



Hi3520 / Hi3515 媒体处理软件

开发参考

文档版本	08
发布日期	2011-05-05
部件编码	N/A

深圳市海思半导体有限公司为客户提供全方位的技术支持，用户可与就近的海思办事处联系，也可直接与公司总部联系。

深圳市海思半导体有限公司

地址：深圳市龙岗区坂田华为基地华为电气生产中心 邮编：518129

网址：<http://www.hisilicon.com>

客户服务电话：+86-755-28788858

客户服务传真：+86-755-28357515

客户服务邮箱：support@hisilicon.com

版权所有 © 深圳市海思半导体有限公司 2009~2011。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。



前言

概述

本文为使用 Hi3520/Hi3515 媒体处理芯片进行开发的程序员而写，目的是供您在开发过程中查阅媒体处理软件开发包的各种参考信息，包括 API、头文件、错误码等。

本文档描述了 Hi3520/Hi3515 媒体处理软件各个 API 的使用方法，以及相关的数据结构和错误码。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3520 H.264 编解码处理器	V100
Hi3515 H.264 编解码处理器	V100

读者对象


本文档主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师





约定

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。



符号	说明
 警告	表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
 注意	表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
 窍门	表示能帮助您解决某个问题或节省您的时间。
 说明	表示是正文的附加信息，是对正文的强调和补充。

通用格式约定

格式	说明
宋体	正文采用宋体表示。
黑体	一级、二级、三级标题采用 黑体 。
楷体	警告、提示等内容一律用楷体，并且在内容前后增加线条与正文隔离。
“Terminal Display” 格式	“Terminal Display” 格式表示屏幕输出信息。此外，屏幕输出信息中夹杂的用户从终端输入的信息采用加粗字体表示。
“ ”	用双引号表示文件路径。如 “C:\Program Files\Huawei”。

命令行格式约定

格式	意义
粗体	命令行关键字（命令中保持不变、必须照输的部分）采用 加粗 字体表示。
<i>斜体</i>	命令行参数（命令中必须由实际值进行替代的部分）采用 <i>斜体</i> 表示。
[]	表示用 “[]” 括起来的部分在命令配置时是可选的。
{ x y ... }	表示从两个或多个选项选取一个。
[x y ...]	表示从两个或多个选项选取一个或者不选。
{ x y ... } *	表示从两个或多个选项选取多个，最少选取一个，最多选取所有选项。



格式	意义
[x y ...] *	表示从两个或多个选项中选取多个或者不选。

数值单位约定

数据容量、频率、数据速率等的表达方式说明如下。

类别	符号	对应的数值
数据容量（如 RAM 容量）	1K	1024
	1M	1,048,576
	1G	1,073,741,824
频率、数据速率等	1k	1000
	1M	1,000,000
	1G	1,000,000,000

地址、数据的表达方式说明如下。

符号	举例	说明
0x	0xFE04、0x18	用 16 进制表示的数据值、地址值。
0b	0b000、0b00 00000000	表示 2 进制的数据值以及 2 进制序列（寄存器描述中除外）。
X	00X、1XX	在数据的表达方式中，X 表示 0 或 1。 例如：00X 表示 000 或 001； 1XX 表示 100、101、110 或 111。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2011-05-04	08	第 3 章 视频输入 3.3 API 参考中新增 HI_MPL_VI_SetUserPicEx ； HI_MPL_VI_SetUserPic 的【注意】中增加“增加用户图片和 ViOverlay 不能共同使用”的说明；HI_MPL_VI_SetSrcCfg 的【注意】中增加“无中断检测功能”的说明；数据结构



修订日期	版本	修订说明
		<p>VI_SRC_CFG_S 增加成员 bEnNoIntDet; 新增 VI_USERPIC_MODE_E、VI_USERPIC_BGC_S 和 VI_USERPIC_ATTR_S。</p> <p>第 9 章 音频</p> <p>表 9-1 中 AAC Decoder 协议支持的采样率改为“兼容全部速率”。</p> <p>HI_MPI_AI_EnableReSmp、HI_MPI_AO_EnableReSmp、的【注意】中增加音频重采样属性 enReSampleType 的采样类型 6 到 1 倍。</p> <p>AUDIO_RESAMPLE_TYPE_E 中增加成员 AUDIO_RESAMPLE_1X6 和 AUDIO_RESAMPLE_6X1 及其描述。</p> <p>第 10 章 Proc 调试信息</p> <p>10.11 MD、新增 ddr16 及描述; 10.13 AI、10.14 AO 新增 use_dd16 及描述。</p> <p>修改 10.15 AENC 的音频编码通道状态参数及其描述。</p>
2010-10-09	07	<p>第 3 章 视频输入</p> <p>3.3 API 参考中新增 HI_MPI_VI_SetSrcCfg 和 HI_MPI_VI_GetSrcCfg。</p> <p>3.4 数据类型中新增 VI_SRC_FIELD_E 和 VI_SRC_CFG_S。</p> <p>第 4 章 视频输出</p> <p>4.4 数据类型中 VO_PUB_ATTR_S 下【注意事项】中第三个 list 中 enIntfSync 的取值范围由[5, 9]改为[7, 13]; 第四个 list 中 enIntfSync 的取值范围由[2, 4]改为[2, 6]。</p> <p>第 5 章 视频前处理</p> <p>5.4 数据类型中 VIDEO_PREPROC_CONF_S 的【定义】中新增时域滤波的三个参数; 【成员】中增加成员关于默认值的描述及 s32TfBase、s32TfRate 和 s32ChrTFDelta 的描述。</p> <p>第 6 章 视频编码</p> <p>6.4 数据类型中 VENC_ATTR_H264_S 的成员 u32ViFramerate 和 u32TargetFramerate 的描述中删除 N/P 制之分, 最大限制由 30 修改为 60。</p> <p>第 9 章 音频</p> <p>9.2 重要概念中“音频接口时序”下增加“SIO 接口支持 PCM 模式”和“Hi3520/Hi3515 SIO 支持标准 PCM 模式和自定义 PCM 模式”; “AI、AO 通道排列”增加 AO 功能相关对接 codec 说明。</p> <p>9.3.1 音频输入的 HI_MPI_AI_SetPubAttr 下【注意】的第一个 list 中增加工作模式支持 PCM 模式的说明; 删除采样位宽可设置 32bit 的说明。</p>



修订日期	版本	修订说明
2010-08-15	06	<p>第 4 章 视频输出</p> <p>4.3 API 参考中新增 HI_MPI_VO_SetSolidDraw 和 HI_MPI_VO_GetSolidDraw。</p> <p>HI_MPI_VO_EnableVideoLayer 的【举例】中增加 stRect 的定义。</p> <p>HI_MPI_VO_ChnPause 的【举例】中代码有更新。</p> <p>HI_MPI_VO_SyncGroupStart 的【举例】中删除参数 u64BasePts 后面原多余的部分。</p> <p>HI_MPI_VO_GetChnPts 的【举例】中最后的代码有更新。</p> <p>4.4 数据类型中新增 VO_DRAW_CFG_S 和 VO_DRAW_ATTR_S。</p> <p>第 5 章 视频前处理</p> <p>5.2 重要概念中增加 CoverEx Region</p> <p>HI_MPI_VPP_CreateRegion 的【注意】中增加“扩展视频遮挡区域”的描述。</p> <p>表 5-1 中增加扩展视频遮挡 COVEREX 一行。</p> <p>5.4 数据类型增加 MAX_COVEREX_REGION_NUM、COVEREX_ATTR_S 和 COVEREX_S。</p> <p>REGION_ATTR_U 的【定义】和【成员】中新增 stCoverEx 的定义和描述；【相关数据类型及接口】中增加 COVEREX_ATTR_S。</p> <p>第 10 章 Proc 调试信息</p> <p>10.4 LOG 的【调试信息分析】中修改“cat /proc/umap/mstlog 用于获取 Hi3520 的主 ARM 上的 log 级别信息”。</p> <p>10.6 VO 的【调试信息】中修改 DEV CONFIG 下 Mux 和 OutMode、CHN INFO 下 Field、CHN STATUS 下 SndTyp 的取值。</p> <p>10.12 VDEC 的【参数说明】中补充 MODULE PARAM 下 nc_alg 取值为 1 是默认值的说明。</p>
2010-06-29	05	<p>第 2 章 系统控制</p> <p>2.4.1 基本数据类型中 VO 设备的最大个数由 3 改为 6，其中包括 3 个虚拟设备；增加 VO 设备中物理设备的最大个数、VO 最小显示 buffer 数、VO 最大显示 buffer 数、VO 最小虚拟设备 buffer 数、VO 最大虚拟设备 buffer 数的定义。</p> <p>第 4 章 视频输出</p> <p>HI_MPI_VO_SetPubAttr 的注意中增加“虚拟设备不受公共属性限制，因此设置虚拟设备属性时该接口返回成功。”的描述。</p> <p>HI_MPI_VO_GetScreenFrame 增加 2 条注意。</p> <p>HI_MPI_VO_DisableVideoLayer 在【错误码】中增加错误码 HI_ERR_VB_BUSY 并增加一条注意。</p>



修订日期	版本	修订说明
		<p>HI_MPI_VO_GetScreenFrame 在【注意】中增加一条注意。</p> <p>HI_MPI_VO_SetDispBufLen 函数中 u32BufLen 的参数中增加物理设备和虚拟设备的取值范围；【注意】中缓冲长度的默认值区分物理设备和虚拟设备；删除“缓冲长度不是默认值的相关描述”；“缓冲长度越大，即用户可以更长时间的持有显示数据。”改为“缓冲长度越大，即用户可以持有的显示图像越多。”；新增 2 条注意事项。</p> <p>VO_DEV 的成员 VO_DEV 的描述中增加取值 3~5 对应的设备；【注意】中增加关于虚拟设备的注意。</p> <p>VO_VIDEO_LAYER_ATTR_S 的[注意]中增加关于虚拟设备的注意。</p> <p>第 6 章 视频编码</p> <p>HI_MPI_VENC_RegisterChn 的【注意】中修改同组的主次码流宽高必须符合如下约束。</p> <p>HI_MPI_VENC_SetH264RcPara 的【注意】中增加 3 条事项。</p> <p>HI_MPI_VENC_GetH264RcPara 的【注意】中增加 1 条事项。</p> <p>VENC_ATTR_H264_RC_S 的【定义】和【成员】中增加 bFrameLostAllow；修改【成员】中 3 个成员的描述。</p> <p>第 8 章 视频解码</p> <p>8.1 概述中增加 VDEC 实现 MJPEG、JPEG 解码功能的说明。</p> <p>8.2 重要概念中图象输出方式中的“快速输出”改为“直接输出”；增加“超大帧”的描述</p> <p>HI_MPI_VDEC_CreateChn 的【举例】中增加 JPEG 的举例。</p> <p>HI_MPI_VDEC_SetChnAttr 的【举例】中增加 JPEG 的举例。</p> <p>HI_MPI_VDEC_SendStream 的【注意】中增加 1 条事项。</p> <p>HI_MPI_VDEC_ResetChn 的【注意】中增加 1 条事项。</p> <p>增加 HI_MPI_VDEC_SetChnParam、 HI_MPI_VDEC_GetChnParam。</p> <p>8.4 数据类型</p> <p>增加 VDEC_CHN_PARAM_S。</p> <p>VDEC_ATTR_H264_S 的成员 u32RefFrameNum 的描述有修改。</p> <p>第 9 章 音频</p> <p>修改表 9-1 中 AAC Encoder 协议的帧长；增加 AAC Encoder 协议中对 aacPlus1 和 aacPlus2 的补充说明。</p> <p>修改表 9-2 和表 9-3 的表描述；调整表 9-2 的表达样式；统一将 NO 改为 Not supported。</p> <p>HI_MPI_AI_SetPubAttr 的【注意】中对 buffer 的大小重新定义。</p> <p>HI_MPI_AO_SetPubAttr 的【注意】中修改“I²S 主模式时，同一 SIO 下的 AI 和 AO 设备的时钟选择配置必须相同”。</p> <p>HI_MPI_AENC_CreateChn 的【注意】中增加对 buffer 的建议</p>



修订日期	版本	修订说明
		<p>配置值。</p> <p>HI_MPI_ADEC_CreateChn 的【注意】中增加对 buffer 的建议配置值。</p> <p>HI_MPI_ADEC_SendStream 的【注意】中增加“AAC 解码只支持 stream 方式；LPCM 解码只支持 pack 方式；其他音频格式的解码两种方式都支持，但正式应用中建议只使用 pack 方式”的描述。</p> <p>ADEC_ATTR_AMR_S 的【定义】有修改。</p> <p>第 10 章 Proc 调试信息</p> <p>10.1 概述中删除“不同版本中，这些调试信息可能不同，现以 Hi3520_SDK_V1.0.2.4 版本为准”的说明；修改关于“Hi3520 与 Hi3515 显示的 proc 信息格式”的相关描述。</p> <p>10.6 VO 增加级联的相关调试信息及参数说明。</p> <p>10.8 VENC 修改 VENC CHN STATE 的 UserGet 和 userRls 参数说明。</p> <p>10.12 VDEC 增加 MODULE PARAM 的相关调试信息及参数说明。</p> <p>10.18 H264E 更新调试信息及参数说明。</p> <p>10.19 H264D 中的 Const 增加参数 FrmMax 及相应说明、Err Stat 增加参数 ErrBigF 及相应说明。</p>
2010-05-29	04	<p>第 3 章 视频输入</p> <p>修改表 3-1 下的表说明 b 中色度重采样 ChromaRes 的配置要求，由“一般配置为 False 既可”改为“详细说明请参考《Hi3520 H.264 编解码处理器用户指南》，没有特殊要求则配置为 False。”</p> <p>表 3-1 下的关于采集区域 CapRect 的配置说明中增加“CapRect 中所有项的取值都必须是偶数，其中宽度值还要求必须是 4 的整数倍”。</p> <p>第 4 章 视频输出</p> <p>对 API 进行分类及顺序调整。</p> <p>HI_MPI_VO_SetChnFilter 的【描述】中将“利用 TDE 单元进行缩放”改为“利用 DSU 单元进行缩放”；将系统内部提供滤波系数的相关说明改为系统提供的滤波参数和系数的相关说明；将“通过此接口可以设置缩放系数”改为“设置缩放类型”；【注意】中增加“设置 enFilterType”和“当用户对缩放效果有特定需求时”的注意事项。</p> <p>HI_MPI_VO_SetZoomInRatio 的【注意】中局部放大比例结构体中的设置值由原来的“表示为百分之多少”改为“表示千分之多少”，同时删除“最后一位表示精度”的说明；增加“当混合输入的时候，起始坐标的横坐标、宽度在大图像上会自从以 4 对齐（向靠近 0 的方向）”的说明。</p> <p>增加设置有效区处理模式时的视频层属性的相关概念示意图</p>



修订日期	版本	修订说明
		<p>4-2。</p> <p>VO_CHN_FILTER_S 的【定义】和【成员】中增加 enFiltType 的描述。</p> <p>增加图 4-3 局部放大原理示意。</p> <p>第 5 章 视频前处理</p> <p>增加接口 HI_MPI_VPP_SetDsuFiltParam 和 HI_MPI_VPP_GetDsuFiltParam。</p> <p>接口 HI_MPI_VPP_SetConf 的【注意】中增加关于滤波参数、滤波类型和滤波系数的描述。</p> <p>增加 DSU_HFILTER_PARAM_NUM、DSU_VFILTER_PARAM_NUM、DSU_HSCALE_FILTER_E 和 DSU_FILTER_PARAM_S。</p> <p>VIDEO_PREPROC_CONF_S 的【定义】和【成员】中增加 enFilterType 的相关内容；【注意事项】中增加滤波类型、滤波参数和滤波系数的描述。</p> <p>第 6 章 视频编码</p> <p>增加接口 HI_MPI_VENC_SetMeParam 和 HI_MPI_VENC_GetMeParam。</p> <p>增加数据类型 VENC_ME_PARAM_H264_S 和 VENC_ME_PARAM_S。</p> <p>HI_MPI_VENC_SetParamSet 的【注意】中增加“系统有默认的编码参数，一般情况下无需调用该接口进行设置”的说明。</p> <p>VENC_PARAM_SET_H264_S 的【定义】和【成员】中增加 9 各 H.264 协议语法元素的成员并修改原成员 chroma_qp_index_offset 的描述；【注意事项】中增加“编码参数的默认值，未在本文档描述。请通过接口 HI_MPI_VENC_GetParamSet() 查询最新默认参数”的描述。</p> <p>VENC_ATTR_H264_RC_S 的【定义】和【成员】中增加 s32MaxQP 和 bFrameFixQP 的相关描述。</p> <p>第 9 章 音频</p> <p>表 9-1 前增加 Hi3520、Hi3515 SDK 音频部分的编解码功能说明。</p> <p>表 9-4 前增加海思语音帧结构的说明。</p> <p>HI_MPI_AI_SetPubAttr 的【注意】中主模式下支持的采样率配置；增加 I²S 主模式时的相关说明。</p> <p>第 10 章 Proc 调试信息说明</p> <p>10.1 概述的【信息查看方法】增加“使用其他常用的文件操作命令查看信息”的说明。</p> <p>10.2 SYS 的【调试信息】和【参数说明】中删除参数 PinMuxCtl。</p> <p>10.4 LOG 的【调试信息分析】中增加描述。</p> <p>10.5 VI、10.6 VO、10.7 DSU 的【调试信息】和【参数说明】</p>



修订日期	版本	修订说明
		更新。 10.8 VENC 的【参数说明】中的 RI Frm 和 FrmU 参数说明有更改。 10.9 GROUP、10.11 MD 的【调试信息】有更新。 10.13 AI 的【调试信息】和【参数说明】更新。 10.18 H264E【参数说明】中 rcv、encode、disc、skip、bufLeak、rcLost、back 有更新。 10.19 H264D 的【参数说明】中 ErrNalu、LostPic、ErrSlc、LostSlc、Overlap、ErrW、ErrH、ErrRef、ErrCncl、ErrBigF、RecvPic、DecPic、NoFB 有更新。
2010-04-29	03	第 4 章 视频输出 增加以下接口及相关数据结构：HI_MPI_VO_SetDispBufLen、HI_MPI_VO_GetDispBufLen 第 5 章 视频前处理 增加 ViOverlay 的相关使用说明的数据结构描述。 第 6 章 视频编码 增加以下接口及相关数据结构：HI_MPI_VENC_SetParamSet、HI_MPI_VENC_GetParamSet； HI_MPI_VENC_CreateChn 接口描述中增加分数帧率的配置说明。 第 9 章 音频 修改结构体 AIO_ATTR_S 中 u32ClkSel 项的描述 第 10 章 Proc 信息 更新 VI、VPP、VO、AUDIO 等模块的 Proc 调试信息。



修订日期	版本	修订说明
2010-03-27	02	<p>第 3 章 视频输入</p> <p>增加接口 HI_MPI_VI_ClearChnMinorAttr。</p> <p>第 4 章 视频输出</p> <p>增加以下接口及相应数据结构：HI_MPI_VO_SetValidImgRect、HI_MPI_VO_GetValidImgRect、HI_MPI_VO_SetZoomInRatio、HI_MPI_VO_GetZoomInRatio、HI_MPI_VO_SetScreenFilter、HI_MPI_VO_GetScreenFilter、HI_MPI_VO_SetChnSrcAttr、HI_MPI_VO_SetDevCSC、HI_MPI_VO_GetDevCSC、HI_MPI_VO_SetChnFilter、HI_MPI_VO_GetChnFilter；DSU_HSCALE_FILTER_E、DSU_VSCALE_FILTER_E、VO_CHN_FILTER_S、VO_SCREEN_HFILTER_E、VO_SCREEN_VFILTER_E、VO_SCREEN_FILTER_S、VO_ZOOM_RATIO_S、VO_CSC_S、VO_SRC_ATTR_S。</p> <p>第 5 章 视频前处理</p> <p>删除数据结构 VPP_SCALE_FILTER_E，增加数据结构 DSU_HSCALE_FILTER_E、DSU_VSCALE_FILTER_E；修改数据结构 VIDEO_PREPROC_CONF_S 的定义。</p> <p>第 6 章 视频编码</p> <p>HI_MPI_VENC_CreateChn 的接口描述中将码流宽高 16 对齐修改为 8 对齐，增加支持 176 * 120 分辨率的 NTSC 制 QCIF 码流，增加 1080P 编码规格。</p> <p>第 9 章 音频</p> <p>增加 AI、AO 的重采样功能的详细说明，增加接口 HI_MPI_AO_ClearChnBuf、HI_MPI_ADEC_ClearChnBuf。</p> <p>第 10 章 Proc 调试信息</p> <p>H264E、H264D、JPEGE 中新增加 16 像素对齐的宽高信息及说明。</p>
2009-12-23	01	正式发布。
2009-09-30	00B02	优化部分描述，修改部分错误码。
2009-08-31	00B01	第 1 次发布。



目 录

1 概述.....	1-1
1.1 API 函数参考域.....	1-1
1.2 数据类型参考域.....	1-1



表格目录

表 1-1 API 函数参考域说明	1-1
表 1-2 数据类型参考域说明.....	1-1



1 概述

1.1 API 函数参考域

本文档对 API 参考信息使用以下域来描述，具体信息如[表 1-1](#) 所示。

表1-1 API 函数参考域说明

参考域	作用
描述	描述 API 的功能。
语法	显示 API 的语法样式。
参数	列出 API 的参数、参数说明及其属性。
返回值	列出 API 的返回值及其返回值说明。
错误码	列出 API 的错误码及其错误码说明。
需求	列出本 API 要包含的头文件和 API 编译时要链接的库文件。
注意	使用 API 时应注意的事项。
举例	使用 API 的实例。
相关主题	同本 API 的其他相关信息。

1.2 数据类型参考域

本参考对数据类型使用以下域来描述，具体信息如[表 1-2](#) 所示。

表1-2 数据类型参考域说明

参考域	作用
说明	简单描述结构体所实现的功能。



参考域	作用
定义	列出结构体的定义。
成员	列出数据结构的成员及含义。
注意事项	列出使用数据类型时应注意的事项。
相关数据类型和接口	列出与本数据类型相关联的其他数据类型和接口。



目 录

2 系统控制.....	2-1
2.1 概述.....	2-1
2.2 重要概念.....	2-1
2.3 API 参考	2-1
2.4 数据类型.....	2-23
2.4.1 基本数据类型.....	2-23
2.4.2 系统控制数据类型.....	2-31
2.4.3 视频公共类型.....	2-33
2.5 错误码.....	2-39
2.5.1 系统控制错误码.....	2-39
2.5.2 视频缓存池错误码.....	2-39



表格目录

表 2-1 系统控制 API 错误码	2-39
表 2-2 视频缓存池 API 错误码	2-39



2 系统控制

2.1 概述

单板上电后，启动进入内核，加载 MPP 系统中的各个 ko 以及外围芯片驱动，应用程序启动 MPP 业务前，必须完成 MPP 系统初始化工作。同理，应用程序退出 MPP 业务后，也要完成 MPP 系统去初始化工作，释放资源。

MPP 系统初始化，主要针对视频缓存池、系统控制两大部分进行初始化。同时提供当前 MPP 系统的版本信息，便于产品的维护、更新。

2.2 重要概念

系统控制

系统控制的功能：主要根据 Hi3520/Hi3515 芯片特性，完成硬件各个部件的复位、基本初始化工作，同时负责完成 MPP 系统各个业务模块的初始化、去初始化以及管理 MPP 系统各个业务模块的工作状态。

视频缓存池

视频缓存池的功能：主要向媒体业务提供大块物理内存，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缓存池。在创建编码通道或者解码通道时，MPP 系统会为该通道创建一个独享缓存池。在销毁通道时，同时也会销毁相应的缓存池。而对于视频输入通道，则需要使用公共缓存池。所有的视频输入通道都可以从公共缓存池中获取缓存块。由于视频输入通道不存在创建和销毁，因此，在系统初始化之前，必须为视频输入通道配置公共缓存池。根据业务的不同，公共缓存池的数量、缓存块的大小和数量会有所不同。

2.3 API 参考

系统控制实现 MPP（Media Process Platform）系统初始化、获取 MPP 版本号、视频缓存池初始化、创建视频缓存池等功能。



该功能模块提供以下 MPI：

- [HI_MPI_SYS_SetConf](#)：配置系统控制参数。
- [HI_MPI_SYS_GetConf](#)：获取系统控制参数。
- [HI_MPI_SYS_Init](#)：初始化 MPP 系统。
- [HI_MPI_SYS_Exit](#)：去初始化 MPP 系统。
- [HI_MPI_SYS_GetVersion](#)：获取 MPP 的版本号。
- [HI_MPI_SYS_GetCurPts](#)：获取当前时间戳。
- [HI_MPI_SYS_InitPtsBase](#)：初始化 MPP 时间戳。
- [HI_MPI_SYS_SyncPts](#)：同步 MPP 时间戳。
- [HI_MPI_VB_SetConf](#)：设置 MPP 视频缓存池属性。
- [HI_MPI_VB_GetConf](#)：获取 MPP 视频缓存池属性。
- [HI_MPI_VB_Init](#)：初始化 MPP 视频缓存池。
- [HI_MPI_VB_Exit](#)：去初始化 MPP 视频缓存池。
- [HI_MPI_VB_CreatePool](#)：创建一个视频缓存池。
- [HI_MPI_VB_DestroyPool](#)：销毁一个视频缓存池。
- [HI_MPI_VB_GetBlock](#)：获取一个缓存块。
- [HI_MPI_VB_ReleaseBlock](#)：释放一个已经获取的缓存块。
- [HI_MPI_VB_Handle2PhysAddr](#)：获取一个缓存块的物理地址。
- [HI_MPI_VB_Handle2PoolId](#)：获取一个帧缓存块所在缓存池的 ID。

HI_MPI_SYS_SetConf

【描述】

配置系统控制参数。

【语法】

```
HI_S32 HI_MPI_SYS_SetConf(const MPP\_SYS\_CONF\_S *pstSysConf);
```

【参数】

参数名称	描述	输入/输出
pstSysConf	系统控制参数指针。 静态属性（指只能在系统未初始化、未启用设备或通道时，才能设置的属性）。	输入

【返回值】

返回值	描述
0	成功。



返回值	描述
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_SYS_NULL_PTR	空指针错误。
HI_ERR_SYS_NOT_PERM	操作不允许。此错误通常是在系统已经初始化后，调用此接口导致。
HI_ERR_SYS_ILLEGAL_PARAM	输入参数非法。此错误通常是系统控制参数中有非法参数导致。

【需求】

- 头文件：hi_comm_sys.h、mpi_sys.h
- 库文件：libmpi.a

【注意】

只有在 MPP 整个系统处于未初始化状态，才可调用此函数配置 MPP 系统，否则会配置失败。video buf 根据不同的应用场景需要不同的配置。

【举例】

```
HI_S32 s32ret;
MPP_SYS_CONF_S struSysConf;

struSysConf.u32AlignWidth = 16;

/* set config of mpp system*/
s32ret = HI_MPI_SYS_SetConf(&struSysConf);
if (HI_SUCCESS != s32ret)
{
    printf("Set mpp sys config failed!\n");
    return s32ret;
}

/* init system*/
s32ret = HI_MPI_SYS_Init();
if (HI_SUCCESS != s32ret)
{
```



```
        printf("Mpi init failed!\n");
        return s32ret;
    }

    /* ..... */

    /* exit system*/
    s32ret = HI_MPI_SYS_Exit();
    if (HI_SUCCESS != s32ret)
    {
        printf("Mpi exit failed!\n");
        return s32ret;
    }
}
```

【相关主题】

[HI_MPI_VB_GetConf](#)

HI_MPI_SYS_GetConf

【描述】

获取系统控制参数。

【语法】

```
HI_S32 HI_MPI_SYS_GetConf(MPP\_SYS\_CONF\_S *pstSysConf);
```

【参数】

参数名称	描述	输入/输出
pstSysConf	系统控制参数指针。 静态属性。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_SYS_NULL_PTR	空指针错误。
HI_ERR_SYS_NOT_PERM	操作不允许。此错误通常是由于未配置过系统控制参数导致。

【需求】

- 头文件：hi_comm_sys.h、mpi_sys.h
- 库文件：libmpi.a

【注意】

必须先调用 [HI_MPI_SYS_SetConf](#) 成功后才能获取配置。

【举例】

无。

【相关主题】

[HI_MPI_SYS_SetConf](#)

HI_MPI_SYS_Init

【描述】

初始化 MPP 系统。

【语法】

```
HI_S32 HI_MPI_SYS_Init(HI_VOID);
```

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_SYS_NOTREADY	系统参数未配置。



接口返回值	含义
HI_ERR_SYS_BUSY	系统忙，通常是由于系统正在去初始化导致。
HI_FAILURE	MPP 系统中有模块初始化失败，通常是由于未加载 hidmac.ko 导致。

【需求】

- 头文件：hi_comm_sys.h、mpi_sys.h
- 库文件：libmpi.a

【注意】

- 必须先调用 [HI_MPI_SYS_SetConf](#) 配置 MPP 系统后才能初始化，否则初始化会失败。
- 由于 MPP 系统的正常运行依赖于缓存池，因此必须先调用 [HI_MPI_VB_Init](#) 初始化缓存池，再初始化 MPP 系统。
- 如果多次初始化，仍会返回成功，但实际上系统不会对 MPP 的运行状态有任何影响。
- 只要有一个进程进行初始化即可，不需要所有的进程都做系统初始化的操作。

【举例】

请参见 [HI_MPI_SYS_SetConf](#) 的举例。

【相关主题】

[HI_MPI_SYS_Exit](#)

HI_MPI_SYS_Exit

【描述】

去初始化 MPP 系统。除了音频的编解码通道外，所有的音频输入输出、视频输入输出、视频编码、视频解码通道都会被销毁或者禁用，整个 MPP 系统恢复到系统未初始化的状态。

【语法】

```
HI_S32 HI_MPI_SYS_Exit(HI_VOID);
```

【参数】

无。

【返回值】

返回值	描述
0	成功。



返回值	描述
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_SYS_BUSY	系统忙，通常是由于还有用户进程阻塞 MPI 接口导致。

【需求】

- 头文件：[hi_comm_sys.h](#)、[mpi_sys.h](#)
- 库文件：[libmpi.a](#)

【注意】

- 去初始化时，如果有阻塞在 MPI 上的用户进程，则去初始化会失败。如果所有阻塞在 MPI 上的调用都返回，则可以成功去初始化。
- 可以反复去初始化，不返回失败。
- 由于系统去初始化不会销毁音频的编解码通道，因此这些通道的销毁需要用户主动进行。如果创建这些通道的进程退出，则通道随之被销毁。

【举例】

请参见 [HI_MPI_SYS_SetConf](#) 的举例。

【相关主题】

[HI_MPI_SYS_Init](#)

HI_MPI_SYS_GetVersion

【描述】

获取 MPP 的版本号。

【语法】

```
HI_S32 HI_MPI_SYS_GetVersion(MPP\_VERSION\_S *pstVersion);
```

【参数】

参数名称	描述	输入/输出
pstVersion	版本号描述指针。 动态属性（指在任何时刻都可以设置的属性）。	输出

**【返回值】**

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_SYS_NULL_PTR	空指针错误。

【需求】

- 头文件：hi_comm_sys.h、mpi_sys.h
- 库文件：libmpi.a

【注意】

无。

【举例】

```
HI_S32 s32ret;  
MPP\_VERSION\_S stVersion;  
  
s32ret = HI_MPI_SYS_GetVersion(&stVersion);  
if (HI_SUCCESS != s32ret)  
{  
    return s32ret;  
}  
printf("mpi version is %s\n", stVersion.aVersion);
```

【相关主题】

无。

HI_MPI_SYS_GetCurPts

【描述】

获取 MPP 的当前时间戳。

【语法】



```
HI_S32 HI_MPI_SYS_GetCurPts (HI_U64 *pu64CurPts);
```

【参数】

参数名称	描述	输入/输出
pu64CurPts	当前时间戳指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为系统错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
非 0	设备打开失败。

【需求】

- 头文件：hi_comm_sys.h、mpi_sys.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。

HI_MPI_SYS_InitPtsBase

【描述】

初始化 MPP 的时间戳基准。

【语法】

```
HI_S32 HI_MPI_SYS_InitPtsBase (HI_U64 u64PtsBase);
```

【参数】



参数名称	描述	输入/输出
u64PtsBase	时间戳基准。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为系统错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
非 0	设备打开失败。

【需求】

- 头文件：hi_comm_sys.h、mpi_sys.h
- 库文件：libmpi.a

【注意】

初始化时间戳基准会将当前系统的时间戳强制置成 u64PtsBase，与系统原有时间戳没有任何约束。因此，建议在媒体业务没有启动时（例如操作系统刚启动），调用这个接口。如果媒体业务已经启动，建议调用 [HI_MPI_SYS_SyncPts](#) 进行时间戳微调。

【举例】

无。

【相关主题】

无。

HI_MPI_SYS_SyncPts

【描述】

同步 MPP 的时间戳。

【语法】

```
HI_S32 HI_MPI_SYS_SyncPts(HI_U64 u64PtsBase);
```

【参数】



参数名称	描述	输入/输出
u64PtsBase	时间戳基准。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为系统错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
非 0	设备打开失败。

【需求】

- 头文件：hi_comm_sys.h、mpi_sys.h
- 库文件：libmpi.a

【注意】

对当前系统时间戳进行微调，微调后不会出现时间戳回退现象。在多片之间做同步时，由于单板的时钟源误差可能比较大，建议一秒钟进行一次时间戳微调。

【举例】

无。

【相关主题】

无。

HI_MPI_VB_SetConf

【描述】

设置 MPP 视频缓存池属性。

【语法】

```
HI_S32 HI_MPI_VB_SetConf (const VB_CONF_S *pstVbConf);
```

【参数】



参数名称	描述	输入/输出
pstVbConf	视频缓存池属性指针。 静态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VB_NULL_PTR	空指针错误。
HI_ERR_VB_BUSY	系统忙，通常是已经调用 HI_MPI_SYS_Init 对系统进行了初始化导致。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

只能在系统处于未初始化的状态下，才可以设置缓存池属性，否则会返回失败。

【举例】

```
HI_S32 s32ret;  
VB_CONF_S stVbConf;  
  
memset(&stVbConf, 0, sizeof(VB_CONF_S));  
stVbConf.u32MaxPoolCnt = 128;  
stVbConf.astCommPool[0].u32BlkSize = 768*576*2;  
stVbConf.astCommPool[0].u32BlkCnt = 20;  
stVbConf.astCommPool[1].u32BlkSize = 384*288*2;  
stVbConf.astCommPool[1].u32BlkCnt = 40;  
  
s32ret = HI_MPI_VB_SetConf(&stVbConf);
```



```
if (HI_SUCCESS != s32ret)
{
    printf("set vb err:0x%x\n", s32ret);
    return s32ret;
}

s32ret = HI_MPI_VB_Init();
if (HI_SUCCESS != s32ret)
{
    printf("init vb err:0x%x\n", s32ret);
    return s32ret;
}

/* ... */

(void)HI_MPI_VB_Exit();
```

【相关主题】

[HI_MPI_VB_GetConf](#)

HI_MPI_VB_GetConf

【描述】

获取 MPP 视频缓存池属性。

【语法】

```
HI_S32 HI_MPI_VB_GetConf (VB_CONF_S *pstVbConf);
```

【参数】

参数名称	描述	输入/输出
pstVbConf	视频缓存池属性指针。 静态属性。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VB_NULL_PTR	空指针错误。
HI_ERR_VB_NOTREADY	缓存池属性未配置。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

必须先调用 [HI_MPI_VB_SetConf](#) 设置 MPP 视频缓存池属性，再获取属性。

【举例】

无。

【相关主题】

[HI_MPI_VB_SetConf](#)

HI_MPI_VB_Init

【描述】

初始化 MPP 视频缓存池。

【语法】

```
HI_S32 HI_MPI_VB_Init (HI_VOID);
```

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_VB_NOTREADY	缓存池属性未配置。
HI_ERR_VB_NOMEM	内存不够。
HI_FAILURE	失败。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

- 必须先调用 [HI_MPI_VB_SetConf](#) 配置缓存池属性，再初始化缓存池，否则会失败。
- 可反复初始化，不返回失败。

【举例】

请参见 [HI_MPI_VB_SetConf](#) 的举例。

【相关主题】

[HI_MPI_VB_Exit](#)

HI_MPI_VB_Exit

【描述】

去初始化 MPP 视频缓存池。

【语法】

```
HI_S32 HI_MPI_VB_Exit (HI_VOID);
```

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VB_BUSY	系统忙，通常是由于 MPP 系统未去初始化导致。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

- 必须先调用 [HI_MPI_SYS_Exit](#) 去初始化 MPP 系统，再去初始化缓存池，否则返回失败。
- 可以反复去初始化，不返回失败。
- 去初始化不会清除先前对缓存池的配置。

【举例】

请参见 [HI_MPI_VB_SetConf](#) 的举例。

【相关主题】

[HI_MPI_VB_Init](#)

HI_MPI_VB_CreatePool

【描述】

创建一个视频缓存池。

【语法】

```
VB_POOL HI_MPI_VB_CreatePool(HI_U32 u32BlkSize, HI_U32 u32BlkCnt);
```

【参数】

参数名称	描述	输入/输出
u32BlkSize	缓存池中每个缓存块的大小。 取值范围：（0, 2^{32} ），以 byte 为单位。	输入
u32BlkCnt	缓存池中缓存块的个数。 取值范围：（0, 2^{32} ）。	输入

【返回值】



返回值	描述
非 VB_INVALID_POOLID	有效的缓存池 ID 号。
VB_INVALID_POOLID	创建缓存池失败，可能是参数非法或者保留内存不够。

【错误码】

无。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

该缓存池是从保留内存中分配的，一个缓存池包含若干个大小相同的缓存块。如果该缓存池的大小超过了保留内存中的空闲空间，则创建缓存池会失败。

【举例】

```
VB_POOL VbPool;
VB_BLK VbBlk;
HI_U32 u32BlkSize = 768*576*2;
HI_U32 u32BlkCnt = 15;
HI_U32 u32Addr;

/* create a video buffer pool*/
VbPool = HI_MPI_VB_CreatePool(u32BlkSize,u32BlkCnt);
if ( VB_INVALID_POOLID == VbPool )
{
    printf("create vb err\n");
    return HI_FAILURE;
}

/* get a buffer block from pool*/
VbBlk = HI_MPI_VB_GetBlock(VbPool, u32BlkSize);
if (VB_INVALID_HANDLE == VbBlk )
{
    printf("get vb block err\n");
    (void)HI_MPI_VB_DestroyPool(VbPool);
    return HI_FAILURE;
}

/* get the physical address of buffer block*/
u32Addr = HI_MPI_VB_Handle2PhysAddr(VbBlk);
```



```
if (HI_NULL_PTR == u32Addr)
{
    printf("blk to physaddr err\n");
    (void)HI_MPI_VB_ReleaseBlock(VbBlk);
    (void)HI_MPI_VB_DestroyPool(VbPool);
    return HI_FAILURE;
}

/* use this address do something ...*/

/* then release the buffer block*/
(void)HI_MPI_VB_ReleaseBlock(VbBlk);

/* destroy video buffer pool */
(void)HI_MPI_VB_DestroyPool(VbPool);
```

