ANYKA
芯 随 你 动

版本: **1.0.4 2014** 年 **06** 月

视频驱动库接口说明

声　　明

联 系 方 式

安凯（广州）微电子技术有限公司

地址：广州科学城科学大道 182 号创新大厦 C1 区 3 楼
电话: (86)-20-3221 9000
传真: (86)-20-3221 9258
邮编: 510663

销售热线:

(86)-20-3221 9499

电子邮箱:

sales@anyka.com

主页:

http: //www. anyka.com

版 本 变 更 说 明

以下表格对于本文档的版本变更做一个简要的说明。版本变更仅限于技术内容的变更，不包括版式、格式、句法等的变更。

版本	说明	完成日期
V1.0.0	正式发布	2011 年 12 月
V1.0.1	更新描述	2011 年 12 月
V1.0.2	增加 VideoStream_GetNextPts 和 VideoStream_SpecialDecode	2012 年 05 月
V1.0.3	增加 VideoStream_SetRotate	2012 年 05 月
V1.0.4	增加 VideoStream_GetItemNum 增加 AK37XXC 芯片支持情况； 增加 VideoStream_Enc_SetTimeStamp 和 VideoStream_Enc_SetVideoBitrate, VideoStream_Enc_Reset	2014 年 06 月

视频驱动库接口说明与视频驱动库对应的关系

Version	Corresponding video driver lib
V1.0.4	V1.10.00
V1.0.3	V1.9.00
V1.0.2	V1.8.00
V1.0.1	V1.7.08
V0.2.0	V0.7.6
V0.1.1	V0.7.2
V0.1.0	V0.7.0
V0.0.1	V0.6.2

目录

1 模块介绍

1.1 功能概述

本模块主要在 AK88/98XX 以及 AK37XX/AK37XXC 芯片基础上使用。

AK88XX 芯片支持情况：

- 视频解码最大分辨率为 H263/MPEG4/FLV263 格式 VGA（H×V：640×480），H264 格式 D1（H×V：720×576），MPEG4 ASP（720P，H×V：1280×720），RV（H×V：1024×1024），MJPEG（1280×1280）
- 编码支持 H263

AK98XX 芯片支持情况：

- 视频解码最大分辨率为 H263/MPEG4/FLV263 格式 D1（H×V：720×576），H264 格式 720P（H×V：1280×720），MPEG4 ASP（720P），RV（720P），MJPEG（1280×1280）
- 编码支持 H263，MPEG4 SP

AK37XX 芯片支持情况：

- 视频解码最大分辨率为 H263/MPEG4/FLV263 格式 D1（H×V：720×576），MJPEG（1280×1280）

AK37XXC 芯片支持情况：

- 视频解码最大分辨率为 H263/MPEG4/FLV263 格式 VGA（H×V：640×480），MJPEG（1280×1280）
- 编码支持 H263，MPEG4 SP

主要用于 CMMB 播放和多线程播放。

1.2 与其它模块关系

本模块依赖于以下模块：

- 内存管理模块
- 系统功能模块

本模块需要调用以上模块相关接口进行内存分配释放等，具体见第 5 章“依赖接口说明”。

2 相关文档

《音视频库总体使用说明》。

3 集成指南

首次使用视频驱动库，请首先阅读《音视频库总体使用说明》。

3.1 层次结构

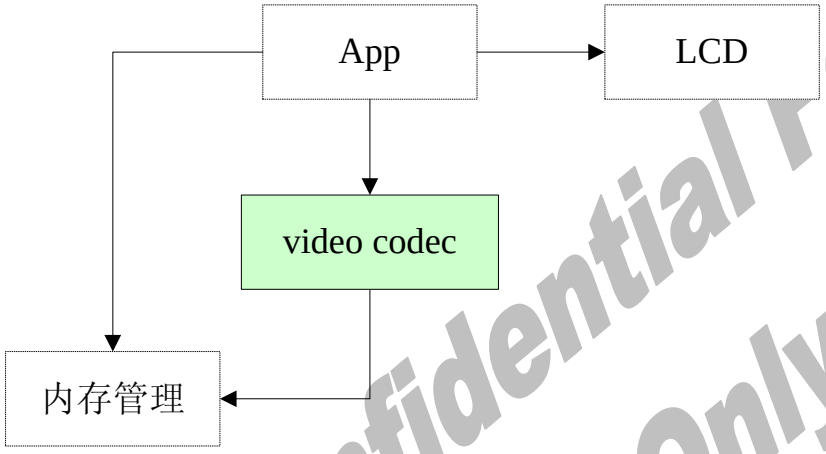

图 3-1 模块关系图

视频驱动库包含视频解码和视频编码，集成后在系统中的位置如上图所示。视频驱动库依赖的模块都应该在使用前由系统初始化。App 应用模块实现播放界面，调用视频驱动模块、控制 LCD 实现编解码和显示；内存管理模块用于申请和释放内存。箭头标明在运行播放过程中模块的调用情况。

