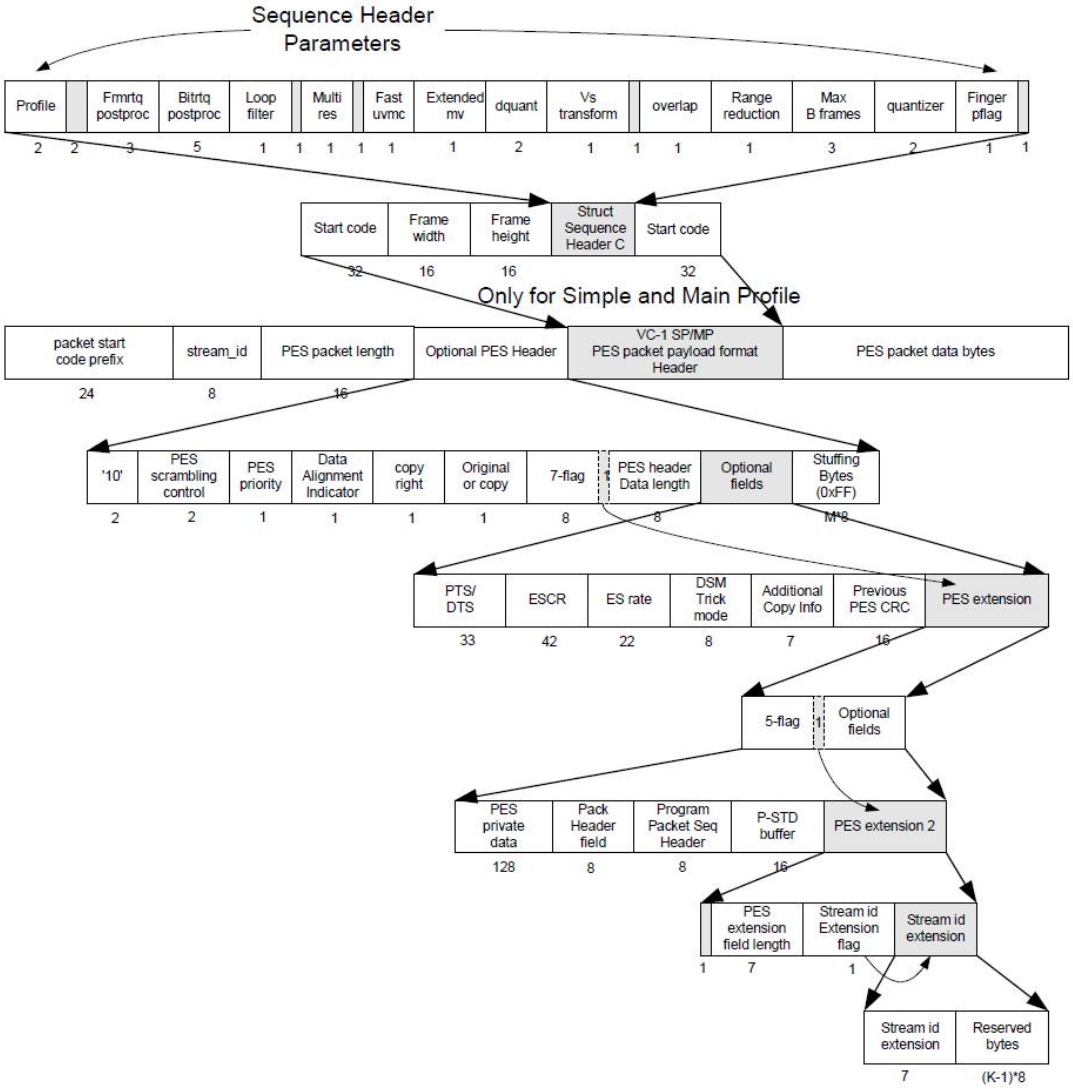
MPEG2 解码器的 init data 存储的是 sequence 信息，其数据格式如下：

Lens(bits)	Contents
8	Mpeg1 解码器标记， 若为 mpeg2 则此 byte 为 0
32	Sequence header code (0x000001B3)
12	Horizontal size value
12	Vertical Size value
4	Aspect ratio information
4	Frame rate code
18	Bit rate value
1	Marker bit
10	Vbv buffer size value
1	Constrained parameters flag
1	Load intra quantiser matrix
8*64	if(Load intra quantiser matrix == 1) Intra quantiser matrix[64]
1	Load non intra quantiser matrix
8*64	if(Load non intra quantiser matrix == 1) Non intra quantiser matrix[64]