【相关主题】

[HI_MPI_VB_DestroyPool](#)

HI_MPI_VB_DestroyPool

【描述】

销毁一个视频缓存池。

【语法】

```
HI_S32 HI_MPI_VB_DestroyPool(VB_POOL Pool);
```

【参数】

参数名称	描述	输入/输出
Pool	缓存池 ID 号。 取值范围：[0, VB_MAX_POOLS)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VB_NOTREADY	缓存池未初始化。
HI_ERR_VB_ILLEGAL_PARAM	输入参数无效，缓存池 ID 非法。
HI_ERR_VB_UNEXIST	缓存池不存在。
HI_ERR_VB_NOT_PERM	操作不允许，试图销毁一个非用户态创建的缓存池。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

- 销毁一个不存在的缓存池，则返回 [HI_ERR_VB_UNEXIST](#)。
- 在去初始化 MPP 缓存池时，所有的缓存池都将被销毁，包括用户态的缓存池。

【举例】

请参见 [HI_MPI_VB_CreatePool](#) 的举例。

【相关主题】

[HI_MPI_VB_CreatePool](#)

HI_MPI_VB_GetBlock

【描述】

用户态获取一个缓存块。

【语法】

```
VB_BLK HI_MPI_VB_GetBlock(VB_POOL Pool, HI_U32 u32BlkSize);
```

【参数】

参数名称	描述	输入/输出
Pool	缓存池 ID 号。 取值范围：[0, VB_MAX_POOLS)。	输入
u32BlkSize	缓存块大小。 取值范围：(0, 2^{32})，以 byte 为单位。	输入

【返回值】



返回值	描述
非 VB_INVALID_HANDLE	有效的缓存块句柄。
VB_INVALID_HANDLE	获取缓存块失败。

【错误码】

无。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

- 用户可以在创建一个缓存池之后，调用本接口从该缓存池中获取一个缓存块；即将第 1 个参数 Pool 设置为创建的缓存池 ID（第 2 个参数 u32BlkSize 不用设置）；获取的缓存块的大小即为用户创建缓存池时指定的缓存块大小。
- 如果用户需要从任意一个公共缓存池中获取一块指定大小的缓存块，则可以将第 1 个参数 Pool 设置为无效 ID 号（VB_INVALID_POOLID），将第 2 个参数 u32BlkSize 设置为需要的缓存块大小。
- 公共缓存池主要用来存放 VIU 的捕获图像，因此，对公共缓存池的不当操作（如占用过多的缓存块）会影响 MPP 系统的正常运行。

【举例】

请参见 [HI_MPI_VB_CreatePool](#) 的举例。

【相关主题】

[HI_MPI_VB_ReleaseBlock](#)

HI_MPI_VB_ReleaseBlock

【描述】

用户态释放一个已经获取的缓存块。

【语法】

```
HI_S32 HI_MPI_VB_ReleaseBlock(VB_BLK Block);
```

【参数】

参数名称	描述	输入/输出
Block	缓存块句柄。	输入

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VB_ILLEGAL_PARAM	输入参数无效，无效的缓存块句柄。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

获取的缓存块使用完后，应该调用此接口释放缓存块。

【举例】

请参见 [HI_MPI_VB_CreatePool](#) 的举例。

【相关主题】

[HI_MPI_VB_GetBlock](#)

HI_MPI_VB_Handle2PhysAddr

【描述】

用户态获取一个缓存块的物理地址。

【语法】

```
HI_U32 HI_MPI_VB_Handle2PhysAddr(VB_BLK Block);
```

【参数】

参数名称	描述	输入/输出
Block	缓存块句柄。	输入

【返回值】



返回值	描述
0	无效返回值，缓存块句柄非法。
非 0	有效物理地址。

【错误码】

无。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

指定的缓存块应该是从 MPP 视频缓存池中获取的有效缓存块。

【举例】

请参见 [HI_MPI_VB_CreatePool](#) 的举例。

【相关主题】

无。

HI_MPI_VB_Handle2PoolId

【描述】

用户态获取一个帧缓存块所在缓存池的 ID。

【语法】

```
VB_POOL HI_MPI_VB_Handle2PoolId (VB_BLK Block);
```

【参数】

参数名称	描述	输入/输出
Block	缓存块句柄。	输入

【返回值】

返回值	描述
非负数	有效的缓存池 ID 号。
负数	无效的缓存池 ID 号。



【错误码】

无。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

指定的缓存块应该是从 MPP 视频缓存池中获取的有效缓存块。

【举例】

```
VB_POOL VbPool;
VB_BLK VbBlk; /* get vb blk id from somewhere*/

/* get pool id */
VbPool = HI_MPI_VB_Handle2PoolId(VbBlk);
if ( VB_INVALID_POOLID != VbPool )
{
    printf("pool id is %d\n", VbPool);
    /* use pool id do something ...*/
}
```

【相关主题】

无。

2.4 数据类型

2.4.1 基本数据类型

基本数据类型定义如下：

```
typedef unsigned char      HI_U8;
typedef unsigned char      HI_UCHAR;
typedef unsigned short     HI_U16;
typedef unsigned int       HI_U32;

typedef char               HI_S8;
typedef short              HI_S16;
typedef int                HI_S32;

#ifdef _M_IX86
typedef unsigned long long HI_U64;
typedef long long          HI_S64;
```



```
#else
typedef __int64          HI_U64;
typedef __int64          HI_S64;
#endif

typedef char             HI_CHAR;
typedef char*            HI_PCHAR;

typedef float            HI_FLOAT;
typedef double            HI_DOUBLE;
typedef void              HI_VOID;

typedef unsigned long    HI_SIZE_T;
typedef unsigned long    HI_LENGTH_T;

typedef enum {
    HI_FALSE    = 0,
    HI_TRUE     = 1,
} HI_BOOL;

#ifndef NULL
#define NULL          0L
#endif
#define HI_NULL        0L
#define HI_NULL_PTR    0L

#define HI_SUCCESS     0
#define HI_FAILURE     (-1)

typedef HI_S32 AI_CHN;
typedef HI_S32 AO_CHN;
typedef HI_S32 AENC_CHN;
typedef HI_S32 ADEC_CHN;
typedef HI_S32 AUDIO_DEV;

typedef HI_S32 VI_DEV;
typedef HI_S32 VI_CHN;
typedef HI_S32 VO_CHN;
typedef HI_S32 VENC_CHN;
typedef HI_S32 VDEC_CHN;
typedef HI_S32 VENC_GRP;

/* 无效的通道号、无效的设备号 */
#define HI_INVALID_CHN    (-1)
```



```
#define HI_INVALID_DEV                (-1)

/* 最大的视频缓存池个数 */
#define VB_MAX_POOLS                  128

/* 所有视频输入设备或通道的最大个数（包括VIU硬件通道和虚拟VI通道）*/
#define VI_MAX_CHN_NUM                32
#define VI_MAX_DEV_NUM                8

/* 硬件视频输入单元设备最大个数*/
#define VIU_MAX_DEV_NUM               4

/* 每个硬件视频输入设备支持的最大通道数目*/
#define VIU_MAX_CHN_NUM_PER_DEV      4

/* 硬件视频输入通道总的最大个数*/
#ifdef hi3515
#define VIU_MAX_CHN_NUM               8
#else
#define VIU_MAX_CHN_NUM               16
#endif

/* VO设备的最大个数 */
#define VO_MAX_DEV_NUM                6

/* VO设备中物理设备的最大个数 */
#define VO_MAX_PHY_DEV                3

/* VO通道的最大个数 */
#define VO_MAX_CHN_NUM                32

/* VO最大同步组数 */
#define VO_SYNC_MAX_GRP               16

/* VO同步组容纳的最多通道数 */
#define VO_SYNC_MAX_CHN               32

/* VO最小显示buffer数 */
#define VO_MIN_DISP_BUF               5

/* VO最大显示buffer数 */
#define VO_MAX_DISP_BUF               15
```



```
/* VO最小虚拟设备buffer数*/
#define VO_MIN_VIRT_BUF          3

/* VO最大虚拟设备buffer数*/
#define VO_MAX_VIRT_BUF          15

/* 最大的编码组个数 */
#define VENC_MAX_GRP_NUM          64

/* 最大的编码通道个数 */
#define VENC_MAX_CHN_NUM          64

/* 最大的解码通道个数 */
#define VDEC_MAX_CHN_NUM          32

/* SIO的个数，即音频设备的个数 */
#define SIO_MAX_NUM                3

/* 最大的AIO通道数 */
#define AIO_MAX_CHN_NUM            16

/* 最大的音频编码通道数 */
#define AENC_MAX_CHN_NUM            32

/* 最大的音频解码通道数 */
#define ADEC_MAX_CHN_NUM            32

/* 调用MPI的阻塞标志 */
#define HI_IO_BLOCK                0

/* 调用MPI的非阻塞标志 */
#define HI_IO_NOBLOCK              1

/* 最大音频帧缓存数 */
#define MAX_AUDIO_FRAME_NUM        50
```

除了上述基本数据类型外，其他基本数据类型定义如下：

- **POINT_S**：定义坐标信息结构体。
- **DIMENSION_S**：定义尺寸信息结构体。



- [RECT_S](#): 定义矩形区域信息结构体。
- [PAYLOAD_TYPE_E](#): 定义音视频净荷类型枚举。

POINT_S

【说明】

定义坐标信息结构体。

【定义】

```
typedef struct hiPOINT_S
{
    HI_S32  s32X;
    HI_S32  s32Y;
}POINT_S;
```

【成员】

成员名称	描述
s32X	横坐标。
s32Y	纵坐标。

【注意事项】

无。

【相关数据类型及接口】

REGION_CTRL_PARAM_U

DIMENSION_S

【说明】

定义尺寸信息结构体。

【定义】

```
typedef struct hiDIMENSION_S
{
    HI_S32  s32Width;
    HI_S32  s32Height;
}DIMENSION_S;
```

【成员】

成员名称	描述
s32Width	宽度。



成员名称	描述
s32Height	高度。

【注意事项】

无。

【相关数据类型及接口】

REGION_CTRL_PARAM_U

RECT_S

【说明】

定义矩形区域信息结构体。

【定义】

```
typedef struct hiRECT_S
{
    HI_S32  s32X;
    HI_S32  s32Y;
    HI_U32  u32Width;
    HI_U32  u32Height;
}RECT_S;
```

【成员】

成员名称	描述
s32X	横坐标。
s32Y	纵坐标。
u32Width	宽度。
u32Height	高度。

【注意事项】

无。

【相关数据类型及接口】

- REGION_ATTR_S
- OVERLAY_ATTR_S
- VI_CHM_ATTR_S
- VI_PUB_ATTR_S



- VO_CHN_ATTR_S

PAYLOAD_TYPE_E

【说明】

定义音视频净荷类型枚举。

【定义】

```
typedef enum
{
    PT_PCMU = 0,
    PT_1016 = 1,
    PT_G721 = 2,
    PT_GSM = 3,
    PT_G723 = 4,
    PT_DVI4_8K = 5,
    PT_DVI4_16K = 6,
    PT_LPC = 7,
    PT_PCMA = 8,
    PT_G722 = 9,
    PT_S16BE_STEREO,
    PT_S16BE_MONO = 11,
    PT_QCELP = 12,
    PT_CN = 13,
    PT_MPEGAUDIO = 14,
    PT_G728 = 15,
    PT_DVI4_3 = 16,
    PT_DVI4_4 = 17,
    PT_G729 = 18,
    PT_G711A = 19,
    PT_G711U = 20,
    PT_G726 = 21,
    PT_G729A = 22,
    PT_LPCM = 23,
    PT_CelB = 25,
    PT_JPEG = 26,
    PT_CUSM = 27,
    PT_NV = 28,
    PT_PICW = 29,
    PT_CPV = 30,
    PT_H261 = 31,
    PT_MPEGVIDEO = 32,
    PT_MPEG2TS = 33,
    PT_H263 = 34,
```



```
PT_SPEG = 35,  
PT_MPEG2VIDEO = 36,  
PT_AAC = 37,  
PT_WMA9STD = 38,  
PT_HEAAC = 39,  
PT_PCM_VOICE = 40,  
PT_PCM_AUDIO = 41,  
PT_AACLC = 42,  
PT_MP3 = 43,  
PT_ADPCMA = 49,  
PT_AEC = 50,  
PT_X_LD = 95,  
PT_H264 = 96,  
PT_D_GSM_HR = 200,  
PT_D_GSM_EFR = 201,  
PT_D_L8 = 202,  
PT_D_RED = 203,  
PT_D_VDVI = 204,  
PT_D_BT656 = 220,  
PT_D_H263_1998 = 221,  
PT_D_MP1S = 222,  
PT_D_MP2P = 223,  
PT_D_BMPEG = 224,  
PT_MP4VIDEO = 230,  
PT_MP4AUDIO = 237,  
PT_VC1 = 238,  
PT_JVC_ASF = 255,  
PT_D_AVI = 256,  
PT_MAX = 257,  
  
PT_AMR = 1001, /* add by mpp */  
PT_MJPEG = 1002,  
}PAYLOAD_TYPE_E;
```

【成员】

略。

【注意事项】

无。

【相关数据类型及接口】

- VENC_CHN_ATTR_S
- VDEC_CHN_ATTR_S
- AENC_CHN_ATTR_S



- ADEC_CHN_ATTR_S

2.4.2 系统控制数据类型

系统控制相关数据类型定义如下：

- [MPP_SYS_CONF_S](#)：定义 MPP 系统控制属性结构体。
- [VB_CONF_S](#)：定义视频缓存池属性结构体。
- [MPP_VERSION_S](#)：定义 MPP 版本描述结构体。

MPP_SYS_CONF_S

【说明】

定义 MPP 系统控制属性结构体。

【定义】

```
typedef struct hiMPP_SYS_CONF_S
{
    /* stride of picture buffer must be aligned with this value.
     * you can choose a value from 1 to 1024,
     * and it except 1 must be multiple of 16.*/
    HI_U32  u32AlignWidth;

    HI_U32  u32PinMuxCtrl;
}MPP_SYS_CONF_S;
```

【成员】

成员名称	描述
u32AlignWidth	整个系统中使用图像的 stride 字节对齐数。 取值范围：[1, 1024]，直接配置成 16 或者 64 即可。 静态属性。
u32PinMuxCtrl	管脚复用配置，暂时不需要配置。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPL_SYS_SetConf](#)

VB_CONF_S

【说明】



定义视频缓存池属性结构体。

【定义】

```
typedef struct hiVB_CONF_S
{
    HI_U32  u32MaxPoolCnt; /* max count of pools, (0,VB_MAX_POOLS] */
    Struct  hiVB_CPOOL_S
    {
        HI_U32  u32BlkSize;
        HI_U32  u32BlkCnt;
    }astCommPool[VB_MAX_COMM_POOLS];
} VB_CONF_S;
```

【成员】

成员名称	描述
u32MaxPoolCnt	整个系统中可容纳的缓存池个数。 取值范围: (0, VB_MAX_POOLS]。 静态属性。 目前, 固定取值 VB_MAX_POOLS。
astCommPool	公共缓存池属性结构体, 成员包括公共缓存池中每个缓存块的大小 (以 byte 为单位) 和缓存块的个数。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPL_SYS_SetConf](#)

MPP_VERSION_S

【说明】

定义 MPI 版本描述结构体。

【定义】

```
typedef struct hiMPP_VERSION_S
{
    HI_CHAR aVersion[64];
}MPP_VERSION_S;
```

【成员】



成员名称	描述
aVersion	版本描述字符串。 比如“HI_VERSION=Hi35xx_MPP_V1.0.4.0”。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_SYS_GetVersion](#)

2.4.3 视频公共类型

视频公共数据类型定义如下：

- [VIDEO_NORM_E](#)：定义视频输入制式类型。
- [PIXEL_FORMAT_E](#)：定义像素格式类型。
- [VIDEO_FIELD_E](#)：定义视频图像帧场类型。
- [VIDEO_FRAME_S](#)：定义视频原始图像帧结构体。
- [VIDEO_FRAME_INFO_S](#)：定义视频图像帧信息结构体。
- [BITMAP_S](#)：定义位图图像信息结构。
- [VIDEO_VBI_INFO_S](#)：定义视频 VBI 信息结构体。

VIDEO_NORM_E

【说明】

定义视频输入制式类型。

【定义】

```
typedef enum hiVIDEO_NORM_E
{
    VIDEO_ENCODING_MODE_PAL=0,
    VIDEO_ENCODING_MODE_NTSC,
    VIDEO_ENCODING_MODE_AUTO,
    VIDEO_ENCODING_MODE_BUTT,
} VIDEO_NORM_E;
```

【成员】

成员名称	描述
VIDEO_ENCODING_MODE_PAL	PAL 制式。
VIDEO_ENCODING_MODE_NTSC	NTSC 制式。



成员名称	描述
VIDEO_ENCODING_MODE_AUTO	自动识别制式。

【注意事项】

自动识别制式目前暂不支持。

【相关数据类型及接口】

VI_PUB_ATTR_S

PIXEL_FORMAT_E

【说明】

定义像素格式类型。

【定义】

```
typedef enum hiPIXEL_FORMAT_E
{
    PIXEL_FORMAT_RGB_1BPP = 0,
    PIXEL_FORMAT_RGB_2BPP,
    PIXEL_FORMAT_RGB_4BPP,
    PIXEL_FORMAT_RGB_8BPP,
    PIXEL_FORMAT_RGB_444,
    PIXEL_FORMAT_RGB_4444,
    PIXEL_FORMAT_RGB_555,
    PIXEL_FORMAT_RGB_565,
    PIXEL_FORMAT_RGB_1555,

    PIXEL_FORMAT_RGB_888,
    PIXEL_FORMAT_RGB_8888,
    PIXEL_FORMAT_RGB_PLANAR_888,
    PIXEL_FORMAT_RGB_BAYER,

    PIXEL_FORMAT_YUV_A422,
    PIXEL_FORMAT_YUV_A444,

    PIXEL_FORMAT_YUV_PLANAR_422,
    PIXEL_FORMAT_YUV_PLANAR_420,
    PIXEL_FORMAT_YUV_PLANAR_444,
    PIXEL_FORMAT_YUV_SEMIPLANAR_422,
    PIXEL_FORMAT_YUV_SEMIPLANAR_420,
    PIXEL_FORMAT_YUV_SEMIPLANAR_444,
```



```
PIXEL_FORMAT_UYVY_PACKAGE_422,  
PIXEL_FORMAT_YCbCr_PLANAR,  
  
PIXEL_FORMAT_BUTT  
} PIXEL_FORMAT_E;
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

- VI_CHN_ATTR_S
- OVERLAY_ATTR_S

VIDEO_FIELD_E

【说明】

定义视频图像帧场类型。

【定义】

```
typedef enum hiVIDEO_FIELD_E  
{  
    VIDEO_FIELD_TOP          = 0x01,    /* even field */  
    VIDEO_FIELD_BOTTOM       = 0x02,    /* odd field */  
    VIDEO_FIELD_INTERLACED   = 0x03,    /* two interlaced fields */  
    VIDEO_FIELD_FRAME        = 0x04,    /* frame */  
  
    VIDEO_FIELD_BUTT  
} VIDEO_FIELD_E;
```

【成员】

成员名称	描述
VIDEO_FIELD_TOP	顶场类型。
VIDEO_FIELD_BOTTOM	底场类型。
VIDEO_FIELD_INTERLACED	两场间插类型。
VIDEO_FIELD_FRAME	帧类型。

【相关数据类型及接口】

[VIDEO_FRAME_S](#)



VIDEO_FRAME_S

【说明】

定义视频原始图像帧结构。

【定义】

```
typedef struct hiVIDEO_FRAME_S
{
    PIXEL_FORMAT_E  enPixelFormat;
    HI_U32           u32Width;
    HI_U32           u32Height;

    VIDEO_FIELD_E   u32Field;

    HI_U32  u32PhyAddr[3];
    HI_VOID *pVirAddr[3];
    HI_U32  u32Stride[3];

    HI_U16  u16OffsetTop;        /* top offset of show area */
    HI_U16  u16OffsetBottom;     /* bottom offset of show area */
    HI_U16  u16OffsetLeft;       /* left offset of show area */
    HI_U16  u16OffsetRight;      /* right offset of show area */

    HI_U64  u64pts;
    HI_U32  u32TimeRef;

    HI_U32  u32PrivateData;
}VIDEO_FRAME_S;
```

【成员】

成员名称	描述
enPixelFormat	视频图像像素格式。
u32Width	图像宽度。
u32Height	图像高度。
u32Field	帧场模式。
u32PhyAddr	物理地址。
pVirAddr	虚拟地址。
u32Stride	图像跨距。
u16OffsetTop	图像顶部剪裁宽度。



成员名称	描述
u16OffsetBottom	图像底部剪裁宽度。
u16OffsetLeft	图像左侧剪裁宽度。
u16OffsetRight	图像右侧剪裁宽度。
u64pts	图像时间戳。
u32TimeRef	图像帧序列号。
u32PrivateData	私有数据。

【注意事项】

无。

【相关数据类型及接口】

VI_PUB_ATTR_S

VIDEO_FRAME_INFO_S

【说明】

定义视频图像帧信息结构体。

【定义】

```
typedef struct hiVIDEO_FRAME_INFO_S
{
    VIDEO_FRAME_S    stVFrame;
    HI_U32            u32PoolId;
}VIDEO_FRAME_INFO_S;
```

【成员】

成员名称	描述
stVFrame	视频图像帧。
u32PoolId	视频缓存池 ID。

【注意事项】

无。

【相关数据类型及接口】

VIDEO_FRAME_S



BITMAP_S

【说明】

定义位图图像信息结构。

【定义】

```
typedef struct hiBITMAP_S
{
    PIXEL_FORMAT_E  enPixelFormat;
    HI_U32           u32Width;
    HI_U32           u32Height;
    HI_VOID          *pData;
} BITMAP_S;
```

【成员】

成员名称	描述
enPixelFormat	位图像素格式。
u32Width	位图宽度。
u32Height	位图高度。
pData	位图数据。

【注意事项】

无。

【相关数据类型及接口】

REGION_CTRL_PARAM_U

VIDEO_VBI_INFO_S

【说明】

定义视频 VBI 信息结构体。

【定义】

```
typedef struct hiVIDEO_VBI_INFO_S
{
    HI_U32  au32Data[VIU_MAX_VBI_LEN];
    HI_U32  u32Len;
} VIDEO_VBI_INFO_S;
```

【成员】



成员名称	描述
au32Data	VBI 数据。
u32Len	VBI 数据长度。

【注意事项】

无。

2.5 错误码

2.5.1 系统控制错误码

系统控制 API 错误码如表 2-1 所示。

表2-1 系统控制 API 错误码

错误代码	宏定义	描述
0xA0028003	HI_ERR_SYS_ILLEGAL_PARAM	参数设置无效
0xA0028006	HI_ERR_SYS_NULL_PTR	空指针错误
0xA0028009	HI_ERR_SYS_NOT_PERM	操作不允许
0xA0028010	HI_ERR_SYS_NOTREADY	系统控制属性未配置
0xA0028012	HI_ERR_SYS_BUSY	系统忙

2.5.2 视频缓存池错误码

视频缓存池 API 错误码如表 2-2 所示。

表2-2 视频缓存池 API 错误码

错误代码	宏定义	描述
0xA0018003	HI_ERR_VB_ILLEGAL_PARAM	参数设置无效
0xA0018005	HI_ERR_VB_UNEXIST	视频缓存池不存在
0xA0018006	HI_ERR_VB_NULL_PTR	参数空指针错误
0xA0018009	HI_ERR_VB_NOT_PERM	操作不允许
0xA001800C	HI_ERR_VB_NOMEM	分配内存失败
0xA001800D	HI_ERR_VB_NOBUF	分配缓存失败



错误代码	宏定义	描述
0xA0018010	HI_ERR_VB_NOTREADY	系统控制属性未配置
0xA0018012	HI_ERR_VB_BUSY	系统忙
0xA0018040	HI_ERR_VB_2MPOOLS	创建缓存池多



目 录

3 视频输入.....	3-1
3.1 概述.....	3-1
3.2 重要概念.....	3-1
3.3 API 参考	3-2
3.4 数据类型.....	3-53
3.5 错误码.....	3-61



表格目录

表 3-1 各种视频输入接口模式下的通道属性.....	3-11
表 3-2 常用分辨率图像的 VI 参考配置	3-12
表 3-3 主属性和次属性的配置关系.....	3-39
表 3-4 4D1 模式时 AdChnId 的默认配置	3-47
表 3-5 2D1 模式时 AdChnId 的默认配置	3-47
表 3-6 视频输入 API 错误码	3-61



3 视频输入

3.1 概述

Hi3520/Hi3515 芯片的视频输入（VI）模块实现的功能：将芯片外的视频数据，通过 ITU-R BT.656/601/1120 接口或 digital camera 接口，存入到指定的内存区域。Hi3520 芯片有 4 个 VI 设备端口，最大支持同时 16 路视频输入；Hi3515 芯片也有 4 个 VI 设备端口，但最大支持同时 8 路视频输入。

3.2 重要概念

- 视频输入设备
Hi3520/Hi3515 芯片均有 4 个 VI 设备端口，SDK 将其分别标示为 ViDev0、ViDev1、ViDev2 和 ViDev3。每个设备又可接 1~4 路实际的捕获通道，具体支持的通道数与该设备的输入接口模式和外接 AD 芯片相关。
- 视频输入接口模式
Hi3520/Hi3515 芯片支持多种视频接口协议，且提供灵活多样的配置与各种外围 codec 对接。支持的接口时序包括 ITU-R BT.656、ITU-R BT.601、ITU-R BT.1120、digital camera 等，具体的时序支持情况请参见《Hi3520 H.264 编解码处理器用户指南》和《Hi3515 H.264 编解码处理器用户指南》中“第 6 章 视频接口”。
4 个视频输入设备都能支持 ITU-R BT.656 接口模式，但只有设备 0 和设备 2 才支持其他几种接口模式。支持 BT.656 接口模式时，又能支持几种不同的多路复用工作模式，例如 54M 2D1、54M 4CIF 以及 108M 4D1。
- 视频输入通道
视频输入通道与物理上的实际视频采集通道一一对应，视频输入通道归属于视频输入设备，视频输入设备的接口模式决定了此设备下能支持的通道个数，一个设备最大支持 4 个通道，依次用 ViChn0、ViChn1、ViChn2、ViChn3 标示；除了有多路复用功能的 BT.656 接口模式能支持多个通道外，其他接口模式时每个设备下只能支持 1 个通道。



3.3 API 参考

视频输入（VI）实现启用视频输入设备、视频输入通道、绑定视频输入通道到某个视频输出通道等功能。

该功能模块提供以下 MPI：

- [HI_MPI_VI_SetPubAttr](#)：设置 VI 设备属性。
- [HI_MPI_VI_GetPubAttr](#)：获取 VI 设备属性。
- [HI_MPI_VI_Enable](#)：启用 VI 设备。
- [HI_MPI_VI_Disable](#)：禁用 VI 设备。
- [HI_MPI_VI_SetChnAttr](#)：设置 VI 通道属性。
- [HI_MPI_VI_GetChnAttr](#)：获取 VI 通道属性。
- [HI_MPI_VI_EnableChn](#)：启用 VI 通道。
- [HI_MPI_VI_DisableChn](#)：禁用 VI 通道。
- [HI_MPI_VI_GetChnLuma](#)：获取 VI 通道图像亮度。
- [HI_MPI_VI_GetFrame](#)：获取 VI 原始帧图像。
- [HI_MPI_VI_ReleaseFrame](#)：释放原始图像数据所占的缓存。
- [HI_MPI_VI_BindOutput](#)：绑定 VI、VO 通道。
- [HI_MPI_VI_UnBindOutput](#)：解绑定 VI、VO 通道。
- [HI_MPI_VI_SetSrcFrameRate](#)：设置视频输入通道的原始帧率。
- [HI_MPI_VI_GetSrcFrameRate](#)：获取视频输入通道的原始帧率。
- [HI_MPI_VI_SetFrameRate](#)：设置视频输入通道的目标帧率。
- [HI_MPI_VI_GetFrameRate](#)：获取视频输入通道的目标帧率。
- [HI_MPI_VI_SetUserPic](#)：设置用户图片的帧信息。
- [HI_MPI_VI_SetUserPicEx](#)：设置用户图片的扩展接口。
- [HI_MPI_VI_EnableUserPic](#)：启用插入用户图片。
- [HI_MPI_VI_DisableUserPic](#)：禁用插入用户图片。
- [HI_MPI_VI_SetMinorChnAttr](#)：设置 VI 通道次属性。
- [HI_MPI_VI_GetMinorChnAttr](#)：获取 VI 通道次属性。
- [HI_MPI_VI_ClearChnMinorAttr](#)：清除 VI 通道次属性。
- [HI_MPI_VI_EnableCascade](#)：启用 VI-VO 级联。
- [HI_MPI_VI_DisableCascade](#)：禁用 VI-VO 级联。
- [HI_MPI_VI_SetAdChnId](#)：设置与 VI 通道对应的 ADC 的 CHID。
- [HI_MPI_VI_GetAdChnId](#)：获取与 VI 通道对应的 ADC 的 CHID。
- [HI_MPI_VI_SetSrcCfg](#)：设置 VI 通道视频源配置。
- [HI_MPI_VI_GetSrcCfg](#)：获取 VI 通道视频源配置。
- [HI_MPI_VI_GetFd](#)：获取 VI 通道的设备文件句柄。



HI_MPI_VI_SetPubAttr

【描述】

设置 VI 设备属性。

【语法】

```
HI_S32 HI_MPI_VI_SetPubAttr(VI_DEV ViDevId, const VI_PUB_ATTR_S  
*pstPubAttr);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
pstPubAttr	VI 设备属性指针。 静态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_PARA	视频输入参数无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_FAILED_NOTDISABLE	视频输入设备未禁用。
HI_ERR_VI_NOT_SUPPORT	操作不支持，当前版本不支持此配置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h



- 库文件：libmpi.a

【注意】

- 在调用前要保证 VI 设备处于禁用状态。如果 VI 设备已处于使能状态，可以使用 [HI_MPI_VI_Disable](#) 来禁用设备。
- 参数 pstPubAttr 主要用来配置指定 VI 设备的视频接口模式，用于与外围 codec 对接，支持的接口模式包括 BT.656、BT.601、digital camera、BT.1120 逐行以及 BT.1120 隔行模式。
- 当视频接口模式为 BT.656 时，需要设置时分复用的工作模式 enWorkMode，目前支持 54M 2D1 模式和 108M 4 D1 模式；VI 的工作模式必须和 ADC 的工作模式一致才能正常捕获视频数据。
 - 2D1 工作模式时，每个 VI 设备下支持 2 个 VI 通道，Hi3520 有 4 个 VI 设备，因此此模式下共支持 8 个 VI 通道；
 - 4D1 工作模式时，每个 VI 设备下支持 4 个 VI 通道，Hi3520 有 4 个 VI 设备，因此此模式下共支持 16 个 VI 通道。
- 当数据接收模式为 BT.1120 逐行或隔行模式时，只能使用 VI 设备 0 和设备 2，且每个设备下只支持 1 个通道。需要配置以下选项：
 - 色度通道 bIsChromaChn: BT.1120 接口模式时需要占用两个 VI 通道（设备 0 和设备 1 的通道 0，或者设备 2 和设备 3 的通道 0），用来传输单独的亮度和色度数据，软件只需要配置和启动设备 0 或设备 2，bIsChromaChn 即用于配置此 VI 设备是否传输色度数据；
 - 色度交换 bChromaSwap: 用于配置色度的存储顺序，TRUE 表示 Cb1Cr1Cb0Cr0，FALSE 表示 Cr1Cb1Cr0Cb0。
- 当数据接收模式为 BT.601 时，只需要设置视频制式，公共属性结构体中的其他项不用设置；只能使用 VI 设备 0 和设备 2，且每个设备下只支持 1 个通道。
- 当数据接收模式为 Digital Camera 时，公共属性结构体中的其他所有项都不用设置；只能使用 VI 设备 0 和设备 2，且每个设备下只支持 1 个通道。

【举例】

```
HI_S32 s32ret;
VI_DEV ViDevId = 0;
VI_CHN ViChn = 0;
VO_DEV VoDevId = 2;
VO_CHN VoChn = 0;
VI_PUB_ATTR_S stPubAttr;
VI_CHN_ATTR_S stChnAttr;

stPubAttr.enInputMode = VI_MODE_BT656;
stPubAttr.enWorkMode = VI_WORK_MODE_4D1;
/* set public attribute of vi */
s32ret = HI_MPI_VI_SetPubAttr(ViDevId, &stPubAttr);
if (HI_SUCCESS != s32ret)
{
    printf("set vi pub attr err:0x%x\n", s32ret);
}
```




```
        return s32ret;
    }

    /* enable vi device*/
    s32ret = HI_MPI_VI_Enable(ViDevId);
    if (HI_SUCCESS != s32ret)
    {
        printf("enable vi dev err:0x%x\n", s32ret);
        return s32ret;
    }

    stChnAttr.enCapSel = VI_CAPSEL_BOTH;
    stChnAttr.bDownScale = HI_FALSE;
    stChnAttr.stCapRect.s32X = 0;
    stChnAttr.stCapRect.s32Y = 0;
    stChnAttr.stCapRect.u32Width = 704;
    stChnAttr.stCapRect.u32Height = 288;
    stChnAttr.enViPixFormat = PIXEL_FORMAT_YUV_SEMIPLANAR_420;
    stChnAttr.bChromaResample = HI_FALSE;
    stChnAttr.bHighPri = HI_FALSE;
    /* set channel attribute for vi chn */
    s32ret = HI_MPI_VI_SetChnAttr(ViDevId, ViChn, &stChnAttr);
    if (HI_SUCCESS != s32ret)
    {
        printf("set vi chn attr err:0x%x\n", s32ret);
        return s32ret;
    }

    /* enable vi chn */
    s32ret = HI_MPI_VI_EnableChn(ViDevId, ViChn);
    if (HI_SUCCESS != s32ret)
    {
        printf("enable vi chn err:0x%x\n", s32ret);
        return s32ret;
    }

    /* bind vi to vo */
    s32ret = HI_MPI_VI_BindOutput(ViDevId, ViChn, VoDevId, VoChn);
    if (HI_SUCCESS != s32ret)
    {
        printf("bind vi to vo err:0x%x\n", s32ret);
        return s32ret;
    }
}
```



```
/* ... .. */

/* unbind vi to vo */
s32ret = HI_MPI_VI_UnBindOutput(ViDevId, ViChn, VoDevId ,VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("unbind vi to vo err:0x%x\n", s32ret);
    return s32ret;
}

/* disable vi chn */
s32ret = HI_MPI_VI_DisableChn(ViDevId, ViChn);
if (HI_SUCCESS != s32ret)
{
    printf("disale vi chn err:0x%x\n", s32ret);
    return s32ret;
}

/* disable vi device*/
s32ret = HI_MPI_VI_Disable(ViDevId);
if (HI_SUCCESS != s32ret)
{
    printf("disale vi dev err:0x%x\n", s32ret);
    return s32ret;
}
```

【相关主题】

[HI_MPI_VI_GetPubAttr](#)

HI_MPI_VI_GetPubAttr

【描述】

获取 VI 设备属性。

【语法】

```
HI_S32 HI_MPI_VI_GetPubAttr(VI_DEV ViDevId, VI\_PUB\_ATTR\_S *pstPubAttr);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
pstPubAttr	VI 设备属性指针。	输出



【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入设备属性未设置。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

如果未设置 VI 设备属性，该接口将返回失败。

【举例】

```
HI_S32 s32ret;
VI_DEV ViDevId = 0;
VI\_PUB\_ATTR\_S stPubAttr;

/* first set public attribute of vi and enable it*/

s32ret = HI_MPI_VI_GetPubAttr(ViDevId, &stPubAttr);
if (HI_SUCCESS != s32ret)
{
    printf("set vi pub attr err:0x%x\n", s32ret);
    return s32ret;
}
```

【相关主题】

[HI_MPI_VI_SetPubAttr](#)



HI_MPI_VI_Enable

【描述】

启用 VI 设备。

【语法】

```
HI_S32 HI_MPI_VI_Enable(VI_DEV ViDevId);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入设备属性未设置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 启用前必须已经设置设备属性，否则返回失败。
- 可重复启用，不返回失败。

【举例】

请参见 [HI_MPI_VI_SetPubAttr](#) 的举例。



【相关主题】

[HI_MPI_VI_Disable](#)

HI_MPI_VI_Disable

【描述】

禁用 VI 设备。

【语法】

```
HI_S32 HI_MPI_VI_Disable(VI_DEV ViDevId);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_FAILED_CHNOTDISABLE	视频输入通道未禁用。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 必须先禁用所有 VI 通道后，再禁用 VI 设备。
- 可重复禁用，不返回失败。



【举例】

请参见 [HI_MPI_VI_SetPubAttr](#) 的举例。

【相关主题】

[HI_MPI_VI_Enable](#)

HI_MPI_VI_SetChnAttr

【描述】

设置 VI 通道属性。

【语法】

```
HI_S32 HI_MPI_VI_SetChnAttr(VI_DEV ViDevId, VI_CHN ViChn, const
VI_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstAttr	VI 通道属性指针。 动态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_PARA	视频输入参数无效。



接口返回值	含义
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入设备或通道的属性未配置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 必须先设置 VI 设备属性，再设置通道属性。
- VI 通道属性可以动态设置，即在 VI 通道工作过程中（启动 VI 通道）也可以调用此接口设置 VI 通道的各项属性。
- 各种视频输入接口模式下的通道属性说明如表 3-1 所示。

表3-1 各种视频输入接口模式下的通道属性

属性	BT656	BT1120P	BT1120I	DC	BT601
采集区域 CapRect	支持	支持	支持	支持	支持
帧场选择 CapSel	支持配置为 BOTH 和 BOTTOM	只支持配置 为 BOTH	支持配置为 BOTH 和 BOTTOM	只支持配置 为 BOTH	支持配置为 BOTH 和 BOTTOM
水平缩放 DownScale	支持	不支持	不支持	支持	支持
像素格式 PixelFormat	支持 SP420 和 SP422	支持 SP420 和 SP422	支持 SP420 和 SP422	支持 SP420 和 SP422	支持 SP420 和 SP422
优先级 HighPri ^a	支持	支持	支持	支持	支持
色度重采样 ChromaRes ^b	支持	支持	支持	支持	支持

a: 优先级 HighPri 用于配置当前通道的内部处理优先级别，一般配置为 False 既可。

b: 色度重采样 ChromaRes 用于配置当前通道图像的色度是否由 co-sited 到 interspersed 转换，详细说明请参见《Hi3520 H.264 编解码处理器用户指南》，没有特殊要求则配置为 False。