3.2 集成步骤

3.2.1 获得视频驱动库相关文件

视频驱动库包括库二进制文件、头文件和相关文档。

3.2.2 实现视频驱动库依赖的系统API

请仔细阅读本文的第五部分，准备好视频驱动库所依赖的系统 API，为系统控制视频驱动库提供足够的灵活性，需要在集成时根据实际情况由系统集成人员实现。

3.2.3 集成链接测试

把视频驱动库、依赖的系统 API 实现和目标系统进行链接。

3.2.4 联合调试

在目标系统上联合调试一般分成以下步骤，调试和使用视频驱动库常见问题请参考本文的相关部分。

1、调试某种格式的解码，确定视频解码基本正常运作

2、调试各种格式的解码，确定视频解码库正常运作

3、调试 H263 编码，确定视频编码基本正常运作

4、调试各种格式的编码，确定视频编码库正常运作

3.3 注意事项

调用视频驱动库的函数前必须调用 MediaLib_Init 或 VideoStream_Init 进行全局初始化。

4 接口说明

4.1 模块功能概述

本模块起视频编解码的作用，请参考功能概述。

4.2 数据结构/格式

4.2.1 回调函数定义

见第 5 章“依赖接口说明”。

4.2.2 解码器打开输入结构体

```
typedef enum
{
    VIDEO_DRV_UNKNOWN = 0,
    VIDEO_DRV_H263,
    VIDEO_DRV_MPEG,
    VIDEO_DRV_FLV263,
    VIDEO_DRV_H264,
    VIDEO_DRV_RV,
    VIDEO_DRV_AVC1,
    VIDEO_DRV_MJPEG,
    VIDEO_DRV_MPEG2
}T_eVIDEO_DRV_TYPE;
```

结构名	T_eVIDEO_DRV_TYPE	
定义概述	编解码类型	
成员说明	VIDEO_DRV_UNKNOWN	未知类型，非法值
	VIDEO_DRV_H263	H263 格式
	VIDEO_DRV_MPEG	Mpeg4 格式
	VIDEO_DRV_FLV263	Flv263 格式

结构名	T_eVIDEO_DRV_TYPE	
	VIDEO_DRV_H264	带 start code 的 H264 格式，一般在 avi 或 cmmb 的视频流中出现
	VIDEO_DRV_RV	Real video 格式
	VIDEO_DRV_AVC1	不带 start code 的 H264 格式，一般在 mp4/flv/mkv 等的视频流中出现
	VIDEO_DRV_MJPEG	Mjpeg 格式
	VIDEO_DRV_MPEG2	MPEG2 格式

typedef struct

{

MEDIALIB_CALLBACK_FUN_PRINTF m_FunPrintf;

MEDIALIB_CALLBACK_FUN_MALLOC m_FunMalloc;

MEDIALIB_CALLBACK_FUN_FREE m_FunFree;

MEDIALIB_CALLBACK_FUN_MMU_INVALIDATEDCACHE m_FunMMUInvalidateDCache;

MEDIALIB_CALLBACK_FUN_MMU_INVALIDATEDCACHE m_FunCleanInvalidateDcache;

MEDIALIB_CALLBACK_FUN_CHECK_DEC_BUF m_FunCheckDecBuf;

MEDIALIB_CALLBACK_FUN_RTC_DELAY m_FunRtcDelay;

//add for Linux and CE

MEDIALIB_CALLBACK_FUN_DMA_MALLOC m_FunDMAMalloc;

MEDIALIB_CALLBACK_FUN_DMA_FREE m_FunDMAFree;

MEDIALIB_CALLBACK_FUN_VADDR_TO_PADDR m_FunVaddrToPaddr;

MEDIALIB_CALLBACK_FUN_MAP_ADDR m_FunMapAddr;

MEDIALIB_CALLBACK_FUN_UNMAP_ADDR m_FunUnmapAddr;

MEDIALIB_CALLBACK_FUN_REG_BITS_WRITE m_FunRegBitsWrite;

//add for hardware mutex

```
MEDIALIB_CALLBACK_FUN_VIDEO_HW_LOCK          m_FunVideoHWLock;
MEDIALIB_CALLBACK_FUN_VIDEO_HW_UNLOCK        m_FunVideoHWUnlock;

//add for using api about fifo in multithread
MEDIALIB_CALLBACK_FUN_MUTEX_CREATE           m_FunMutexCreate;
MEDIALIB_CALLBACK_FUN_MUTEX_LOCK             m_FunMutexLock;
MEDIALIB_CALLBACK_FUN_MUTEX_UNLOCK           m_FunMutexUnlock;
MEDIALIB_CALLBACK_FUN_MUTEX_RELEASE          m_FunMutexRelease;
}T_VIDEOLIB_CB;

typedef enum
{
        MEDIALIB_ROTATE_0,
        MEDIALIB_ROTATE_90,
        MEDIALIB_ROTATE_180,
        MEDIALIB_ROTATE_270
}T_eMEDIALIB_ROTATE;

typedef struct
{
        T_eVIDEO_DRV_TYPE  m_VideoType;
    T_U32   m_ulBufSize;     //if 0 == m_ulBufSize, use VideoStream_Decode to decode;
        T_BOOL                  m_bNeedSYN;
        T_BOOL                  m_bFixedStream;
        T_VIDEOLIB_CB           m_CBFunc;
        T_eMEDIALIB_ROTATE      m_Rotate;

        T_U16                   m_uWidth;
        T_U16                   m_uHeight;
}T_VIDEOLIB_OPEN_INPUT;
```