6.1.3. WMV3 解码器的 Initial Data

WMV3(VC1) 解码器的 initData 中保存的是码流的 sequence 信息。数据组成如下表和图所示：

Sequence information parameters		
Lens (bits)	Contents	value
32	Sequence start code	0x0000010f
16	frame width	
16	frame height	
32	Struct sequence_header_c	
32*n	frame start code	0x0000010d



6.1.4. RXG2 解码器的 Initial Data

包含 RXG2 码流的文件可能是 MKV 或者 RX。RXG2 解码器的 initData 数据格式为：

Lens(bits)	Contents	
32	Optional Unrestricted Motion Vector (UMV) mode (1 or 0)	
32	Optional Advanced Prediction (AP) mode (1 or 0)	
32	Optional Advanced INTRA Coding (AIC) mode (1 or 0)	
32	Optional Deblocking Filter (DF) mode (1 or 0)	
32	Optional Slice Structured (SS) mode (1 or 0)	
32	Optional Reference Picture Selection (RPS) mode (1 or 0)	
32	Optional Independent Segment Decoding (ISD) mode (1 or 0)	
32	Optional Alternative INTER VLC (AIV) mode (1 or 0)	
32	Optional Modified Quantization (MQ) mode （1 or 0）	
32	iRvSubIdVersion	
16	uNum_RPR_Sizes	
16	Source format:	Width x height
	0x000: reserved	
	0x001: sub-QCIF	128 x 96
	0x010: QCIF	176 x 144
	0x011: CIF	352 x 288
	0x100: 4CIF	704 x 506
	0x101: 16CIF	1408 x 1152
	0x110: custom source format	
	0x111: reserved	
32	RmCodecID	

6.2. 视频码流传输格式

6.3. 视频像素格式

6.3.1. YUV420、YUV422 和 YUV444 采样方式

视频编码时，输入图像通常是 YUV420 格式的图像序列。因此，各种视频解码器的输出通常也是标准的 YUV420 图像。YUV420 格式采样如图 6.3.1-1 所示：

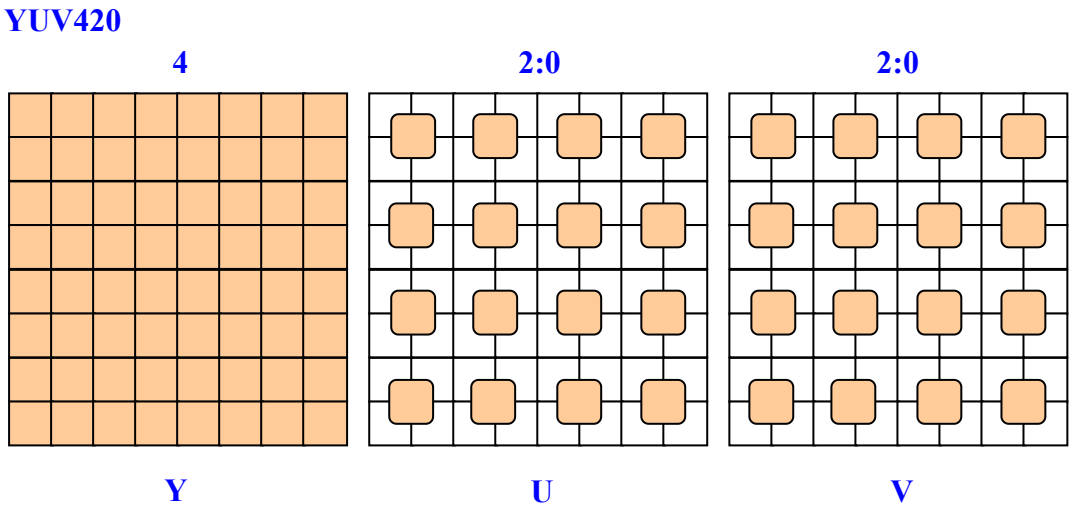


图 6.3.1-1. YUV420 格式的采样

按 YUV420 采样时，图像为 WIDTH*HEIGHT 大小时，图像 Y 分量也为 WIDTH*HEIGHT 大小。由于人眼对色度分量不敏感，因此 U 和 V 两个色差分量被降采样，每四个像素点采样一个 U 分量和一个 V 分量，因此 U 的数据量为 Y 的四分之一，V 的数据量也为 Y 的四分之一。