其中采集区域 CapRect 的配置说明：

- 采集区域用于配置需要采集的矩形图像范围及相对与原始图像起始点的位置。起始点位置的横坐标以像素为单位，纵坐标以行为单位。



- 如果接口模式为隔行采集模式（如 BT.656、BT.601、BT.1120I 模式），高度 u32Height 需要配置为一场的高度，例如 BT.656 的 D1 采集，需要配置为 288 或 240；如果接口模式为逐行采集模式（如 BT.1120P、DC 模式），高度 u32Height 则配置为整帧图像的实际高度。
- 采集区域的宽度需要配置为水平 1/2 缩放之前的图像宽度，例如 CIF 图像的采集则将宽度配置为 704，再选择 DownScale 为 TRUE。
- 配置的区域大小不应该超出外围 ADC 输出图像的大小范围。
- CapRect 中所有项的取值都必须是偶数，其中宽度值还要求必须是 4 的整数倍。

常用分辨率图像的 VI 参考配置如表 3-2 所示。

表3-2 常用分辨率图像的 VI 参考配置

属性	D1	Half-D1	2CIF	CIF	VGA	QVGA	720P	1080I
采集区域横坐标 CapRect.s32X	8	8	8	8	0	0	0	0
采集区域纵坐标 CapRect.s32Y	0	0	0	0	0	0	0	0
采集区域宽度 CapRect. u32Width	704	704	704	704	640	320	1280	1920
采集区域高度 CapRect. u32Height	288/240	288/240	288/240	288/240	480	240	720	540
帧场选择 CapSel	BOTH	BOTTOM	BOTH	BOTTOM	BOTH	BOTH	BOTH	BOTH
水平缩放 DownScale	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE

【举例】

请参见 [HI_MPI_VI_SetPubAttr](#) 的举例。

【相关主题】

[HI_MPI_VI_GetChnAttr](#)

HI_MPI_VI_GetChnAttr

【描述】

获取 VI 通道属性。



【语法】

```
HI_S32 HI_MPI_VI_GetChnAttr(VI_DEV ViDevId, VI_CHN ViChn, VI_CHN_ATTR_S  
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstAttr	VI 通道属性指针。 动态属性。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入设备属性未配置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】



必须先设置通道属性再获取属性，否则将返回 [HI_ERR_VI_FAILED_NOTCONFIG](#)。

【举例】

```
HI_S32 s32ret;
VI_DEV ViDevId = 0;
VI_CHN ViChn = 0;
VI_CHN_ATTR_S stChnAttr;

/* first enable vi device and vi chn */

/* get channel attribute for vi chn */
s32ret = HI_MPI_VI_GetChnAttr(ViDevId, ViChn, &stChnAttr);
if (HI_SUCCESS != s32ret)
{
    printf("get vi chn attr err:0x%x\n", s32ret);
    return s32ret;
}
```

【相关主题】

[HI_MPI_VI_SetChnAttr](#)

HI_MPI_VI_EnableChn

【描述】

启用 VI 通道。

【语法】

```
HI_S32 HI_MPI_VI_EnableChn(VI_DEV ViDevId,VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM))。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。



【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_FAILED_NOTENABLE	视频输入设备或通道未启用。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入通道属性未设置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 必须先启用 VI 设备，再启用其下的 VI 通道。
- VI 通道启用后，该 VI 通道即可开始正常工作并采集视频数据，此后即可通过调用 [HI_MPI_VI_GetFrame](#) 获取视频数据，需要将 VI 数据发送到 VO 和 VENC，则需要再调用相应的绑定接口。
- 启用 VI 通道前，必须已经设置通道属性，否则返回失败。
- 可重复启用 VI 通道，不返回失败。

【举例】

请参见 [HI_MPI_VI_SetPubAttr](#) 的举例。

【相关主题】

[HI_MPI_VI_Disable](#)

HI_MPI_VI_DisableChn

【描述】

禁用 VI 通道。

【语法】

```
HI_S32 HI_MPI_VI_DisableChn(VI_DEV ViDevId, VI_CHN ViChn);
```

【参数】



参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 禁用 VI 通道后，此 VI 通道即停止采集视频输入数据，如果已经绑定 VO 或 VENC，则 VO 或 VENC 将不会再接收到视频图像。
- 可重复禁用 VI 通道，不返回失败。

【举例】

请参见 [HI_MPI_VI_SetPubAttr](#) 的举例。

【相关主题】

[HI_MPI_VI_EnableChn](#)



HI_MPI_VI_GetChnLuma

【描述】

获取 VI 通道图像的亮度接口。

【语法】

```
HI_S32 HI_MPI_VI_GetChnLuma(VI_DEV ViDevId, VI_CHN ViChn, VI_CH_LUM_S
*pstLuma);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstLuma	VI 通道亮度信息指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_FAILED_NOTENABLE	视频输入设备或通道未启用。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h



- 库文件：libmpi.a

【注意】

此接口获取的亮度值是 VI 原始捕获图像（即水平缩放和丢场之前的图像）的所有像素亮度累加值。

【举例】

```
HI_S32 s32ret;
VI_DEV ViDevId = 0;
VI_CHN ViChn = 0;
VI_CH_LUM_S stLuma;
s32ret = HI_MPI_VI_GetChnLuma(ViDevId, ViChn, &stLuma);
if (HI_SUCCESS != s32ret)
{
    printf("get vi luma err:0x%x\n", s32ret);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_VI_GetFrame

【描述】

获取原始图像数据。

【语法】

```
HI_S32 HI_MPI_VI_GetFrame(VI_DEV ViDevId,VI_CHN ViChn,VIDEO_FRAME_INFO_S
*pstFrame);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstFrame	视频图像帧信息结构指针。	输出

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_FAILED_NOTENABLE	视频输入设备或通道未启用。
HI_ERR_VI_BUF_EMPTY	视频输入缓存为空。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 此接口可以获取指定 VI 通道当前采集的视频图像信息，图像信息主要包括图像的宽度、高度、像素格式、时间戳以及 YUV 各分量的物理地址。
- 此接口需在通道已启用后才有效。
- 获取的物理地址信息来自 MPP 内部使用的 VideoBuffer，因此使用完之后，必须要调用 HI_MPI_VI_ReleaseFrame 接口释放其内存。
- pstFrame->stVFrame->u32PhyAddr[0]和 pstFrame->stVFrame->u32PhyAddr[1]分别指向图像的亮度分量和色度分量的物理地址。
- pstFrame 结构中图像虚拟地址无效。

【举例】

```
HI_S32 s32ret;
VI_DEV ViDevId = 0;
VI_CHN ViChn = 0;
VIDEO_FRAME_INFO_S stFrame;

/* get video frame from vi chn */
s32ret = HI_MPI_VI_GetFrame(ViDevId, ViChn, &stFrame)
```



```
if (HI_SUCCESS != s32ret)
{
    printf("get vi frame err:0x%x\n", s32ret);
    return s32ret;
}

/* deal with video frame ... */

/* release video frame */
(void)HI_MPI_VI_ReleaseFrame(ViDevId, ViChn, &stFrame);
```

【相关主题】

[HI_MPI_VI_ReleaseFrame](#)

HI_MPI_VI_ReleaseFrame

【描述】

释放原始图像数据所占的缓存。

【语法】

```
HI_S32 HI_MPI_VI_ReleaseFrame(VI_DEV ViDevId,VI_CHN ViChn,const
VIDEO_FRAME_INFO_S *pstRawFrame);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstFrame	视频图像帧数据存储结构指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_INVALID_PARA	输入参数无效。
HI_ERR_VI_FAILED_NOTENABLE	视频输入设备或通道未启用。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 该接口需在通道已启用后才有效。
- 要求用户每次使用完获取原始图像接口后，调用本接口释放原始图像数据（即必须与 [HI_MPI_VI_GetFrame](#) 配对使用）。

【举例】

请参见 [HI_MPI_VI_GetFrame](#) 的举例。

【相关主题】

[HI_MPI_VI_GetFrame](#)

HI_MPI_VI_BindOutput

【描述】

视频输入通道绑定到视频输出通道。

【语法】

```
HI_S32 HI_MPI_VI_BindOutput(VI_DEV ViDevId, VI_CHN ViChn, VO_DEV VoDev,  
VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入



参数名称	描述	输入/输出
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
VoDevId	VO 设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	VO 通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入或输出通道号无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 绑定后将在 VO 直接显示 VI 通道捕获的图像，不需作用户态获取和释放的操作即可实现预览的功能。
- 支持一个 VI 通道绑定到多个 VO 通道，但一个 VO 通道同时只能被一个 VI 通道绑定。
- 可以不解绑定，而直接绑定到另外一个 VO 通道。
- 可以重复绑定，不返回失败。

【举例】

请参见 [HI_MPI_VI_SetPubAttr](#) 的举例。



【相关主题】

[HI_MPI_VI_UnBindOutput](#)

HI_MPI_VI_UnBindOutput

【描述】

视频输入通道与视频输出通道解绑定。

【语法】

```
HI_S32 HI_MPI_VI_UnBindOutput(VI_DEV ViDevId, VI_CHN ViChn, VO_DEV  
VoDevId, VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
VoDevId	VO 设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	VO 通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入或输出通道号无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。



【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

无。

【举例】

请参见 [HI_MPI_VI_SetPubAttr](#) 的举例。

【相关主题】

[HI_MPI_VI_BindOutput](#)

HI_MPI_VI_SetSrcFrameRate

【描述】

设置视频输入通道的原始帧率。

【语法】

```
HI_S32 HI_MPI_VI_SetSrcFrameRate(VI_DEV ViDevId,VI_CHN ViChn,HI_U32
u32ViFramerate);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
u32ViFramerate	原始帧率。 取值范围：大于 0。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_PARA	输入参数无效。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 这里设置的原始帧率即外围 codec 输出给 Hi3520 VI 的每秒帧数，例如 PAL 制为 25fps，NTSC 制为 30fps，720P 为 60fps。
- 与原始帧率配合使用的是目标帧率，VI 软件内部根据用户设置的原始帧率和目标帧率，进行视频图像帧捕获的帧率控制。

【举例】

```
HI_S32 s32ret;
HI_U32 u32SrcFrmRate = 25;
HI_U32 u32FrmRate = 5;

/* set public attribute of VI device*/
/* enable VI device*/
/* set attribute of VI channel*/
/* ... .. */

/* set SRC framerate of VI channel*/
s32ret = HI_MPI_VI_SetSrcFrameRate(0, 0, u32SrcFrmRate);
if (s32ret)
{
    return -1;
}

/* set target framerate of VI channel*/
s32ret = HI_MPI_VI_SetFrameRate(0, 0, u32FrmRate);
if (s32ret)
{
    return -1;
}

/* enable VI channel*/
```



```
/* ... */  
/* also, you can set target framerate afer enable vi channel */
```

【相关主题】

无。

HI_MPI_VI_GetSrcFrameRate

【描述】

获取视频输入通道的目标帧率。

【语法】

```
HI_S32 HI_MPI_VI_GetSrcFrameRate(VI_DEV ViDevId, VI_CHN ViChn, HI_U32  
*pu32ViFramerate);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pu32ViFramerate	目标帧率指针。	输出

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_FAILED_NOTCONFIG	原始帧率未配置。



【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

如果未设置原始帧率，则返回原始帧率未配置的错误码。

【举例】

无。

【相关主题】

[HI_MPI_VI_SetFrameRate](#)

HI_MPI_VI_SetFrameRate

【描述】

设置视频输入通道的目标帧率。

【语法】

```
HI_S32 HI_MPI_VI_SetFrameRate(VI_DEV ViDevId, VI_CHN ViChn, HI_U32  
u32ViFramerate);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
u32ViFramerate	目标帧率。 取值范围：[0, 30]	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_NOT_SUPPORT	操作不支持。
HI_ERR_VI_INVALID_PARA	输入参数无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 支持视频输入通道设置指定范围内的任意目标帧率，以实现低帧率采集视频帧，降低系统性能消耗。
- 必须先设置 VI 通道的原始输入帧率，才能调用此接口设置目标帧率，且目标帧率不能大于原始帧率。
- 支持 VI 通道启用后再动态设置目标帧率。
- 如果用户未调用此接口设置 VI 目标帧率，VI 软件内部不做任何帧率控制。

【举例】

请参见 [HI_MPI_VI_SetPubAttr](#) 的举例。

【相关主题】

无。

HI_MPI_VI_GetFrameRate

【描述】

获取视频输入通道的目标帧率。

【语法】

```
HI_S32 HI_MPI_VI_GetFrameRate(VI_DEV ViDevId,VI_CHN ViChn,HI_U32
*pu32ViFramerate);
```

【参数】



参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pu32ViFramerate	目标帧率指针。	输出

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_FAILED_NOTCONFIG	目标帧率未设置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

如果未设置目标帧率，则返回目标帧率未配置的错误码。

【举例】

无。

【相关主题】

无。



HI_MPI_VI_SetUserPic

【描述】

设置用户图片的帧信息。

【语法】

```
HI_S32 HI_MPI_VI_SetUserPic(VI_DEV ViDevId, VI_CHN ViChn,  
VIDEO_FRAME_INFO_S *pstVFrame);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstVFrame	用户图片的帧信息结构指针。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_NOT_PERM	操作不允许。
HI_ERR_VI_INVALID_PARA	参数设置无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h



- 库文件：libmpi.a

【注意】

- 设置 VI 用户图片用于将 VI 通道输出的视频帧图像修改为用户配置的指定 YUV 图像数据，而不是 codec 的视频数据；此接口需要与 [HI_MPI_VI_EnableUserPic](#)、[HI_MPI_VI_DisableUserPic](#) 配合使用。
- 目前此接口中的参数 VI 设备和通道号未被使用，即设置用户图像针对的是所有 VI 通道，而不是具体某个 VI 通道。
- 设置完用户图片后，即可调用 [HI_MPI_VI_EnableUserPic](#) 接口对指定 VI 通道启用插入用户图片，此时 VI 通道输出的数据即为所配置的图片 YUV 数据；一般用于视频信号丢失时，VI 通道输出 NoVideo 图片。
- 配置的用户图片大小应该与 VI 通道大小相一致；如果不一致，VI 内部会自动将其缩放为 VI 通道大小。
- 用户图片的视频帧信息结构中，需要设置图片的宽度、高度、行间隔、YUV 格式以及 Y 分量和 C 分量数据的物理地址；可以从 MPP 公共缓冲池中获取一块相应大小的视频缓存块，从缓存块信息中得到存放 YUV 数据的物理地址，然后将物理地址映射到用户空间，即可对这块内存进行 YUV 数据的填充操作；注意只支持 semi-planar YUV420、semi-planar YUV422 的格式，因此填充数据时需要遵循先存 Y 分量数据，再存 UV 分量间插数据的存储顺序（小端字节序先 V 后 U）。
- VIU 模块在启用插入用户图片时，会直接使用设置的用户图片帧信息中的物理地址，因此设置完用户图片后，不应该释放或销毁其视频缓存块，除非确认不再使用。可以通过再次调用此接口设置另外一块 VideoBuffer 以修改图片信息。
- 启用 VI 通道或启用插入用户图片时，此接口都可以动态调用。
- 由于所有 VI 通道共用固定的用户图片内存，因此用户图片启用时，象 VI Overlay、VICoverEx 这样能更改源图像的功能是不允许同时开启的。

【举例】

```
HI_S32  s32ret;
HI_U32  u32Width;
HI_U32  u32Height;
HI_U32  u32LStride;
HI_U32  u32CStride;
HI_U32  u32LumaSize;
HI_U32  u32ChrmSize;
HI_U32  u32Size;
VB_BLK  VbBlk;
HI_U32  u32PhyAddr;
HI_U8   *pVirAddr;

/* you need get width and height of pictrue */
u32LumaSize = (u32LStride * u32Height);
u32ChrmSize = (u32CStride * u32Height) >> 2; /* 420 */
u32Size = u32LumaSize + (u32ChrmSize << 1);

/* get video buffer block form common pool */
```



```
VbBlk = HI_MPI_VB_GetBlock(VB_INVALID_POOLID, u32Size);
if (VB_INVALID_HANDLE == VbBlk)
{
    return -1;
}
/* get physical address*/
u32PhyAddr = HI_MPI_VB_Handle2PhysAddr(VbBlk);
if (0 == u32PhyAddr)
{
    return -1;
}

/* mmap physical address to virtual address*/
/* ... */

/* get pool id */
pstVFrameInfo->u32PoolId = HI_MPI_VB_Handle2PoolId(VbBlk);
if (VB_INVALID_POOLID == pstVFrameInfo->u32PoolId)
{
    return -1;
}

pstVFrameInfo->stVFrame.u32PhyAddr[0] = u32PhyAddr;
pstVFrameInfo->stVFrame.u32PhyAddr[1] = pstVFrameInfo->
>stVFrame.u32PhyAddr[0] + u32LumaSize;
pstVFrameInfo->stVFrame.u32PhyAddr[2] = pstVFrameInfo->
>stVFrame.u32PhyAddr[1] + u32ChrmSize;

pstVFrameInfo->stVFrame.pVirAddr[0] = pVirAddr;
pstVFrameInfo->stVFrame.pVirAddr[1] = pstVFrameInfo->
>stVFrame.pVirAddr[0] + u32LumaSize;
pstVFrameInfo->stVFrame.pVirAddr[2] = pstVFrameInfo->
>stVFrame.pVirAddr[1] + u32ChrmSize;

pstVFrameInfo->stVFrame.u32Width = u32Width;
pstVFrameInfo->stVFrame.u32Height = u32Height;
pstVFrameInfo->stVFrame.u32Stride[0] = u32LStride;
pstVFrameInfo->stVFrame.u32Stride[1] = u32CStride;
pstVFrameInfo->stVFrame.u32Stride[2] = u32CStride;
pstVFrameInfo->stVFrame.enPixelFormat =
PIXEL_FORMAT_YUV_SEMIPLANAR_420;

/* now you need get YUV Semi Palnar Data ,fill them to the virtual
address */
```



```
/* ... */
/* ... */

/* enabel VI channel ... */

/* first set user pic info*/
s32ret = HI_MPI_VI_SetUserPic(0, 0, pstVFrameInfo);
if (s32ret)
{
    return -1;
}

/* ... */

/* enable insert user pic if you need */
s32ret = HI_MPI_VI_EnableUserPic(0, 0);
if (s32ret)
{
    return -1;
}

/* ... */

/* disable insert user pic if you don't need */
s32ret = HI_MPI_VI_DisableUserPic(0, 0);
if (s32ret)
{
    return -1;
}
```

【相关主题】

无。

HI_MPI_VI_SetUserPicEx

【描述】

设置用户图片的扩展接口。

【语法】

```
HI_S32 HI_MPI_VI_SetUserPicEx(VI_DEV ViDevId, VI_CHN ViChn,
VI_USERPIC_ATTR_S *pstUsrPicAttr);
```

【参数】



参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstUsrPicAttr	用户图片的帧信息结构指针。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_PARA	参数无效
HI_ERR_VI_NOT_SUPPORT	操作不支持。
HI_ERR_VI_INVALID_PARA	参数设置无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 本接口覆盖了 HI_MPI_VI_SetUserPic 的功能，另外增加了一种用户图片模式：单色填充模式。
- 单色填充的用户图片模式一般需要与 VI Overlay 功能同时使用，VI_USERPIC_ATTR_S 中的 enUsrPicMode 配置为 VI_USERPIC_MODE_BGC，即为单色填充的用户图片模式，stUsrPicBg 配置为需要填充的颜色值。然后调用 HI_MPI_VI_EnableUserPic 及 HI_MPI_VI_DisableUserPic。用户图片功能启用时，



MPP 内部会按照配置的颜色填充 VI 视频帧，用户如果启用了 VI Overlay 功能，VI Overlay 会叠加到填充颜色的视频帧图像上（注意：VI Overlay 使用 TDE 实现，建议只用于 TDE 使用较少的板卡等产品，否则可能会造成系统负荷超出）。

- bPub 项目目前只支持设置为 TRUE，即所有 VI 通道共用相同配置。

【举例】

无。

【相关主题】

无。

HI_MPI_VI_EnableUserPic

【描述】

启用插入用户图片。

【语法】

```
HI_S32 HI_MPI_VI_EnableUserPic(VI_DEV ViDevId, VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。



接口返回值	含义
HI_ERR_VI_NOT_PERM	操作不允许。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 启用插入用户图片之前，需要先设置用户图片帧信息。
- 启用插入用户图片之后，当前 VI 通道即输出配置的用户图片帧数据。
- 此接口可以重复调用。

【举例】

请参见 [HI_MPI_VI_SetUserPic](#) 的举例。

【相关主题】

无。

HI_MPI_VI_DisableUserPic

【描述】

禁用插入用户图片。

【语法】

```
HI_S32 HI_MPI_VI_DisableUserPic(VI_DEV ViDevId, VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。



返回值	描述
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_NOT_PERM	操作不允许。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- VI 通道不再需要输出用户图片时，应该调用此接口以恢复输出 AD 的原始视频数据。
- 此接口可以重复调用。

【举例】

请参见 [HI_MPI_VI_SetUserPic](#) 的举例。

【相关主题】

无。

HI_MPI_VI_SetMinorChnAttr

【描述】

设置 VI 通道次属性。

【语法】

```
HI_S32 HI_MPI_VI_SetMinorChnAttr(VI_DEV ViDevId, VI_CHN ViChn, const  
VI_CHN_ATTR_S *pstAttr);
```

【参数】



参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstAttr	VI 通道次属性指针。 动态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_PARA	输入参数无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入设备或通道的属性未配置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 默认情况下，VI 通道将按 [HI_MPI_VI_SetChnAttr](#) 接口中设置的分辨率来采集连续的视频帧图像；使用 [HI_MPI_VI_SetMinorChnAttr](#) 接口的目的是设置另外一个分辨率，VI 通道将会按照配置的帧率来轮流采集两种不同分辨率的视频图像。



- 所谓“通道次（要）属性”是相对主（要）属性而言的，主属性在函数 [HI_MPI_VI_SetChnAttr](#) 中配置；主、次属性的概念只有在对 VI 做了帧率控制时才有效，VI 采集图像时，按配置的目标帧率采集主属性大小的图像，其他帧则按次属性大小采集。

例如：设置原始帧率为 25，目标帧率为 5，主属性大小为 D1，次属性大小为 CIF，则实际视频捕获则为 5 帧输出 D1 大小图像，剩余 20 帧输出 CIF 大小图像。主属性和次属性的配置关系如表 3-3 所示。

表3-3 主属性和次属性的配置关系

属性	AD	主属性	次属性
分辨率等	-	SetChnAttr	SetChnMinorAttr
帧率	SetSrcFrameRate	FrameRate	SrcFrameRate - FrameRate

- 如果已配置 VI 帧率，但未配置通道次属性，则按照目标帧率只输出主属性大小的视频图像。
- 如果已经配置了通道次属性，而中途需要关闭次属性图像采集，则可以调用 [HI_MPI_VI_ClearChnMinorAttr](#) 接口将此属性清除。
- FrameRate 配置为 0，表示不输出主属性图像，全部输出次属性图像。
- FrameRate 配置为与 SrcFrameRate 相等，则表示不输出次属性图像，全部输出主属性图像。
- 可以在 VI 通道启用后动态更改通道次属性。
- 使用此方式输出图像大小持续变化的 VI 视频图像给 VENC 和 VO 模块时，需要对 VENC 和 VO 的图像显示做特殊配置：
 - 调用 [HI_MPI_VPP_SetConf](#) 设置编码通道组缩放模式为 [VPP_SCALE_MODE_USEBOTTOM2](#)。
 - 调用 [HI_MPI_VO_SetChnField](#) 设置 VO 显示底场。

【举例】

```
HI_S32 s32ret;
HI_U32 u32SrcFrmRate = 25;
HI_U32 u32FrmRate = 5;
VI_CHN_ATTR_S stAttr;

/* set public attribute of VI device*/
/* ... */

/* enable VI device*/
/* ... */

/* set main attribute of VI channel, size is D1 */
/* ... */
```



```
HI_MPI_VI_GetChnAttr(ViDevId, ViChn, &stAttr);
stAttr.enCapSel = VI_CAPSEL_BOTTOM;
stAttr.bDownScale = HI_TRUE;

/* set minor attribute of VI channel, size is CIF */
if (HI_MPI_VI_SetChnMinorAttr(ViDevId, ViChn, &stAttr))
{
    BBIT_ERR("set chn attr ex fail\n");
    return -1;
}

/* set SRC framerate of VI channel*/
s32ret = HI_MPI_VI_SetSrcFrameRate(0, 0, u32SrcFrmRate);
if (s32ret)
{
    return -1;
}

/* set target framerate of VI channel*/
s32ret = HI_MPI_VI_SetFrameRate(0, 0, u32FrmRate);
if (s32ret)
{
    return -1;
}

/* enable VI channel*/
/* ... */

/* enable VO dev and VO chn */
/* ... */

/* set vo field bottom */
HI_MPI_VO_SetChnField(i, VO_FIELD_BOTTOM);

/* create group/venc */
/* ... */

/* set group scale mode VPP_SCALE_MODE_USEBOTTOM2 */
VIDEO_PREPROC_CONF_S stConf;
HI_MPI_VPP_GetConf(s32Grp, &stConf);
stConf.enScaleMode = VPP_SCALE_MODE_USEBOTTOM2;
HI_MPI_VPP_SetConf(s32Grp, &stConf);
/* ... */
```



```
/* if you not need minor attr capture */  
if (HI_MPI_VI_ClearChnMinorAttr(ViDevId, ViChn))  
{  
    BBIT_ERR("clear chn minor attr fail\n");  
    return -1;  
}
```

【相关主题】

[HI_MPI_VI_GetMinorChnAttr](#)

HI_MPI_VI_GetMinorChnAttr

【描述】

获取 VI 通道次属性。

【语法】

```
HI_S32 HI_MPI_VI_GetMinorChnAttr(VI_DEV ViDevId, VI_CHN ViChn,  
VIDEO_FRAME_INFO_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstAttr	VI 通道属性指针。 动态属性。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入属性未配置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：[hi_comm_vi.h](#)、[mpi_vi.h](#)
- 库文件：[libmpi.a](#)

【注意】

必须先设置通道属性再获取属性，否则将返回 [HI_ERR_VI_FAILED_NOTCONFIG](#)。

【举例】

请参见 [HI_MPI_VI_SetMinorChnAttr](#) 的举例。

【相关主题】

[HI_MPI_VI_SetChnAttr](#)

HI_MPI_VI_ClearChnMinorAttr

【描述】

清除 VI 通道次属性。

【语法】

```
HI_S32 HI_MPI_VI_ClearChnMinorAttr(VI_DEV ViDevId,VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 如果已经配置通道次属性，可以调用此接口再将其取消。
- 本接口可以在通道启用状态下调用。

【举例】

请参见 [HI_MPI_VI_SetMinorChnAttr](#) 的举例。

【相关主题】

[HI_MPI_VI_SetChnAttr](#)

HI_MPI_VI_EnableCascade

【描述】

VI 启用 VI-VO 级联功能。

【语法】

```
HI_S32 HI_MPI_VI_EnableCascade(VI_DEV ViDevId, VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入



参数名称	描述	输入/输出
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_NOT_SUPPORT	操作不支持。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- VIVO 多片级联环境中，主片应该调用此接口启用 VI 的级联功能。
- 启用级联功能的 VI 通道必须配置为 BT.1120 逐行模式。
- Hi3515 不支持视频级联功能。

【举例】

无。

【相关主题】

- [HI_MPI_VI_DisableCascade](#)
- [HI_MPI_VI_SetChnAttr](#)



HI_MPI_VI_DisableCascade

【描述】

VI 禁用 VIVO 级联功能。

【语法】

```
HI_S32 HI_MPI_VI_DisableCascade(VI_DEV ViDevId, VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

无。

【举例】



无。

【相关主题】

[HI_MPI_VI_EnableCascade](#)

HI_MPI_VI_SetAdChnId

【描述】

设置与 VI 通道对应的 ADC 的 CHID。

【语法】

```
HI_S32 HI_MPI_VI_SetAdChnId(VI_DEV ViDevId, VI_CHN ViChn, HI_U32  
u32AdChnId);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
u32AdChnId	ADC 的 CHID 序号。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】



- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 接口时序为 BT.656 格式时，支持 108M 4D1 和 54M 2D1 时分复用工作模式，与有此类功能的 codec 对接时，Hi3520 VI 内部需要通过 codec 提供的 CHID 标识来识别不同的通道。
- Hi3520/Hi3515 通过 SAV/EAV 中纠错位的低 2bit 来读取通道号标识。
- 4D1 模式时默认的 AdChnId 配置如表 3-4 所示。

表3-4 4D1 模式时 AdChnId 的默认配置

类别	编号															
ViDev	0				1				2				3			
ViChn	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
AdChn	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

- 2D1 模式时默认的 AdChnId 配置如表 3-5 所示。

表3-5 2D1 模式时 AdChnId 的默认配置

类别	编号							
ViDev	0		1		2		3	
ViChn	0	1	0	1	0	1	0	1
AdChn	0	1	2	3	0	1	2	3

- 如果默认 AdChn 配置不能满足要求，需要调用本接口进行修改。由于 SDK 内部是在调用 HI_MPI_VI_SetPubAttr 接口中载入默认配置，因此请在调用 HI_MPI_VI_SetPubAttr 接口后再调用本接口修改 AdChnId 配置。

【举例】

无。

【相关主题】

[HI_MPI_VI_GetAdChnId](#)

HI_MPI_VI_GetAdChnId

【描述】

获取与 VI 通道对应的 ADC 的 CHID。

【语法】



```
HI_S32 HI_MPI_VI_GetAdChnId(VI_DEV ViDevId, VI_CHN ViChn, HI_U32
*pu32AdChnId);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pu32AdChnId	ADC 的 CHID 序号。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】



HI_MPI_VI_SetAdChnId

HI_MPI_VI_SetSrcCfg

【描述】

设置 VI 通道视频源配置。

【语法】

```
HI_S32 HI_MPI_VI_SetSrcCfg(VI_DEV ViDevId, VI_CHN ViChn, const  
VI_SRC_CFG_S *pstSrcCfg);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstSrcCfg	视频源配置结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_VI_INVALID_PARA	参数无效。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h



- 库文件：libmpi.a

【注意】

- 视频源配置结构体中的每一项都有预设的默认值
 - 如果不需要修改，则不必调用此接口。
 - 如果需要修改某项，建议先调用 `HI_MPI_VI_GetSrcCfg` 获取属性，修改后再调用 `HI_MPI_VI_SetSrcCfg` 设置属性。
- 视频源配置结构体中的 `enSrcField`，表示视频源数据的帧场格式，包括隔行间插模式、逐行帧模式以及自动模式；默认是自动模式，即根据用户配置的 VI 接口时序，内部自动判断是隔行还是逐行，例如 BT656、BT601 自动选择为隔行，DC 模式则自动选择为逐行；如果自动选择的模式与实际的视频源数据格式不匹配，则用户可以调用此接口手动修改为正确的格式。
- 视频源配置结构体中的 `bEnNoIntDet`，表示是否启用输入源的无中断检测，此功能需要与 VI 的用户图片相关接口配合使用，一般用于无视频信号时 VI 输出 NoVideo 用户图片，详细使用流程请参考开发包中的 `sample_vio.c` 相应样例。

【举例】

无。

【相关主题】

[HI_MPI_VI_GetSrcCfg](#)

HI_MPI_VI_GetSrcCfg

【描述】

获取 VI 通道视频源配置。

【语法】

```
HI_S32 HI_MPI_VI_GetSrcCfg(VI_DEV ViDevId, VI_CHN ViChn, VI_SRC_CFG_S  
*pstSrcCfg);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstSrcCfg	视频源配置结构体指针。	输出

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

[HI_MPI_VI_SetSrcCfg](#)

HI_MPI_VI_GetFd

【描述】

获取 VI 通道对应的设备文件句柄。

【语法】

```
HI_S32 HI_MPI_VI_GetFd(VI_DEV ViDevId, VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入



参数名称	描述	输入/输出
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

无。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

用户可以获取文件句柄实现多通道 select 获取视频帧数据。

【举例】

```
VI_DEV ViDevId = 0;
VI_CHN ViChn = 0;
VIDEO_NORM_E stNorm;
HI_S32 s32ViFd = 0;

/* enable vi dev and vi chn */

/* get video frame from vi chn */
s32ViFd = HI_MPI_VI_GetFd(ViDevId, ViChn);
if (s32ViFd <= 0)
{
    return HI_FAILURE;
}
```

【相关主题】

无。



3.4 数据类型

视频输入相关数据类型定义如下：

- [VI_CAPSEL_E](#)：定义视频输入捕获图像的帧场选择。
- [VI_INPUT_MODE_E](#)：定义视频输入接口模式。
- [VI_WORK_MODE_E](#)：定义视频输入工作模式。
- [VI_CHN_ATTR_S](#)：定义视频输入通道属性结构体。
- [VI_PUB_ATTR_S](#)：定义视频输入设备的公共属性结构体。
- [VI_CH_LUM_S](#)：定义视频输入通道的图像亮度信息结构体。
- [VI_SRC_FIELD_E](#)：定义视频输入通道的视频源图像的帧场格式。
- [VI_SRC_CFG_S](#)：定义视频输入通道的视频源配置。

VI_CAPSEL_E

【说明】

定义视频输入捕获图像的帧场选择。

【定义】

```
typedef enum hiVI_CAPSEL_E
{
    VI_CAPSEL_TOP=0,
    VI_CAPSEL_BOTTOM,
    VI_CAPSEL_BOTH,
    VI_CAPSEL_BUTT
} VI_CAPSEL_E;
```

【成员】

成员名称	描述
VI_CAPSEL_TOP	选择顶场。
VI_CAPSEL_BOTTOM	选择底场（推荐）。
VI_CAPSEL_BOTH	选择顶底两场。

【注意事项】

无。

【相关数据类型及接口】

[VI_CHN_ATTR_S](#)



VI_INPUT_MODE_E

【说明】

定义视频输入接口模式。

【定义】

```
typedef enum hiVI_INPUT_MODE_E
{
    VI_MODE_BT656=0,
    VI_MODE_BT601,
    VI_MODE_DIGITAL_CAMERA,
    VI_MODE_BT1120_PROGRESSIVE,
    VI_MODE_BT1120_INTERLACED,
    VI_MODE_BT601_SEP,
    VI_MODE_DIGITAL_CAMERA_SEP,
    VI_MODE_BUTT
} VI_INPUT_MODE_E;
```

【成员】

成员名称	描述
VI_MODE_BT656	ITUR BT.656 模式。
VI_MODE_BT601	ITUR BT.601 模式。
VI_MODE_BT1120_PROGRESSIVE	ITUR BT.1120 逐行模式。
VI_MODE_BT1120_INTERLACED	ITUR BT.1120 隔行模式。
VI_MODE_DIGITAL_CAMERA	数字摄像头模式。
VI_MODE_BT601_SEP	ITUR BT.601 亮色度分离模式。
VI_MODE_DIGITAL_CAMERA_SEP	数字摄像头亮色度分离模式。

【注意事项】

无。

【相关数据类型及接口】

[VI_PUB_ATTR_S](#)

VI_WORK_MODE_E

【说明】

定义视频输入工作模式。

【定义】



```
typedef enum hiVI_WORK_MODE_E
{
    VI_WORK_MODE_1D1=0,
    VI_WORK_MODE_2D1,
    VI_WORK_MODE_4HALFD1,
    VI_WORK_MODE_4D1,
    VI_WORK_MODE_BUTT
} VI_WORK_MODE_E;
```

【成员】

成员名称	描述
VI_WORK_MODE_1D1	1D1 工作模式。
VI_WORK_MODE_2D1	2D1 工作模式。
VI_WORK_MODE_4HALFD1	4Half D1 工作模式。
VI_WORK_MODE_4D1	4D1 工作模式。

【注意事项】

无。

【相关数据类型及接口】

[VI_PUB_ATTR_S](#)

VI_CHN_ATTR_S

【说明】

定义视频输入通道属性结构体。

【定义】

```
typedef struct hiVI_CHN_ATTR_S
{
    RECT_S          stCapRect;
    VI\_CAPSEL\_E     enCapSel;
    HI_BOOL         bDownScale;
    HI_BOOL         bChromaResample;
    HI_BOOL         bHighPri;
    PIXEL_FORMAT_E  enViPixFormat;
}VI_CHN_ATTR_S;
```

【成员】



成员名称	描述
stCapRect	通道捕获区域属性。 动态属性。
enCapSel	帧场选择。 动态属性。
bDownScale	1/2 水平压缩选择。 动态属性。
bChromaResample	色度重采样选择。 动态属性。
bHighPri	高优先级选择。 动态属性。
enViPixelFormat	像素格式。 动态属性。

【注意事项】

无。

【相关数据类型及接口】

- RECT_S
- [VI_CAPSEL_E](#)
- [HI_MPI_VI_SetChnAttr](#)

VI_PUB_ATTR_S

【说明】

定义视频输入设备的公共属性结构体。

【定义】

```
typedef struct hiVI_PUB_ATTR_S
{
    VI\_INPUT\_MODE\_E enInputMode;
    VI\_WORK\_MODE\_E enWorkMode;
    VIDEO_NORM_E enViNorm;
    HI_BOOL bIsChromaChn;
    HI_BOOL bChromaSwap;
}VI_PUB_ATTR_S;
```

【成员】



成员名称	描述
enInputMode	视频输入接口模式。 静态属性。
enWorkMode	视频输入工作模式。 静态属性。
enViNorm	接口制式。 静态属性。
bIsChromaChn	是否色度通道。
bChromaSwap	是否色度数据交换。

【注意事项】

无。

【相关数据类型及接口】

- [VI_INPUT_MODE_E](#)
- [VI_WORK_MODE_E](#)
- [HI_MPI_VI_SetPubAttr](#)

VI_CH_LUM_S

【说明】

定义视频输入通道的图像亮度信息结构体。

【定义】

```
typedef struct hiVI_CH_LUM_S
{
    HI_U32  u32FramId;
    HI_U32  u32Lum;
}VI_CH_LUM_S;
```

【成员】

成员名称	描述
u32FramId	原始图像帧序号。
u32Lum	亮度值。

【注意事项】

无。



【相关数据类型及接口】

[HI_MPI_VI_GetChnLuma](#)

VI_SRC_FIELD_E

【说明】

定义视频输入通道的视频源图像的帧场格式。

【定义】

```
typedef enum hiVI_SRC_FIELD_E
{
    VI_SRC_FIELD_AUTO          = 0,
    VI_SRC_FIELD_INTERLACED,
    VI_SRC_FIELD_FRAME,
    VI_SRC_FIELD_BUTT
} VI_SRC_FIELD_E;
```

【成员】

成员名称	描述
VI_SRC_FIELD_AUTO	自动选择（根据 VI 设备属性中的接口模式）。
VI_SRC_FIELD_INTERLACED	隔行模式。
VI_SRC_FIELD_FRAME	逐行模式。

【注意事项】

请参考 [HI_MPI_VI_SetSrcCfg](#) 接口说明。

【相关数据类型及接口】

[VI_SRC_CFG_S](#)

VI_SRC_CFG_S

【说明】

定义视频输入通道的视频源配置。

【定义】