结构名	T_VIDEOLIB_OPEN_INPUT
定义概述	打开时输入参数

结构名	T_VIDEOLIB_OPEN_INPUT	
成员说明	m_VideoType	见 T_eVIDEO_DRV_TYPE
	m_ulBufSize	解码缓冲空间，不小于 200K；为 0 时表示不能使用同步解码接口
	m_bNeedSYN	调用同步解码接口时是否采用同步解码的方式
	m_bFixedStream	为 true 表示从文件或固定 buffer 数据播放，为 false 表示解码接收到的实时码流，如 cmmb
	m_CBFunc	见 T_VIDEOLIB_CB，T_VIDEOLIB_CB 中的各回调详见《音视频库总体使用说明》
	m_Rotate	旋转角度，见 T_eMEDIALIB_ROTATE
	m_uWidth	待解码的图像宽度
	m_uHeight	待解码的图像宽度

4.2.3 解码器状态

```
typedef enum
{
        VIDEO_STREAM_OK,
        VIDEO_STREAM_SYS_ERR,      //such as malloc failed
        VIDEO_STREAM_IDR_ERR,      //idr frame not found
        VIDEO_STREAM_SYN_FAST,     //video time is too fast
        VIDEO_STREAM_CODEC_ERR, //maybe data error or codec error
        VIDEO_STREAM_SYN_ERR       //audio and video pts is out of range, should be error
}T_eVIDEOLIB_ERROR;
```

结构名	T_eVIDEOLIB_ERROR	
定义概述	打开结果	
成员说明	VIDEO_STREAM_OK	正常
	VIDEO_STREAM_SYS_ERR	系统错误
	VIDEO_STREAM_IDR_ERR	IDR 错误，找不到第一个关键帧
	VIDEO_STREAM_SYN_FAST	视频解码超前

结构名	T_eVIDEOLIB_ERROR	
	VIDEO_STREAM_CODEC_ERR	视频编解码出错
	VIDEO_STREAM_SYN_ERR	同步错误，pts 相差太远

4.2.4 编码器打开输入结构体

```
typedef struct
{
    T_eVIDEO_DRV_TYPE       m_VideoType;
    T_U32                   m_ulWidth;      //宽度
    T_U32                   m_ulHeight;     //高度
    T_U32                   m_ulMaxVideoSize;
    T_VIDEOLIB_CB           m_CBFunc;
}T_VIDEOLIB_ENC_OPEN_INPUT;
```

结构名	T_VIDEOLIB_OPEN_INPUT	
定义概述	视频信息	
成员说明	m_VideoType	只能为 VIDEO_DRV_H263
	m_ulWidth	宽度，必须为 16 的倍数
	m_ulHeight	高度，必须为 16 的倍数
	m_ulMaxVideoSize	存放码流的 buffer 空间大小，不可小于 Width*Height/2，最好是 512 的整数倍
	m_CBFunc	回调函数

4.2.5 编码输入输出结构体

```
typedef struct
{
    T_pDATA         p_curr_data;
    T_pDATA         p_vlc_data;
    T_S32           QP;
    T_U8            mode;
```

```
    T_BOOL          bInsertP;
}T_VIDEOLIB_ENC_IO_PAR;
```

结构名	T_VIDEOLIB_ ENC_IO_PAR	
定义概述	视频信息	
成员说明	p_curr_data	原始图像地址
	p_vlc_data	编码后存放码流地址
	QP	编码 QP 值，取值范围[1, 31]；一般取 10 左右；对于同一帧图像，QP 越小编码压缩率越小；注意，当使用码率控制时该参数无效
	mode	编码类型：0－关键帧；1－非关键帧
	bInsertP	是否编码一帧插帧，只有在 mode 为 1 时有效

4.2.6 时间戳参数结构体

```
typedef struct
{
    T_U8    *m_pTimestampY;
    T_U8    *m_pTimestampU;
    T_U8    *m_pTimestampV;

    T_U32   m_ulOffsetX;
    T_U32   m_ulOffsetY;
    T_U32   m_ulWidth;
    T_U32   m_ulHeight;
}T_VIDEOLIB_ENC_TIMESTAMP_PAR;
```

结构名	T_VIDEOLIB_ENC_TIMESTAMP_PAR	
定义概述	编码时间戳参数结构体	
成员说明	m_pTimestampY	输入编码时间戳图像的 Y 数据地址
	m_pTimestampU	输入编码时间戳图像的 U 数据地址
	m_pTimestampV	输入编码时间戳图像的 V 数据地址
	m_ulOffsetX	输入编码时间戳图像的宽(必须是 16 的倍数)

结构名	T_VIDEOLIB_ENC_TIMESTAMP_PAR	
	m_ulOffsetY	输入编码时间戳图像的高(必须是 16 的倍数)
	m_ulWidth	输入编码时间戳图像相对源图像的水平偏移坐标(必须是 16 的倍数)
	m_ulHeight	输入编码时间戳图像相对源图像的垂直偏移坐标(必须是 16 的倍数)

备注：

1.当 m_pTimestampU 或 m_pTimestampV 至少一个为 NULL 时，m_pTimestampU 和 m_pTimestampV 在库内部使用 m_pTimestampY 计算获得。

2.时间戳图像存放格式为 YUV420

3.当输入时间戳图像 YUV 数据为全 0 时，时间戳图像为透明色，透明色图像不会覆盖源图像

4.2.7 码率控制参数结构体

```
typedef struct
{
    T_U32   m_nvbps;            //bits per second, zero to disable rate control
    T_U16   m_nFPS;             //frame rate
    T_U16   m_nAdjustInterval;  //how many frames should we adjust the QP for encode
    T_U8    m_nMinQP;           //[1,31], and no more than m_nMaxQP
    T_U8    m_nMaxQP;           //[1,31], and no less than m_nMinQP
}T_VIDEOLIB_ENC_RC_PAR;
```