有时候视频编码采用的输入数据位 YUV422 格式，即 U 分量和 V 分量的数据是每两个像素点取一个，如图 6.3.1-2 所示。

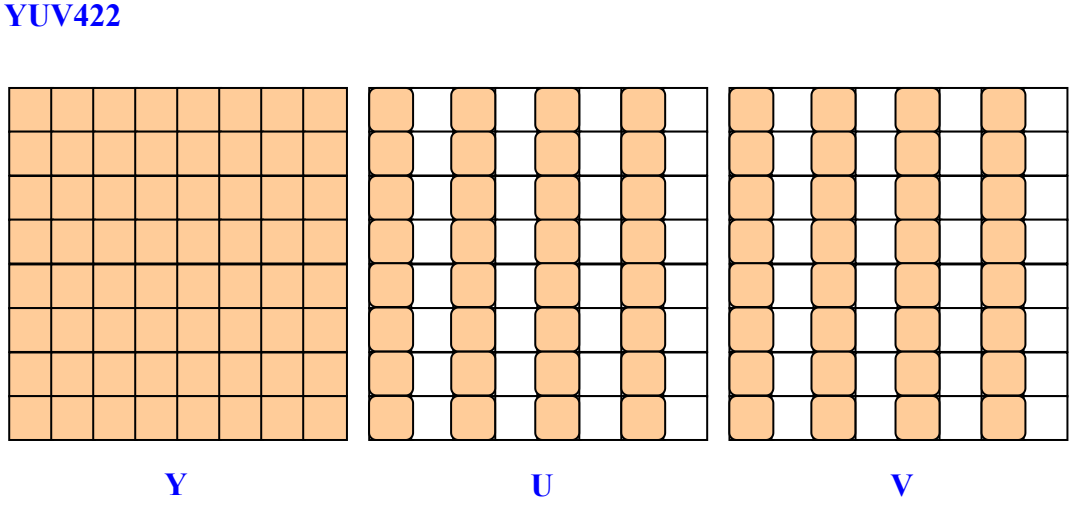


图 6.3.1-2. YUV422 格式的采样

有时候视频编码甚至采用的输入数据位 YUV444 格式，即 U 分量和 V 分量的采样精度与 Y 分量一样，如图 6.3.1-3 所示。

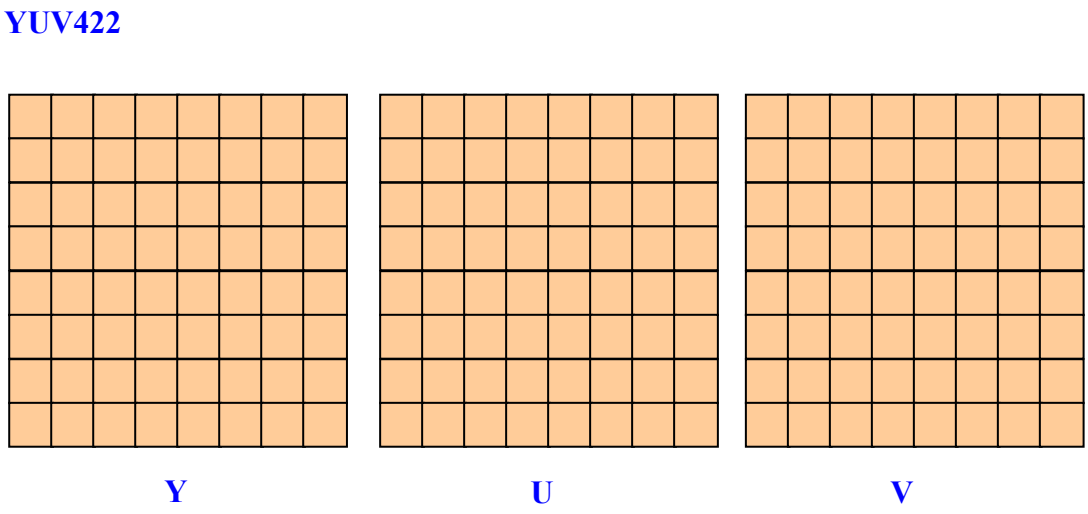


图 6.3.1-3. YUV444 格式的采样

在实际应用中，最常用的是 YUV420 采样方式。

6.3.2. YUVPlaner 排列格式与 YV12 排列格式

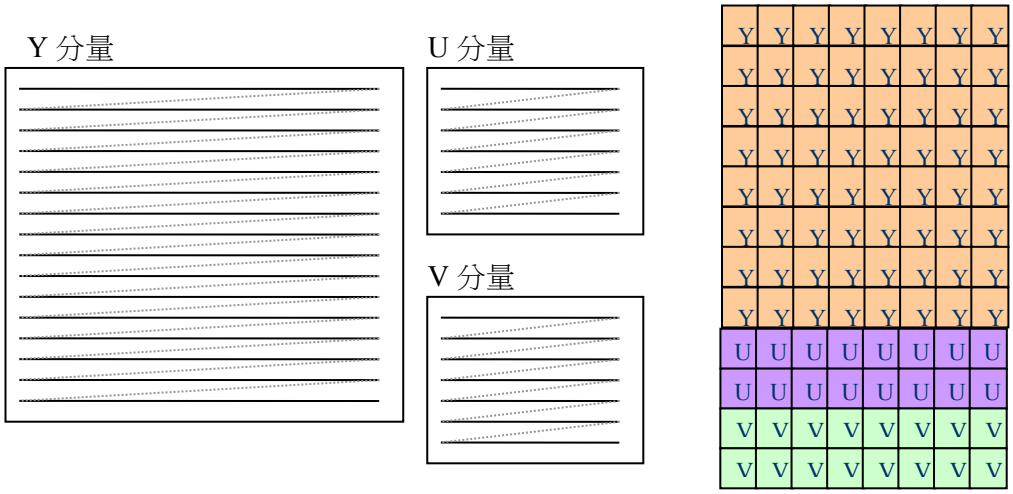


图 6.3.2-1. YUVPlaner 格式图像的光栅扫描和数据存放

在内存中，YUV 数据一般按照光栅顺序扫描存放，内存依次存放 Y 分量的第一行、第二行、第三行，直到 Y 分量存放完后，接着存放 U 分量的第一行、第二行、第三行，直到 U 分量存放完后接着存放 V 分量，如图 6.3.2-1 所示。这种存放格式称为 YUVPlaner 格式。YUVPlaner 格式是一种非常通用的像素排列方式。