```
typedef struct hiVI_SRC_CFG_S
{
    VI_SRC_FIELD_E  enSrcField;
    HI_BOOL         bEnNoIntDet;
    HI_U32          u32Resev;
} VI_SRC_CFG_S;;
```



【成员】

成员名称	描述
enSrcField	视频源图像的帧场格式。
bEnNoIntDet	是否启用无中断检测。
u32Resev	保留项（无意义）。

【注意事项】

请参考 [HI_MPI_VI_SetSrcCfg](#) 接口说明。

【相关数据类型及接口】

- [VI_SRC_FIELD_E](#)
- [HI_MPI_VI_SetSrcCfg](#)
- [HI_MPI_VI_GetSrcCfg](#)

VI_USERPIC_MODE_E

【说明】

定义用户图片模式。

【定义】

```
typedef enum hi_VI_USERPIC_MODE_E
{
    VI_USERPIC_MODE_PIC = 0,
    VI_USERPIC_MODE_BGC,
    VI_USERPIC_MODE_BUTT,
} VI_USERPIC_MODE_E;
```

【成员】

成员名称	描述
VI_USERPIC_MODE_PIC	YUV 图片模式。
VI_USERPIC_MODE_BGC	背景色模式。

【相关数据类型及接口】

- [VI_SRC_FIELD_ES](#)

VI_USERPIC_BGC_S

【说明】



定义用户图片背景色的配置。

【定义】

```
typedef struct hiVI_USERPIC_BGC_S
{
    HI_U32          u32BgColor;
} VI_USERPIC_BGC_S;
```

【成员】

成员名称	描述
u32BgColor	背景色。

【相关数据类型及接口】

- [VI_SRC_FIELD_ES](#)

VI_USERPIC_ATTR_S

【说明】

定义用户图片属性配置结构体。

【定义】

```
typedef struct hiVI_USERPIC_ATTR_S
{
    HI_BOOL          bPub;
    VI_USERPIC_MODE_E    enUsrPicMode;
    union
    {
        VIDEO_FRAME_INFO_S    stUsrPicFrm;
        VI_USERPIC_BGC_S      stUsrPicBg;
    }unUsrPic;
} VI_USERPIC_ATTR_S;
```

【成员】

成员名称	描述
bPub	是否公共属性。
enUsrPicMode	用户图片模式。
unUsrPic	用户图片属性。

【相关数据类型及接口】



- [VI_SRC_FIELD_E](#)

3.5 错误码

视频输入 API 错误码如[表 3-6](#) 所示。

表3-6 视频输入 API 错误码

错误代码	宏定义	描述
0xA0108001	HI_ERR_VI_INVALID_DEVID	视频输入设备号无效
0xA0108002	HI_ERR_VI_INVALID_CHNID	视频输入通道号无效
0xA0108003	HI_ERR_VI_INVALID_PARA	视频输入参数设置无效
0xA0108006	HI_ERR_VI_INVALID_NULL_PTR	输入参数空指针错误
0xA0108007	HI_ERR_VI_FAILED_NOTCONFIG	视频输入通道属性未配置
0xA0108008	HI_ERR_VI_NOT_SUPPORT	操作不支持
0xA0108009	HI_ERR_VI_NOT_PERM	操作不允许
0xA010800C	HI_ERR_VI_NOMEM	分配内存失败
0xA010800E	HI_ERR_VI_BUF_EMPTY	视频输入缓存为空
0xA010800F	HI_ERR_VI_BUF_FULL	视频输入缓存为满
0xA0108010	HI_ERR_VI_SYS_NOTREADY	视频输入系统未初始化
0xA0108012	HI_ERR_VI_BUSY	视频输入系统忙
0xA0108040	HI_ERR_VI_FAILED_NOTENABLE	视频输入设备或通道未启用
0xA0108041	HI_ERR_VI_FAILED_NOTDISABLE	视频输入设备未禁用
0xA0108042	HI_ERR_VI_FAILED_CHNOTDISABLE	视频输入通道未禁用



目 录

4 视频输出.....	4-1
4.1 概述.....	4-1
4.2 重要概念.....	4-1
4.3 API 参考	4-2
4.4 数据类型.....	4-109
4.5 错误码.....	4-129



插图目录

图 4-1 视频层属性的相关概念示意（基本处理模式）	4-114
图 4-2 视频层属性的相关概念示意（设置有效区处理模式）	4-115
图 4-3 局部放大原理示意.....	4-124



表格目录

表 4-1 视频输出 API 错误码	4-129
--------------------------	-------



4 视频输出

4.1 概述

VOU (Video Output Unit) 模块主动从内存相应位置读取视频和图形数据, 并通过相应的显示设备输出。Hi3520 支持 HD (高清)、AD (辅助) 和 SD (标清) 三个显示设备, 同时支持 5 个图形层。Hi3515 支持 HD、SD 两个显示设备, 同时支持 4 个图形层。VOU 的每个设备最多支持 32 路通道视频图像显示, 通道之间支持按优先级叠加。

4.2 重要概念

- 视频级联

从片预览图像的输出以 BT.1120 数据格式连接到下级芯片的视频输入端, 通过这样的方式连接多个芯片, 将芯片上的图像传输到主片同时输出显示。Hi3520 的 HD 设备支持视频级联。视频级联的详细信息请参见《Hi3520 视频级联应用指南》。



说明

由于 Hi3515 不支持视频级联, 文档中涉及视频级联的 API、数据类型及相关描述仅针对 Hi3520。

- 同步回放

将多个视频输出通道的解码图像按照其时间戳进行统一输出。为了达到多个画面的回放步调一致, 先将多个通道注册到一个同步组中, 然后同步组按照播放帧率将该时刻的图像输出进行同步。

- 通道优先级

当同一输出设备上有多通道同时输出显示时, 按照优先级顺序对输出图像进行叠加。当各个通道的画面有重叠区域时, 优先级高的图像显示在上层。如果各个通道优先级一致, 则通道号越大的默认优先级越高。

- 分辨率

分辨率主要有以下 3 种概念:

- 设备分辨率指该设备的输出有效像素点数, 由设备时序决定。
- 显示分辨率指画面在显示设备上的有效显示区域。



– 图像分辨率指图像本身的有效像素点数。

- 局部放大

将显示画面上的一部分图像进行放大显示，放大显示的源区域从源图像上截取，放大显示的目标区域是该显示通道的通道大小。

- 图形层绑定

Hi3520/Hi3515 有多个视频层和图形层。其中有两个图形层 G1 和 HC（硬件鼠标层）可以动态绑定到高清设备或者标清设备（这里的标清设备指 Hi3520 上的 AD 或者 Hi3515 上的 SD）上，即这两个图形层上显示的画面可以在高清设备和标清设备上移动。

4.3 API 参考

视频输出（VO）实现启用视频输出设备或通道、发送视频数据到输出通道等功能。

该功能模块提供以下 MPI：

设备操作：

- [HI_MPI_VO_Enable](#)：启用视频输出设备。
- [HI_MPI_VO_Disable](#)：禁用视频输出设备。
- [HI_MPI_VO_SetPubAttr](#)：设置视频输出设备的公共属性。
- [HI_MPI_VO_GetPubAttr](#)：获取视频输出设备的公共属性。
- [HI_MPI_VO_CloseFd](#)：关闭所有视频输出设备的 Fd。

视频层操作：

- [HI_MPI_VO_EnableVideoLayer](#)：使能视频层。
- [HI_MPI_VO_DisableVideoLayer](#)：禁止视频层。
- [HI_MPI_VO_SetVideoLayerAttr](#)：设置视频层属性。
- [HI_MPI_VO_GetVideoLayerAttr](#)：获取视频层属性。
- [HI_MPI_VO_SetDisplayRect](#)：设置视频输出的显示区域。
- [HI_MPI_VO_GetDisplayRect](#)：获取视频输出的显示区域。
- [HI_MPI_VO_SetValidImgRect](#)：设置图像输出有效区域。
- [HI_MPI_VO_GetValidImgRect](#)：获取图像输出有效区域。

通道操作：

- [HI_MPI_VO_EnableChn](#)：启用指定的视频输出通道。
- [HI_MPI_VO_DisableChn](#)：禁用指定的视频输出通道。
- [HI_MPI_VO_SetChnAttr](#)：设置指定视频输出通道的属性。
- [HI_MPI_VO_GetChnAttr](#)：获取指定视频输出通道的属性。
- [HI_MPI_VO_SendFrame](#)：将视频图像送入指定视频输出通道显示。
- [HI_MPI_VO_SetChnField](#)：设置指定视频输出通道的帧场显示策略。
- [HI_MPI_VO_GetChnField](#)：获取指定视频输出通道的帧场显示策略。



- [HI_MPI_VO_SetChnFrameRate](#): 设置指定视频输出通道的显示帧率。
- [HI_MPI_VO_GetChnFrameRate](#): 获取指定视频输出通道的显示帧率。
- [HI_MPI_VO_ChnPause](#): 暂停指定的视频输出通道。
- [HI_MPI_VO_ChnResume](#): 恢复指定的视频输出通道。
- [HI_MPI_VO_ChnStep](#): 单帧播放指定的视频输出通道。
- [HI_MPI_VO_SetChnFilter](#): 设置指定视频输出通道的缩放系数。
- [HI_MPI_VO_GetChnFilter](#): 获取指定视频输出通道的缩放系数。
- [HI_MPI_VO_SetZoomInWindow](#): 设置视频输出局部放大窗口。
- [HI_MPI_VO_GetZoomInWindow](#): 获取视频输出局部放大窗口参数。
- [HI_MPI_VO_SetZoomInRatio](#): 按比例设置视频输出窗口的局部放大。
- [HI_MPI_VO_GetZoomInRatio](#): 获取视频输出局部放大比例。
- [HI_MPI_VO_GetChnPts](#): 获取视频输出通道当前播放图像的时间戳。
- [HI_MPI_VO_ChnShow](#): 设置显示通道。
- [HI_MPI_VO_ChnHide](#): 设置隐藏通道。
- [HI_MPI_VO_QueryChnStat](#): 查询视频输出通道状态。
- [HI_MPI_VO_SetChnSrcAttr](#): 设置通道数据源类型。
- [HI_MPI_VO_GetChnFrame](#): 获取输出通道图像数据。
- [HI_MPI_VO_ReleaseChnFrame](#): 释放输出通道图像数据。
- [HI_MPI_VO_ClearChnBuffer](#): 清除视频输出通道缓冲。
- [HI_MPI_VO_SetAttrBegin](#): 设置属性开始。
- [HI_MPI_VO_SetAttrEnd](#): 设置属性结束。

VBI 操作:

- [HI_MPI_VO_SetVbiInfo](#): 设置图像附加数据信息。
- [HI_MPI_VO_ClrVbiInfo](#): 清除图像附加数据信息。
- [HI_MPI_VO_GetVbiInfo](#): 获取图像附加数据信息。

显示操作:

- [HI_MPI_VO_GetScreenFrame](#): 获取输出屏幕图像数据。
- [HI_MPI_VO_ReleaseScreenFrame](#): 释放输出屏幕图像数据。
- [HI_MPI_VO_SetDevCSC](#): 设置设备输出图像效果。
- [HI_MPI_VO_GetDevCSC](#): 获取设备输出图像效果。
- [HI_MPI_VO_SetScreenFilter](#): 设置指定视频层的滤波系数。
- [HI_MPI_VO_GetScreenFilter](#): 获取指定视频层的滤波系数。
- [HI_MPI_VO_SetDispBufLen](#): 设置显示缓冲的长度。
- [HI_MPI_VO_GetDispBufLen](#): 获取显示缓冲的长度。
- [HI_MPI_VO_SetSolidDraw](#): 设置屏幕绘制属性。
- [HI_MPI_VO_GetSolidDraw](#): 获取屏幕绘制属性。

级联操作:



- [HI_MPI_VO_EnableCascade](#): 使能视频级联。
- [HI_MPI_VO_DisableCascade](#): 禁止视频级联。
- [HI_MPI_VO_SetCascadeMode](#): 设置视频级联模式。
- [HI_MPI_VO_GetCascadeMode](#): 获取视频级联模式。
- [HI_MPI_VO_SetCascadePattern](#): 设置视频级联画面样式。
- [HI_MPI_VO_GetCascadePattern](#): 获取视频级联画面样式。
- [HI_MPI_VO_CascadePosBindChn](#): 绑定级联区域与视频输出通道。
- [HI_MPI_VO_CascadePosUnBindChn](#): 解绑定级联区域与视频输出通道。

同步组操作:

- [HI_MPI_VO_CreateSyncGroup](#): 创建视频输出同步组。
- [HI_MPI_VO_DestroySyncGroup](#): 销毁视频输出同步组。
- [HI_MPI_VO_RegisterSyncGroup](#): 注册视频输出通道到视频输出同步组。
- [HI_MPI_VO_UnRegisterSyncGroup](#): 注销视频输出通道到视频输出同步组。
- [HI_MPI_VO_SyncGroupStart](#): 启动视频输出同步组。
- [HI_MPI_VO_SyncGroupStop](#): 停止视频输出同步组。
- [HI_MPI_VO_SetSyncGroupFrameRate](#): 设置视频输出同步组帧率。
- [HI_MPI_VO_GetSyncGroupFrameRate](#): 获取视频同步组帧率。
- [HI_MPI_VO_PauseSyncGroup](#): 暂停视频输出同步组。
- [HI_MPI_VO_ResumeSyncGroup](#): 恢复视频输出同步组。
- [HI_MPI_VO_StepSyncGroup](#): 单帧播放视频输出同步组。
- [HI_MPI_VO_SetSyncGroupBase](#): 设置多通道同步组的基准时间戳。
- [HI_MPI_VO_GetSyncGroupBase](#): 获取多通道同步组的基准时间戳。

HI_MPI_VO_Enable

【描述】

启用视频输出设备。

【语法】

```
HI_S32 HI_MPI_VO_Enable (VO_DEV VoDev);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围: [0, VO_MAX_DEV_NUM)。	输入

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备 Fd 未打开。
HI_ERR_VO_DEV_NOT_CONFIG	视频输出设备没有配置。
HI_ERR_VO_DEV_HAS_ENABLED	视频输出设备已经使能。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 由于系统没有初始化设备为使能状态，所以在使用视频输出功能前必须先进行设备使能操作。
- 在调用设备使能前，必须对设备公共属性进行配置，否则返回设备未配置错误。
- 为适应开机画面与正常操作界面间顺畅切换，此处需要检查 VO 硬件是否已经使能，如果已使能则返回 [HI_ERR_VO_DEV_HAS_ENABLED](#)，且沿用已有时序配置。如果希望更改 VO 的时序配置，则需要先调用 [HI_MPI_VO_Disable](#) 接口，强制关闭 VO 硬件后再使能。

【举例】

```
HI_S32 s32Ret;
VO_DEV VoDev = 0;
VO_PUB_ATTR_S stPubAttr;

s32Ret = HI_MPI_VO_GetPubAttr(VoDev, &stPubAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Get device attributes failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}
```



```
stPubAttr.u32BgColor = 0xff;
stPubAttr.enIntfType = VO_INTF_VGA;
stPubAttr.enIntfSync = VO_OUTPUT_1280x1024_60;

s32Ret = HI_MPI_VO_SetPubAttr(VoDev, &stPubAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Set device attributes failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VO_Enable(VoDev);
if (s32Ret != HI_SUCCESS)
{
    printf("Enable vo dev %d failed with errno %#x!\n", VoDev, s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VO_Disable(VoDev);
if (s32Ret != HI_SUCCESS)
{
    printf("Enable vo dev %d failed with errno %#x!\n", VoDev, s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VO_CloseFd();
if (s32Ret != HI_SUCCESS)
{
    printf("Some device is not disable with errno %#x!\n", VoDev, s32Ret);
    return HI_FAILURE;
}
```

【相关主题】[HI_MPI_VO_Disable](#)

HI_MPI_VO_Disable

【描述】

禁用视频输出设备。

【语法】

```
HI_S32 HI_MPI_VO_Disable(VO\_DEV VoDev);
```

【参数】



参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备Fd未打开。
HI_ERR_VO_VIDEO_NOT_DISABLE	视频层未禁止。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 设备禁止前必须先禁止该设备上的视频层。
- 调用设备使能接口后，如果未调用该接口进行禁止，则VO设备将一直保持使能状态，并且下次设置设备属性时不会生效。

【举例】

请参见[HI_MPI_VO_Enable](#)的举例。

【相关主题】

[HI_MPI_VO_Enable](#)

HI_MPI_VO_SetPubAttr

【描述】

配置视频输出设备的公共属性。



【语法】

```
HI_S32 HI_MPI_VO_SetPubAttr(VO_DEV VoDev, const VO_PUB_ATTR_S
*pstPubAttr);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstPubAttr	视频输出设备公共属性结构体指针。 静态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_NULL_PTR	参数指针为空。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备Fd未打开。
HI_ERR_VO_ILLEGAL_PARAM	参数无效。
HI_ERR_VO_NOT_SUPPORT	系统不支持的视频输出属性。
HI_ERR_VO_DEV_NOT_DISABLE	设备未禁止。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 视频输出设备属性为静态属性，必须在执行 HI_MPI_VO_Enable 前配置。



- 虚拟设备不受公共属性限制，因此设置虚拟设备属性时该接口返回成功。

【举例】

请参见 [HI_MPI_VO_Enable](#) 的举例。

【相关主题】

[HI_MPI_VO_GetPubAttr](#)

HI_MPI_VO_GetPubAttr

【描述】

获取视频输出设备的公共属性。

【语法】

```
HI_S32 HI_MPI_VO_GetPubAttr(VO_DEV VoDev, VO_PUB_ATTR_S *pstPubAttr);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstPubAttr	视频输出设备公共属性结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_NULL_PTR	参数指针为空。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

**【注意】**

建议在设置设备公共属性前先获取属性，这样就可以只设置需要改变的配置项。

【举例】

请参见 [HI_MPI_VO_Enable](#) 的举例。

【相关主题】

[HI_MPI_VO_SetPubAttr](#)

HI_MPI_VO_CloseFd

【描述】

关闭所有视频输出设备的 Fd。

【语法】

```
HI_S32 HI_MPI_VO_CloseFd(HI_VOID);
```

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_BUSY	设备忙，即有设备还未禁止。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

该接口必须在所有设备都禁止的情况下使用。

【举例】

请参见 [HI_MPI_VO_Enable](#) 的举例。



【相关主题】

[HI_MPI_VO_Enable](#)

HI_MPI_VO_EnableVideoLayer

【描述】

使能视频层。

【语法】

```
HI_S32 HI_MPI_VO_EnableVideoLayer (VO_DEV VoDev);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备Fd未打开。
HI_ERR_VO_DEV_NOT_ENABLE	视频输出设备未使能。
HI_ERR_VO_DEV_NOT_CONFIG	设备未配置。
HI_ERR_VO_NO_MEM	没有可用的内存。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

**【注意】**

- 视频层使能前必须保证该视频层所在的设备处于使能状态。
- 视频层使能前必须保证该视频层已经配置。

【举例】

```
HI_S32 s32Ret;
VO_DEV VoDev = 0;
RECT_S stRect;
VO_VIDEO_LAYER_ATTR_S stLayerAttr;

s32Ret = HI_MPI_VO_GetVideoLayerAttr(VoDev, &stLayerAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Get video layer attributes failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}

stLayerAttr.stDispRect.s32X      = 0;
stLayerAttr.stDispRect.s32Y      = 0;
stLayerAttr.stDispRect.u32Width  = 720;
stLayerAttr.stDispRect.u32Height = 576;

stLayerAttr.stImageSize.u32Width  = 720;
stLayerAttr.stImageSize.u32Height = 576;

stLayerAttr.s32PiPChn = VO_DEFAULT_CHN;

s32Ret = HI_MPI_VO_SetVideoLayerAttr(VoDev, &stLayerAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Set video layer attributes failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VO_EnableVideoLayer(VoDev);
if (s32Ret != HI_SUCCESS)
{
    printf("Enable video layer failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VO_GetDisplayRect(VoDev, &stRect);
if (s32Ret != HI_SUCCESS)
```




```
{
    printf("Get disp rect failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}

stRect.s32X = 128;
stRect.s32Y = 128;

s32Ret = HI_MPI_VO_SetDisplayRect(VoDev, &stRect);
if (s32Ret != HI_SUCCESS)
{
    printf("Set disp rect failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VO_DisableVideoLayer(VoDev);
if (s32Ret != HI_SUCCESS)
{
    printf("Disable video layer failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}
```

【相关主题】

[HI_MPI_VO_DisableVideoLayer](#)

HI_MPI_VO_DisableVideoLayer

【描述】

禁止视频层。

【语法】

```
HI_S32 HI_MPI_VO_DisableVideoLayer(VO_DEV VoDev);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入

【返回值】

返回值	描述
0	成功。



返回值	描述
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备 Fd 未打开。
HI_ERR_VO_CHN_NOT_DISABLE	该视频层上还有未禁止的通道。
HI_ERR_VO_GRP_NOT_DESTROY	该视频层上还有同步组未销毁。
HI_ERR_VB_BUSY	VB 资源没有释放。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 视频层禁止前必须保证其上的通道全部禁止。
- 视频层禁止前必须保证其上的同步组全部销毁。
- 在禁止视频层时，如果用户没有释放从 VO 获取的图像 buffer 资源，该接口会返回 HI_ERR_VB_BUSY 的错误码，表示 VO 创建的 VB 资源没有释放。多见于用户调用获取屏幕图像未释放的情况下。

【举例】

请参见 [HI_MPI_VO_EnableVideoLayer](#) 的举例。

【相关主题】

[HI_MPI_VO_EnableVideoLayer](#)

HI_MPI_VO_SetVideoLayerAttr

【描述】

设置视频层属性。

【语法】

```
HI_S32 HI_MPI_VO_SetVideoLayerAttr(VO_DEV VoDev, const
VO_VIDEO_LAYER_ATTR_S *pstLayerAttr);
```



【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstLayerAttr	视频层属性结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备 Fd 未打开。
HI_ERR_VO_ILLEGAL_PARAM	无效参数。
HI_ERR_VO_VIDEO_NOT_DISABLE	视频层未禁止。
HI_ERR_VO_NULL_PTR	参数指针为空。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

设置视频层属性必须在视频层禁止的情况下进行。

【举例】

请参见 [HI_MPI_VO_EnableVideoLayer](#) 的举例。

【相关主题】

[HI_MPI_VO_GetVideoLayerAttr](#)



HI_MPI_VO_GetVideoLayerAttr

【描述】

获取视频层属性。

【语法】

```
HI_S32 HI_MPI_VO_GetVideoLayerAttr(VO_DEV VoDev, VO_VIDEO_LAYER_ATTR_S
*pstLayerAttr);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstLayerAttr	视频层属性结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备 Fd 未打开。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_NULL_PTR	参数指针为空。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 获取视频层属性不要求视频层属性已设置或者视频层使能。
- 建议在设置视频层属性前先获取。



【举例】

请参见 [HI_MPI_VO_EnableVideoLayer](#) 的举例。

【相关主题】

[HI_MPI_VO_SetVideoLayerAttr](#)

HI_MPI_VO_SetDisplayRect

【描述】

设置视频输出的显示区域。

【语法】

```
HI_S32 HI_MPI_VO_SetDisplayRect(VO_DEV VoDev, RECT_S *pstRect);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstRect	视频层显示区域结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备Fd未打开。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_NULL_PTR	参数指针为空。
HI_ERR_VO_ILLEGAL_PARAM	无效参数。

【需求】



- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 该接口设置的显示区域和视频层属性中的显示区域是同一概念。
- 调用此接口前必须保证视频层已经配置。
- 此接口为动态属性，在视频层属性配置时会设置显示区域的初始值。如果需要动态改变显示区域，应该使用这个接口。
- 显示区域如果超出设备分辨率就进行剪裁。

【举例】

请参见 [HI_MPI_VO_EnableVideoLayer](#) 的举例。

【相关主题】

[HI_MPI_VO_GetDisplayRect](#)

HI_MPI_VO_GetDisplayRect

【描述】

获取视频输出的显示区域。

【语法】

```
HI_S32 HI_MPI_VO_GetDisplayRect(VO_DEV VoDev, RECT_S *pstRect);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstRect	视频层属性结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备Fd未打开。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_NULL_PTR	参数指针为空。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

调用此接口前必须保证视频层已经配置。

【举例】

请参见[HI_MPI_VO_EnableVideoLayer](#)的举例。

【相关主题】

[HI_MPI_VO_SetDisplayRect](#)

HI_MPI_VO_SetValidImgRect

【描述】

设置图像输出有效区域。

【语法】

```
HI_S32 HI_MPI_VO_SetValidImgRect(VO\_DEV VoDev, const RECT_S *pstRect);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstRect	图像有效区域矩形指针。	输入

【返回值】

返回值	描述
0	成功。



返回值	描述
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_ILLEGAL_PARAM	无效参数。
HI_ERR_VO_VIDEO_NOT_CONFIG	视频层未配置。
HI_ERR_VO_DEV_NOT_ENABLE	设备未使能。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备 Fd 未打开。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_NULL_PTR	参数指针为空。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用此接口前必须保证视频层已经配置。
- 有效区域的起始点和宽高必须是偶数，并且宽高不小于 16 像素。
- 有效区域不能超出视频层中设置的 stImageSize。
- 该接口属于动态属性，一般应用于多画面切换时想改变画面布局的情况。例如在 HD 设备上，从一画面切换到四画面时，为了防止图像在垂直方向上进行缩放后再 DIE 引起图像质量下降的现象，将有效区域扩大。

【举例】

无。

【相关主题】

[HI_MPI_VO_GetValidImgRect](#)

HI_MPI_VO_GetValidImgRect

【描述】

获取图像输出有效区域。

【语法】



```
HI_S32 HI_MPI_VO_GetValidImgRect(VO_DEV VoDev, RECT_S *pstRect);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstRect	图像有效区域矩形指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_VIDEO_NOT_CONFIG	视频层未配置。
HI_ERR_VO_DEV_NOT_ENABLE	设备未使能。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备 Fd 未打开。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_NULL_PTR	参数指针为空。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

调用此接口前必须保证视频层已经配置。

【举例】

无。

【相关主题】

[HI_MPI_VO_SetValidImgRect](#)



HI_MPI_VO_SetScreenFilter

【描述】

设置指定视频层的滤波系数。

视频输出设备(HD/AD)提供视频缩放功能，将通过 TDE 缩放、拼接后的图像缩放后按照指定时序输出。默认情况下，SDK 会按照算法选择最佳滤波系数，对拼接后的图像进行缩放，但用户也可根据需要使用该接口选择滤波系数，调整图像质量。

【语法】

```
HI_S32 HI_MPI_VO_SetScreenFilter(VO_DEV VoDev, const VO_SCREEN_FILTER_S
*pstFilter)
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstFilter	VO 缩放滤波系数属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	设备号超出正常范围。
HI_ERR_VO_ILLEGAL_PARAM	参数错误。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h



- 库文件：libmpi.a

【注意】

- 只有 HD 和 AD 支持 VO 缩放功能，如果在 SD 上调用该接口，会返回参数错的错误码。
- 一般用户不需要设置该接口，SDK 自动选择默认滤波系数。
- 所提供的滤波系数中，值越高图像越清晰。

【举例】

无。

【相关主题】

[HI_MPI_VO_GetScreenFilter](#)

HI_MPI_VO_GetScreenFilter

【描述】

获取指定视频层的滤波系数。

【语法】

```
HI_S32 HI_MPI_VO_GetScreenFilter(VO_DEV VoDev, VO_SCREEN_FILTER_S
*pstFilter);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstFilter	视频通道滤波系数属性。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_VO_INVALID_DEVID	设备号超出正常范围。
HI_ERR_VO_ILLEGAL_PARAM	参数错误。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

只有 HD 和 AD 支持 VO 缩放功能，如果在 SD 上调用该接口，会返回参数错的错误码。

【举例】

无。

【相关主题】

[HI_MPI_VO_SetScreenFilter](#)

HI_MPI_VO_EnableChn

【描述】

启用指定的视频输出通道。

【语法】

```
HI_S32 HI_MPI_VO_EnableChn(VO\_DEV VoDev, VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)	输入
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	视频输出通道号无效。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_VIDEO_NOT_ENABLE	视频层未使能。
HI_ERR_VO_CHN_NOT_ALLOC	通道资源未分配。
HI_ERR_VO_CHN_NOT_CONFIG	通道未配置。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，否则将返回失败。
- 调用前必须使能相应设备上的视频层。
- 通道使能前必须进行通道配置，否则返回通道未配置的错误。
- 允许重复使能同一视频输出通道，不返回失败。

【举例】

```
VO_DEV VoDev = 0;
VO_CHN VoChn = 0;
VO_CHN_ATTR_S stChnAttr;

s32Ret = HI_MPI_VO_GetChnAttr(VoDev, VoChn, &stChnAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Get channel attr failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}

stChnAttr.u32Priority = 0;
```



```
stChnAttr.stRect.s32X = 0;
stChnAttr.stRect.s32Y = 0;
stChnAttr.stRect.u32Width = 720;
stChnAttr.stRect.u32Height = 576;
stChnAttr.bZoomEnable = HI_FALSE;
stChnAttr.bDeflicker = HI_FALSE;

s32Ret = HI_MPI_VO_SetChnAttr(VoDev, VoChn, &stChnAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Set channel attr failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VO_EnableChn(VoDev, VoChn);
if (s32Ret != HI_SUCCESS)
{
    printf("Enable channel failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VO_DisableChn(VoDev, VoChn);
if (s32Ret != HI_SUCCESS)
{
    printf("Disable channel failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}
```

【相关主题】

[HI_MPI_VO_DisableChn](#)

HI_MPI_VO_DisableChn

【描述】

禁用指定的视频输出通道。

【语法】

```
HI_S32 HI_MPI_VO_DisableChn(VO_DEV VoDev, VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入



参数名称	描述	输入/输出
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_VIDEO_NOT_ENABLE	视频层未使能。
HI_ERR_VO_CHN_NOT_ALLOC	通道资源未配置。
HI_ERR_VO_GRP_CHN_NOT_UNREG	同步组通道未注销

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

允许重复禁用同一视频输出通道，不返回失败。

【举例】

请参见 [HI_MPI_VO_EnableChn](#) 的举例。

【相关主题】

[HI_MPI_VO_EnableChn](#)

HI_MPI_VO_SetChnAttr

【描述】

配置指定视频输出通道的属性。



【语法】

```
HI_S32 HI_MPI_VO_SetChnAttr(VO_DEV VoDev, VO_CHN VoChn, VO_CHN_ATTR_S
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstAttr	视频通道属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_ILLEGAL_PARAM	参数超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 属性中的优先级，数值越大优先级越高，最大值为 31，最小值为 0。
 - 当多个通道有重叠的显示区域时，优先级高的通道图像将覆盖优先级低的通道。



- 优先级相同的各通道有重叠时，默认通道号大的图像将覆盖通道号小的通道图像。
- 通道显示区域不能超过视频层属性中设定的显示分辨率范围(stImageSize 大小)。
- 如果缩放标识为 TRUE，则通道的输入图像将被缩放到通道大小；否则对输入图像进行剪裁以适合通道大小。
- 该接口为动态设置接口，可在 VO 设备使能的情况下调用。
- 通道的起始位置和宽高必须能被 2 整除。
- 选项 bDeflicker 表示该通道是否进行抗闪烁。一般情况下，逐行设备上不需要进行抗闪烁；在隔行设备上，如果该通道通过单场图像缩小得到，则需要抗闪烁处理。需要注意的是，抗闪烁可能带来图像清晰度的下降。

【举例】

请参见 [HI_MPI_VO_EnableChn](#) 的举例。

【相关主题】

[HI_MPI_VO_GetChnAttr](#)

HI_MPI_VO_GetChnAttr

【描述】

获取指定视频输出通道的属性。

【语法】

```
HI_S32 HI_MPI_VO_GetChnAttr(VO_DEV VoDev, VO_CHN VoChn,  
                             VO_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstAttr	视频通道属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。



【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

无。

【举例】

请参见 [HI_MPI_VO_EnableChn](#) 的举例。

【相关主题】

[HI_MPI_VO_SetChnAttr](#)

HI_MPI_VO_SendFrame

【描述】

将视频图像送入指定输出通道显示。

【语法】

```
HI_S32 HI_MPI_VO_SendFrame(VO\_DEV VoDev, VO_CHN VoChn,  
                             VIDEO_FRAME_INFO_S *pstVFrame);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstVFrame	视频数据信息指针。	输入



【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_CHN_NOT_ENABLE	通道未使能。
HI_ERR_VO_ILLEGAL_PARAM	传入的参数非法。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数或输出 buffer 已满。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

调用该接口前必须保证通道已经使能。

【举例】

无。

【相关主题】

无。

HI_MPI_VO_SetChnField

【描述】

设置指定视频输出通道的帧场显示策略。

主要的应用场合有两个：一是在预览 CIF 图像时，将视频输入设置为两场 CIF，对应 VO 通道设置成显示两场，这样比只显示一场的流畅性要好；二是在低帧率 2 场图像预览的情况下，如在低帧率的 D1 预览时，应该将对应 VO 通道设置成只显示一场，否则会出现画面上下抖动的现象。



【语法】

```
HI_S32 HI_MPI_VO_SetChnField(VO_DEV VoDev, VO_CHN VoChn, const
VO_DISPLAY_FIELD_E enField)
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
enField	视频通道显示帧场策略。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_ILLEGAL_PARAM	参数非法。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 如果不显式调用此接口，VO 默认为两场显示（即 VO_FIELD_BOTH）。
- 如果送 VO 显示的图像为帧图像（例如单场 CIF 图像），则调用此接口对图像显示无影响。



【举例】

```
/* 这是一个预览低帧率的例子，先将视频输入设置成8帧每秒，然后对应VO通道用底场显示 */
VO_DISPLAY_FIELD_E enField;
if (HI_SUCCESS!=HI_MPI_VI_SetFrameRate(0,0,8))
{
    printf("HI_MPI_VI_SetFrameRate failed !\n");
    return HI_FAILURE;
}
if (HI_SUCCESS!=HI_MPI_VO_SetChnField(0,VO_FIELD_BOTTOM))
{
    printf("HI_MPI_VO_SetChnField failed !\n");
    return HI_FAILURE;
}
if (HI_SUCCESS!=HI_MPI_VO_GetChnField(0, &enField))
{
    printf("HI_MPI_VO_GetChnField failed !\n");
    return HI_FAILURE;
}
```

【相关主题】

无。

HI_MPI_VO_GetChnField

【描述】

获取指定视频输出通道的帧场显示策略。

【语法】

```
HI_S32 HI_MPI_VO_GetChnField(VO_DEV VoDev,VO_CHN VoChn,const
VO_DISPLAY_FIELD_E *penField)
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
penField	视频通道显示帧场策略。	输出

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

如未配置则返回系统默认策略 VO_FIELD_BOTH。

【举例】

请参见 HI_MPI_VO_SetChnField 的举例。

【相关主题】

无。

HI_MPI_VO_SetChnFrameRate

【描述】

设置指定视频输出通道的显示帧率。

【语法】

```
HI_S32 HI_MPI_VO_SetChnFrameRate (VO_DEV VoDev, VO_CHN VoChn, HI_S32
s32VoFramerate);
```

【参数】



参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
s32VoFramerate	视频通道显示帧率。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_ILLEGAL_PARAM	参数非法。
HI_ERR_VO_CHN_NOT_CONFIG	通道未配置。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 需先设置 VO 的公共属性，才能调用此接口，否则返回失败。
- 帧率可设置为 Nx（其中 N 为[-64, +64]的任意整数，x 为 P 制或 N 制下的帧率）。负数的倍数用于倒放操作，此时需要用户来倒序送图像到 VO，即送时间戳递减的图像。

【举例】

无。

**【相关主题】**

无。

HI_MPI_VO_GetChnFrameRate**【描述】**

获取指定视频输出通道的显示帧率。

【语法】

```
HI_S32 HI_MPI_VO_GetChnFrameRate(VO_DEV VoDev, VO_CHN VoChn, HI_S32  
*ps32VoFramerate);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
ps32VoFramerate	视频通道显示帧率。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】



- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

获取的为配置用户帧率，如未设置则返回帧率。。

【举例】

无。

【相关主题】

无。

HI_MPI_VO_ChnPause

【描述】

暂停指定的视频输出通道。

【语法】

```
HI_S32 HI_MPI_VO_ChnPause(VO_DEV VoDev, VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)	输入
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	视频输出通道号无效。
HI_ERR_VO_CHN_NOT_ENABLE	通道未使能。

**【需求】**

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，否则将返回失败。
- 调用前必须使能视频输出通道，否则将返回通道未使能错误码 [HI_ERR_VO_CHN_NOT_ENABLE](#)。
- VI-VO 预览时不推荐使用此接口。
- 允许重复暂停同一通道，不返回失败。
- 如果该通道在同步组中注册，则调用该接口无效。

【举例】

```
HI_S32 s32ret;
VO_DEV VoDev = 0;
VO_CHN VoChn = 0;

...

/* enable vo chn */
s32ret = HI_MPI_VO_EnableChn(VoDev, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("enable vo chn failed! \n");
    return s32ret;
}

/* pause current vo channel */
s32ret = HI_MPI_VO_ChnPause(VoDev, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("pause vo chn failed! \n");
    return s32ret;
}

/* resume current vo channel */
s32ret = HI_MPI_VO_ChnResume(VoDev, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("resume vo chn failed! \n");
    return s32ret;
}
```



```
while (getchar() != 'q')
{
    /* step forward current vo channel */
    s32ret = HI_MPI_VO_ChnStep(VoDev, VoChn);
    if (HI_SUCCESS != s32ret)
    {
        printf("step play vo chn failed! \n");
        return s32ret;
    }
}

/* resume current vo channel */
s32ret = HI_MPI_VO_ChnResume(VoDev, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("resume vo chn failed! \n");
    return s32ret;
}

(void)HI_MPI_VO_DisableChn(VoDev, VoChn);
```

【相关主题】

[HI_MPI_VO_ChnResume](#)

HI_MPI_VO_ChnResume

【描述】

恢复指定的视频输出通道。

【语法】

```
HI_S32 HI_MPI_VO_ChnResume(VO\_DEV VoDev, VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)	输入
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	视频输出通道号无效。
HI_ERR_VO_CHN_NOT_ENABLE	通道未使能。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，否则将返回失败。
- 如果未启用通道将无法正常工作，所以通道使用前必须启用成功。
- 允许重复恢复同一通道，不返回失败。
- 如果该通道在同步组中注册，则调用该接口无效。

【举例】

请参见 [HI_MPI_VO_ChnPause](#) 的举例。

【相关主题】

[HI_MPI_VO_ChnResume](#)

HI_MPI_VO_ChnStep

【描述】

单帧播放指定的视频输出通道。

【语法】