结构名	T_VIDEOLIB_ENC_RC_PAR	
定义概述	编码码率控制参数结构体	
成员说明	m_nvbps	比特率，设置为 0 时表示不进行码率控制，且其它参数也无效
	m_nFPS	帧率
	m_nAdjustInterval	调整 QP 的关键帧帧间隔
	m_nMinQP	编码最小 QP 值，取值范围[1, 31]
	m_nMaxQP	编码最大 QP 值，取值范围[1, 31]

备注：

1、码率的调整是通过升降 QP 值来达到目标码率。QP 含义是量化系数，为 Quantization Parameter 的英文简写。QP 值是编码器用于控制编码后码流解码图像效果的重要参数，QP 值越大，则码率越小，图像效果越差，QP 值越小，则码率越大，图像效果越好。

2、m_nvbps 为需要设置的目标码率，其值不能小于(5<<10),若小于则码率设置无效。

3、m_nAdjustInterval 为调整 QP 的关键帧帧间隔，其值若设置为 0，则编码库以一秒内的数据量来调整一次 QP 值。若 m_nAdjustInterval 不为 0，编码库则以一个 m_nAdjustInterval 间隔内的数据量来调整一次 QP，以此来达到目标码率值。

4、m_nMinQP 和 m_nMaxQP 为提供给外部用来灵活控制编码 QP 的最小值和最大值。

4.3 接口函数列表

4.3.1 VideoLib_GetVersion

原　　型	const T_CHR *VideoLib_GetVersion(T_VOID)
功能概述	获取版本号
参数说明	-
返回值说明	返回版本号信息
返回值说明	版本号字符串
注意事项	无
调用示例	-

4.3.2 VideoStream_Init

原　　型	T_BOOL VideoStream_Init(const T_MEDIALIB_INIT_INPUT *init_input, const T_MEDIALIB_INIT_CB *init_cb_fun)	
功能概述	初始化媒体播放库，全局资源初始化	
参数说明	init_input	初始化输入信息：其中芯片类型必须设置正确，否则可能造成死机
	init_cb_fun	全局使用的回调函数，主要是打印函数
返回值说明	初始化的结果	
返回值说明	AK_TRUE　　初始化成功 AK_FALSE　　初始化失败	
注意事项	1. 在所有播放操作之前，即系统启动时必须首先调用本函数，整个系统运行过程中必须并且只能调用一次 2. 在执行该函数时必须提供相应的回调函数	

原　　型	T_BOOL VideoStream_Init(const T_MEDIALIB_INIT_INPUT *init_input, const T_MEDIALIB_INIT_CB *init_cb_fun)
调用示例	见典型调用示例

4.3.3 VideoStream_Destroy

原　　型	T_VOID VideoStream_Destroy(T_VOID)
功能概述	全局资源释放
参数说明	-
返回值说明	-
注意事项	-
调用示例	见典型调用示例

4.3.4 VideoStream_Open

原　　型	T_pVOID VideoStream_Open(T_VIDEOLIB_OPEN_INPUT *open_input);	
功能概述	打开流式视频解码器，并初始化相关资源	
参数说明	open_input	见 T_VIDEOLIB_OPEN_INPUT
返回值说明	返回解码器句柄	
返回值说明	AK_NULL：打开失败； 其它：打开成功	
注意事项	-	
调用示例	见典型调用示例	

4.3.5 VideoStream_Close

原　　型	T_BOOL VideoStream_Close(T_pVOID hVS);	
功能概述	关闭流式视频解码器，并释放相关资源	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
返回值说明	返回关闭解码器的结果	
返回值说明	AK_TRUE　　关闭成功 AK_FALSE　　关闭失败	

原　　型	T_BOOL VideoStream_Close(T_pVOID hVS);
注意事项	-
调用示例	见典型调用示例

4.3.6 VideoStream_GetLastError

原　　型	T_eVIDEOLIB_ERROR VideoStream_GetLastError(T_pVOID hVS);	
功能概述	获取最近一次解码状态	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
返回值说明	返回解码状态	
返回值说明	见 T_eVIDEOLIB_ERROR	
注意事项	必须已经成功执行 VideoStream_Open 函数	
调用示例	见典型调用示例	

4.3.7 VideoStream_Reset

原　　型	T_BOOL VideoStream_Reset(T_pVOID hVS);	
功能概述	解码器复位	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
返回值说明	返回复位结果	
返回值说明	AK_TRUE　　复位成功 AK_FALSE　　复位失败	
注意事项	必须已经成功执行 VideoStream_Open 函数	
调用示例	见典型调用示例	

4.3.8 VideoStream_GetAddr

原　　型	T_pVOID VideoStream_GetAddr(T_pVOID hVS, T_U32 size);	
功能概述	获取缓冲地址	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
	size	需缓冲的数据空间大小
返回值说明	返回缓冲地址	

原　　型	T_pVOID VideoStream_GetAddr(T_pVOID hVS, T_U32 size);
返回值说明	AK_NULL：获取失败； 其它：获取成功
注意事项	必须已经成功执行 VideoStream_Open 函数
调用示例	见典型调用示例