目前 YV12 格式也是一种比较通用的格式，各种 GPU 都选择此格式为默认输入格式。YV12 格式是指把 YUV420 采样的数据，按照光栅扫描顺序进行存放，但与 YUVPlaner 存放方式不一样，它把 V 分量放在 U 分量的前面。图 6.3.2-2 所示是 YV12 格式的数据存放方式。这里需要注意，YV12 格式下，YUV 数据是按 YUV420 的方式采样的。

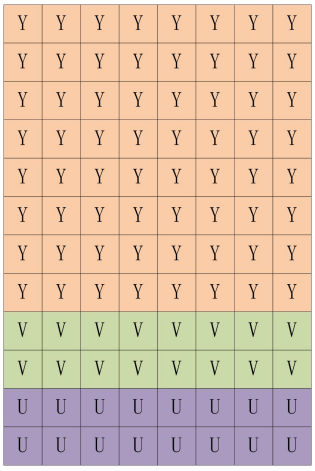


图 6.3.2-2. YV12 格式数据存放

6.3.3. NV21 排列格式

NV21 属于 YUV420 格式， 是一种 two-plane 模式， 即 Y 和 UV 分为两个 Plane，且 UV 为交错存储，图 6.3.3-1 是 NV21 格式的数据存储方式。

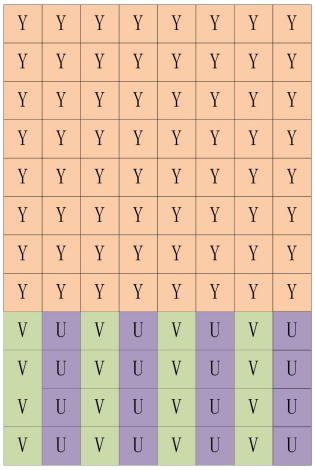


图 6.3.3-1. NV21 格式数据存放

6.3.4. MB32 排列格式

VE 视频解码硬件默认输出 MB32 格式的图像。MB 是 Macro Block 的意思，MB32 是指以 32*32 大小的宏块作为一个存储单元，宏块的数据连续存放。因此，按照 MB32 格式存放 YUV420 数据时，存放顺序不是按照光栅扫描，而是按照如图 6.3.4-1 所示的扫描顺序：