```
HI_S32 HI_MPI_VO_ChnStep(VO\_DEV VoDev, VO_CHN VoChn);
```

【参数】



参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)	输入
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	视频输出通道号无效。
HI_ERR_VO_CHN_NOT_ENABLE	通道未使能。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，否则将返回失败。
- 如果未启用通道将无法正常工作，所以通道使用前必须启用成功。
- 恢复正常播放使用 [HI_MPI_VO_ChnResume](#) 接口。
- VI-VO 预览时不推荐使用此接口。
- 如果该通道在同步组中注册，则调用该接口无效。

【举例】

请参见 [HI_MPI_VO_ChnPause](#) 的举例。

【相关主题】

[HI_MPI_VO_ChnResume](#)



HI_MPI_VO_SetChnFilter

【描述】

设置指定视频输出通道的缩放系数。

当 VO 视频层属性 s32PiPChn 设置为 VO_DEFAULT_CHN，且通道输入图像大小和其显示大小不一致时，利用 DSU 缩放单元对图像进行缩放。系统内部提供多套滤波参数，每套滤波参数又包括水平和垂直的多组滤波系数。系统内部默认选择通用类型（FILTER_PARAM_TYPE_NORM）滤波参数，并根据算法自动选定其下的一组最优的滤波系数进行缩放滤波。若用户希望对缩放质量进行微调或有其它特殊需求可通过此接口设置相应的缩放类型和系数以便调节图像质量。

【语法】

```
HI_S32 HI_MPI_VO_SetChnFilter(VO_DEV VoDev, VO_CHN VoChn, const  
VO_CHN_FILTER_S *pstFilter);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstFilter	视频通道滤波系数属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_NULL_PTR	空指针。
HI_ERR_VO_ILLEGAL_PARAM	参数非法。
HI_ERR_VO_CHN_NOT_CONFIG	通道未配置。



接口返回值	含义
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 一般用户不需要设置此接口，系统会自动选择通用的最优滤波系数。
- 当滤波系数结构 pstFilter 中的相应滤波系数配置为 DSU_XSCALE_FILTER_DEFAULT 时，即为系统自动选择滤波系数。
- 滤波系数的值越大，图像越清晰但闪烁感越明显；值越低，图像越模糊。
- 该接口中 enFilterType 仅支持设置为 FILTER_PARAM_TYPE_NORM、FILTER_PARAM_TYPE_USER1 或 FILTER_PARAM_TYPE_USER2，否则返回参数错误。
当设置 enFilterType 为 FILTER_PARAM_TYPE_NORM 时，可通过调节 enHFilter/ enVFilterL/ enVFilterC 选择不同截止频率的滤波参数达到不同的滤波效果。
- 当用户对缩放效果有特定需求时，需先通过接口 HI_MPI_VPP_SetDsuFiltParam 设置用户自定义的滤波参数，然后在此指定 enFilterType 为相应的 FILTER_PARAM_TYPE_USERX，即可在 DSU 缩放时启动用户自定义滤波参数了。设置用户自定义滤波参数请参见接口 HI_MPI_VPP_SetDsuFiltParam 说明。

【举例】

无。

【相关主题】

无。

HI_MPI_VO_GetChnFilter

【描述】

获取指定视频输出通道的缩放系数。

【语法】

```
HI_S32 HI_MPI_VO_GetChnFilter(VO_DEV VoDev, VO_CHN VoChn, VO_CHN_FILTER_S  
*pstFilter);
```

【参数】



参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstFilter	视频通道滤波系数属性。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。



HI_MPI_VO_CreateSyncGroup

【描述】

创建视频输出同步组。

【语法】

```
HI_S32 HI_MPI_VO_CreateSyncGroup(VO_GRP VoGroup, VO_DEV VoDev);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_ID	同步通道组组号无效。
HI_ERR_VO_INVALID_DEVID	输出设备号无效。
HI_ERR_VO_SYS_NOTREADY	系统未初始化。
HI_ERR_VO_GRP_HAS_CREATED	指定的同步组已经创建。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并对视频输出进行属性配置。



- 指定的同步组必须保证未被创建，否则返回已经创建错误码。
- 创建同步组时通过指定输出设备号，表明同步组内的通道在该设备上显示。但同步组内的通道可以是来自不同设备的通道。
- 最大同时支持 16 个通道组，每组内最多支持注册 32 路通道。

【举例】

请参见 [HI_MPI_VO_SyncGroupStart](#) 的举例。

【相关主题】

[HI_MPI_VO_DestroySyncGroup](#)

HI_MPI_VO_DestroySyncGroup

【描述】

销毁视频输出同步组。

【语法】

```
HI_S32 HI_MPI_VO_DestroySyncGroup(VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_ID	同步组 ID 无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_CHN_NOT_EMPTY	同步组通道组不为空。
HI_ERR_VO_GRP_NOT_STOP	同步组未停止。



【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须保证同步通道已经创建。
- 调用前必须保证同步组已停止。
- 如果同步组中的视频输出通道没有全部注销，调用会返回同步组非空的错误码。

【举例】

请参见 [HI_MPI_VO_SyncGroupStart](#) 的举例。

【相关主题】

[HI_MPI_VO_CreateSyncGroup](#)

HI_MPI_VO_RegisterSyncGroup

【描述】

注册视频输出通道到视频输出同步组。

【语法】

```
HI_S32 HI_MPI_VO_RegisterSyncGroup(VO_DEV VoDev, VO_CHN VoChn,  
                                     VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)	输入
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。



【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_FAILED_NOTENABLE	没有使能视频输出设备。
HI_ERR_VO_INVALID_CHNID	视频输出通道号无效。
HI_ERR_VO_GRP_INVALID_ID	同步组 ID 无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_CHN_FULL	同步组注册数已满。
HI_ERR_VO_GRP_CHN_HAS_REG	通道已经注册。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 调用前必须保证同步组已经创建，并且非满。
- 调用前必须保证通道未在其它同步组中注册。
- 重复注册同一通道到一个同步组中，返回成功。

【举例】

请参见 [HI_MPI_VO_SyncGroupStart](#) 的举例。

【相关主题】

[HI_MPI_VO_UnRegisterSyncGroup](#)

HI_MPI_VO_UnRegisterSyncGroup

【描述】

注销视频输出通道从视频输出同步组。

【语法】

```
HI_S32 HI_MPI_VO_UnRegisterSyncGroup(VO\_DEV VoDev, VO_CHN VoChn,  
                                       VO\_GRP VoGroup);
```

【参数】



参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)	输入
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	视频输出通道号无效。
HI_ERR_VO_CHN_NOT_ENABLE	视频输出通道未使能。
HI_ERR_VO_GRP_INVALID_ID	同步组 ID 无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_CHN_EMPTY	同步通道组中没有视频输出通道。
HI_ERR_VO_GRP_CHN_NOT_REG	通道没有注册。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 不能从同步组里注销一个不在同步组内的通道。

【举例】



请参见 [HI_MPI_VO_SyncGroupStart](#) 的举例。

【相关主题】

[HI_MPI_VO_RegisterSyncGroup](#)

HI_MPI_VO_SyncGroupStart

【描述】

启动视频输出同步组。

【语法】

```
HI_S32 HI_MPI_VO_SyncGroupStart(VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_CHN_NOT_ENABLE	通道未使能。
HI_ERR_VO_GRP_INVALID_ID	同步组 ID 无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_INVALID_BASE_PTS	同步通道组基准 PTS 无效。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a



【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证同步通道组已经创建，并且有视频输出通道注册。
- 重复启动返回成功。

【举例】

```
HI_S32 s32ret;
HI_U32 u32FrameRate = 12;
VO_DEV VoDev = 0;
VO_CHN VoChn = 0;
VO_GRP VoGroup = 0;
HI_U64 u64BasePts = 0;

/* create synchronous group */

u64BasePts = 64094025454LLU;
s32ret = HI_MPI_VO_CreateSyncGroup(VoGroup, VoDev);
if (HI_SUCCESS != s32ret)
{
    printf("vo create synchronous group failed! \n");
    return s32ret;
}

/* register a vo channel to synchronous group */
s32ret = HI_MPI_VO_RegisterSyncGroup(VoDev, VoChn, VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("vo channel register synchronous group failed! \n");
    return s32ret;
}

/* start synchronous group */
s32ret = HI_MPI_VO_SyncGroupStart(VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("start synchronous group failed! \n");
    return s32ret;
}

s32ret = HI_MPI_VO_SetSyncGroupBase (VoGroup, &stSyncBase);
if (HI_SUCCESS != s32ret)
{
    printf("set synchronous group base pts failed! \n");
```



```
        return s32ret;
    }

    s32ret = HI_MPI_VO_GetSyncGroupBase (VoGroup, &stSyncBase);
    if (HI_SUCCESS != s32ret)
    {
        printf("get synchronous group base pts failed! \n");
        return s32ret;
    }
    else
    {
        printf("current base pts = %llu\n", stSyncBase. u64BasePts);
    }
    /* set frame rate of synchronous group */
    s32ret = HI_MPI_VO_SetSyncGroupFrameRate(VoGroup, u32FrameRate);
    if (HI_SUCCESS != s32ret)
    {
        printf("set synchronous group frame rate failed! \n");
        return s32ret;
    }

    /* pause group */
    s32ret = HI_MPI_VO_PauseSyncGroup(VoGroup);
    if (HI_SUCCESS != s32ret)
    {
        printf("pause synchronous group failed! \n");
        return s32ret;
    }

    /* step group */
    s32ret = HI_MPI_VO_StepSyncGroup(VoGroup);
    if (HI_SUCCESS != s32ret)
    {
        printf("step synchronous group failed! \n");
        return s32ret;
    }

    /* resume group */
    s32ret = HI_MPI_VO_ResumeSyncGroup(VoGroup);
    if (HI_SUCCESS != s32ret)
    {
        printf("resume synchronous group failed! \n");
        return s32ret;
    }
}
```




```
/* stop synchronous group */
s32ret = HI_MPI_VO_SyncGroupStop(VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("stop synchronous group failed! \n");
    return s32ret;
}

/* un-register vo channel from group */
s32ret = HI_MPI_VO_UnRegisterSyncGroup(VoDev, VoChn, VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("un-register vo from synchronous group failed! \n");
    return s32ret;
}

/* destroy synchronous group */
s32ret = HI_MPI_VO_DestroySyncGroup(VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("destroy synchronous group failed! \n");
    return s32ret;
}
```

【相关主题】

[HI_MPI_VO_SyncGroupStop](#)

HI_MPI_VO_SyncGroupStop

【描述】

停止视频输出通道组。

【语法】

```
HI_S32 HI_MPI_VO_SyncGroupStop(VO\_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入

**【返回值】**

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_ID	同步组 ID 无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_NOT_START	同步通道组没有启动。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 调用前必须保证同步通道组已经创建，并且启动。

【举例】

请参见 [HI_MPI_VO_SyncGroupStart](#) 的举例。

【相关主题】

[HI_MPI_VO_SyncGroupStart](#)

HI_MPI_VO_SetSyncGroupFrameRate

【描述】

设置视频输出同步通道组帧率。

【语法】

```
HI_S32 HI_MPI_VO_SetSyncGroupFrameRate(VO\_GRP VoGroup, HI_U32  
u32VoGrpFramerate);
```

【参数】



参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入
u32VoGrpFramerate	同步输出组预设帧率。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_ID	同步组 ID 无效。
HI_ERR_VO_ILLEGAL_PARAM	参数无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证同步通道组已经创建。
- 目前同步组帧率上限为满帧率的 64 倍。

【举例】

请参见 [HI_MPI_VO_SyncGroupStart](#) 的举例。

【相关主题】

[HI_MPI_VO_GetSyncGroupFrameRate](#)

HI_MPI_VO_GetSyncGroupFrameRate

【描述】



获取视频输出通道组帧率。

【语法】

```
HI_S32 HI_MPI_VO_GetSyncGroupFrameRate(VO\_GRP VoGroup, HI_U32  
*pu32VoGrpFramerate);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入
pu32VoGrpFramerate	同步输出组帧率指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_ID	同步组 ID 无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_NULL_PTR	空指针。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证同步通道组已经创建。

【举例】

请参见 [HI_MPI_VO_SyncGroupStart](#) 的举例。



【相关主题】

[HI_MPI_VO_SetSyncGroupFrameRate](#)

HI_MPI_VO_PauseSyncGroup

【描述】

暂停同步输出通道组。

【语法】

```
HI_S32 HI_MPI_VO_PauseSyncGroup(VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_ID	同步组 ID 无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_NOT_START	同步通道组没有启动。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。



- 调用前必须保证同步通道组已经创建并启动。

【举例】

请参见 [HI_MPI_VO_SyncGroupStart](#) 的举例。

【相关主题】

[HI_MPI_VO_ResumeSyncGroup](#)

HI_MPI_VO_ResumeSyncGroup

【描述】

恢复同步输出通道组。

【语法】

```
HI_S32 HI_MPI_VO_ResumeSyncGroup(VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_ID	同步组 ID 无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_NOT_START	同步通道组没有启动。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a



【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 调用前必须保证同步通道组已经创建并启动。

【举例】

请参见 [HI_MPI_VO_SyncGroupStart](#) 的举例。

【相关主题】

[HI_MPI_VO_PauseSyncGroup](#)

HI_MPI_VO_StepSyncGroup

【描述】

单帧播放同步输出通道组。

【语法】

```
HI_S32 HI_MPI_VO_StepSyncGroup(VO\_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_ID	同步组 ID 无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_NOT_START	同步通道组没有启动。

**【需求】**

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 调用前必须保证同步通道组已经创建并启动。

【举例】

请参见 [HI_MPI_VO_SyncGroupStart](#) 的举例。

【相关主题】

[HI_MPI_VO_PauseSyncGroup](#)

HI_MPI_VO_SetSyncGroupBase

【描述】

设置多通道同步组的基准时间戳。

【语法】

```
HI_S32 HI_MPI_VO_SetSyncGroupBase(VO_GRP VoGroup, HI_U64 u64BasePts);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入
u64BasePts	同步基准 PTS。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_VO_GRP_INVALID_ID	同步组 ID 无效。
HI_ERR_VO_INVALID_CHNID	无效的 VO 通道号。
HI_ERR_VO_GRP_NOT_CREATE	同步组未创建。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，否则将返回失败。
- 调用前必须使能视频输出通道，否则返回失败。
- 调用前必须创建同步组，否则返回失败。
- 启动同步组前或后均可设置时间基准。
- 可以多次设置基准时间戳。

【举例】

请参见 [HI_MPI_VO_SyncGroupStart](#) 的举例。

【相关主题】

[HI_MPI_VO_GetSyncGroupBase](#)

HI_MPI_VO_GetSyncGroupBase

【描述】

获取多通道同步组的基准时间戳。

【语法】

```
HI_S32 HI_MPI_VO_GetSyncGroupBase(VO\_GRP VoGroup, HI_U64 *pu64BasePts);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GRP)。	输入
pu64BasePts	同步基准 PTS。	输出

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_ID	同步通道组组号无效。
HI_ERR_VO_INVALID_CHNID	无效的 VO 通道号。
HI_ERR_VO_GRP_NOT_CREATE	同步组未创建。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，否则将返回失败。
- 调用前必须使能视频输出通道，否则返回失败。
- 调用前必须创建同步组，否则返回失败。
- 启动同步组前或后均可获取时间基准。
- 可以多次获取基准时间戳。

【举例】

请参见 [HI_MPI_VO_SyncGroupStart](#) 的举例。

【相关主题】

[HI_MPI_VO_SetSyncGroupBase](#)

HI_MPI_VO_SetZoomInWindow

【描述】

设置视频输出局部放大窗口。

【语法】

```
HI_S32 HI_MPI_VO_SetZoomInWindow(VO\_DEV VoDev, VO\_CHN VoChn,  
                                   const VO\_ZOOM\_ATTR\_S *pstZoomAttr);
```

【参数】



参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstZoomAttr	局部放大属性结构体。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道号无效。
HI_ERR_VO_INVALID_RECT_PARA	矩形结构体参数无效。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 局部放大窗口参数为非负数，并且 2 像素对齐；如果视频层属性 s32PiPChn 模式不是默认模式，则起始点横坐标需要 16 像素对齐。
- 局部放大窗口的起始点和宽高都是相对于输入图像数据而言，而不是屏幕坐标，可根据需要将屏幕坐标转换到输入图像坐标。
- 取消局部放大时，只要将局部放大属性的窗口矩形设置为 (0, 0, 0, 0)。
- 在视频级联的主片上，利用局部放大功能在级联的图像上选取不同区域进行多画面拼接。

【举例】



```
/* define input parameter */
HI_S32 s32Ret;
VO_DEV VoDev = 0;
VO_CHN VoChn = 0;
VO_ZOOM_ATTR_S stZoomWindow, stZoomWindowGet;

/*
 * we should enable VO and VO channel first!
 * TODO: enable operation.
 */

stZoomWindow.enField = VIDEO_FIELD_FRAME;
stZoomWindow.stZoomRect.s32X = 128;
stZoomWindow.stZoomRect.s32Y = 128;
stZoomWindow.stZoomRect.u32Width = 200;
stZoomWindow.stZoomRect.u32Height = 200;

/* set zoom window */
s32Ret = HI_MPI_VO_SetZoomInWindow(VoDev, VoChn, &stZoomWindow);
if (s32Ret != HI_SUCCESS)
{
    printf("Set zoom attribute failed, ret = %#x.\n", s32Ret);
    return HI_FAILURE;
}

/* get current zoom window parameter */
s32Ret = HI_MPI_VO_GetZoomInWindow(VoDev, VoChn, &stZoomWindowGet);
if (s32Ret != HI_SUCCESS)
{
    printf("Get zoom attribute failed, ret = %#x.\n", s32Ret);
    return HI_FAILURE;
}
else
{
    printf("Current zoom window is (%d,%d,%d,%d)!\n",
        stZoomWindowGet.stZoomRect.s32X,
        stZoomWindowGet.stZoomRect.s32Y,
        stZoomWindowGet.stZoomRect.u32Width,
        stZoomWindowGet.stZoomRect.u32Height);
}

/*
 * TODO: something else ...
 */
```



```
stZoomWindow.stZoomRect.s32X = 0;
stZoomWindow.stZoomRect.s32Y = 0;
stZoomWindow.stZoomRect.u32Width = 0;
stZoomWindow.stZoomRect.u32Height = 0;

/* cancel zoom window, we use (0,0,0,0) to recover */
s32Ret = HI_MPI_VO_SetZoomInWindow(VoDev, VoChn, &stZoomWindow);
if (s32Ret != HI_SUCCESS)
{
    printf("Recover zoom attribute failed, ret = %x.\n", s32Ret);
    return HI_FAILURE;
}
```

【相关主题】

[HI_MPI_VO_GetZoomInWindow](#)

HI_MPI_VO_GetZoomInWindow

【描述】

获取视频输出局部放大窗口。

【语法】

```
HI_S32 HI_MPI_VO_GetZoomInWindow(VO\_DEV VoDev, VO\_CHN VoChn,
                                   VO\_ZOOM\_ATTR\_S *pstZoomAttr);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstZoomAttr	局部放大属性结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。



【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道号无效。
HI_ERR_VO_CHN_NOT_ENABLE	通道未使能。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 如果没有设置局部放大窗口，那么默认获取到 (0, 0, 0, 0)。

【举例】

请参见 [HI_MPI_VO_SetZoomInWindow](#) 的举例。

【相关主题】

[HI_MPI_VO_SetZoomInWindow](#)

HI_MPI_VO_SetZoomInRatio

【描述】

按比例设置视频输出窗口的局部放大。

【语法】

```
HI_S32 HI_MPI_VO_SetZoomInRatio(VO\_DEV VoDev, VO\_CHN VoChn, const
                                VO\_ZOOM\_RATIO\_S *pstZoomRatio);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstZoomRatio	局部放大比例结构体。	输入



【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	设备号无效。
HI_ERR_VO_INVALID_CHNID	通道号无效。
HI_ERR_VO_SYS_NOTREADY	系统未初始化。
HI_ERR_VO_NULL_PTR	空指针。
HI_ERR_VO_VIDEO_NOT_CONFIG	未设置视频层属性。
HI_ERR_VO_CHN_NOT_ALLOC	未设置通道属性。
HI_ERR_VO_INVALID_RECT_PARA	矩形结构体参数无效。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频层已配置。
- 局部放大比例指显示坐标上需缩放区域与该通道显示区域之间的比例。
- 取消局部放大时，只需将局部放大比例结构体中的宽比例或高比例设置为 0。
- 局部放大比例结构体中的设置值均表示千分之多少。
例如，设置值为 425，表示 42.5%。
- 局部放大比例结构体中的设置值取值范围为[0, 1000]。
- 根据比例在通道源图象中截取的缩放区域宽、高、起始坐标均会自动以 2 对齐（向靠近 0 的方向）。当混合输入的时候，起始坐标的横坐标、宽度在大图像上会自从以 4 对齐（向靠近 0 的方向）。
- 根据放大比例结构体中比例在通道源图象中截取的需缩放区域宽、高均需大于 16，否则不缩放。

【相关主题】



HI_MPI_VO_GetZoomInRatio

HI_MPI_VO_GetZoomInRatio

【描述】

获取视频输出局部放大比例。

【语法】

```
HI_S32 HI_MPI_VO_GetZoomInWindow(VO_DEV VoDev, VO_CHN VoChn,  
                                   VO_ZOOM_ATTR_S *pstZoomAttr);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstZoomRatio	局部放大比例结构体。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	设备号无效。
HI_ERR_VO_INVALID_CHNID	通道号无效。
HI_ERR_VO_SYS_NOTREADY	系统未初始化。
HI_ERR_VO_NULL_PTR	空指针。
HI_ERR_VO_VIDEO_NOT_CONFIG	未设置视频层属性。
HI_ERR_VO_CHN_NOT_ALLOC	未设置通道属性。



【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频层已配置。
- 默认局部放大的比例全为 0。

【相关主题】

[HI_MPI_VO_SetZoomInRatio](#)

HI_MPI_VO_GetChnPts

【描述】

获取指定视频输出通道当前显示图像的时间戳。

【语法】

```
HI_S32 HI_MPI_VO_GetChnPts(VO_DEV VoDev, VO_CHN VoChn,  
                             HI_U64 *pu64VoChnPts);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pu64VoChnPts	通道时间戳指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_CHN_NOT_ENABLE	通道未使能。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

调用前需保证视频设备使能，视频通道使能。

【举例】

```
VO_DEV VoDev = 0;
VO_CHN VoChn = 0;
HI_U64 u64ChnPts;

s32Ret = HI_MPI_VO_GetChnPts(VoDev, VoChn, &u64ChnPts);
if (s32Ret != HI_SUCCESS)
{
    printf("Get channel %d pts failed with errno %#x!\n", VoChn, s32Ret);
    return HI_FAILURE;
}

printf("Channel %d pts is %llu.\n", VoChn, u64ChnPts);
```

【相关主题】

无。

HI_MPI_VO_SetAttrBegin

【描述】

设置属性开始。

【语法】

```
HI_S32 HI_MPI_VO_SetAttrBegin(VO_DEV VoDev);
```

【参数】



参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_CHN_NOT_ENABLE	通道未使能。
HI_ERR_VO_SETBEGIN_ALREADY	视频输出设置属性已经开始。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- BEGIN 和 END 接口要配对使用，否则 BEGIN 后设置的通道属性不会生效。

【举例】

```
HI_S32 s32ret;
HI_U32 i;
VO_DEV VoDev = 0;
VO_PUB_ATTR_S VoAttr;
VO_CHN_ATTR_S VoChnAttr[4];

memset(&VoAttr, 0, sizeof(VO_PUB_ATTR_S));
VoAttr.enIntfType = VO_INTF_CVBS; /*CVBS */
VoAttr.enIntfSync = VO_OUTPUT_PAL ; /*PAL Mode*/
VoAttr.u32BgColor = 0x0;
```



```
/* set public attribute of vo device */
s32ret = HI_MPI_VO_SetPubAttr(VoDev, &VoAttr);
if (HI_SUCCESS != s32ret)
{
    printf("vo set pub attr failed! \n");
    return s32ret;
}

/* enable vo device */
s32ret = HI_MPI_VO_Enable(VoDev);
if (HI_SUCCESS != s32ret)
{
    printf("enable vo device failed! \n");
    return s32ret;
}

/* configure and enable vo channel */
for (i = 0; i < 4; i++)
{
    VoChnAttr[i].bZoomEnable = HI_TRUE;
    VoChnAttr[i].u32Priority = 1;
    VoChnAttr[i].stRect.s32X = 360*(i%2);
    VoChnAttr[i].stRect.s32Y = 288*(i/2);
    VoChnAttr[i].stRect.u32Width = 360;
    VoChnAttr[i].stRect.u32Height = 288;

    /* set attribute of vo chn*/
    s32ret = HI_MPI_VO_SetChnAttr(VoDev, i, &VoChnAttr[i]);
    if (HI_SUCCESS != s32ret)
    {
        printf("vo set dev %d chn %d attr failed! \n", VoDev, i);
        return s32ret;
    }

    /* enable vo chn */
    s32ret = HI_MPI_VO_EnableChn(VoDev, i);
    if (HI_SUCCESS != s32ret)
    {
        printf("enable vo dev %d chn %d failed! \n", VoDev, i);
        return s32ret;
    }
}
```



```
/* do somethng else */
sleep(20);

/* set channel attributes begin */
s32ret = HI_MPI_VO_SetAttrBegin(VoDev);
if (HI_SUCCESS != s32ret)
{
    printf("set attributes begin failed!\n");
    return s32ret;
}

/* reset channel attributes
 * we just show channel 0 and 1 on screen, and scale them as vertical
 * half D1
 */
for (i = 0; i < 2; i++)
{
    VoChnAttr[i].bZoomEnable = HI_TRUE;
    VoChnAttr[i].u32Priority = 1;
    VoChnAttr[i].stRect.s32X = 352*(i%2);
    VoChnAttr[i].stRect.s32Y = 288*(i/2);
    VoChnAttr[i].stRect.u32Width = 360;
    VoChnAttr[i].stRect.u32Height = 576;

    /* set attribute of vo chn*/
    s32ret = HI_MPI_VO_SetChnAttr(VoDev, i, &VoChnAttr[i]);
    if (HI_SUCCESS != s32ret)
    {
        printf("vo set chn %d attr failed! \n", i);
        return s32ret;
    }
}

/* hide channel 2 and 3
 * we do this just for saving bandwidth
 */
for (i = 2; i < 4; i++)
{
    s32ret = HI_MPI_VO_ChnHide(VoDev, i);
    if (s32ret != HI_SUCCESS)
    {
        printf("vo hide channel %d failed!\n", i);
        return HI_FAILURE;
    }
}
```



```
    }

    /* set channel attributes end */
    s32ret = HI_MPI_VO_SetAttrEnd(VoDev);
    if (HI_SUCCESS != s32ret)
    {
        printf("set attributes end failed!\n");
        return s32ret;
    }

    /* do somethng else */
    sleep(20);

    /* set channel attributes begin */
    s32ret = HI_MPI_VO_SetAttrBegin(VoDev);
    if (HI_SUCCESS != s32ret)
    {
        printf("set attributes begin failed!\n");
        return s32ret;
    }

    /* reset channel attributes
     * now we set them back as 4 cif on screen
     */
    for (i = 0; i < 4; i++)
    {
        VoChnAttr[i].bZoomEnable = HI_TRUE;
        VoChnAttr[i].u32Priority = 1;
        VoChnAttr[i].stRect.s32X = 360*(i%2);
        VoChnAttr[i].stRect.s32Y = 288*(i/2);
        VoChnAttr[i].stRect.u32Width = 360;
        VoChnAttr[i].stRect.u32Height = 288;

        /* set attribute of vo chn*/
        s32ret = HI_MPI_VO_SetChnAttr(VoDev,i, &VoChnAttr[i]);
        if (HI_SUCCESS != s32ret)
        {
            printf("vo set chn %d attr failed! \n", i);
            return s32ret;
        }
    }

    /* show channel 2 and 3 */
    for (i = 2; i < 4; i++)
```



```
{
    s32ret = HI_MPI_VO_ChnShow(VoDev,i);
    if (s32ret != HI_SUCCESS)
    {
        printf("vo show channel %d failed!\n", i);
        return HI_FAILURE;
    }
}

/* set channel attributes end */
s32ret = HI_MPI_VO_SetAttrEnd(VoDev);
if (HI_SUCCESS != s32ret)
{
    printf("set attributes end failed!\n");
    return s32ret;
}

/* do somethng else */
sleep(20);

/* disable all channels */
for (i = 0; i < 4; i++)
{
    s32ret = HI_MPI_VO_DisableChn(VoDev , i);
    if (HI_SUCCESS != s32ret)
    {
        printf("vo disable chn %d failed!\n", i);
        return s32ret;
    }
}

/* disable vou */
(void)HI_MPI_VO_Disable(VoDev);
```

【相关主题】

[HI_MPI_VO_SetAttrEnd](#)

HI_MPI_VO_SetAttrEnd

【描述】

设置属性结束。

【语法】

```
HI_S32 HI_MPI_VO_SetAttrEnd(VO\_DEV VoDev);
```



【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_SETBEGIN_NOTYET	视频输出设置属性没有开始。
HI_ERR_VO_SETEND_ALREADY	视频输出设置属性未完成。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- BEGIN 和 END 接口要配对使用，否则 BEGIN 后设置的通道属性不会生效。

【举例】

请参见 [HI_MPI_VO_SetAttrBegin](#) 的举例。

【相关主题】

[HI_MPI_VO_SetAttrBegin](#)

HI_MPI_VO_ChnShow

【描述】

显示指定通道。



【语法】

```
HI_S32 HI_MPI_VO_ChnShow(VO\_DEV VoDev, VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_CHN_NOT_ENABLE	视频输出设备未使能。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- 默认情况下通道处于显示状态。

【举例】

请参见 [HI_MPI_VO_SetAttrBegin](#) 的举例。

【相关主题】

[HI_MPI_VO_ChnHide](#)



HI_MPI_VO_ChnHide

【描述】

隐藏指定通道。

【语法】

```
HI_S32 HI_MPI_VO_ChnHide(VO\_DEV VoDev, VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_CHN_NOT_ENABLE	视频输出设备未使能。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。

【举例】



请参见 [HI_MPI_VO_SetAttrBegin](#) 的举例。

【相关主题】

[HI_MPI_VO_ChnShow](#)

HI_MPI_VO_QueryChnStat

【描述】

查询视频输出通道状态。

【语法】

```
HI_S32 HI_MPI_VO_QueryChnStat (VO_DEV VoDev, VO_CHN VoChn,  
                                VO_QUERY_STATUS_S *pstStatus);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstStatus	通道状态结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_CHN_NOT_ENABLE	视频输出通道未使能。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_CHN_NOT_ALLOC	通道资源未配置。

**【需求】**

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- 可以多次调用获取通道状态接口。

【举例】

```
HI_S32 s32Ret;
VO_DEV VoDev = 0;
VO_CHN VoChn = 0;
VO_QUERY_STATUS_S stVoQueryStatus;

/* enable vo device and vo channel */
...

s32Ret = HI_MPI_VO_QueryChnStat(VoDev, VoChn, &stVoQueryStatus);
if (HI_SUCCESS != s32Ret)
{
    printf("Query channel status failed with errorcode %#x!\n", s32Ret);
    return HI_FAILURE;
}
printf("Current vo dev %d channel %d has %d VB occupied!\n",
        VoDev, VoChn, stVoQueryStatus.u32ChnBufUsed );
```

【相关主题】

无。

HI_MPI_VO_SetChnSrcAttr

【描述】

设置通道数据源类型。

【语法】

```
HI_S32 HI_MPI_VO_SetChnSrcAttr(VO_DEV VoDev, VO_CHN VoChn, VO_SRC_ATTR_S
*pstSrcAttr);
```

【参数】



参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstSrcAttr	源属性结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_INVALID_DEVID	视频输出设备号无效。
HI_ERR_VO_CHN_NOT_ALLOC	通道资源未配置。
HI_ERR_VO_CHN_NOT_CONFIG	通道属性未配置。
HI_ERR_VO_NULL_PTR	空指针。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 该接口用于如下特殊场景：当输入图像源是假帧（即实际是隔行数据），需要进行 DIE 或其它处理后才能逐行设备上输出，需要调用此接口将该通道设置为隔行源。
- 调用前需要保证该通道属性已经配置。
- 当需要撤销对源的设置时，将结构体成员 bInterlaced 置为 HI_FALSE。

【举例】



无。

【相关主题】

无。

HI_MPI_VO_GetChnFrame

【描述】

获取输出通道图像数据。

【语法】

```
HI_S32 HI_MPI_VO_GetChnFrame(VO_DEV VoDev, VO_CHN VoChn,
                              VIDEO_FRAME_INFO_S *pstFrame);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstFrame	获取的输出通道图像数据信息结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_CHN_NOT_ENABLE	通道未使能。
HI_ERR_VO_ILLEGAL_PARAM	视频输出参数无效。
HI_ERR_VO_WAIT_TIMEOUT	获取图像数据超时。



【需求】

- 头文件: mpi_vo.h、hi_comm_vo.h
- 库文件: libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证视频层使能。
- 获取后应保证及时的释放。

【举例】

```
...
VIDEO_FRAME_INFO_S *pstFrame =
(VIDEO_FRAME_INFO_S*)malloc(sizeof(VIDEO_FRAME_INFO_S));

if (HI_SUCCESS != HI_MPI_VO_GetScreenFrame(VoDev, pstFrame))
{
    printf("Get screen frame failed!\n");
    return HI_FAILURE;
}

/* do some process, i.e. store file or do JPEG encodeing ... */

if (HI_SUCCESS != HI_MPI_VO_ReleaseScreenFrame(VoDev, pstFrame))
{
    printf("Release screen frame failed!\n");
    return HI_FAILURE;
}

/* enable vo chn */
s32ret = HI_MPI_VO_EnableChn(VoDev, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("enable vo chn failed! \n");
    return s32ret;
}

if (HI_SUCCESS != HI_MPI_VO_GetChnFrame(VoDev, VoChn, pstFrame))
{
    printf("Get channel frame failed!\n");
    return HI_FAILURE;
}

/* do some process, i.e. store file or do JPEG encodeing ... */
```



```
if (HI_SUCCESS != HI_MPI_VO_ReleaseChnFrame(VoDev, VoChn, pstFrame))
{
    printf("Release screen frame failed!\n");
    return HI_FAILURE;
}

(void)HI_MPI_VO_DisableChn(VoDev, VoChn);
```

【相关主题】

[HI_MPI_VO_ReleaseChnFrame](#)

HI_MPI_VO_ReleaseChnFrame

【描述】

释放输出通道图像数据。

【语法】

```
HI_S32 HI_MPI_VO_ReleaseChnFrame(VO_DEV VoDev, VO_CHN
                                   VoChn, VIDEO_FRAME_INFO_S *pstFrame);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstFrame	释放的输出通道图像数据信息结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_CHN_NOT_ENABLE	通道未使能。
HI_ERR_VO_ILLEGAL_PARAM	视频输出参数无效。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 获取操作应保证与释放操作配对。

【举例】

请参见 [HI_MPI_VO_GetChnFrame](#) 的举例。

【相关主题】

[HI_MPI_VO_GetChnFrame](#)

HI_MPI_VO_GetScreenFrame

【描述】

获取输出屏幕图像数据。

【语法】

```
HI_S32 HI_MPI_VO_GetScreenFrame(VO_DEV VoDev,  
                                VIDEO_FRAME_INFO_S *pstFrame);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstFrame	获取的输出屏幕图像数据信息结构体指针。	输出

【返回值】

返回值	描述
0	成功。



返回值	描述
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_CHN_NOT_ENABLE	通道未使能。
HI_ERR_VO_ILLEGAL_PARAM	视频输出参数无效。
HI_ERR_VO_WAIT_TIMEOUT	获取图像数据超时。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- 可以多次调用，获取后应保证及时的释放。
- 如果需要获取更多连续的输出图像，用户需要适当地加显示 buffer 长度。
- 该接口属于阻塞接口，超时时间为 1s，所以在显示帧率较低时，由于系统的波动可能导致接口返回超时，此时属于正常情况。
- 如果用户获取了屏幕图像没有进行释放，则在禁止视频层的时候会返回 busy 的错误，这时用户应该释放获取的 buffer，再次调用禁止视频层操作，这样才能保证禁止成功。
- 该接口获取的图像含有时间戳，该时间戳表示图像拼接时的时间信息，如果用户不希望采用该时间戳进行编码，可以自行修改时间戳信息。

【举例】

请参见 [HI_MPI_VO_GetChnFrame](#) 的举例。

【相关主题】

- [HI_MPI_VO_ReleaseScreenFrame](#)
- [HI_MPI_VO_SetDispBufLen](#)

HI_MPI_VO_ReleaseScreenFrame

【描述】



释放输出屏幕图像数据。

【语法】

```
HI_S32 HI_MPI_VO_ReleaseScreenFrame(VO\_DEV VoDev,  
                                     VIDEO_FRAME_INFO_S *pstFrame);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstFrame	释放的输出屏幕图像数据信息结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_CHN_NOT_ENABLE	视频输出通道未使能。
HI_ERR_VO_ILLEGAL_PARAM	视频输出参数无效。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- 可以多次调用，获取操作应保证与释放操作配对。

【举例】

请参见 [HI_MPI_VO_GetChnFrame](#) 的举例。



【相关主题】

[HI_MPI_VO_GetScreenFrame](#)

HI_MPI_VO_SetVbiInfo

【描述】

设置图像附加数据信息。

【语法】

```
HI_S32 HI_MPI_VO_SetVbiInfo(VO_DEV VoDev, VO_VBI_INFO_S *pstVbiInfo);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstVbiInfo	图像附加信息结构体。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_ILLEGAL_PARAM	无效参数。
HI_ERR_VO_NULL_PTR	参数指针为空。
HI_ERR_VO_NO_MEM	系统没有为附加信息准备好内存空间。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】



- Hi3520 芯片的附加信息长度不能大于 768×5 bytes。
- 设置信息成功后如果不清除，附加信息将始终位于图像上。
- Hi3520 的 VBI 信息会影响图像显示。Hi3515 芯片增加了真实的 VBI 功能，信息在消隐区，不会影响图像显示。

【举例】

```
HI_U32 u32Ret;
HI_CHAR vbi[100] = "good morning sir!";
HI_CHAR gvbi[100];

VIDEO_VBI_INFO_S stVbiInfo, stVbiInfoGet;

stVbiInfo.u32X = 0;
stVbiInfo.u32Y = 0;
stVbiInfo.u32InfoLen = 100;
stVbiInfo.pu8VbiInfo = (HI_U32 *)vbi;

stVbiInfoGet.pu8VbiInfo = (HI_U32 *)gvbi;

/* e.g. Enable Vi to Vo preview */
...

u32Ret = HI_MPI_VO_SetVbiInfo(VoDev, &stVbiInfo);
if (HI_SUCCESS != u32Ret)
{
    printf("set vbi error with return value %#x\n", u32Ret);
    return HI_FAILURE;
}

u32Ret = HI_MPI_VO_GetVbiInfo(VoDev, &stVbiInfoGet);
if (HI_SUCCESS != u32Ret)
{
    printf("get vbi error with return value %#x\n", u32Ret);
    return HI_FAILURE;
}

printf("[%d,%d,%d,%s]\n", stVbiInfoGet.u32X, stVbiInfoGet.u32Y,
        stVbiInfoGet.u32InfoLen, stVbiInfoGet.pu8VbiInfo);

sleep(5);

u32Ret = HI_MPI_VO_ClrVbiInfo(VoDev);
if (HI_SUCCESS != u32Ret)
{
```



```
printf("Clear Vbi info with return value %#x\n", u32Ret);  
return HI_FAILURE;  
}  
  