4.3.9 VideoStream_UpdateAddr

原　　型	T_BOOL VideoStream_UpdateAddr(T_pVOID hVS, T_pVOID pInaddr, T_U32 size);	
功能概述	更新缓冲	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
	pInaddr	VideoStream_GetAddr 的返回值
	size	与 VideoStream_GetAddr 输入相同的 size
返回值说明	返回更新是否成功	
返回值说明	AK_TRUE　　停止成功 AK_FALSE　　停止失败	
注意事项	1. 必须已经成功执行 VideoStream_Open 函数 2. 必须已经成功执行 VideoStream_GetAddr 函数	
调用示例	见典型调用示例	

4.3.10 VideoStream_GetFreeSpace

原　　型	T_U32 VideoStream_GetFreeSpace(T_pVOID hVS);	
功能概述	获取剩余空间信息	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
返回值说明	剩余空间大小，字节为单位	
返回值说明	-	
注意事项	必须已经成功执行 VideoStream_Open 函数	
调用示例	见典型调用示例	

4.3.11 VideoStream_GetItemNum

原　　型	T_S32 VideoStream_GetItemNum(T_pVOID hVS);	
功能概述	获取库内部空间中未解码的视频帧数	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
返回值说明	库内部空间中未解码的视频帧数	
返回值说明	<0：获取失败； >=0：库内部空间中未解码的视频帧数	
注意事项	必须已经成功执行 VideoStream_Open 函数，且是使用库内部空间存储视频码流的方式（即：VideoStream_Open 时，m_ulBufSize 参数不为 0）	
调用示例	VideoStream_GetItemNum(hVS);	

4.3.12 VideoStream_SYNDecodeHeader

原　　型	T_S32 VideoStream_SYNDecodeHeader(T_pVOID hVS);	
功能概述	执行解码头信息	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
返回值说明	解码头信息结果	
返回值说明	<=0：解码失败； 1：解码成功 2：未知或错误的头数据	
注意事项	必须已经成功执行 VideoStream_Open 函数，与 VideoStream_SYNDecode 函数搭配使用	
调用示例	见典型调用示例	

4.3.13 VideoStream_SYNDecode

```
typedef struct
{
        T_U32   m_ulSize;
        T_U16   m_uDispWidth;
        T_U16   m_uDispHeight;
        T_U8    *m_pBuffer;
```

```
    T_U16   m_uOriWidth;            //原始宽，不一定是 16 的倍数
    T_U16   m_uOriHeight;           //原始高，不一定是 16 的倍数
    T_U8    *m_pBuffer_u;
    T_U8    *m_pBuffer_v;
}T_VIDEO_DECODE_OUT;
```

结构名	T_VIDEO_DECODE_OUT	
定义概述	视频缓冲区结构体	
成员说明	m_ulSize	缓冲区实际大小
	m_uDispWidth	用于显示的图像宽（16 的倍数）
	m_uDispHeight	用于显示的图像高（16 的倍数）
	m_pBuffer	输出图像 Y 的 Buffer 指针
	m_uOriWidth	原始宽，不一定是 16 的倍数
	m_uOriHeight	原始高，不一定是 16 的倍数
	m_pBuffer_u	输出图像 U 的 Buffer 指针
	m_pBuffer_v	输出图像 V 的 Buffer 指针

原　　型	T_S32 VideoStream_SYNDecode(T_pVOID hVS, T_VIDEO_DECODE_OUT *pVideoDecOut, T_S32 ulMilliSec);	
功能概述	执行同步解码	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
	pVideoDecOut	见 T_VIDEO_DECODE_OUT
	ulMilliSec	同步时间
返回值说明	视频时间	
返回值说明	小于 0 表示解码异常； 其它表示视频时间，以毫秒为单位	
注意事项	必须已经成功执行 VideoStream_Open 函数	
调用示例	见典型调用示例	

4.3.14 VideoStream_DecodeHeader

原　　型	T_S32 VideoStream_DecodeHeader(T_pVOID hVS, T_pDATA vlc_data, T_U32 vlc_len)	
功能概述	解码头信息	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
	vlc_data	码流首地址
	vlc_len	码流长度
返回值说明	解码头信息的结果	
返回值说明	<=0：解码失败； 1：解码成功 2：未知或错误的头数据	
注意事项	1. 必须已经成功执行 VideoStream_Open 函数 2. 与 VideoStream_Decode 函数搭配使用	
调用示例	-	

4.3.15 VideoStream_Decode

原　　型	T_S32 VideoStream_Decode(T_pVOID hVS, T_pVOID pStreamBuf, T_U32 ulStreamLen, T_VIDEO_DECODE_OUT *pVideoDecOut)	
功能概述	解码一帧	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
	pStreamBuf	码流首地址，前 8 字节保留，实际码流数据从 8 字节后开始
	ulStreamLen	码流长度，实际码流字节数加 8
	pVideoDecOut	返回解码后的图像，以及宽高信息，见 T_VIDEO_DECODE_OUT
返回值说明	解码是否成功	
返回值说明	<0：解码失败； >=0：解码成功	
注意事项	1、必须已经成功执行 VideoStream_Open 函数 2、 执行该函数时由调用方实现同步算法	
调用示例	-	

4.3.16 VideoStream_DecodeChangeSpeed

原　　型	T_S32 VideoStream_DecodeChangeSpeed(T_pVOID hVS, T_S8 nStep)	
功能概述	调节解码速度	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
	nStep	解码级别的调节尺度
返回值说明	调节后的解码速度级别	
返回值说明	解码速度级别值越小表示丢帧越少	
注意事项	1. 必须已经成功执行 VideoStream_Open 函数 2. 在视频落后于音频时 nStep 为正数，在视频快过音频时 nStep 为负数 3. 与 VideoStream_Decode 函数搭配使用	
调用示例	-	