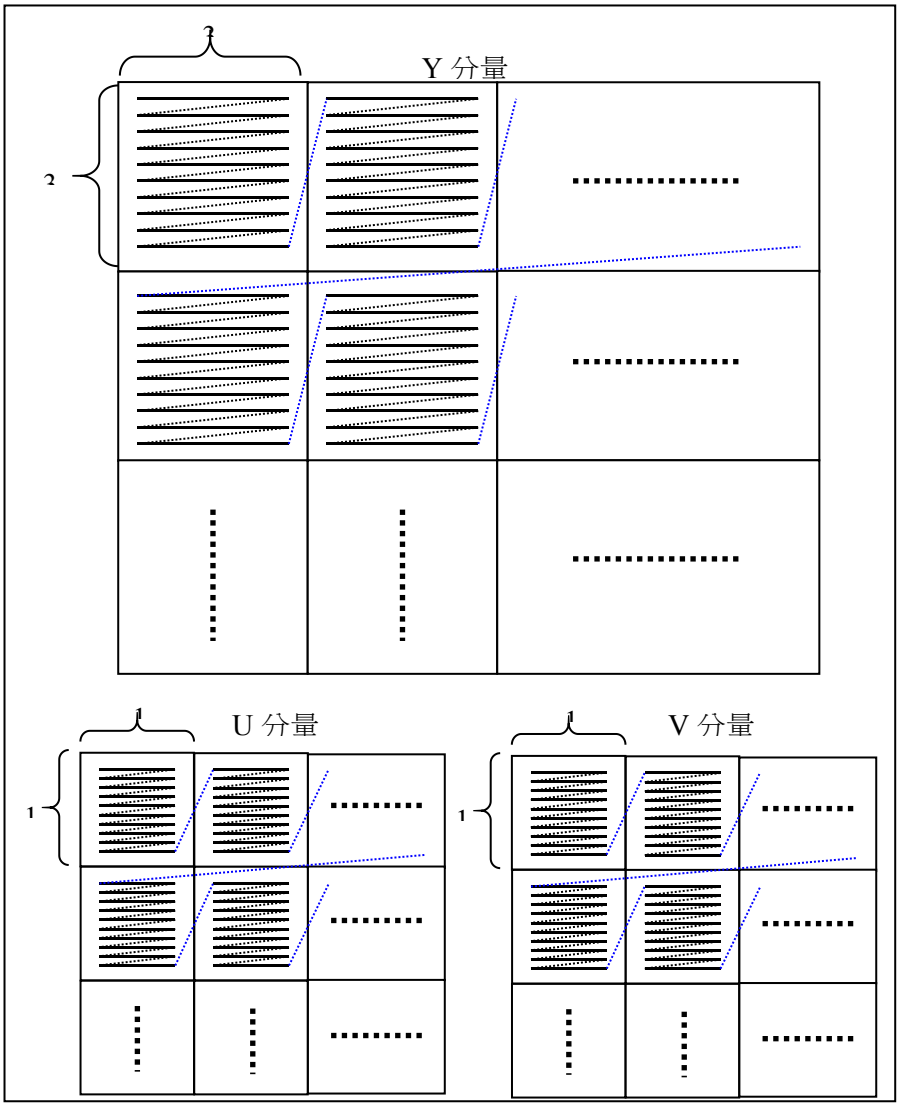


图 6.3.4-1. 按照 MB32 格式存放 YUV 数据时的扫描顺序

按照 MB32 格式存放 YUV 数据时，Y 分量和 UV 分量可以分开存放，但 UV 分量是存放在一起，且是间隔存放的，如图 6.3.4-2 所示：

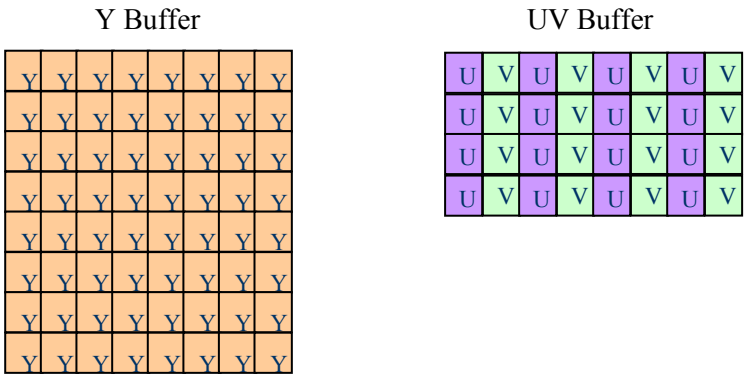


图 6.3.4-2. 按 MB32 格式存放 YUV420 数据

实际应用中，图像的宽度和高度可能不满足 32 像素对齐，此时需要将图像扩展到宽度和高度都满足 32 像素对齐（相应的，U 分量和 V 分量为 16 对齐）。

6.3.5. MB32 排列格式如何转换成 YUVPlaner 排列格式

从以上对格式的介绍可以知道，把 MB32 格式的 YUV 图像转换成 YUVPlaner 格式的 YUV 图像，只需把 YUV 数据重新排列。下面针对最常用的 YUV420 采样方式为例，介绍如何将 MB32 格式的 YUV420 图像转换为 YUVPlaner 格式的 YUV420 图像。