/* e.g. Disable Vi to Vo preview */  
...
```

【相关主题】[HI_MPI_VO_GetVbiInfo](#)

HI_MPI_VO_GetVbiInfo

【描述】

获取图像附加数据信息。

【语法】

```
HI_S32 HI_MPI_VO_GetVbiInfo(VO\_DEV VoDev, VIDEO_VBI_INFO_S *pstVbiInfo);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstVbiInfo	图像附加信息结构体。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_NULL_PTR	参数指针为空。

【需求】



- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

附加信息若没有设置会返回失败。

【举例】

请参见 [HI_MPI_VO_SetVbiInfo](#) 的举例。

【相关主题】

[HI_MPI_VO_SetVbiInfo](#)

HI_MPI_VO_ClrVbiInfo

【描述】

清除图像附加数据信息。

【语法】

```
HI_S32 HI_MPI_VO_ClrVbiInfo (VO_DEV VoDev);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	无效设备号。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h



- 库文件：libmpi.a

【注意】

- 不需要附加信息的时候应及时清除。
- 该接口可以重复调用。

【举例】

请参见 [HI_MPI_VO_SetVbiInfo](#) 的举例。

【相关主题】

[HI_MPI_VO_SetVbiInfo](#)

HI_MPI_VO_ClearChnBuffer

【描述】

清空指定输出通道的缓存 buff 数据。

【语法】

```
HI_S32 HI_MPI_VO_ClearChnBuffer(VO_DEV VoDev, VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道号无效。
HI_ERR_VO_CHN_NOT_ENABLE	通道未使能。



【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

该接口功能在解码 HI_MPI_VDEC_ResetChn 中已经涵盖。用户也可以单独调用此接口实现清除 VO 通道 buffer 的功能。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

[HI_MPI_VO_SetVbiInfo](#)

HI_MPI_VO_SetDevCSC

【描述】

设置设备输出图像效果。

【语法】

```
HI_S32 HI_MPI_VO_SetDevCSC(VO\_DEV VoDev, const VO\_CSC\_S *pstPubCSC);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstPubCSC	图像输出效果结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_VO_INVALID_CHNID	通道号无效。
HI_ERR_VO_ILLEGAL_PARAM	无效参数。
HI_ERR_VO_NULL_PTR	参数指针为空。
HI_ERR_VO_DEV_NOT_CONFIG	设备未配置。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 该接口主要用于调节图像的输出效果，包括亮度、对比度、色调、饱和度，其取值范围在 0 到 100 之间。
- 调用该接口前确保设备属性已经配置。

【举例】

无。

【相关主题】

[HI_MPI_VO_GetDevCSC](#)

HI_MPI_VO_GetDevCSC

【描述】

获取设备输出图像效果。

【语法】

HI_S32 HI_MPI_VO_GetDevCSC([VO_DEV](#) VoDev, [VO_CSC_S](#) *pstPubCSC);

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstPubCSC	图像输出效果结构体指针。	输出

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道号无效。
HI_ERR_VO_ILLEGAL_PARAM	无效参数。
HI_ERR_VO_NULL_PTR	参数指针为空。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

该接口主要用于获取图像的输出生效果，包括亮度、对比度、色调、饱和度，其取值范围在 0 到 100 之间。

【举例】

无。

【相关主题】

[HI_MPI_VO_SetDevCSC](#)

HI_MPI_VO_SetDispBufLen

【描述】

设置显示缓冲的长度。

【语法】

```
HI_S32 HI_MPI_VO_SetDispBufLen(VO\_DEV VoDev, HI_U32 u32BufLen);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入



参数名称	描述	输入/输出
u32BufLen	显示缓冲的长度。 物理设备取值范围: [VO_MIN_DISP_BUF, VO_MAX_DISP_BUF]; 虚拟设备取值范围: [VO_MIN_VIRT_BUF, VO_MAX_VIRT_BUF]	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	设备号无效。
HI_ERR_VO_DEV_NOT_ENABLE	设备未使能。
HI_ERR_VO_ILLEGAL_PARAM	无效参数。
HI_ERR_VO_NOT_PERMIT	操作不允许。

【需求】

- 头文件: mpi_vo.h、hi_comm_vo.h
- 库文件: libmpi.a

【注意】

- 调用前确保视频层未使能。
- 缓冲长度具有默认值，物理设备默认值为 VO_DEF_DISP_BUF_LEN，虚拟设备默认值为 VO_DEF_VIRT_BUF_LEN。
- 该接口主要用于配合屏幕图像获取接口使用，如果用户需要获取连续的视频输出图像数据，需要将显示缓冲的长度按照上述要求设置。缓冲长度越大，即用户可以持有的显示图像越多。
- 默认情况下，允许用户获取一帧显示图像。
- 在单通道直通的情况下，不允许用户获取输出图像，事实上用户可以获取通道图像或者源图像代替。



- 实际使用上，用户应该尽量保证获取图像后及时释放。

【举例】

无。

【相关主题】

[HI_MPI_VO_GetScreenFrame](#)

HI_MPI_VO_GetDispBufLen

【描述】

获取显示缓冲的长度。

【语法】

```
HI_S32 HI_MPI_VO_GetDispBufLen(VO_DEV VoDev, HI_U32 *pu32BufLen);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pu32BufLen	显示缓冲的长度指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	设备号无效。
HI_ERR_VO_DEV_NOT_ENABLE	设备未使能。
HI_ERR_VO_NULL_PTR	参数指针为空。

【需求】



- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

缓冲长度默认值为 VO_DEF_DISP_BUF_LEN。

【举例】

无。

【相关主题】

[HI_MPI_VO_SetDispBufLen](#)

HI_MPI_VO_SetSolidDraw

【描述】

设置屏幕绘制属性。该接口用于在视频层绘制线条或者矩形区域。

【语法】

```
HI_S32 HI_MPI_VO_SetSolidDraw(VO_DEV VoDev, const VO_DRAW_CFG_S
*pstDrawCfg);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstDrawCfg	绘制属性结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	设备号无效。
HI_ERR_VO_DEV_NOT_ENABLE	设备未使能。



接口返回值	含义
HI_ERR_VO_NULL_PTR	参数指针为空。
HI_ERR_VO_VIDEO_NOT_CONFIG	视频层未配置。
HI_ERR_VO_ILLEGAL_PARAM	参数非法。
HI_ERR_VO_NO_MEM	内存不足。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 使用该接口进行线条绘制或者色块填充会影响系统性能，尤其在系统负荷紧张的情况下。因为该接口的绘制图像刷新率和视频层一致，从而进一步地增加了 TDE 的负荷。
- 当设置绘制计数为 0 时，所有划线取消。
- 当将某个绘制区的宽或者高设置为 0 时，取消该绘制区。

【举例】

无。

【相关主题】

[HI_MPI_VO_GetSolidDraw](#)

HI_MPI_VO_GetSolidDraw

【描述】

获取屏幕绘制属性。

【语法】

```
HI_S32 HI_MPI_VO_GetSolidDraw(VO_DEV VoDev, VO_DRAW_CFG_S *pstDrawCfg);
```

【参数】

参数名称	描述	输入/输出
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
pstDrawCfg	绘制属性结构体指针。	输出

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_DEVID	设备号无效。
HI_ERR_VO_DEV_NOT_ENABLE	设备未使能。
HI_ERR_VO_NULL_PTR	参数指针为空。
HI_ERR_VO_NO_MEM	内存不足。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

设置绘制属性前建议先获取绘制属性，这样就不至于在设置新的绘制区时影响原来的绘制区。

【举例】

无。

【相关主题】

[HI_MPI_VO_SetSolidDraw](#)

HI_MPI_VO_EnableCascade

【描述】

使能视频级联。

【语法】

```
HI_S32 HI_MPI_VO_EnableCascade (HI_VOID);
```

【参数】

无。

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备 Fd 未打开。
HI_ERR_VO_DEV_NOT_ENABLE	设备未使能。
HI_ERR_VO_VIDEO_NOT_ENABLE	视频层未使能。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 级联只能在高清设备上进行。
- 级联使能前必须保证高清设备使能和该设备上的视频层使能。

【举例】

请参见《Hi3520 视频级联应用指南》。

【相关主题】

[HI_MPI_VO_DisableCascade](#)

HI_MPI_VO_DisableCascade

【描述】

禁止视频级联。

【语法】

```
HI_S32 HI_MPI_VO_DisableCascade(HI_VOID);
```

【参数】

无。

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备Fd未打开。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

无。

【举例】

请参见《Hi3520 视频级联应用指南》。

【相关主题】

[HI_MPI_VO_EnableCascade](#)

HI_MPI_VO_SetCascadeMode

【描述】

设置视频级联模式。

【语法】

```
HI_S32 HI_MPI_VO_SetCascadeMode(HI_BOOL bSlave);
```

【参数】

参数名称	描述	输入/输出
bSlave	视频输出级联模式。	输入

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备Fd未打开。
HI_ERR_VO_BUSY	视频输出设备忙。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 设置视频级联模式必须在设备使能之前进行，即必须保证设备已禁止。
- 级联的第一芯片设置为主模式，其它级联芯片设置为从模式。

【举例】

请参见《Hi3520 视频级联应用指南》。

【相关主题】

[HI_MPI_VO_GetCascadeMode](#)

HI_MPI_VO_GetCascadeMode

【描述】

获取视频级联模式。

【语法】

```
HI_S32 HI_MPI_VO_GetCascadeMode(HI_BOOL *pbSlave);
```

【参数】

参数名称	描述	输入/输出
pbSlave	视频输出级联模式指针。	输出

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备 Fd 未打开。
HI_ERR_VO_NULL_PTR	参数空指针。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

无。

【举例】

请参见《Hi3520 视频级联应用指南》。

【相关主题】

[HI_MPI_VO_SetCascadeMode](#)

HI_MPI_VO_SetCascadePattern

【描述】

设置视频级联画面样式。

【语法】

```
HI_S32 HI_MPI_VO_SetCascadePattern(HI_U32 u32Pattern);
```

【参数】

参数名称	描述	输入/输出
u32Pattern	视频级联画面布局样式。 取值范围：[0, 127]。	输入

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备 Fd 未打开。
HI_ERR_VO_INVALID_PATTERN	无效的样式。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

画面布局样式是一个小于 128 的正整数，它的主要作用是检测各片之间传输图像的一致性。即如果要想保证图像传输的正确性，就必须保证各片的 pattern 一致。

- 用 pattern 来标识当前画面布局样式，如 4 画面、9 画面、16 画面的 pattern 对应设置为 4、9、16。设置的 pattern 值不必和画面布局样式中的画面个数一致，只要保证各片设置的 pattern 值一致即可，如也可以用 1 来表示每个片子上都是 16 分屏的画面布局。
- 使用 pattern 的另一个作用是确保多画面切换时的同步性。当级联的芯片正在进行 4 画面显示时，其中一片先切换到 9 画面，而其它片子陆续切换为 9 画面，如果 pattern 始终不变，画面就会在切换过程中出现错误的几帧。但是如果在 4 画面时 pattern 设置为 4，在 9 画面时 pattern 设置为 9，那么切换过程中出现的错误图像就会被丢弃，保证画面切换的正确性。

【举例】

请参见《Hi3520 视频级联应用指南》。

【相关主题】

[HI_MPI_VO_GetCascadePattern](#)

HI_MPI_VO_GetCascadePattern

【描述】

获取视频级联画面样式。

【语法】



```
HI_S32 HI_MPI_VO_GetCascadePattern(HI_U32 *pu32Pattern);
```

【参数】

参数名称	描述	输入/输出
pu32Pattern	视频级联画面布局样式指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备 Fd 未打开。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

无。

【举例】

请参见《Hi3520 视频级联应用指南》。

【相关主题】

[HI_MPI_VO_SetCascadePattern](#)

HI_MPI_VO_CascadePosBindChn

【描述】

绑定级联区域与视频输出通道。

【语法】

```
HI_S32 HI_MPI_VO_CascadePosBindChn (HI_U32 u32Pos, VO_CHN VoChn);
```



【参数】

参数名称	描述	输入/输出
u32Pos	视频级联位置编号。 取值范围：[0, 31]。	输入
VoChn	视频层通道号。 取值范围：[0, 31]。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备 Fd 未打开。
HI_ERR_VO_DEV_NOT_ENABLE	视频输出设备未使能。
HI_ERR_VO_INVALID_POSITION	无效的级联位置编号。
HI_ERR_VO_INVALID_CHNID	通道号无效。
HI_ERR_VO_CHN_NOT_ALLOC	通道资源未分配。
HI_ERR_VO_CHN_NOT_CONFIG	通道未配置。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 视频级联位置编号必须在所有的级联芯片上是唯一标识的，也就是级联最多可以输出 32 个通道。
- Pos 的值不能有重复，如果存在两个或者两个以上通道的 Pos 值相同，那么传输图像就会错误。Pos 相当于给 Chn 取了个别名，所有需要级联的通道都必须有一个名字区别于其它通道，这样才能保证传输图像的正确性。



【举例】

请参见《Hi3520 视频级联应用指南》。

【相关主题】

[HI_MPI_VO_CascadePosUnBindChn](#)

HI_MPI_VO_CascadePosUnBindChn

【描述】

解绑定级联区域与视频输出通道。

【语法】

```
HI_S32 HI_MPI_VO_CascadePosUnBindChn(HI_U32 u32Pos, VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
u32Pos	视频级联位置编号。 取值范围：[0, 31]。	输入
VoChn	视频层通道号。 取值范围：[0, 31]。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_SYS_NOTREADY	系统未准备好，这里指设备 Fd 未打开。
HI_ERR_VO_DEV_NOT_ENABLE	视频输出设备未使能。
HI_ERR_VO_INVALID_POSITION	无效的级联位置编号。
HI_ERR_VO_INVALID_CHNID	通道号无效。
HI_ERR_VO_CHN_NOT_ALLOC	通道资源未分配。



接口返回值	含义
HI_ERR_VO_CHN_NOT_CONFIG	通道未配置。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

视频级联位置编号必须在所有的级联芯片上是唯一标识的，也就是级联最多可以输出32个通道。

【举例】

请参见《Hi3520 视频级联应用指南》。

【相关主题】

[HI_MPI_VO_CascadePosBindChn](#)

4.4 数据类型

视频输出相关数据类型定义如下：

- [VO_DEV](#)：定义设备号。
- [VO_GRP](#)：定义同步组号。
- [VO_PUB_ATTR_S](#)：定义视频输出设备属性结构体。
- [VO_VIDEO_LAYER_ATTR_S](#)：定义视频层属性结构体。
- [DSU_HSCALE_FILTER_E](#)：定义水平缩放系数。
- [DSU_VSCALE_FILTER_E](#)：定义垂直缩放系数。
- [VO_CHN_FILTER_S](#)：定义 VO 通道图像缩放的滤波系数属性结构体。
- [VO_SCREEN_HFILTER_E](#)：定义 VO 缩放水平缩放系数。
- [VO_SCREEN_VFILTER_E](#)：定义 VO 缩放垂直缩放系数。
- [VO_SCREEN_FILTER_S](#)：定义 VO 缩放滤波系数属性结构体。
- [VO_CHN_ATTR_S](#)：定义视频输出通道属性结构体。
- [VO_DISPLAY_FIELD_E](#)：定义视频输出时的顶底场模式。
- [VO_QUERY_STATUS_S](#)：定义视频输出通道状态结构体。
- [VO_VBI_INFO_S](#)：定义 VBI 信息结构体。
- [VO_ZOOM_ATTR_S](#)：定义局部放大属性结构体。
- [VO_ZOOM_RATIO_S](#)：定义按比例局部放大结构体。
- [VO_CSC_S](#)：定义图像输出效果结构体。
- [VO_SRC_ATTR_S](#)：定义视频源属性结构体。



- [VO_DRAW_CFG_S](#): 定义绘制区属性。
- [VO_DRAW_ATTR_S](#): 定义单个绘制区属性。

VO_DEV

【说明】

定义设备号。

【定义】

```
typedef HI_S32 VO_DEV;
```

【成员】

成员名称	描述
VO_DEV	视频输出模块有三个视频输出设备，分别进行如下定义： 0: HD 设备，即高清显示设备。 1: AD 设备，即辅助显示设备。 2: SD 设备，即标清轮询设备。 3: 虚拟设备。 4: 虚拟设备。 5: 虚拟设备。

【注意事项】

- 高清设备可以输出 VGA/YPBPR/BT.1120 等高清模拟、数字信号。
- 辅助设备可以输出 VGA 和 CVBS、BT.656 信号。
- 标清轮询设备只能输出 CVBS 信号。
- 每个设备最多容纳 64 个通道。
- Hi3515 不支持 AD 设备，即只有 HD 和 SD 可用。
- 虚拟设备不进行实际的显示输出，只是将通道图像进行拼接处理。因此，所有与视频输出硬件相关的操作对于虚拟设备来说都是无效的，调用相应的接口会返回成功。例如色彩空间转换系数、设备公共属性、级联等。

【相关数据类型及接口】

[VO_PUB_ATTR_S](#)

VO_GRP

【说明】

定义同步组号。

【定义】

```
typedef HI_S32 VO_GRP;
```



【成员】

成员名称	描述
VO_GRP	视频输出同步组组号。 同步组最大值：VO_SYNC_MAX_GRP。 最多容纳通道数 VO_SYNC_MAX_CHN。

【注意事项】

无。

【相关数据类型及接口】

同步组所有操作。

VO_PUB_ATTR_S

【说明】

定义视频输出公共属性结构体。

【定义】

```
typedef struct hiVO_PUB_ATTR_S
{
    /* background color of device [RGB] */
    HI_U32          u32BgColor;

    /* vo inteface type */
    VO_INTF_TYPE_E  enIntfType;

    /* vo interface synchronization */
    VO_INTF_SYNC_E  enIntfSync;

    /* video synchronizing information */
    VO_SYNC_INFO_S  stSyncInfo;
} VO_PUB_ATTR_S;
```

【成员】

成员名称	描述
u32BgColor	设备背景色，表示方法 RGB888。



成员名称	描述
enIntfType	<p>接口类型典型配置，原型定义：</p> <pre>typedef enum hiVO_INTF_TYPE_E { VO_INTF_CVBS = 0, VO_INTF_BT656 = 1, VO_INTF_VGA = 2, VO_INTF_YPBPR = 3, VO_INTF_BT1120 = 4, VO_INTF_LCD = 5, VO_INTF_BUTT } VO_INTF_TYPE_E;</pre>
enIntfSync	<p>接口时序典型配置，原型定义：</p> <pre>typedef enum hiVO_INTF_SYNC_E { VO_OUTPUT_PAL = 0, VO_OUTPUT_NTSC = 1, VO_OUTPUT_720P60 = 2, VO_OUTPUT_1080I50 = 3, VO_OUTPUT_1080I60 = 4, VO_OUTPUT_1080P25 = 5, VO_OUTPUT_1080P30 = 6, VO_OUTPUT_800x600_60 = 7, VO_OUTPUT_1024x768_60 = 8, VO_OUTPUT_1280x1024_60 = 9, VO_OUTPUT_1366x768_60 = 10, VO_OUTPUT_1440x900_60 = 11, VO_OUTPUT_800x600_75 = 12, VO_OUTPUT_1024x768_75 = 13, VO_OUTPUT_USER = 14, VO_OUTPUT_BUTT } VO_INTF_SYNC_E;</pre>



成员名称	描述
stSyncInfo	接口时序结构体，原型定义： <pre>typedef struct tagVO_SYNC_INFO_S { HI_BOOL bSynm; HI_BOOL bIop; HI_U8 u8Intfb; HI_U16 u16Vact ; HI_U16 u16Vbb; HI_U16 u16Vfb; HI_U16 u16Hact; HI_U16 u16Hbb; HI_U16 u16Hfb; HI_U16 u16Bvact; HI_U16 u16Bvbb; HI_U16 u16Bvfb; HI_U16 u16Hpw; HI_U16 u16Vpw; HI_BOOL bIdv; HI_BOOL bIhs; HI_BOOL bIvs; } VO_SYNC_INFO_S;</pre>

【注意事项】

- 当接口时序配置为 VO_OUTPUT_USER 时，stSyncInfo 定义的时序结构才会生效，表示用户自定义的时序结构。
- 接口类型配置为 VO_INTF_CVBS 或 VO_INTF_BT656 时，enIntfSync 的取值范围为[0, 1]。
- 接口类型为 VO_INTF_VGA 或者 VO_INTF_LCD 时，enIntfSync 的取值范围为[7, 13]。
- 接口类型为 VO_INTF_YPBPR 或 VO_INTF_BT1120 时，enIntfSync 的取值范围为[2, 6]。
- 配置设备属性在设备下一次打开时生效。



- Hi3515 的 HD 只支持 VO_INTF_VGA 接口，同时其时序只支持 VO_OUTPUT_800x600_60 到 VO_OUTPUT_1440x900_60 的 VGA 显示时序，当然，用户也可以自定义时序，但必须属于 VGA 接口系列。
- Hi3515 的 SD 支持 VO_INTF_CVBS 和 VO_INTF_BT656 两种接口，时序支持典型时序 VO_OUTPUT_PAL 和 VO_OUTPUT_NTSC。

【相关数据类型及接口】

[HI_MPI_VO_SetPubAttr](#)

VO_VIDEO_LAYER_ATTR_S

【说明】

定义视频层属性。

在视频层属性中存在三个概念，即设备分辨率、显示分辨率和图像分辨率。每种分辨率的概念可以从图 4-1 和图 4-2 中看出。

TDE 缩放引擎负责将各个通道的画面拼接成多画面的合成图像，这个合成画面的分辨率即图像分辨率，也就是视频层数据结构中的 stImageSize。

VO 缩放引擎负责将合成图像缩放到屏幕显示区域，当进行缩小时，最大缩小能力为 1/2，如果小于 1/2，画面将被剪裁（而不是缩小）成需要的比例；放大无此限制。

图4-1 视频层属性的相关概念示意（基本处理模式）

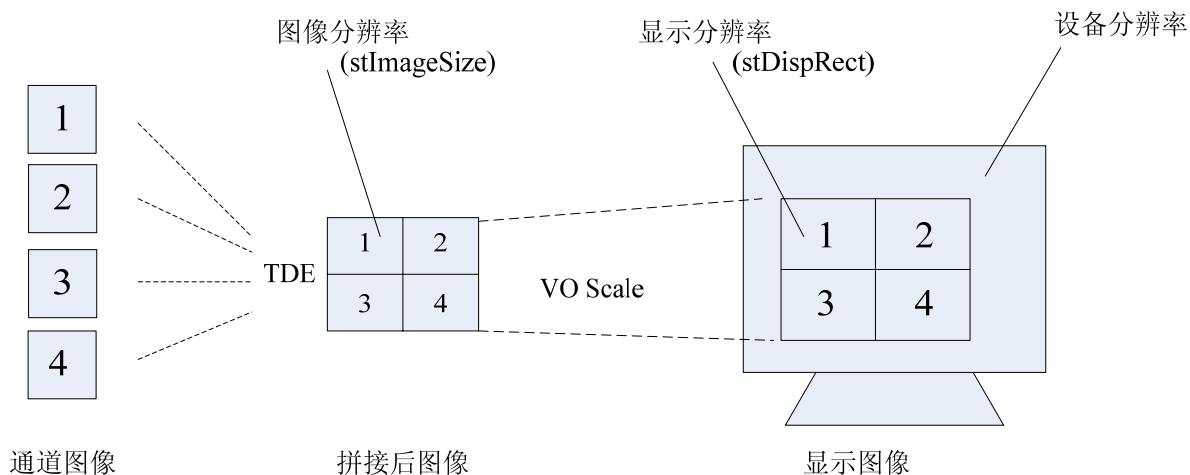
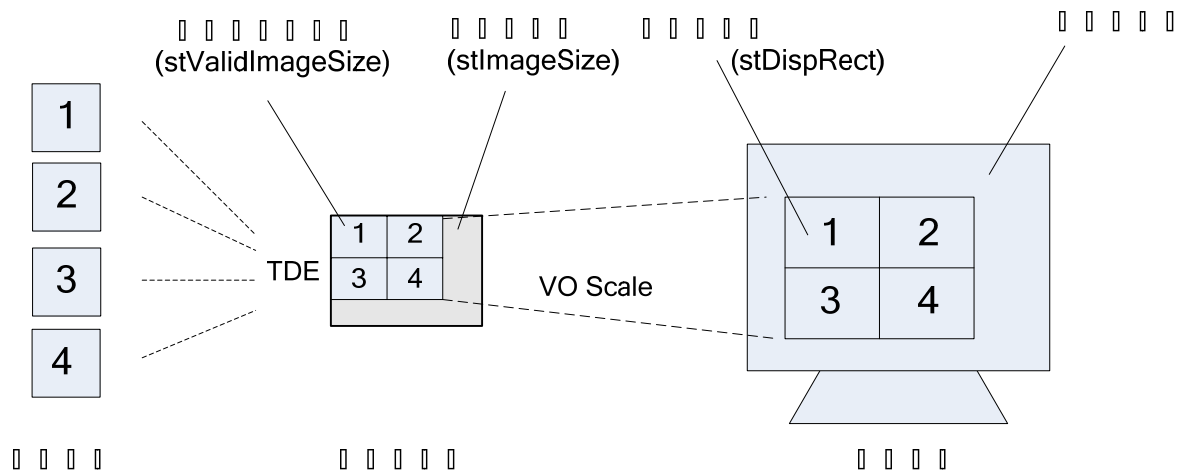




图4-2 视频层属性的相关概念示意（设置有效区处理模式）



【定义】

```
typedef struct hiVO_VIDEO_LAYER_ATTR_S
{
    RECT_S stDispRect;           /* display window */
    SIZE_S stImageSize;          /* display resolution */
    HI_U32 u32DispFrmRt;         /* display frame rate */
    PIXEL_FORMAT_E enPixelFormat; /* support sp420 and sp422 */

    HI_S32 s32PiPChn; /* (1) VO_DEFAULT_CHN: default value */
                  /* (2) [0~31]: use the channel buffer as pip buffer */
} VO_VIDEO_LAYER_ATTR_S;
```

【成员】

成员名称	描述
stDispRect	视频显示区域矩形结构体。
stImageSize	图像分辨率结构体，即合成画面尺寸。
u32DispFrmRt	视频显示帧率。
enPixelFormat	视频层输入像素格式，SPYCbCr420 或者 SPYCbCr422。
s32PiPChn	画面合成路径标识。 默认值为 VO_DEFAULT_CHN。

【注意事项】

- stDispRect 范围不能超出设备分辨率。



- `stImageSize` 如果大于 `stDispRect`，合成图像将被剪裁。
- `u32DispFrmRt` 是视频显示帧率，实质上是拼接图像的帧率。该值一般设置为通道满帧率，即：
 - 如果源是 PAL 制式，设置为 25；
 - 如果源是 NTSC 制式，设置为 30。
 如果设置的帧率比通道的源帧率高，则会造成不必要的性能浪费。
- SD 设备上不支持 VO 的缩放，所以对于 SD 上的输出，通常保持图像分辨率和显示分辨率一致。
- `s32PiPChn` 标识画面合成路径，默认模式 `VO_DEFAULT_CHN` 表示所有图像都经过 TDE 缩放引擎搬移，该模式适合于任何情况，建议用户采用。单通道输出时，如果该标识不是默认路径，输入图像就会将 TDE 缩放引擎 BYPASS。
- `s32PiPChn` 标识的非默认模式不仅可以用作单通道直通输出，而且在多通道情况下可以节省拼接图像的公共 `buffer`。在多通道情况下，用户可以设置 `s32PiPChn` 为 `[0, VO_MAX_CHN_NUM-1]` 中的某一值，即将指定的通道作为拼接图像的目标区域，其它通道的图像都叠加到该图像上输出。
- 对于虚拟设备，`stDispRect` 无效，因为虚拟设备不会将图像输出显示。

【相关数据类型及接口】

[HI_MPI_VO_SetPubAttr](#)

DSU_HSCALE_FILTER_E

【说明】

定义水平缩放系数。

【定义】

```
typedef enum hiDSU_HSCALE_FILTER_E
{
    DSU_HSCALE_FILTER_DEFAULT = 0,
    DSU_HSCALE_FILTER_C_65M,
    DSU_HSCALE_FILTER_CG_56M,
    DSU_HSCALE_FILTER_LC_45M,
    DSU_HSCALE_FILTER_CG_3M,
    DSU_HSCALE_FILTER_CG_2M,
    DSU_HSCALE_FILTER_CG_1M,
    DSU_HSCALE_FILTER_BUTT
}DSU_HSCALE_FILTER_E;
```

【成员】

成员名称	描述
<code>DSU_HSCALE_FILTER_DEFAULT</code>	默认缩放系数模式。



成员名称	描述
VPP_HSCALE_FILTER_n M	缩放系数。n 值越小，滤波后的图像越模糊。

【注意事项】

系统默认采用 DSU_HSCALE_FILTER_DEFAULT，即内部自动选择最优滤波系数。一般不需要使用者配置滤波系数。

【相关数据类型及接口】

- [HI_MPI_VO_SetChnFilter](#)
- [HI_MPI_VO_GetChnFilter](#)

DSU_VSCALE_FILTER_E

【说明】

定义垂直缩放系数。

【定义】

```
typedef enum hiDSU_VSCALE_FILTER_E
{
    DSU_VSCALE_FILTER_DEFAULT = 0,
    DSU_VSCALE_FILTER_S_6M,
    DSU_VSCALE_FILTER_S_5M,
    DSU_VSCALE_FILTER_S_4M,
    DSU_VSCALE_FILTER_S_38M,
    DSU_VSCALE_FILTER_S_37M,
    DSU_VSCALE_FILTER_S_36M,
    DSU_VSCALE_FILTER_S_25M,
    DSU_VSCALE_FILTER_S_2M,
    DSU_VSCALE_FILTER_S_15M,
    DSU_VSCALE_FILTER_S_12M,
    DSU_VSCALE_FILTER_S_11M,
    DSU_VSCALE_FILTER_S_1M,
    DSU_VSCALE_FILTER_BUTT
}DSU_VSCALE_FILTER_E;
```

【成员】

成员名称	描述
DSU_VSCALE_FILTER_DEFAULT	默认缩放系数模式。
VPP_VSCALE_FILTER_n M	缩放系数。n 值越小，滤波后的图像越模糊。



【注意事项】

系统默认采用 DSU_VSCALE_FILTER_DEFAULT，即内部自动选择最优滤波系数。一般不需要使用者配置滤波系数。

【相关数据类型及接口】

- [HI_MPI_VO_SetChnFilter](#)
- [HI_MPI_VO_GetChnFilter](#)

VO_CHN_FILTER_S

【说明】

定义 VO 通道图像缩放的滤波系数属性结构体。

【定义】

```
typedef struct hi_VO_CHN_FILTER_S
{
    DSU_FILTER_PARAM_TYPE    enFiltType;
    DSU_HSCALE_FILTER_E      enHFilter;
    DSU_VSCALE_FILTER_E      enVFilterL;
    DSU_VSCALE_FILTER_E      enVFilterC;
} VO_CHN_FILTER_S;
```

【成员】

成员名称	描述
enFiltType	滤波类型，每种滤波类型都包含水平/垂直亮度/垂直色度滤波系数
enHFilter	水平滤波系数选项（亮度和色度使用相同的水平滤波系数）
enVFilterL	亮度的垂直滤波系数选项
enVFilterC	色度的垂直滤波系数选项

【注意事项】

无。

【相关数据类型及接口】

- [HI_MPI_VO_SetChnFilter](#)
- [HI_MPI_VO_GetChnFilter](#)



VO_SCREEN_HFILTER_E

【说明】

定义 VO 缩放水平缩放系数。

【定义】

```
typedef enum hiVO_SCREEN_HFILTER_E
{
    VO_SCREEN_HFILTER_DEF    = 0,
    VO_SCREEN_HFILTER_8M,
    VO_SCREEN_HFILTER_6M,
    VO_SCREEN_HFILTER_5M,
    VO_SCREEN_HFILTER_4M,
    VO_SCREEN_HFILTER_3M,
    VO_SCREEN_HFILTER_2M,
    VO_SCREEN_HFILTER_BUTT

} VO_SCREEN_HFILTER_E;
```

【成员】

成员名称	描述
VO_SCREEN_HFILTER_DEF	默认缩放系数模式。
VO_SCREEN_HFILTER_n M	缩放系数。n 值越小，滤波后的图像越模糊。

【注意事项】

系统默认采用 VO_SCREEN_HFILTER_DEF，即内部自动选择最优滤波系数。一般不需要使用者配置滤波系数。

【相关数据类型及接口】

- [HI_MPI_VO_SetScreenFilter](#)
- [HI_MPI_VO_GetScreenFilter](#)

VO_SCREEN_VFILTER_E

【说明】

定义 VO 缩放垂直缩放系数。

【定义】

```
typedef enum hiVO_SCREEN_VFILTER_E
{
    VO_SCREEN_VFILTER_DEF    = 0,
```



```

VO_SCREEN_VFILTER_8M,
VO_SCREEN_VFILTER_6M,
VO_SCREEN_VFILTER_5M,
VO_SCREEN_VFILTER_4M,
VO_SCREEN_VFILTER_3M,
VO_SCREEN_VFILTER_2M,
VO_SCREEN_VFILTER_BUTT
} VO_SCREEN_VFILTER_E;

```

【成员】

成员名称	描述
VO_SCREEN_VFILTER_DEF	默认缩放系数模式。
VO_SCREEN_VFILTER_n M	缩放系数。n 值越小，滤波后的图像越模糊。

【注意事项】

系统默认采用 VO_SCREEN_VFILTER_DEF，即内部自动选择最优滤波系数。一般不需要使用者配置滤波系数。

【相关数据类型及接口】

- [HI_MPI_VO_SetScreenFilter](#)
- [HI_MPI_VO_GetScreenFilter](#)

VO_SCREEN_FILTER_S

【说明】

定义 VO 缩放滤波系数属性结构体。

【定义】

```

typedef struct hiVO_SCALE_FILTER_S
{
    VO_SCREEN_HFILTER_E enHFilter;
    VO_SCREEN_VFILTER_E enVFilter;
} VO_SCREEN_FILTER_S;

```

【成员】

成员名称	描述
enHFilter	水平滤波系数选项（亮度色度均已包含）
enVFilter	垂直滤波系数选项（亮度色度均已包含）



【注意事项】

无。

【相关数据类型及接口】

- [HI_MPI_VO_SetScreenFilter](#)
- [HI_MPI_VO_GetScreenFilter](#)

VO_CHN_ATTR_S

【说明】

定义视频输出通道属性。

【定义】

```
typedef struct hiVO_CHN_ATTR_S
{
    HI_U32  u32Priority;
    RECT_S  stRect;
    HI_BOOL bZoomEnable;
    HI_BOOL bDeflicker;
}VO_PUB_ATTR_S;
```

【成员】

成员名称	描述
u32Priority	视频通道叠加优先级，优先级高的在上层。 取值范围：[0, 31]。 动态属性。
stRect	通道矩形显示区域。以屏幕的左上角为原点。该矩形的左上角座标必须是 2 对齐，且该矩形区域必须在屏幕范围之内。 动态属性。
bZoomEnable	缩放开关标识。 取值范围： <ul style="list-style-type: none">• HI_TRUE：将输入图像缩放成 stRect 定义的尺寸在屏幕上显示。• HI_FALSE：输入图像上剪裁 stRect 定义的矩形区域进行显示。 动态属性。
bDeflicker	通道抗闪烁开关。 取值范围： <ul style="list-style-type: none">• HI_TRUE：将输入图像做抗闪烁处理后显示。• HI_FALSE：不对通道输入图像做抗闪烁处理，直接显示。 动态属性。



【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VO_SetChnAttr](#)

VO_DISPLAY_FIELD_E

【说明】

定义视频输出时的顶底场模式。

【定义】

```
typedef enum hiVO_DISPLAY_FIELD_E
{
    VO_FIELD_TOP,      /* top field*/
    VO_FIELD_BOTTOM,   /* bottom field*/
    VO_FIELD_BOTH,     /* top and bottom field*/
    VO_FIELD_BUTT
} VO_DISPLAY_FIELD_E;
```

【成员】

成员名称	描述
VO_FIELD_TOP	视频输出缩放时只处理顶场。
VO_FIELD_BOTTOM	视频输出缩放时只处理底场。
VO_FIELD_BOTH	视频输出缩放时两场都处理。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VO_SetChnField](#)

VO_QUERY_STATUS_S

【说明】

视频输出通道状态结构体。

【定义】

```
typedef struct hiVO_QUERY_STATUS_S
{
```



```
        HI_U32 u32ChnBufUsed;        /* channel buffer that been occupied */  
    } VO_QUERY_STATUS_S;
```

【成员】

成员名称	描述
u32ChnBufUsed	视频输出通道当前占用的视频 buffer 数目。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VO_QueryChnStat](#)

VO_VBI_INFO_S

【说明】

定义图像附加信息结构体。

【定义】

```
typedef struct hiVO_VBI_INFO_S  
{  
    HI_U32 u32X;  
    HI_U32 u32Y;  
    HI_U8 u8VbiInfo[128];  
    HI_U32 u32InfoLen;  
  