4.3.17 VideoStream_DecodeGetPTS

原　　型	T_S32 VideoStream_DecodeGetPTS(T_pVOID hVS)	
功能概述	获取时间戳信息	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
返回值说明	输出视频帧的时间戳	
返回值说明	-	
注意事项	1. 必须已经成功执行 VideoStream_Open 函数 2. 必须在输入视频码流时给入了正确的时间戳信息	
调用示例		

4.3.18 VideoStream_DecodeGetInfo

```
typedef struct
{
        T_U32   m_Width;        //图像宽度，16 象素倍数
        T_U32   m_Height;       //图像高度，16 象素倍数
        T_U32   m_OriWidth;     //图像宽度，可能非 16 象素倍数
        T_U32   m_OriHeight;    //图像高度，可能非 16 象素倍数
}T_VIDEOLIB_INFO;
```

原　　型	T_BOOL VideoStream_DecodeGetInfo(T_pVOID hVS, T_VIDEOLIB_INFO *pInfo)	
功能概述	获取视频尺寸信息	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
	pInfo	输出值，视频尺寸信息
返回值说明	获取是否成功	
返回值说明	-	
注意事项	1. 必须已经成功执行 VideoStream_Open 函数 2. 必须成功解码至少一帧之后调用	
调用示例	-	

4.3.19 VideoStream_GetNextPts

原　　型	T_S32 VideoStream_GetNextPts(T_pVOID hVS)	
功能概述	获取下一帧待解码码流数据中的时间戳信息	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
返回值说明	时间戳信息	
返回值说明	小于 0 表示异常； 其它表示视频时间戳，以毫秒为单位	
注意事项	1、必须已经成功执行 VideoStream_Open 函数 2、获取到的是视频 FIFO 中接下来解码的一帧的时间戳，一般在全关键帧解码时使用	
调用示例	见 VideoStream_SpecialDecode	

4.3.20 VideoStream_SpecialDecode

原　　型	T_S32 VideoStream_SpecialDecode(T_pVOID hVS, T_VIDEO_DECODE_OUT *pVideoDecOut, T_BOOL bDrop)	
功能概述	在全 I 帧解码时，根据需求进行解码或丢帧	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
	pVideoDecOut	见 T_VIDEO_DECODE_OUT
	bDrop	AK_TRUE：丢弃待解码的一帧； AK_FALSE：解码一帧；

原　　型	T_S32 VideoStream_SpecialDecode(T_pVOID hVS, T_VIDEO_DECODE_OUT *pVideoDecOut, T_BOOL bDrop)
返回值说明	视频时间
返回值说明	小于 0 表示解码异常； 其它表示视频时间，以毫秒为单位 当 bDrop 为 AK_TRUE 时，不进行解码，返回的是上次解码的时间信息
注意事项	1、必须已经成功执行 VideoStream_Open 函数 2、在视频缓冲中的数据全是关键帧时调用，是为了配合媒体库在快进/快退状态的解码 3、必须保证是全关键帧，否则在设置丢帧后再解码图像可能异常

调用示例	T_S32 video_pts;
	T_BOOL bDrop = AK_FALSE;
	...
	//进入全关键帧解码
	VideoStream_Reset(hVS);
	while (!bExit)
	{
	video_pts = VideoStream_GetNextPts(hVS);
	if (video_pts >= 0)
	{
	//判决接下来的帧是否丢弃，判决条件由调用者决定
	bDrop = Judge_Drop(video_pts);
	//解码或丢弃
	video_pts = VideoStream_SpecialDecode(hVS, pVideoDecOut, bDrop);
	}
	}
	//退出全关键帧解码
	VideoStream_Reset(hVS);

4.3.21 VideoStream_SetRotate

原　　型	T_BOOL VideoStream_SetRotate(T_pVOID hVS, T_eMEDIALIB_ROTATE rotate)	
功能概述	设置解码旋转角度	
参数说明	hVS	VideoStream_Open 时返回的解码器句柄
	rotate	旋转角度，见 T_eMEDIALIB_ROTATE
返回值说明	设置是否成功	
返回值说明	AK_TRUE　　设置成功 AK_FALSE　　设置失败	
注意事项	1、必须已经成功执行 VideoStream_Open 函数 2、目前该函数仅对于 h263、mpeg4sp、flv263 三种视频解码格式有效	
调用示例	VideoStream_SetRotate (hVS, MEDIALIB_ROTATE_90);	

4.3.22 VideoStream_Enc_Open

原　　型	T_pVOID VideoStream_Enc_Open(const T_VIDEOLIB_ENC_OPEN_INPUT *open_input)	
功能概述	打开视频编码器，并初始化相关资源	
参数说明	open_input	见 T_VIDEOLIB_ENC_OPEN_INPUT
返回值说明	AK_NULL：打开失败； 其它：打开成功	
注意事项	-	
调用示例	见典型调用示例	

4.3.23 VideoStream_Enc_Close

原　　型	T_VOID VideoStream_Enc_Close(T_pVOID hVS);	
功能概述	关闭视频编码器，并释放相关资源	
参数说明	hVS	VideoStream_Enc_Open 时返回的编码器句柄
返回值说明	T_VOID	
返回值说明	无	
注意事项	必须已经成功执行 VideoStream_Enc_Open 函数	