6.3.5.1. Y 分量的转换

Y 分量的转换方法如下面代码所示，注意转换时图像高度、宽度是 32 对齐的。

```
static void WriteBack_Y(u32 Width, u32 Height, u8 *src, u8 *dst)
{
	u32 i;
	u32 j;
	u32 x;
	u32 y;
	u8* srcT;
	u8* dstT;

	srcT = src;
	dstT = dst;

	for(y=0; y<Height; y+=32)
	{
		dstT = dst + Width*y;
		srcT = src + Width*y;
		for(x=0; x<Width; x+=32)
		{
			for(i=0; i<32; i++)
			{
				for(j=0; j<32; j++)
					dstT[i*Width + j] = srcT[i*32 + j];
			}
			dstT += 32;
			srcT += 32*32;
		}
	}
}
```

6.3.5.2. U 分量和 V 分量的转换

UV 分量的转换方法如下面代码所示，注意转换时图像高度、宽度是 32 对齐的。

```
static void WriteBack_UV(u32 width, u32 height, u8* srcChrom, u8* dstU, u8* dstV)
{
	u32 i;
	u32 j;
	u32 x;
	u32 y;
	u8* C;
	u8* U;
	u8* V;

	C    = srcChrom;

	for(y=0; y<height/2; y+=32)
	{
		U = dstU + width/2*y;
		V = dstV + width/2*y;
		C = srcChrom    + width*y;

		for(x=0; x<width; x+=32)
		{
			for(i=0; i<32; i++)
			{
				for(j=0; j<16; j++)
				{
					if ((y+i)<(height/2))
					{
						U[i*width/2 + j] = C[i*32 + 2*j];
						V[i*width/2 + j] = C[i*32 + 2*j + 1];
					}
				}
			}

			U += 16;
			V += 16;
			C += 32*32;
		}
                                                        }
}
```

6.3.5.3. 完整的转换

转换一幅图像时，需要注意图像的宽度和高度的 32 对齐问题，完整的转换如下面代码所示。

```
static u8* transform_to_yuv420_planer(u32 Width,   u32 Height,
                                      u8* ySrc,        u8* cSrc,
                                      u8** pY,         u8** pU,   u8** pV)
{
    u8* yuv_plane;
    u32 ysize;
    u32 csize;

    //* width and height with 16 pixels align.
    u32 width16;
    u32 height16;

    //* width and height with 32 pixels align.
    u32 width32, height32;

    //* height with 64 pixels align.
    u32 height64;

    //* destination YUV.
    u8* Y;
    u8* U;
    u8* V;

    width16    = (Width   + 15) & ~15;
    height16   = (Height + 15) & 15;
    width32    = (Width   + 31) & ~31;
    height32   = (Height + 31) & ~31;
    height64   = (Height + 63) & ~63;

    ysize = width32*height32;   //* for y.
    csize = width32*height64/2;         //* for u and v together.

    yuv_plane = (u8*)malloc(ysize + csize);
    if(yuv_plane == NULL)
        return;

   Y = yuv_plane;
   WriteBack_Y(width32, height32, ySrc, Y);

   U = Y + width32*height32;
   V = U + (width32*height64/4);
   WriteBack_UV(width32, height64, cSrc, U, V);

   *pY = Y;
   *pU = U;
   *pV = V;
                                                  return;
}
```

6.3.5.4. YUV422 和 YUV444 采样格式下如何转换

从以上介绍可知，MB32 和 YUVPlaner 格式指的是一种 YUV 数据的排列方式，YUV420、YUV422 或 YUV444 采样所得的 YUV 数据都可按照此这两种方式排列。本文档上面一节提供了 YUV420 采样时 MB32 和 YUVPlaner 格式的转换方法，对于 YUV422、YUV444 采样的 YUV 数据，不同之处知识 UV 分量的数据量有所增加，数据的排列方式、UV 交错排列的方式都是一样的。因此 YUV422 和 YUV444 采样的 YUV 数据可简单类推。

7. Declaration

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.