} VO_VBI_INFO_S;
```

【成员】

成员名称	描述
u32X	图像附加信息起始点横坐标。
u32Y	图像附加信息起始点纵坐标。
u8VbiInfo	图像附加信息数组。
u32InfoLen	图像附加信息长度。

【注意事项】

无。

【相关数据类型及接口】



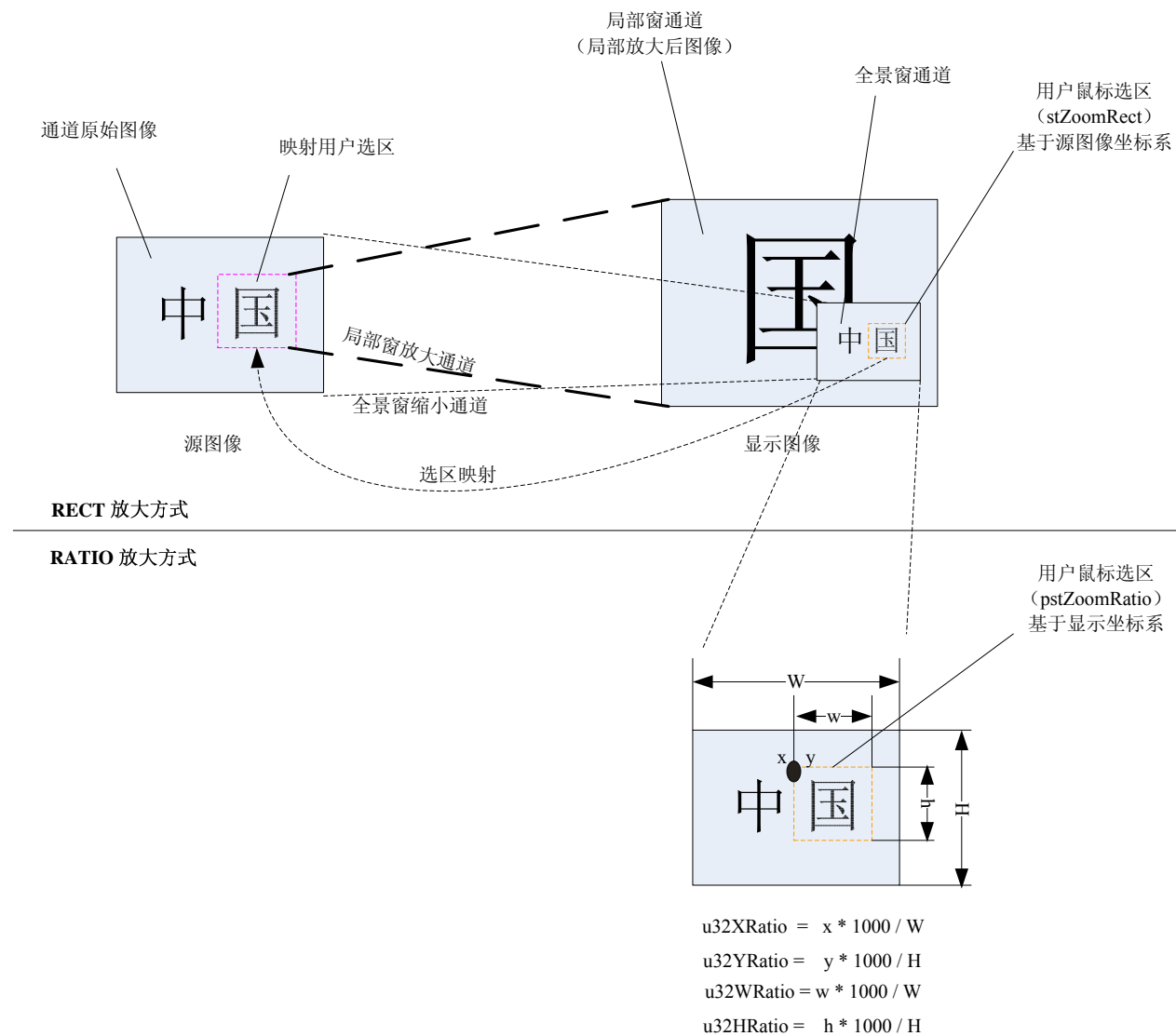
HI_MPI_VO_SetVbiInfo

VO_ZOOM_ATTR_S

【说明】

定义局部放大属性结构体。局部放大原理示意如图 4-3 所示。

图4-3 局部放大原理示意



【定义】

```
typedef struct hiVO_ZOOM_ATTR_S
{
    RECT_S stZoomRect;
    VIDEO_FIELD_E enField;
```




```
} VO_ZOOM_ATTR_S;
```

【成员】

成员名称	描述
stZoomRect	局部放大窗口矩形坐标。
enField	局部放大图像的帧场信息。

【注意事项】

- 取消局部放大功能，将该放大区域的矩形坐标设置为（0,0,0,0）。
- 帧场标识可以在级联时标识该放大区域是否要进行 DIE 操作，即如果该区域帧场标识设置为 VIDEO_FIELD_INTERLACED，则在逐行设备上就要进行 DIE 操作。

【相关数据类型及接口】

[HI_MPI_VO_SetZoomInWindow](#)

VO_ZOOM_RATIO_S

【说明】

定义按比例局部放大结构体。

【定义】

```
typedef struct hiVO_ZOOM_RATIO_S
{
    HI_U32 u32XRatio;
    HI_U32 u32YRatio;
    HI_U32 u32WRatio;
    HI_U32 u32HRatio;
} VO_ZOOM_RATIO_S;
```

【成员】

成员名称	描述
u32XRatio	屏幕坐标上，待缩放区域起始点横坐标与显示通道宽的比例。
u32YRatio	屏幕坐标上，待缩放区域起始点纵坐标与显示通道高的比例。
u32WRatio	屏幕坐标上，待缩放区域宽与显示通道宽的比例。
u32HRatio	屏幕坐标上，待缩放区域高与显示通道高的比例。

【注意事项】



- 结构体成员变量的取值范围[0, 1000]。
- 取消局部放大功能时，只要将该结构体中的成员都置 0 即可。

【相关数据类型及接口】

[HI_MPI_VO_SetZoomInRatio](#)

VO_CSC_S

【说明】

定义图像输出效果结构体。

【定义】

```
typedef struct hiVO_CSC_S
{
    VO_CSC_E enCSCType;
    HI_U32    u32Value;
} VO_CSC_S;
```

其中VO_CSC_E定义如下

```
typedef enum hiVO_CSC_E
{
    VO_CSC_LUMA      = 0,          /* luminance:  [0 ~ 100] */
    VO_CSC_CONTR     = 1,          /* contrast  :  [0 ~ 100] */
    VO_CSC_HUE       = 2,          /* hue       :   [0 ~ 100] */
    VO_CSC_SATU      = 3,          /* satuation:  [0 ~ 100] */
    VO_CSC_BUTT
} VO_CSC_E;
```

【成员】

成员名称	描述
enCSCType	图像效果类型。
u32Value	图像效果值。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VO_SetDevCSC](#)

VO_SRC_ATTR_S

【说明】



定义视频源属性结构体。

【定义】

```
typedef struct hiVO_SRC_ATTR_S
{
    HI_BOOL bInterlaced;          /* interlaced source */

} VO_SRC_ATTR_S;
```

【成员】

成员名称	描述
bInterlaced	图像源是否隔行输入。 取值范围： <ul style="list-style-type: none">• HI_TRUE：隔行输入。• HI_FALSE：逐行输入。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VO_SetChnSrcAttr](#)

VO_DRAW_CFG_S

【说明】

定义绘制区属性。

【定义】

```
typedef struct hiVO_DRAW_CFG_S
{
    HI_U32 u32DrawCount;
    VO_DRAW_ATTR_S stDrawAttr[VO_MAX_SOLIDDRAW];

} VO_DRAW_CFG_S;
```

【成员】

成员名称	描述
u32DrawCount	绘制区计数。 取值范围：[0, VO_MAX_SOLIDDRAW]。 动态属性。



成员名称	描述
stDrawAttr	绘制区域属性结构体。 动态属性。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VO_SetSolidDraw](#)

VO_DRAW_ATTR_S

【说明】

定义单个绘制区属性。

【定义】

```
typedef struct hiVO_DRAW_ATTR_S
{
    RECT_S stDrawRect;
    HI_U32 u32Color;
} VO_DRAW_ATTR_S;
```

【成员】

成员名称	描述
stDrawRect	绘制区范围。 绘制区域不能超出 stImgSize，同时绘制区的起始点和宽高都要求 2 对齐。
u32Color	绘制区颜色。 颜色以 RGB888 表示，不支持 alpha 操作。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VO_SetSolidDraw](#)



4.5 错误码

视频输出 API 错误码如表 4-1 所示。

表4-1 视频输出 API 错误码

错误代码	宏定义	描述
0xA00F8001	HI_ERR_VO_INVALID_DEVID	设备 ID 超出合法范围
0xA00F8002	HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围
0xA00F8003	HI_ERR_VO_ILLEGAL_PARAM	参数超出合法范围
0xA00F8006	HI_ERR_VO_NULL_PTR	函数参数中有空指针
0xA00F8008	HI_ERR_VO_NOT_SUPPORT	不支持的操作
0xA00F8009	HI_ERR_VO_NOT_PERMIT	操作不允许
0xA00F800C	HI_ERR_VO_NO_MEM	内存不足
0xA00F8010	HI_ERR_VO_SYS_NOTREADY	系统未初始化
0xA00F8012	HI_ERR_VO_BUSY	资源忙
0xA00F8040	HI_ERR_VO_DEV_NOT_CONFIG	设备未配置
0xA00F8041	HI_ERR_VO_DEV_NOT_ENABLE	设备未使能
0xA00F8042	HI_ERR_VO_DEV_NOT_DISABLE	设备未禁止
0xA00F8045	HI_ERR_VO_VIDEO_NOT_ENABLE	视频层未使能
0xA00F8046	HI_ERR_VO_VIDEO_NOT_DISABLE	视频层未禁止
0xA00F8047	HI_ERR_VO_VIDEO_NOT_CONFIG	视频层未配置
0xA00F8048	HI_ERR_VO_CHN_NOT_DISABLE	通道未禁止
0xA00F8049	HI_ERR_VO_CHN_NOT_ENABLE	通道未使能
0xA00F804A	HI_ERR_VO_CHN_NOT_CONFIG	通道未配置
0xA00F804B	HI_ERR_VO_CHN_NOT_ALLOC	通道未分配资源
0xA00F804C	HI_ERR_VO_INVALID_PATTERN	无效样式
0xA00F804D	HI_ERR_VO_INVALID_POSITION	无效级联位置
0xA00F804E	HI_ERR_VO_WAIT_TIMEOUT	等待超时
0xA00F804F	HI_ERR_VO_INVALID_VFRAME	无效视频帧
0xA00F8050	HI_ERR_VO_INVALID_RECT_PARA	无效矩形参数
0xA00F8051	HI_ERR_VO_SETBEGIN_ALREADY	BEGIN 已设置



错误代码	宏定义	描述
0xA00F8052	HI_ERR_VO_SETBEGIN_NOTYET	BEGIN 没有设置
0xA00F8053	HI_ERR_VO_SETEND_ALREADY	END 已设置
0xA00F8054	HI_ERR_VO_SETEND_NOTYET	END 没有设置
0xA00F8055	HI_ERR_VO_GRP_INVALID_ID	同步组 ID 无效
0xA00F8056	HI_ERR_VO_GRP_NOT_CREATE	同步组未创建
0xA00F8057	HI_ERR_VO_GRP_HAS_CREATED	同步组已经创建
0xA00F8058	HI_ERR_VO_GRP_NOT_DESTROY	同步组未销毁
0xA00F8059	HI_ERR_VO_GRP_CHN_FULL	同步组通道注册满
0xA00F805A	HI_ERR_VO_GRP_CHN_EMPTY	同步组无注册通道
0xA00F805B	HI_ERR_VO_GRP_CHN_NOT_EMPTY	同步组通道不为空
0xA00F805C	HI_ERR_VO_GRP_INVALID_SYN_MODE	无效同步模式
0xA00F805D	HI_ERR_VO_GRP_INVALID_BASEPTS	无效基准时间戳
0xA00F805E	HI_ERR_VO_GRP_NOT_START	同步组未启动
0xA00F805F	HI_ERR_VO_GRP_NOT_STOP	同步组未停止
0xA00F8060	HI_ERR_VO_GRP_INVALID_FRMRATE	同步组无效帧率
0xA00F8061	HI_ERR_VO_GRP_CHN_HAS_REG	通道已经注册
0xA00F8062	HI_ERR_VO_GRP_CHN_NOT_REG	通道未注册
0xA00F8063	HI_ERR_VO_GRP_CHN_NOT_UNREG	通道未注销
0xA00F8064	HI_ERR_VO_GRP_BASE_NOT_CFG	基准时间戳未配置
0xA00F8065	HI_ERR_VO_DEV_HAS_ENABLED	视频输出设备已经使能



目 录

5 视频前处理.....	5-1
5.1 概述.....	5-1
5.2 重要概念.....	5-1
5.3 API 参考	5-2
5.4 数据类型.....	5-30
5.5 错误码.....	5-59



表格目录

表 5-1 各种 VPP 区域的对比	5-10
表 5-2 视频前处理 API 错误码	5-59



5 视频前处理

5.1 概述

视频前处理模块对输入或者是 VI 捕获的图像所作的处理都在编码前进行，主要提供设置和获取视频前处理配置、创建和销毁 VPP 区域、控制 VPP 区域、创建和等待缩放任务完成功能。

5.2 重要概念

- **Cover Region**
视频遮挡区域。
- **Overlay Region**
视频叠加区域，针对码流里打的叠加区域，即通常所说的码流 OSD。
- **ViOverlay Region**
VI 视频叠加区域，是在 VI 的原始视频图像叠加的“前端 OSD”，与 VI 绑定的 VO、VENC 等视频流中都会该叠加区域显示。只推荐应用于 PCI 板卡业务中。
- **CoverEx Region**
扩展视频遮挡区域。Cover Region 受限于 VI 硬件，每个通道最多有 4 个遮挡区域，这可能在某些场景下不足，因此用 TDE 的 fill 操作在 VI 的图像上添加遮挡区域，同时该遮挡区域支持对顶场、底场单独进行遮挡操作。
- **区域层次 Layer**
区域层次是针对遮挡区域、软叠加区域而言，范围是[0, 100]，0 表示最底层，100 表示最上层。
- **区域透明度**
区域透明度是针对叠加区域、软叠加区域而言，当前叠加区域对于其下一层图像的透明程度，范围是[0, 128]，0 表示完全透明，128 表示完全不透明。



5.3 API 参考

视频前处理模块对用户输入或 VI 捕获的图像所作的处理都在编码前进行。视频前处理模块主要提供设置和获取视频前处理配置、创建和销毁 VPP 区域、控制 VPP 区域、创建和等待图像缩放任务完成功能。

该功能模块提供以下 MPI：

- [HI_MPI_VPP_SetConf](#)：设置视频前处理配置。
- [HI_MPI_VPP_GetConf](#)：获取视频前处理配置。
- [HI_MPI_VPP_SetDsuFiltParam](#)：设置用户自定义的 DSU 缩放滤波参数。
- [HI_MPI_VPP_GetDsuFiltParam](#)：获取用户自定义的 DSU 滤波参数。
- [HI_MPI_VPP_CreateRegion](#)：创建 VPP 区域。
- [HI_MPI_VPP_DestroyRegion](#)：销毁 VPP 区域。
- [HI_MPI_VPP_ControlRegion](#)：控制 VPP 区域。
- [HI_MPI_VPP_CreateScaleTask](#)：创建一个图像缩放任务。
- [HI_MPI_VPP_WaitScaleTask](#)：等待一个图像缩放任务完成。

HI_MPI_VPP_SetConf

【描述】

设置视频前处理配置。

【语法】

```
HI_S32 HI_MPI_VPP_SetConf(VENC_GRP VeGroup, const VIDEO_PREPROC_CONF_S
*pstConf);
```

【参数】

参数名称	描述	输入/输出
VeGroup	编码通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入
pstConf	视频前处理属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_INVALID_CHNID	通道组号错误或无效区域句柄。
HI_ERR_VPP_NULL_PTR	空指针。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_ILLEGAL_PARAM	非法参数。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块（指各个 MPI 对应的内核模块）。

【需求】

- 头文件：hi_comm_vpp.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】

- 视频前处理的各项配置都有默认值，一般情况下不需要用户再进行设置，目前的默认配置为：彩色转灰色关闭、时域去噪打开、滤波类型为 FILTER_PARAM_TYPE_NORM、滤波系数为默认、缩放方式为采用保留底场方式缩放、编码输入图像的剪切功能关闭。
- 如果需要更改默认配置，应该先调用 HI_MPI_VPP_GetConf 接口获取配置，更改需要配置的部分选项后，再调用此接口进行设置。
- 滤波类型及系数在编码业务需要用到 DSU 模块进行缩放时使用。用户可指定使用某种类型的滤波参数，然后指定使用其下的某组水平/垂直滤波参数（或选择默认系数类型，SDK 内部会根据一定策略在该类型滤波参数中自动选择合适的水平/垂直滤波参数）。
- 该接口中滤波类型 enFilterType 仅支持设置为 FILTER_PARAM_TYPE_NORM、FILTER_PARAM_TYPE_USER1 或 FILTER_PARAM_TYPE_USER2，否则返回参数错误。
当设置 enFilterType 为 FILTER_PARAM_TYPE_NORM 时，可通过调节 enHFilter/enVFilterL/enVFilterC 选择不同截止频率的滤波参数达到不同的滤波效果
- 当用户对缩放效果有特定需求时，需先通过接口 HI_MPI_VPP_SetDsuFiltParam 设置用户自定义的滤波参数，然后在此指定 enFilterType 为相应的 FILTER_PARAM_TYPE_USERX，即可在 DSU 缩放时启动用户自定义滤波参数了。设置用户自定义滤波参数请参见接口 HI_MPI_VPP_SetDsuFiltParam 说明。
- 通道组创建之后，才能调用此接口，否则返回 HI_ERR_VPP_NOT_PERM。
- 此接口可动态调用，即在编码时也可以调用此接口。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```



```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_vpp.h"
#include "mpi_vpp.h"

HI_S32 VppSetAndGetConf(HI_VOID)
{
    HI_S32 s32Ret;
    VENC_GRP VeGroup = 0;
    VIDEO_PREPROC_CONF_S stConf;

    /* first get current vpp config */
    s32Ret = HI_MPI_VPP_GetConf(VeGroup, &stConf);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VPP_GetConf failed 0x%x!\n", s32Ret);
        return HI_FAILURE;
    }

    /* modify some config you need */
    stConf.bColorToGrey = HI_TRUE;
    stConf.bTemporalDenoise = HI_TRUE;
    stConf.enScaleMode = VPP_SCALE_MODE_USEBOTTOM;
    stConf.enFilter = VPP_SCALE_FILTER_DEFAULT;

    s32Ret = HI_MPI_VPP_SetConf(VeGroup, &stConf);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VPP_SetConf failed 0x%x!\n", s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```



【相关主题】

无。

HI_MPI_VPP_GetConf

【描述】

获取视频前处理配置。

【语法】

```
HI_S32 HI_MPI_VPP_GetConf(VENC_GRP VeGroup,  
VIDEO_PREPROC_CONF_S *pstConf);
```

【参数】

参数名称	描述	输入/输出
VeGroup	编码通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入
pstConf	视频前处理属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_INVALID_CHNID	通道组号错误或无效区域句柄。
HI_ERR_VPP_NULL_PTR	空指针。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_vpp.h、mpi_vpp.h
- 库文件：libmpi.a

**【注意】**

- 如果不设置视频前处理属性，则返回系统默认的前处理配置，不会返回失败。
- 视频前处理的默认配置为：彩色转灰色关闭、时域去噪打开、滤波系数为默认系数、缩放方式为采用保留底场方式缩放、编码输入图像的剪切功能关闭。
- 通道组创建之后，才能调用此接口，否则返回 [HI_ERR_VPP_NOT_PERM](#)。

【举例】

请参见 [HI_MPI_VPP_SetConf](#) 的举例。

【相关主题】

无。

HI_MPI_VPP_SetDsuFiltParam

【描述】

设置用户自定义的 DSU 缩放滤波参数。

【语法】

```
HI_S32 HI_MPI_VPP_SetDsuFiltParam(DSU_FILTER_PARAM_S *pstDsuFiltParam);
```

【参数】

参数名称	描述	输入/输出
pstDsuFiltParam	DSU 滤波参数属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_NULL_PTR	空指针。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_ILLEGAL_PARAM	非法参数。



接口返回值	含义
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块（指各个 MPI 对应的内核模块）。

【需求】

- 头文件：hi_comm.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】

- 若用户对编码/预览图像质量有特殊需求，SDK 支持用户设置自定义的 DSU 滤波参数，以达到实现定制图像质量或解决特定问题。自定义滤波参数必须符合 Hi3520/Hi3515 芯片的要求，一般由 SDK 根据特定客户需求专门定制。若用户设置的滤波参数不符合规范，会导致图像缩放异常。

一套完整的滤波参数包括水平方向和垂直方向的滤波参数。其中水平参数共 6 组，分别具有不同的滤波截止频率；每组又由 132 个数（HI_U8）组成，故一套完整的水平滤波参数包含 $6 \times 132 = 792$ 个数（单位 HI_U8）；垂直参数共 12 组，分别具有不同的滤波截止频率；每组又由 40 个数（HI_U8）组成，故一套完整的垂直滤波参数包含 $40 \times 12 = 480$ 个数（单位 HI_U8）。用户设置自定义滤波类型时，必须提供完整的一套滤波参数。

SDK 共支持 5 套滤波参数，不同套的滤波参数通过 pstDsuFiltParam 结构的滤波类型（enFiltType）指定，其中前 3 套为内置滤波类型，后 2 套为用户自定义类型。因不同套的滤波参数特性不同可能导致图像缩放的滤波效果不同。除非用户指定使用“自定义滤波类型”，否则内部自动选择使用某一类型的内置滤波参数进行 DSU 缩放。

- 一般情况下，用户不需要设置该接口，SDK 会自动选择内置的滤波参数进行 DSU 缩放，并满足通用场景。
- 使用方法：系统初始化后，若有需要，则用户首先设置自定义滤波参数(通过接口 HI_MPI_VPP_SetDsuFiltParam)；然后在预览或编码业务中可通过 VO 接口 (HI_MPI_VO_GetChnFilter)或 VPP 接口(HI_MPI_VPP_SetConf)指定某通道缩放时使用用户自定义滤波系数进行缩放。
- 该接口仅支持设置用户类型的滤波参数，即 DSU_FILTER_PARAM_S 的属性 enFiltType（其指定滤波参数的类型）应设置为 FILTER_PARAM_TYPE_USER1 或者 FILTER_PARAM_TYPE_USER2，否则返回参数错误。

【相关主题】

无。

HI_MPI_VPP_GetDsuFiltParam

【描述】

获取用户自定义的 DSU 滤波参数。

【语法】



```
HI_S32 HI_MPI_VPP_GetDsuFiltParam(DSU_FILTER_PARAM_S *pstDsuFiltParam);
```

【参数】

参数名称	描述	输入/输出
pstDsuFiltParam	DSU 滤波参数属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_NULL_PTR	空指针。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VPP_ILLEGAL_PARAM	非法参数。

【需求】

- 头文件：hi_comm.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】

- 该接口仅支持获取用户类型的滤波参数，即 DSU_FILTER_PARAM_S 的属性 enFiltType（其指定滤波参数的类型）应设置为 FILTER_PARAM_TYPE_USER1 或者 FILTER_PARAM_TYPE_USER2，否则返回参数错误。
- 系统初始化时，SDK 已经为 FILTER_PARAM_TYPE_USER1 类型内置了一套滤波参数，用户可通过获取接口查看该套系数；而 FILTER_PARAM_TYPE_USER2 类型的滤波参数未指定，需要用户设置后方可使用。

【相关主题】

无。



HI_MPI_VPP_CreateRegion

【描述】

创建 VPP 区域。

【语法】

```
HI_S32 HI_MPI_VPP_CreateRegion(const REGION_ATTR_S *pstRegion,  
REGION_HANDLE *pHandle);
```

【参数】

参数名称	描述	输入/输出
pstRegion	区域属性。	输入
pHandle	区域句柄指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_INVALID_DEVID	错误的设备号。
HI_ERR_VPP_INVALID_CHNID	通道组号错误或无效的区域句柄。
HI_ERR_VPP_NULL_PTR	空指针。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_NOMEM	分配内存失败。
HI_ERR_VPP_ILLEGAL_PARAM	非法参数。
HI_ERR_VPP_NOT_SUPPORT	不支持。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】



- 头文件：hi_comm_vpp.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】

- VPP 区域创建后，默认是隐藏的，必须主动调用显示接口才会显示。
- 视频遮挡区域是在 VI 的原始视频图像叠加的单色填充块，在与此 VI 通道绑定的编码通道组以及视频输出中都会有相应的视频遮挡显示；视频遮挡的位置和大小是相对于 VI 原始视频图像（即未做丢场和水平压缩前的输入图像）而言。
- 视频叠加区域是视频编码前叠加在视频上的“前端 OSD”，在视频编码码流中显示，但在视频预览中不会显示；视频叠加区域针对的是整个编码通道组，由于同一编码通道组中的小码流由大码流缩放得到，因此小码流中显示的叠加区域的位置和大小也是等比例缩放。
- VI 视频叠加区域是在 VI 的原始视频图像叠加的“前端 OSD”，由于是加上 VI 的视频数据中，所以在与此 VI 通道绑定的 VO、VENC 等视频流中都会有叠加区域显示。VI 视频叠加区域的大小和位置是相对于其叠加的 VI 原始视频图像的，因此如果 VI 采集大小有变化，VI 视频叠加区域的大小和位置也需要相应变化。
- VI 叠加区域和 VI 遮挡区域的不同在于：VI 遮挡区域为纯色块，而 VI 叠加区域可显示用户指定设置的位图，且可实现透明效果。
- VI 视频叠加区域只建议用于 PCI 板卡等 TDE 性能较充裕的应用场景，用于其它场景时可能出现视频丢帧甚至异常现象。
- 扩展视频遮挡区域的原理和使用方法与视频遮挡区域基本一致，区别在于扩展视频遮挡区域支持顶底场分别叠加遮挡块。
- 表 5-1 为遮挡区域与叠加区域的对比。

表5-1 各种 VPP 区域的对比

属性	视频遮挡 COVER	扩展视频遮挡 COVEREX	视频叠加 OVERLAY	VI 视频叠加
相对通道	VI 通道	VI 通道	Group 通道	VI 通道
区域	起始点坐标必须大于等于 0，高宽必须大于 0，最大为 4095 × 4095。	起始点坐标必须大于等于 0，高宽必须大于 0，最大为 4095 × 4095。且起始点和宽高必须为偶数。	起始点坐标必须大于等于 0 且为偶数，X 坐标必须为 8 的倍数。高、宽都必须为偶数，以像素为单位，最大为 2047 × 2047。	起始点坐标必须大于等于 0 且为偶数；高、宽都必须大于 16 像素的偶数。超出 VI 源图像范围的部分将会被剪裁。
公共区域	支持公共区域，即如果设置为公共区域，所有通道使用其相同属性。	支持公共区域，即如果设置为公共区域，所有通道使用其相同属性。	支持公共区域，即如果设置为公共区域，所有通道使用其相同属性。	不支持公共区域。
数量限制	包括公共区域在内，每个通道最多 4 个。	包括公共区域在内，每个通道最多 16 个。	包括公共区域在内，每个通道最多 4 个。	每个通道最多 8 个。



属性	视频遮挡 COVER	扩展视频遮挡 COVEREX	视频叠加 OVERLAY	VI 视频叠加
像素格式	不支持设置像素格式。	不支持设置像素格式。	即为背景色和填充位图的像素格式，支持 α RGB1555、 α RGB4444 两种格式，同一通道的各区域像素格式必须一致。	即区域内数据的像素格式，只支持 α RGB1555 格式。决定了区域的背景色和位图数据的像素格式。
颜色	视频遮挡区域仅支持单色填充。颜色值采用 RGB888 格式，取值范围为 $0x00000000 \sim 0x00FFFFFF$ ，低 24 位有效，低 8 位为 R 值，中间 8 位为 G 值，高 8 位为 B 值。	视频遮挡区域仅支持单色填充。颜色值采用 RGB888 格式，取值范围为 $0x00000000 \sim 0x00FFFFFF$ ，低 24 位有效，低 8 位为 R 值，中间 8 位为 G 值，高 8 位为 B 值。	对于视频叠加，即为背景色。两种格式的取值范围为 $0x00000000 \sim 0x0000FFFF$ ，低 16 位有效。对 α RGB1555 格式，最高 1 位为 α 位，次高 5 位为 R 值，中间 5 位为 G 值，低 5 位为 B 值；对 α RGB4444 格式，最高 4 位为 α 位，次高 4 位为 R 值，中间 4 位为 G 值，低 4 位为 B 值。	即未被位图数据覆盖到的背景色。 α RGB1555 格式时，与 OVERLAY 的背景色数据格式表示方式相同。
透明度	不支持设置透明度。	不支持设置透明度。	可设置前景透明度和背景透明度。透明度取值范围为 $0 \sim 128$ ，值越小表示越透明。0 表示完全透明，128 表示完全不透明。单色填充叠加区域时，透明度取背景透明度； α RGB1555 格式位图填充时，alpha 位为 1 的点取前景透明度，alpha 位为 0 的点取背景透明度； α RGB4444 格式位图填充时，透明度由位图像素点的前 4 位 α 值决定。	可设置全局透明度（GlobalAlpha），取值范围为 $0 \sim 255$ ，值越小表示越透明，0 表示完全透明，255 表示完全不透明。 α RGB1555 格式时支持 Alpha0/Alpha1。如果 α RGB1555 格式 Alpha 扩展功能未开启，区域内 alpha 位为 0 的像素点为完全透明，alpha 位为 1 的像素点则完全不透明；如果扩展功能开启，alpha 位为 0 和 1 的像素点的透明度分别为配置的指定透明度值（取值范围同全局透明度）。
层次	支持范围为 $0 \sim 100$ 的层次设置，通道内层次不同的区域间可以重叠。	支持范围为 $0 \sim 100$ 的层次设置，通道内层次不同的区域间可以重叠。	不支持设置层次，且通道内各区域间不能有位置重叠。	支持范围为 $0 \sim 100$ 的层次设置，层次值高的区域将会叠加在层次值低的区域上面；层次相同的区域不可以互相重叠，但层次不同的区域可以互相重叠。



属性	视频遮挡 COVER	扩展视频遮挡 COVEREX	视频叠加 OVERLAY	VI 视频叠加
位图填充	不支持填充位图。	不支持填充位图。	可填充 ARGB1555 或者 ARGB4444 格式的位图。	支持位图填充，支持的位图的像素格式取决于区域的像素格式（目前只支持 αRGB1555 格式）。

【举例】

本举例共有两个用例，一个遮挡区域用例，一个叠加区域用例。

```
/*cover region sample*/
HI_S32 VppCtrlCoverRegion(HI_VOID)
{
    HI_S32 i;
    HI_S32 s32Ret;
    HI_S32 s32Cnt = 0;
    REGION_CTRL_CODE_E enCtrl;
    REGION_CTRL_PARAM_U unParam;
    REGION_ATTR_E stRgnAttr;
    REGION_HANDLE handle[5];

    stRgnAttr.enType = COVER_REGION;
    stRgnAttr.unAttr.stCover.bIsPublic = HI_FALSE;
    stRgnAttr.unAttr.stCover.u32Color = 0;
    stRgnAttr.unAttr.stCover.u32Layer = 1;
    stRgnAttr.unAttr.stCover.stRect.s32X = 100;
    stRgnAttr.unAttr.stCover.stRect.s32Y = 200;
    stRgnAttr.unAttr.stCover.stRect.u32Height = 50;
    stRgnAttr.unAttr.stCover.stRect.u32Width = 50;
    stRgnAttr.unAttr.stCover.ViChn = VICHNID;
    stRgnAttr.unAttr.stCover.ViDevId = VIDEVID;

    s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[0]);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VPP_CreateRegion err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    stRgnAttr.enType = COVER_REGION;
    stRgnAttr.unAttr.stCover.bIsPublic = HI_FALSE;
```



```
stRgnAttr.unAttr.stCover.u32Color = 0x0000ff00;
stRgnAttr.unAttr.stCover.u32Layer = 2;
stRgnAttr.unAttr.stCover.stRect.s32X = 200;
stRgnAttr.unAttr.stCover.stRect.s32Y = 200;
stRgnAttr.unAttr.stCover.stRect.u32Height = 50;
stRgnAttr.unAttr.stCover.stRect.u32Width = 50;
stRgnAttr.unAttr.stCover.ViChn = VICHNID;
stRgnAttr.unAttr.stCover.ViDevId = VIDEVID;

s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[1]);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateRegion err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

/*public cover region*/
stRgnAttr.enType = COVER_REGION;
stRgnAttr.unAttr.stCover.bIsPublic = HI_TRUE;
stRgnAttr.unAttr.stCover.u32Color = 0x00ff0000;
stRgnAttr.unAttr.stCover.u32Layer = 3;
stRgnAttr.unAttr.stCover.stRect.s32X = 300;
stRgnAttr.unAttr.stCover.stRect.s32Y = 200;
stRgnAttr.unAttr.stCover.stRect.u32Height = 50;
stRgnAttr.unAttr.stCover.stRect.u32Width = 50;

s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[2]);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateRegion err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

stRgnAttr.enType = COVER_REGION;
stRgnAttr.unAttr.stCover.bIsPublic = HI_FALSE;
stRgnAttr.unAttr.stCover.u32Color = 0x00ff;
stRgnAttr.unAttr.stCover.u32Layer = 4;
stRgnAttr.unAttr.stCover.stRect.s32X = 400;
stRgnAttr.unAttr.stCover.stRect.s32Y = 200;
stRgnAttr.unAttr.stCover.stRect.u32Height = 50;
stRgnAttr.unAttr.stCover.stRect.u32Width = 50;
stRgnAttr.unAttr.stCover.ViChn = VICHNID;
stRgnAttr.unAttr.stCover.ViDevId = VIDEVID;
```



```
s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[3]);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateRegion err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

/*show region*/
enCtrl = REGION_SHOW;
s32Ret = HI_MPI_VPP_ControlRegion(handle[0],enCtrl,&unParam);
if(s32Ret != HI_SUCCESS)
{
    printf("show faild 0x%x!!!\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VPP_ControlRegion(handle[1],enCtrl,&unParam);
if(s32Ret != HI_SUCCESS)
{
    printf("show faild 0x%x!!!\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VPP_ControlRegion(handle[2],enCtrl,&unParam);
if(s32Ret != HI_SUCCESS)
{
    printf("show faild 0x%x!!!\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VPP_ControlRegion(handle[3],enCtrl,&unParam);
if(s32Ret != HI_SUCCESS)
{
    printf("show faild 0x%x!!!\n",s32Ret);
    return HI_FAILURE;
}

/*ctrl the region*/
while(1)
{
    sleep(1);
    s32Cnt++;

    if(s32Cnt <= 10)
    {
```



```
        /*change color*/
        enCtrl = REGION_SET_COLOR;

        if(0 == s32Cnt % 2)
        {
            unParam.u32Color = 0;
        }
        else
        {
            unParam.u32Color = 0xff;
        }

        s32Ret = HI_MPI_VPP_ControlRegion(handle[0], enCtrl, &unParam);

        if(s32Ret != HI_SUCCESS)
        {
            printf("set region position faild 0x%x!!!\n",s32Ret);
            return HI_FAILURE;
        }
    }
    else if(s32Cnt <= 20)
    {
        /*change position*/
        enCtrl = REGION_SET_POSTION;
        unParam.stPoint.s32X = 200 + ((s32Cnt - 10) * 2);
        unParam.stPoint.s32Y = 200 + ((s32Cnt - 10) * 5);

        s32Ret = HI_MPI_VPP_ControlRegion(handle[1] ,enCtrl,&unParam);

        if(s32Ret != HI_SUCCESS)
        {
            printf("set region position faild 0x%x!!!\n",s32Ret);
            return HI_FAILURE;
        }
    }
    else if(s32Cnt <= 30)
    {
        /*change size*/
        enCtrl = REGION_SET_SIZE;
        unParam.stDimension.s32Height = 50 + ((s32Cnt - 20) * 8);
        unParam.stDimension.s32Width = 50 + ((s32Cnt - 20) * 8);

        s32Ret = HI_MPI_VPP_ControlRegion(handle[2] ,enCtrl,&unParam);
```



```
        if(s32Ret != HI_SUCCESS)
        {
            printf("set region position failed 0x%x!!!\n",s32Ret);
            return HI_FAILURE;
        }
    }
    else if(s32Cnt <= 40)
    {
        /*change layer*/
        enCtrl = REGION_SET_LAYER;
        unParam.u32Layer = s32Cnt;

        if(0 == s32Cnt % 2)
        {
            handle[4] = handle[3];
        }
        else
        {
            handle[4] = handle[2];
        }

        s32Ret = HI_MPI_VPP_ControlRegion(handle[4] ,enCtrl, &unParam);

        if(s32Ret != HI_SUCCESS)
        {
            printf("set region position failed 0x%x!!!\n",s32Ret);
            return HI_FAILURE;
        }
    }
    else
    {
        for(i=0; i<4; i++)
        {
            s32Ret = HI_MPI_VPP_DestroyRegion(handle[i]);
            if(s32Ret != HI_SUCCESS)
            {
                printf("HI_MPI_VPP_DestroyRegion err 0x%x!\n",s32Ret);
                return HI_FAILURE;
            }
        }

        break;
    }
}
```




```
    }

    return HI_SUCCESS;
}

/*overlay region sample*/
HI_S32 VppCtrlOverlayRegion(HI_VOID)
{
    HI_S32 i = 0;
    HI_S32 s32Ret;
    HI_S32 s32Cnt = 0;
    char *pFilename;

    REGION_ATTR_S stRgnAttr;
    REGION_CTRL_CODE_E enCtrl;
    REGION_CTRL_PARAM_U unParam;
    REGION_HANDLE handle[4];

    stRgnAttr.enType = OVERLAY_REGION;
    stRgnAttr.unAttr.stOverlay.bPublic = HI_FALSE;
    stRgnAttr.unAttr.stOverlay.enPixelFormat = PIXEL_FORMAT_RGB_1555;
    stRgnAttr.unAttr.stOverlay.stRect.s32X= 104;
    stRgnAttr.unAttr.stOverlay.stRect.s32Y= 100;
    stRgnAttr.unAttr.stOverlay.stRect.u32Width = 180;
    stRgnAttr.unAttr.stOverlay.stRect.u32Height = 144;
    stRgnAttr.unAttr.stOverlay.u32BgAlpha = 128;
    stRgnAttr.unAttr.stOverlay.u32FgAlpha = 128;
    stRgnAttr.unAttr.stOverlay.u32BgColor = 0;
    stRgnAttr.unAttr.stOverlay.VeGroup = 0;
    s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[0]);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VPP_CreateRegion err 0x%x!\n",s32Ret);
        return HI_FAILURE;
    }

    stRgnAttr.enType = OVERLAY_REGION;
    stRgnAttr.unAttr.stOverlay.bPublic = HI_FALSE;
    stRgnAttr.unAttr.stOverlay.enPixelFormat = PIXEL_FORMAT_RGB_1555;
    stRgnAttr.unAttr.stOverlay.stRect.s32X= 304;
    stRgnAttr.unAttr.stOverlay.stRect.s32Y= 100;
    stRgnAttr.unAttr.stOverlay.stRect.u32Width = 180;
    stRgnAttr.unAttr.stOverlay.stRect.u32Height = 144;
    stRgnAttr.unAttr.stOverlay.u32BgAlpha = 128;
```



```
stRgnAttr.unAttr.stOverlay.u32FgAlpha = 128;
stRgnAttr.unAttr.stOverlay.u32BgColor = 0x1f;
stRgnAttr.unAttr.stOverlay.VeGroup = 0;
s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[1]);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateRegion err 0x%x!\n",s32Ret);
    return HI_FAILURE;
}

stRgnAttr.enType = OVERLAY_REGION;
stRgnAttr.unAttr.stOverlay.bPublic = HI_FALSE;
stRgnAttr.unAttr.stOverlay.enPixelFormat = PIXEL_FORMAT_RGB_1555;
stRgnAttr.unAttr.stOverlay.stRect.s32X= 104;
stRgnAttr.unAttr.stOverlay.stRect.s32Y= 300;
stRgnAttr.unAttr.stOverlay.stRect.u32Width = 48;
stRgnAttr.unAttr.stOverlay.stRect.u32Height = 48;
stRgnAttr.unAttr.stOverlay.u32BgAlpha = 70;
stRgnAttr.unAttr.stOverlay.u32FgAlpha = 70;
stRgnAttr.unAttr.stOverlay.u32BgColor = 0x3e0;
stRgnAttr.unAttr.stOverlay.VeGroup = 0;
s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[2]);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateRegion err 0x%x!\n",s32Ret);
    return HI_FAILURE;
}

/*public overlay region*/
stRgnAttr.enType = OVERLAY_REGION;
stRgnAttr.unAttr.stOverlay.bPublic = HI_TRUE;
stRgnAttr.unAttr.stOverlay.enPixelFormat = PIXEL_FORMAT_RGB_1555;
stRgnAttr.unAttr.stOverlay.stRect.s32X= 304;
stRgnAttr.unAttr.stOverlay.stRect.s32Y= 300;
stRgnAttr.unAttr.stOverlay.stRect.u32Width = 48;
stRgnAttr.unAttr.stOverlay.stRect.u32Height = 48;
stRgnAttr.unAttr.stOverlay.u32BgAlpha = 30;
stRgnAttr.unAttr.stOverlay.u32FgAlpha = 30;
stRgnAttr.unAttr.stOverlay.u32BgColor = 0x7c00;

s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[3]);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateRegion err 0x%x!\n",s32Ret);
```



```
        return HI_FAILURE;
    }

    /*show all region*/
    enCtrl = REGION_SHOW;

    for(i=0; i<4; i++)
    {
        s32Ret = HI_MPI_VPP_ControlRegion(handle[i], enCtrl, &unParam);

        if(s32Ret != HI_SUCCESS)
        {
            printf("show faild 0x%x!\n",s32Ret);
            return HI_FAILURE;
        }
    }

    /*ctrl the region*/
    while(1)
    {
        sleep(1);
        s32Cnt++;

        if(s32Cnt <= 10)
        {
            /*change bitmap*/
            if(10 == s32Cnt)
            {
                memset(&unParam, 0, sizeof(REGION_CTRL_PARAM_U));
                pFilename = "mm.bmp";

                /*change bitmap to ARGB1555*/
                SampleLoadBmp(pFilename, &unParam);

                enCtrl = REGION_SET_BITMAP;

                s32Ret = HI_MPI_VPP_ControlRegion(handle[0], enCtrl,
&unParam);

                if(s32Ret != HI_SUCCESS)
                {
                    printf("set region bitmap faild 0x%x!\n",s32Ret);
                    return HI_FAILURE;
                }
            }
        }
    }
}
```



```
        free(unParam.stBitmap.pData);
        unParam.stBitmap.pData = NULL;
    }

}

else if(s32Cnt <= 20)
{
    /*change bitmap*/
    if(20 == s32Cnt)
    {
        pFilename = "huawei.bmp";
        memset(&unParam, 0, sizeof(REGION_CTRL_PARAM_U));
        SampleLoadBmp(pFilename, &unParam);
        enCtrl = REGION_SET_BITMAP;

        s32Ret = HI_MPI_VPP_ControlRegion(handle[1], enCtrl,
&unParam);

        if(s32Ret != HI_SUCCESS)
        {
            printf("REGION_SET_BITMAP 0x%x!\n", s32Ret);
            return HI_FAILURE;
        }

        free(unParam.stBitmap.pData);
        unParam.stBitmap.pData = NULL;
    }

}

else if(s32Cnt <= 30)
{
    /*change position*/
    enCtrl = REGION_SET_POSTION;
    unParam.stPoint.s32X = 300 + (s32Cnt - 20)*8;
    unParam.stPoint.s32Y = 300 + (s32Cnt - 20)*4;

    s32Ret = HI_MPI_VPP_ControlRegion(handle[3], enCtrl, &unParam);

    if(s32Ret != HI_SUCCESS)
    {
        printf("REGION_SET_POSTION faild 0x%x!\n", s32Ret);
        return HI_FAILURE;
    }
}
```



```
    }
}
else if(s32Cnt <=40)
{
    enCtrl = REGION_SET_ALPHA0;
    unParam.u32Alpha = 128 - (s32Cnt - 30)*8;

    s32Ret = HI_MPI_VPP_ControlRegion(handle[0], enCtrl, &unParam);

    if(s32Ret != HI_SUCCESS)
    {
        printf("REGION_SET_ALPHA0 0xxx!\n",s32Ret);
        return HI_FAILURE;
    }

    enCtrl = REGION_SET_ALPHA1;
    unParam.u32Alpha = 128 - (s32Cnt - 30)*8;
    s32Ret = HI_MPI_VPP_ControlRegion(handle[0], enCtrl, &unParam);

    if(s32Ret != HI_SUCCESS)
    {
        printf("REGION_SET_ALPHA1 faild 0xxx!\n",s32Ret);
        return HI_FAILURE;
    }
}
else
{
    break;
}
}

for(i=0; i<4; i++)
{
    s32Ret = HI_MPI_VPP_DestroyRegion(handle[i]);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VPP_DestroyRegion err 0xxx!\n",s32Ret);
        return HI_FAILURE;
    }
}

return HI_SUCCESS;
}
```