原　　型	T_VOID VideoStream_Enc_Close(T_pVOID hVS);
调用示例	见典型调用示例

4.3.24 VideoStream_Enc_Encode

原　　型	T_S32 VideoStream_Enc_Encode(T_pVOID hVS, T_VIDEOLIB_ENC_IO_PAR *video_enc_io_param);	
功能概述	编码一帧	
参数说明	hVS	VideoStream_Enc_Open 时返回的编码器句柄
	video_enc_io_param	见 T_VIDEOLIB_ENC_IO_PAR
返回值说明	编码后码流长度	
返回值说明	>0：编码成功； 其它：编码失败	
注意事项	必须已经成功执行 VideoStream_Enc_Open 函数	
调用示例	见典型调用示例	

4.3.25 VideoStream_Enc_GetDispData

原　　型	T_pDATA VideoStream_Enc_GetDispData(T_pVOID hVS);	
功能概述	获取最后一次调用 VideoStream_Enc_Encode 编码后的重构图像，YUV 格式（YUV420 或 YUV422，与编码输入的 YUV 数据格式一致）； 可用于显示，预览录制效果	
参数说明	hVS	VideoStream_Enc_Open 时返回的编码器句柄
返回值说明	编码后重构图像的地址	
返回值说明	AK_NULL：获取失败； 其它：表示重构图像首地址	
注意事项	必须已经成功执行 VideoStream_Enc_Open 函数	
调用示例	见典型调用示例	

4.3.26 VideoStream_Enc_SetTimeStamp

原　　型	T_S32 VideoStream_Enc_SetTimeStamp(T_pVOID hVS, T_VIDEOLIB_ENC_TIMESTAMP_PAR *timestamp_param);	
功能概述	设置编码时间戳参数	
参数说明	hVS	VideoStream_Enc_Open 时返回的编码器句柄
	timestamp_param	见 T_VIDEOLIB_ENC_TIMESTAMP_PAR
返回值说明	T_S32	
返回值说明	返回-1 表示设置编码时间戳参数失败 返回 0 表示设置编码时间戳参数成功	
注意事项	必须已经成功执行 VideoStream_Enc_Open 函数	
调用示例	-	

4.3.27 VideoStream_Enc_SetBitrate

原　　型	T_BOOL VideoStream_Enc_SetBitrate(T_pVOID hVS, T_VIDEOLIB_ENC_RC_PAR *rc_param);	
功能概述	设置码率控制参数	
参数说明	hVS	VideoStream_Enc_Open 时返回的编码器句柄
	rc_param	见 T_VIDEOLIB_ENC_RC_PAR
返回值说明	T_BOOL	
返回值说明	返回 AK_FALSE 表示设置码率控制参数失败 返回 AK_TRUE 表示设置码率控制参数成功	
注意事项	必须已经成功执行 VideoStream_Enc_Open 函数	
调用示例	-	

4.3.28 VideoStream_Enc_Reset

原　　型	T_BOOL VideoStream_Enc_Reset(T_pVOID hVS);	
功能概述	重置编码库参数值(针对循环录像)	
参数说明	hVS	VideoStream_Enc_Open 时返回的编码器句柄
返回值说明	T_BOOL	

原　　型	T_BOOL VideoStream_Enc_Reset(T_pVOID hVS);
返回值说明	返回 AK_FALSE 表示重置参数失败 返回 AK_TRUE 表示重置参数成功
注意事项	必须已经成功执行 VideoStream_Enc_Open 函数
调用示例	-

4.4 典型调用范例

4.4.1 解码

解码部分 1：使用 fifo 进行同步解码(open_input.m_ulBufSize 不能为 0)

```
T_pVOID g_hVS;
T_VIDEOLIB_OPEN_INPUT open_input;
T_S32 audio_timestamp;
T_S32 video_timestamp;
T_VIDEO_DECODE_OUT VideoDecOut;

memset(&open_input, 0, sizeof(T_VIDEOLIB_OPEN_INPUT));
open_input.m_VideoType = VIDEO_DRV_MPEG;
open_input.m_ulBufSize = 200*1024;
open_input.m_bNeedSYN = AK_TRUE;
open_input.m_bFixedStream = AK_FALSE;
open_input.m_uWidth = 640;
open_input.m_uHeight = 480;

set_callback_func(&open_input.m_CBFunc);          //设置回调函数

g_hVS = VideoStream_Open(&open_input);
if (AK_NULL == g_hVS)
{
        return;
}
```

```
if (VideoStream_SYNDecodeHeader(g_hVS) < 0)
{
        VideoStream_Close(g_hVS);
        return;
}

while(1)
{
        if (pause_flag) //暂停后需要重新缓冲数据
        {
                if (VideoStream_Reset(g_hVS) == AK_FALSE)
                {
                        break;
                }
        }
        audio_timestamp = get_audio_timestap();    //由音频库实现
        video_timestamp = VideoStream_SYNDecode(g_hVS, &VideoDecOut,
                                                audio_timestamp);
        if (VideoDecOut.m_pBuffer != AK_NULL)
        {
                Display(VideoDecOut.m_pBuffer,
                VideoDecOut.m_pBuffer_u,
                VideoDecOut.m_pBuffer_v,
                VideoDecOut.m_uDispWidth,
                VideoDecOut.m_uDispHeight);
        }
        if (stop_flag)//检测到停止标志后退出 while 循环
        {
                break;
        }
}
VideoStream_Close(g_hVS);
```

数据处理部分（即之前提到的回调函数）：

```
T_BOOL video_call_back(DWORD timestamp_ms, void *data_pointer, DWORD length)
{
        T_pDATA video_buf = AK_NULL;
        video_buf = VideoStream_GetAddr(g_hVS, length + 8);
        //预留前 8 字节，因此要多加 8
        if (AK_NULL == video_buf)
        {
                return AK_FALSE;
        }
        memcpy(video_buf + 4, &timestamp_ms, 4);
        //将 timestamp 放入 5-8 字节，1-4 字节保留
        memcpy(video_buf + 8, data_pointer, length); //将视频数据拷入 buffer
        return VideoStream_UpdateAddr(g_hVS, video_buf, length + 8);//更新视频数据
}
```

****

```
//解码部分 2：不使用 fifo，输入一帧解码一帧(open_input.m_ulBufSize 必须为 0)
T_pVOID g_hVS;
T_VIDEOLIB_OPEN_INPUT open_input;
T_VIDEO_DECODE_OUT VideoDecOut;
T_pVOID pVlcBuf;
T_U32 vlc_len;
T_S32 retval;

memset(&open_input, 0, sizeof(T_VIDEOLIB_OPEN_INPUT));
open_input.m_VideoType = VIDEO_DRV_MPEG;
open_input.m_ulBufSize = 0;
open_input.m_bNeedSYN = AK_TRUE;
open_input.m_bFixedStream = AK_FALSE;
open_input.m_uWidth = 640;
open_input.m_uHeight = 480;
```

```
set_callback_func(&open_input.m_CBFunc);          //设置回调函数
g_hVS = VideoStream_Open(&open_input);
if (AK_NULL == g_hVS)
{
        return;
}
vlc_len = get_vlc_data_header(pVlcBuf);    //获取码流数据
if (VideoStream_DecodeHeader(g_hVS, pVlcBuf, vlc_len) != 1)
{
        VideoStream_Close(g_hVS);
        return;
}
while(1)
{
        vlc_len = get_vlc_data(pVlcBuf);   //获取码流数据
        retval = VideoStream_Decode(g_hVS, pVlcBuf, vlc_len, &VideoDecOut);
        if (VideoDecOut.m_pBuffer != AK_NULL)
        {
                Display(VideoDecOut.m_pBuffer,
                VideoDecOut.m_pBuffer_u,
                VideoDecOut.m_pBuffer_v,
                VideoDecOut.m_uDispWidth,
                VideoDecOut.m_uDispHeight);
        }
        if (stop_flag)//检测到停止标志后退出 while 循环
        {
                break;
        }
}
VideoStream_Close(g_ hVS);
```

4.4.2 编码

```
T_pVOID g_hVS;
T_VIDEOLIB_ENC_OPEN_INPUT open_input;
T_VIDEOLIB_ENC_IO_PAR video_enc_io_param;

T_U8 curr_buf[PIC_SIZE];
T_U8 vlc_buf[STREAM_MAX_SIZE];
T_U8 *out_pic = AK_NULL;

memset(&open_input, 0, sizeof(T_VIDEOLIB_ENC_OPEN_INPUT));
open_input.m_VideoType = VIDEO_DRV_H263;
open_input.m_ulWidth = 352;
open_input.m_ulHeight = 288;
open_input.m_ulMaxVideoSize =
                (((open_input.m_ulWidth*open_input.m_ulHeight>>1)+511)/512)*512;

set_callback_func(&open_input.m_CBFunc);          //设置回调函数

g_hVS = VideoStream_Enc_Open(&open_input);
if (AK_NULL == g_hVS)
{
        return;
}

video_enc_io_param.p_curr_data = curr_buf;
video_enc_io_param.p_vlc_data = vlc_buf;
video_enc_io_param.QP = 10;
video_enc_io_param.mode = 0;
video_enc_io_param.bInsertP = AK_FALSE;

while (1)
{
        video_timestamp = VideoStream_Enc_Encode(g_hVS, &video_enc_io_param);
```

```
            out_pic = VideoStream_Enc_GetDispData(g_hVS);
            if (out_pic != AK_NULL)
            {
                    Display(out_pic, open_input.m_ulWidth, open_input.m_ulHeight);
            }
            video_enc_io_param.mode = 1;      //根据需要改变 mode、QP、bInsertP 等

            if (stop_flag)  //检测到停止标志后退出 while 循环
            {
                    break;
            }
    }

    VideoStream_Enc_Close(g_hVS);
```

5 依赖接口说明

从 3.1 层次关系可以看出视频驱动库所依赖的外部接口主要有内存管理及其它一些视频驱动库需要的功能接口，该类函数由目标平台实现。参见《音视频库总体使用说明》。

6 常见问题

6.1 编解码常见问题

1、调用 VideoStream_Open 失败

检查是否成功调用 MediaLib_Init 或 VideoStream_Init；检查输入参数是否正确，注意观察打印。

2、调用 VideoStream_Enc_Open 失败

检查是否成功调用 MediaLib_Init 或 VideoStream_Init；检查输入参数是否正确，注意观察打印。

3、同步解码后显示，出现视频卡的现象

a. 检查是否选择同步解码方式；

b. 检查调用视频的频率是否大于帧率，如果调用视频间隔太长会导致不停进行重同步，表现为视频卡；

c. 是否误码太多，基本无法解码出正常图像；

d. 检查同步解码两次输入的同步时间相差太大。

4、解码后图像显示不正常

检查显示时宽高是否正确。

6.2 其它问题

参见《音视频库总体使用说明》。