**【相关主题】**

无。

HI_MPI_VPP_DestroyRegion**【描述】**

销毁 VPP 区域。

【语法】

```
HI_S32 HI_MPI_VPP_DestroyRegion(REGION_HANDLE Handle);
```

【参数】

参数名称	描述	输入/输出
Handle	区域句柄。 取值范围：创建区域时返回的有效句柄。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_INVALID_CHNID	通道组号错误或无效区域句柄。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_UNEXIST	区域不存在。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_vpp.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】



无。

【举例】

请参见 [HI_MPI_VPP_CreateRegion](#) 的举例。

【相关主题】

无。

HI_MPI_VPP_ControlRegion

【描述】

控制 VPP 区域。

【语法】

```
HI_S32 HI_MPI_VPP_ControlRegion(REGION_HANDLE Handle,  
REGION_CTRL_CODE_E enCtrl, REGION_CTRL_PARAM_U *punParam);
```

【参数】

参数名称	描述	输入/输出
Handle	区域句柄。 取值范围：创建区域时返回的有效句柄。	输入
enCtrl	控制命令。	输入
punParam	控制参数指针。	输入/输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_INVALID_CHNID	通道组号错误或无效区域句柄。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_NOT_SUPPORT	不支持。



接口返回值	含义
HI_ERR_VPP_UNEXIST	区域不存在。
HI_ERR_VPP_ILLEGAL_PARAM	非法参数。
HI_ERR_VPP_NOT_PERM	操作不允许。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_vpp.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】

- 对于遮挡区域，可进行的控制包括显示隐藏区域、设置位置、设置尺寸、设置层次、设置颜色、获取该区域的属性，其他操作不允许。
- 对于叠加区域，可进行的控制包括显示隐藏区域、设置位置、设置透明度、设置颜色、设置填充位图、获取该区域的属性，其他操作不允许。
- 对于 VI 叠加区域，可进行的控制包括显示隐藏区域、设置位置、设置透明度、设置层次、设置颜色、设置填充位图、获取该区域的属性，其他操作不允许。
- 叠加区域填充位图时，位图的格式在创建区域时就已经确定，若位图的格式为 α RGB1555，其中 alpha 位为 0 时，表示该像素使用背景透明度；alpha 为 1 时，表示该像素使用前景透明度。若位图格式为 α RGB4444，则高 4 位表示该像素的透明度。
- 改变叠加区域的位图时，区域原有位图或背景色会被新的位图覆盖。若位图大于区域大小时，会根据创建区域时的大小按照由左到右、由上到下进行裁减。
- 获取所有区域属性时，参数 Handle 将被忽略，并通过 punParam 返回所有区域的属性。

**说明**

Hi3520 暂不支持获取所有区域属性。

- 对于不同的控制，参数的输入或输出类型也是不同的。在获取单个区域属性和所有区域属性时，punParam 是输出参数；在其他的控制操作时都是输入参数。具体的参数类型对应关系，请参见 [REGION_CTRL_PARAM_U](#)。

【举例】

请参见 [HI_MPI_VPP_CreateRegion](#) 的举例。

【相关主题】

无。

HI_MPI_VPP_CreateScaleTask

【描述】

创建一个图像缩放任务。



【语法】

```
HI_S32 HI_MPI_VPP_CreateScaleTask(PIC_SCALE_TASK_S *pstTask,  
VPP_SCALE_CONF_S *pstConf);
```

【参数】

参数名称	描述	输入/输出
pstTask	缩放任务指针。	输入
pstConf	缩放开关和系数指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_NULL_PTR	空指针。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_ILLEGAL_PARAM	非法参数。
HI_ERR_VPP_NOBUF	无缩放任务缓存。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_vpp.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】

- 可以进行 8 倍以内的任意比例的缩小和放大。
- 原始图像和目标图像的格式支持 semi-planar YUV422 和 semi-planar YUV420 两种，原始图像和目标图像的格式可以不同。
- 无论缩小或放大，原始图像和目标图像的最大宽高均为 4096 像素。
- 原始图像与目标图像帧场格式应相同。



- Hi3520/Hi3515 暂不支持该接口。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_vpp.h"
#include "mpi_vpp.h"
#include "mpi_vb.h"
#include "hi_comm_vi.h"
#include "mpi_vi.h"

HI_S32 VPP_MST_007_001(HI_VOID)
{
    HI_S32 s32Ret;
    HI_U32 u32TaskId= 0;
    VIDEO_FRAME_INFO_S stFrame;
    VIDEO_FRAME_S stVideoOut;
    PIC_SCALE_TASK_S stTask;
    VPP_SCALE_CONF_S stConf;
    HI_U32 u32BlockFlag = HI_IO_BLOCK;
    HI_U32 u32Timeout = 0;

    VB_POOL VbPool;
    VB_BLK VbBlk;
    HI_U32 u32Addr;

    /*create the scale task ,must be notice how to use the VB*/
    VbPool = HI_MPI_VB_CreatePool(384*288*2, 1);
    if (VB_INVALID_POOLID == VbPool)
    {
        printf("HI_MPI_VB_CreatePool err!");
        return HI_FAILURE;
    }
```



```
}

/* get blk from a pool */
VbBlk = HI_MPI_VB_GetBlock(VbPool, 0);
if (VB_INVALID_HANDLE == VbBlk)
{
    printf("HI_MPI_VB_GetBlock err!");
    HI_MPI_VB_DestroyPool(VbPool);
    return HI_FAILURE;
}

/* blk handle to physaddr */
u32Addr = HI_MPI_VB_Handle2PhysAddr(VbBlk);
if (0 == u32Addr)
{
    printf("HI_MPI_VB_Handle2PhysAddr err!");
    HI_MPI_VB_DestroyPool(VbPool);
    return HI_FAILURE;
}

stVideoOut.enPixelFormat = PIXEL_FORMAT_YUV_SEMIPLANAR_422;
stVideoOut.u32Width = 352;
stVideoOut.u32Height = 288;
stVideoOut.u32Field = VIDEO_FIELD_FRAME;

/*light physics addr and stride*/
stVideoOut.u32PhyAddr[0] = u32Addr;
stVideoOut.u32Stride[0] = 384;

/*chroma physics addr and stride*/
stVideoOut.u32PhyAddr[1] = u32Addr + 384*288;
stVideoOut.u32Stride[1] = 384;

stTask.stDesPic.stVFrame = stVideoOut;
stTask.stDesPic.u32PoolId = VbPool;

do
{
    /*get D1 frame from VI*/
    if(HI_MPI_VI_GetFrame(0, 0, &stFrame) < 0)
    {
        printf("HI_MPI_VI_GetFrame err!\n");
        break;
    }
}
```



```
memcpy(&stTask.stSrcPic,&stFrame,sizeof(VIDEO_FRAME_INFO_S));

stTask.u32TaskId = u32TaskId;
stTask.stDesPic.stVFrame.u32Field = stFrame.stVFrame.u32Field;

stConf.bColorToGrey = HI_FALSE;
stConf.bDeInterlace = HI_FALSE;
stConf.bTemporalDenoise = HI_FALSE;
stConf.enChoice = VPP_SCALE;
stConf.enCE = VPP_CE_DISABLE;
stConf.enLumaStr = VPP_LUMA_STR_DISABLE;
stConf.enFilter = VPP_SCALE_FILTER_DEFAULT;
stConf.enSpatialDenoise = VPP_DENOISE_ONLYEDAGE;

s32Ret = HI_MPI_VPP_CreateScaleTask(&stTask,&stConf);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateScaleTask err 0x%x!\n",s32Ret);
    break;
}

s32Ret = HI_MPI_VPP_WaitScaleTask(&stTask, u32BlockFlag,
u32Timeout);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_WaitScaleTask err 0x%x!\n",s32Ret);
    break;
}

u32TaskId ++;
printf("task id is %d\n",stTask.u32TaskId);

}while(u32TaskId < 0xff);

s32Ret = HI_MPI_VB_ReleaseBlock(VbBlk);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VB_ReleaseBlock err 0x%x!\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VB_DestroyPool(VbPool);
if(s32Ret != HI_SUCCESS)
```



```
{  
    printf("HI_MPI_VB_DestroyPool err 0x%x!\n",s32Ret);  
    return HI_FAILURE;  
}  
  
return HI_SUCCESS;  
}
```

【相关主题】

无。

HI_MPI_VPP_WaitScaleTask

【描述】

等待一个图像缩放任务完成。

【语法】

```
HI_S32 HI_MPI_VPP_WaitScaleTask(PIC_SCALE_TASK_S *pstTask, HI_U32  
u32BlockFlag, HI_U32 u32Timeout);
```

【参数】

参数名称	描述	输入/输出
pstTask	缩放任务结构指针。	输出
u32BlockFlag	阻塞标志。 取值范围： <ul style="list-style-type: none">• HI_IO_BLOCK：阻塞。• HI_IO_NOBLOCK：非阻塞。	输入
u32Timeout	等待时间。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_VPP_NULL_PTR	空指针。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_BUF_EMPTY	无缩放任务完成。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：[hi_comm_venc.h](#)、[mpi_venc.h](#)
- 库文件：[libmpi.a](#)

【注意】

- 阻塞情况下，等待缩放任务完成还需输入等待的时间，单位为 ms，0 表示无限期等待。
- 如果连续创建多个任务，则需要多次调用等待接口，每次返回一个完成的任务。
- Hi3520/Hi3515 暂不支持该接口。

【举例】

请参见 [HI_MPI_VPP_CreateScaleTask](#) 的举例。

【相关主题】

无。

5.4 数据类型

视频前处理相关数据类型、数据结构定义如下：

- [MAX_COVER_NUM](#)：定义同一个 VI 通道的最大遮挡区域的个数。
- [MAX_OVERLAY_NUM](#)：定义同一个 GROUP 通道组的最大叠加区域的个数。
- [MAX_VIOVERLAY_NUM](#)：定义同一个 VI 通道的最大 VI 视频叠加区域的个数。
- [MAX_COVEREX_REGION_NUM](#)：定义同一个 VI 通道的最大扩展遮挡区域的个数。
- [VI_COVER_REGION](#)：定义所有的遮挡区域的个数最大值。
- [VENC_OVERLAY_REGION](#)：定义所有的叠加区域的个数最大值。
- [VPP_CLIPATTR_NUM](#)：定义剪裁属性个数。
- [VPP_SCALE_DENOISE_CHOICE_E](#)：定义缩放去噪选择。
- [DSU_HSCALE_FILTER_E](#)：定义水平缩放系数。
- [DSU_VSCALE_FILTER_E](#)：定义垂直缩放系数。
- [VPP_CLIP_ATTR_S](#)：定义图像剪裁属性。



- [DSU_HFILTER_PARAM_NUM](#): DSU 水平滤波参数个数。
- [DSU_VFILTER_PARAM_NUM](#): DSU 垂直滤波参数个数。
- [DSU_HSCALE_FILTER_E](#): DSU 滤波参数类型。
- [VPP_DENOISE_E](#): 定义去噪系数。
- [VPP_CE_E](#): 定义色彩增强开关。
- [VPP_LUMA_STR_E](#): 定义对比度拉伸开关。
- [VPP_SCALE_MODE_E](#): 定义缩放方式。
- [DSU_FILTER_PARAM_S](#): 用户自定义 DSU 滤波参数属性。
- [VIDEO_PREPROC_CONF_S](#): 定义视频前处理配置的数据结构体。
- [VPP_SCALE_CONF_S](#): 定义视频缩放任务配置的数据结构体。
- [REGION_HANDLE](#): 定义区域句柄。
- [REGION_TYPE_E](#): 定义区域类型。
- [COVER_ATTR_S](#): 定义遮挡区域属性结构体。
- [OVERLAY_ATTR_S](#): 定义叠加区域属性结构体。
- [VI_OVERLAY_ATTR_S](#): 定义 VI 叠加区域属性结构体。
- [COVEREX_ATTR_S](#): 定义扩展遮挡区域属性结构体。
- [REGION_ATTR_U](#): 定义区域属性联合体。
- [REGION_ATTR_S](#): 定义区域类型结构体。
- [REGION_CTRL_CODE_E](#): 定义区域操作命令。
- [COVER_S](#): 定义所有遮挡区域结构体。
- [OVERLAY_S](#): 定义所有叠加区域结构体。
- [COVEREX_S](#): 定义所有扩展遮挡区域结构体。
- [REGION_CTRL_PARAM_U](#): 定义控制区域参数。
- [PIC_SCALE_TASK_S](#): 定义缩放任务结构体。

MAX_COVER_NUM

【说明】

定义同一个 VI 通道的最大遮挡区域的个数。

【定义】

```
#define MAX_COVER_NUM 4
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。



MAX_OVERLAY_NUM

【说明】

定义同一个 GROUP 通道组的最大叠加区域的个数。

【定义】

```
#define MAX_OVERLAY_NUM    4;
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

MAX_VIOVERLAY_NUM

【说明】

定义同一个 VI 通道的最大 VI 视频叠加区域的个数。

【定义】

```
#define MAX_VIOVERLAY_NUM    8;
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

MAX_COVEREX_REGION_NUM

【说明】

定义同一个 VI 通道的最大扩展遮挡区域的个数。

【定义】

```
#define MAX_COVEREX_REGION_NUM 16
```

【成员】

无。

【注意事项】



无。

【相关数据类型及接口】

无。

VI_COVER_REGION

【说明】

定义所有的遮挡区域的个数最大值。

【定义】

```
#define VI_COVER_REGION (VIU_MAX_DEV_NUM * VIU_MAX_CHN_NUM_PER_DEV  
*MAX_COVER_NUM);
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

[MAX_COVER_NUM](#)

VENC_OVERLAY_REGION

【说明】

定义所有的叠加区域的个数最大值。

【定义】

```
#define VENC_OVERLAY_REGION (VENC_MAX_CHN_NUM*MAX_OVERLAY_NUM);
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

[MAX_OVERLAY_NUM](#)

VPP_CLIPATTR_NUM

【说明】

定义剪裁属性的个数

【定义】



```
#define VPP_CLIPATTR_NUM 2
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

- [VIDEO_PREPROC_CONF_S](#)
- [VPP_CLIP_ATTR_S](#)

VPP_SCALE_DENOISE_CHOICE_E

【说明】

定义缩放去噪选择。

【定义】

```
typedef enum hiVPP_SCALE_DENOISE_CHOICE_E
{
    VPP_SCALE,
    VPP_DENOISE,
    VPP_BUTT,
}VPP_SCALE_DENOISE_CHOICE_E;
```

【成员】

成员名称	描述
VPP_SCALE	缩放。
VPP_DENOISE	空域去噪。

【注意事项】

无。

【相关数据类型及接口】

[VPP_SCALE_CONF_S](#)

DSU_HSCALE_FILTER_E

【说明】

定义水平缩放系数。

【定义】



```
typedef enum hiDSU_HSCALE_FILTER_E
{
    DSU_HSCALE_FILTER_DEFAULT = 0,
    DSU_HSCALE_FILTER_C_65M,
    DSU_HSCALE_FILTER_CG_56M,
    DSU_HSCALE_FILTER_LC_45M,
    DSU_HSCALE_FILTER_CG_3M,
    DSU_HSCALE_FILTER_CG_2M,
    DSU_HSCALE_FILTER_CG_1M,
    DSU_HSCALE_FILTER_BUTT
}DSU_HSCALE_FILTER_E;
```

【成员】

成员名称	描述
DSU_HSCALE_FILTER_DEFAULT	默认缩放系数模式。
VPP_HSCALE_FILTER_n M	缩放系数。nM 数值越小，滤波后的图像越模糊。

【注意事项】

系统默认采用 DSU_HSCALE_FILTER_DEFAULT，即内部自动选择最优滤波系数。一般不需要使用者配置滤波系数。

【相关数据类型及接口】

- [VIDEO_PREPROC_CONF_S](#)
- [VPP_SCALE_CONF_S](#)

DSU_VSCALE_FILTER_E

【说明】

定义垂直缩放系数。

【定义】

```
typedef enum hiDSU_VSCALE_FILTER_E
{
    DSU_VSCALE_FILTER_DEFAULT = 0,
    DSU_VSCALE_FILTER_S_6M,
    DSU_VSCALE_FILTER_S_5M,
    DSU_VSCALE_FILTER_S_4M,
    DSU_VSCALE_FILTER_S_38M,
    DSU_VSCALE_FILTER_S_37M,
    DSU_VSCALE_FILTER_S_36M,
    DSU_VSCALE_FILTER_S_25M,
```



```
DSU_VSCALE_FILTER_S_2M,  
DSU_VSCALE_FILTER_S_15M,  
DSU_VSCALE_FILTER_S_12M,  
DSU_VSCALE_FILTER_S_11M,  
DSU_VSCALE_FILTER_S_1M,  
DSU_VSCALE_FILTER_BUTT  
}DSU_VSCALE_FILTER_E;
```

【成员】

成员名称	描述
DSU_VSCALE_FILTER_DEFAULT	默认缩放系数模式。
VPP_VSCALE_FILTER_n M	缩放系数。nM 数值越小，滤波后的图像越模糊。

【注意事项】

系统默认采用 DSU_VSCALE_FILTER_DEFAULT，即内部自动选择最优滤波系数。一般不需要使用者配置滤波系数。

【相关数据类型及接口】

- [VIDEO_PREPROC_CONF_S](#)
- [VPP_SCALE_CONF_S](#)

VPP_CLIP_ATTR_S

【说明】

图像裁减属性。

【定义】

```
typedef struct hiVPP_CLIP_ATTR_S  
{  
  
    HI_U32 u32SrcWidth;  
  
    HI_U32 u32SrcHeight;  
  
    HI_U32 u32ClipMode;  
  
    RECT_S stClipRect;  
  
} VPP_CLIP_ATTR_S;
```

【成员】

成员名称	描述
u32SrcWidth	拟剪裁的原始图像宽度。



成员名称	描述
u32SrcHeight	拟剪裁的原始图像高度。
u32ClipMode	剪裁模式。 取值范围：[1, 4]。 1：针对顶场进行剪裁。 2：针对底场进行剪裁。 3：针对两场进行剪裁。 4：针对整帧进行剪裁。
stClipRect	剪裁区域设置。

【注意事项】

- u32SrcWidth、u32Srcheight 用于锁定要剪裁的图像，只有高宽均匹配的输入图像才会被剪裁。
- 剪裁区域的起始横坐标必须是 8 的整数倍。

【相关数据类型及接口】

- [VIDEO_PREPROC_CONF_S](#)
- [RECT_S](#)
- [VIDEO_FRAME_S](#)
- [HI_MPI_VPP_SetConf](#)

DSU_HFILTER_PARAM_NUM

【说明】

DSU 水平滤波参数个数。

【定义】

```
#define DSU_HFILTER_PARAM_NUM 792
```

【成员】

无。

【注意事项】

一套完整的滤波参数包含水平和垂直 2 个方向的滤波参数。其中水平参数共 6 组，分别具有不同的滤波截止频率；每组又由 132 个数（HI_U8）组成，故一套完整的水平滤波参数包含 $6 \times 132 = 792$ 个数（单位 HI_U8）。

【相关数据类型及接口】

- [DSU_HFILTER_PARAM_NUM](#)
- [DSU_FILTER_PARAM_S](#)



DSU_VFILTER_PARAM_NUM

【说明】

DSU 垂直滤波参数个数。

【定义】

```
#define DSU_VFILTER_PARAM_NUM    480
```

【成员】

一套完整的滤波参数包含水平和垂直 2 个方向的滤波参数。其中垂直参数共 12 组，分别具有不同的滤波截止频率；每组又由 40 个数（HI_U8）组成，故一套完整的垂直滤波参数包含 $40 \times 12 = 480$ 个数（单位 HI_U8）。

【注意事项】

无。

【相关数据类型及接口】

- [DSU_HFILTER_PARAM_NUM](#)
- [DSU_FILTER_PARAM_S](#)

DSU_HSCALE_FILTER_E

【说明】

DSU 滤波参数类型。

【定义】

```
typedef enum hiDSU_FILTER_PARAM_TYPE
{
    FILTER_PARAM_TYPE_NORM = 0,
    FILTER_PARAM_TYPE_EX,
    FILTER_PARAM_TYPE_EX2,
    FILTER_PARAM_TYPE_USER1,
    FILTER_PARAM_TYPE_USER2,
    FILTER_PARAM_TYPE_BUTT
}DSU_FILTER_PARAM_TYPE;
```

【成员】

成员名称	描述
FILTER_PARAM_TYPE_NORM	通用滤波系数类型
FILTER_PARAM_TYPE_EX	扩展的滤波系数类型 1
FILTER_PARAM_TYPE_EX2	扩展的滤波系数类型 2
FILTER_PARAM_TYPE_USER1	用户自定义滤波系数 1



成员名称	描述
FILTER_PARAM_TYPE_USER2	用户自定义滤波系数 2

【注意事项】

- 一套完整的滤波参数包括水平方向和垂直方向的滤波参数，DSU 缩放时需要被指定使用哪一套滤波参数进行滤波。一种滤波参数类型即对应一套完整的滤波参数。
- 一般情况，若用户不特别指定使用“用户自定义滤波系数类型”（即 FILTER_PARAM_TYPE_USER1 或 FILTER_PARAM_TYPE_USER2），则 SDK 内部自动选择一套滤波系数进行滤波。若用户有特定需求，则可指定使用 FILTER_PARAM_TYPE_USERX 类型滤波类型。

【相关数据类型及接口】

[DSU_FILTER_PARAM_S](#)

VPP_DENOISE_E

【说明】

定义去噪系数。

【定义】

```
typedef enum hiVPP_DENOISE_E
{
    VPP_DENOISE_ONLYEDAGE = 0,
    VPP_DENOISE_LOWNOISE,
    VPP_DENOISE_MIDNOISE,
    VPP_DENOISE_HIGHNOISE,
    VPP_DENOISE_VERYHIGHNOISE,
    VPP_DENOISE_BUTT,
}VPP_DENOISE_E;
```

【成员】

成员名称	描述
VPP_DENOISE_ONLYEDAGE	边缘去噪。
VPP_DENOISE_LOWNOISE	低去噪。
VPP_DENOISE_MIDNOISE	中去噪。
VPP_DENOISE_HIGHNOISE	高去噪。
VPP_DENOISE_VERYHIGHNOISE	极高去噪。



【注意事项】

无。

【相关数据类型及接口】

无。

VPP_CE_E

【说明】

定义色彩增强开关。

【定义】

```
typedef enum hiVPP_CE_E
{
    VPP_CE_DISABLE = 0,
    VPP_CE_ENABLE,
    VPP_CE_BUTT,
}VPP_CE_E;
```

【成员】

成员名称	描述
VPP_CE_DISABLE	色彩增强关。
VPP_CE_ENABLE	色彩增强开。

【注意事项】

无。

【相关数据类型及接口】

[VPP_SCALE_CONF_S](#)

VPP_LUMA_STR_E

【说明】

定义对比度拉伸开关。

【定义】

```
typedef enum hiVPP_LUMA_STR_E
{
    VPP_LUMA_STR_DISABLE = 0,
    VPP_LUMA_STR_ENABLE,
    VPP_LUMA_STR_BUTT,
}VPP_LUMA_STR_E;
```




【成员】

成员名称	描述
VPP_LUMA_STR_DISABLE	对比度拉伸关。
VPP_LUMA_STR_ENABLE	对比度拉伸开。

【注意事项】

无。

【相关数据类型及接口】

- [VIDEO_PREPROC_CONF_S](#)
- [VPP_SCALE_CONF_S](#)

VPP_SCALE_MODE_E

【说明】

定义缩放方式。

【定义】

```
typedef enum hiVPP_SCALE_MODE_E
{
    VPP_SCALE_MODE_USEBOTTOM = 0,    /*use bottom scale*/
    VPP_SCALE_MODE_USETOP,           /*use top scale*/
    VPP_SCALE_MODE_DIRECT,           /*scale directly*/
    VPP_SCALE_MODE_BUTT,
}VPP_SCALE_MODE_E;
```

【成员】

成员名称	描述
VPP_SCALE_MODE_USETOP	取顶场缩放。
VPP_SCALE_MODE_USEBOTTOM	取底场缩放。
VPP_SCALE_MODE_DIRECT	直接缩放。

【注意事项】

- 丢场缩放可以有效的提高系统性能，当不满足丢场缩放时，必须进行直接缩放。类似这种问题可以推广到各种不同的场景，比如 VI 输入 D1 或编码 QVGA 之类大小不严格满足 1/2 和 1/4 的情形。
- 缩放方式只对大码流（JPEG 抓拍和 MPEG4）有效，小码流无效。



- 如果输入图像和编码图像的高度比大于 1/2 则自动使用 SCALE_MODE_DIRECT 方式，其他设置无效。

【相关数据类型及接口】

[VIDEO_PREPROC_CONF_S](#)

DSU_FILTER_PARAM_S

【说明】

用户自定义 DSU 滤波参数属性。

【定义】

```
typedef struct hiDSU_FILTER_PARAM_S
{
    DSU_FILTER_PARAM_TYPE enFiltType;
    HI_U8 au8HParamTable[DSU_HFILTER_PARAM_NUM];
    HI_U8 au8VParamTable[DSU_VFILTER_PARAM_NUM];
}DSU_FILTER_PARAM_S;
```

【成员】

成员名称	描述
enFiltType	滤波参数类型。
au8HParamTable	水平滤波参数数组。
au8VParamTable	垂直滤波参数数组。

【注意事项】

一种类型（enFiltType）的滤波参数饱含水平滤波参数 DSU_HFILTER_PARAM_NUM 个，垂直滤波参数 DSU_VFILTER_PARAM_NUM 个。SDK 共支持 5 套滤波参数，通过用户指定或内部自动选择使用某一套滤波参数进行 DSU 缩放，不同套的滤波参数特性不同可能导致图像缩放的滤波效果不同。

【相关数据类型及接口】

- [HI_MPI_VPP_SetDsuFiltParam](#)
- [HI_MPI_VPP_GetDsuFiltParam](#)

VIDEO_PREPROC_CONF_S

【说明】

定义视频前处理配置的数据结构体。

【定义】

```
typedef struct hiVIDEO_PREPROC_CONF_S
```



```
{  
    HI_BOOL          bTemporalDenoise;  
    HI_S32  s32TfBase;  
    HI_S32  s32TfRate;  
    HI_S32  s32ChrTFDelta;  
    HI_BOOL          bColorToGrey;  
    HI_S32  s32SrcFrmRate;  
    HI_S32  s32TarFrmRate;  
    VPP_SCALE_MODE_E enScaleMode;  
    DSU_FILTER_PARAM_TYPE enFilterType;  
    DSU_HSCALE_FILTER_E enHFilter;  
    DSU_VSCALE_FILTER_E enVFilterL;  
    DSU_VSCALE_FILTER_E enVFilterC;  
    VPP_CLIP_ATTR_S  stClipAttr[VPP_CLIPATTR_NUM];  
}VIDEO_PREPROC_CONF_S;
```

【成员】

成员名称	描述
bTemporalDenoise	时域去噪开关。 取值范围： <ul style="list-style-type: none">• HI_TRUE: 开启时域去噪。• HI_FALSE: 关闭时域去噪。 默认值为 HI_TRUE。
s32TfBase	时域滤波系数为常数的阈值，即若 MAD 值小于此处的配置值，则为常数。 取值范围：[0,255]。默认值为 9。 取值越大，滤波强度越大。
s32TfRate	时域滤波系数相对于 MAD 值的变化率。 取值范围：[0,255]。默认值为 12。 取值越大，滤波强度越大。
s32ChrTFDelta	时域滤波色度滤波系数与亮度的差值。 取值范围：[-127,127]。默认值为 0。
bColorToGrey	彩转灰开关。 取值范围： <ul style="list-style-type: none">• HI_TRUE: 打开彩转灰。• HI_FALSE: 关闭彩转灰。 默认值为 HI_TRUE。



成员名称	描述
s32SrcFrmRate	Group 的原始帧率，与 VI 采集的帧率相等。 默认值为-1，即不做帧率控制。
s32TarFrmRate	Group 的目标帧率。 默认值为-1，即不做帧率控制。
enScaleMode	缩放方式。 默认值为 VPP_SCALE_MODE_USEBOTTOM。
enFilterType	缩放滤波参数类型。 默认值为 FILTER_PARAM_TYPE_NORM。
enHFilter	水平缩放系数。 默认值为 FILTER_PARAM_TYPE_NORM。
enVFilterL	亮度分量垂直方向的缩放系数。 默认值为 DSU_VSCALE_FILTER_DEFAULT。
enVFilterC	色度分量垂直方向的缩放系数。 默认值为 DSU_VSCALE_FILTER_DEFAULT。
stClipAttr	图像裁剪属性。 其中的 u32SrcWidth 和 u32SrcHeight 默认值为 0，即不开启图像裁剪功能。

【注意事项】

- 提供了两组 vpp 剪裁属性设置。当一个通道组的输入图像尺寸变化时，可对它们设置不同的剪裁属性。
- 对剪裁后的图像进行编码，因此，通道组的编码图像大小，应设置为剪裁后的图像大小。
- 滤波类型（enFilterType）和滤波系数（enHFilter/ enVFilterL/ enVFilterC）的关系：一种滤波类型即对应一套滤波参数，SDK 有多套滤波参数供选择，每一套滤波参数都包含水平 6 组和垂直 12 组滤波参数，各组的滤波参数主要是滤波截止频率不一致。用户需指定使用哪一类型的滤波参数（enFilterType），及其下的哪一组水平滤波系数和哪一组垂直滤波系数（enHFilter/ enVFilterL/ enVFilterC）。

【相关数据类型及接口】

- [VPP_SCALE_MODE_E](#)
- [VPP_CLIP_ATTR_S](#)
- [VPP_CLIPATTR_NUM](#)
- [HI_MPI_VPP_SetConf](#)



VPP_SCALE_CONF_S

【说明】

定义视频缩放任务配置的数据结构体。

【定义】

```
typedef struct hiVPP_SCALE_CONF_S
{
    HI_BOOL                bTemporalDenoise;
    HI_BOOL                bDeInterlace;
    HI_BOOL                bColorToGrey;
    VPP_SCALE_DENOISE_CHOICE_E enChoice;
    VPP_CE_E               enCE;
    VPP_LUMA_STR_E         enLumaStr;
    DSU_HSCALE_FILTER_E   enHFilter;
    DSU_VSCALE_FILTER_E   enVFilterL;
    DSU_VSCALE_FILTER_E   enVFilterC;
    VPP_DENOISE_E          enSpatialDenoise;
}VPP_SCALE_CONF_S;
```

【成员】

成员名称	描述
bTemporalDenoise	时域去噪开关。 取值范围： <ul style="list-style-type: none">• HI_TRUE：打开时域去噪。• HI_FALSE：关闭时域去噪。
bDeInterlace	DeInterlace 开关。 取值范围： <ul style="list-style-type: none">• HI_TRUE：打开 Deinterlace。• HI_FALSE：关闭 Deinterlace。
bColorToGrey	彩转灰开关。 取值范围： <ul style="list-style-type: none">• HI_TRUE：打开彩转灰。• HI_FALSE：关闭彩转灰。
enChoice	缩放空域去噪选择。
enCE	色彩增强开关。
enLumaStr	对比度拉伸开关。
enHFilter	水平缩放系数。



成员名称	描述
enVFilterL	亮度分量垂直方向的缩放系数。
enVFilterC	色度分量垂直方向的缩放系数。
enSpatialDenoise	空域去噪系数。

【注意事项】

无。

【相关数据类型及接口】

- [VPP_CLIP_ATTR_S](#)
- [VPP_CLIPATTR_NUM](#)
- [VPP_CE_E](#)
- [VPP_LUMA_STR_E](#)
- [HI_MPI_VPP_CreateScaleTask](#)

**说明**

现版本不支持缩放任务配置中的时域去噪、Deinterlace、彩转灰、空域去噪、色彩增强、对比度拉伸；仅支持缩放。

REGION_HANDLE

【说明】

定义区域句柄。

【定义】

```
typedef HI_U32 REGION_HANDLE;
```

【成员】

成员名称	描述
REGION_HANDLE	区域句柄。

【注意事项】

无。

【相关数据类型及接口】

- [HI_MPI_VPP_CreateRegion](#)
- [HI_MPI_VPP_DestroyRegion](#)
- [HI_MPI_VPP_ControlRegion](#)



REGION_TYPE_E

【说明】

定义区域类型。

【定义】

```
typedef enum hiREGION_TYPE_E
{
    COVER_REGION = 0,
    OVERLAY_REGION,
    VIOVERLAY_REGION,
    REGION_BUTT
} REGION_TYPE_E;
```

【成员】

成员名称	描述
COVER_REGION	遮挡区域。
OVERLAY_REGION	叠加区域。
VIOVERLAY_REGION	VI 叠加区域。

【注意事项】

无。

【相关数据类型及接口】

[REGION_ATTR_S](#)

COVER_ATTR_S

【说明】

定义遮挡区域属性结构体。

【定义】

```
typedef struct hiCOVER_ATTR_S
{
    VI_DEV  ViDevId;
    VI_CHN  ViChn;
    HI_BOOL bIsPublic;
    HI_U32  u32Layer;
    RECT_S  stRect;
    HI_U32  u32Color;
} COVER_ATTR_S;
```



【成员】

成员名称	描述
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。 静态属性。
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。 静态属性。
bIsPublic	是否公共区域。 取值范围： • HI_TRUE：是公共区域。 • HI_FALSE：不是公共区域。 静态属性。
u32Layer	层次。 取值范围：[0, 100]。 动态属性。
stRect	区域的位置和大小。 取值范围：遮挡区域起始点坐标大于等于 0，高宽大于 0，最大为 4095 × 4095，高宽最大值都为 4095。 动态属性。
u32Color	背景色。 取值范围：[0, 0xFFFFFFFF]。 动态属性。

【注意事项】

无。

【相关数据类型及接口】

[REGION_ATTR_U](#)

OVERLAY_ATTR_S

【说明】

定义叠加区域属性结构体。

【定义】

```
typedef struct hiOVERLAY_ATTR_S
```




```
{  
    VENC_GRP          VeGroup;  
    HI_BOOL           bPublic;  
    RECT_S            stRect;  
    PIXEL_FORMAT_E    enPixelFormat;  
    HI_U32             u32FgAlpha;  
    HI_U32             u32BgAlpha;  
    HI_U32             u32BgColor;  
} OVERLAY_ATTR_S;
```

【成员】

成员名称	描述
VeGroup	编码通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。 静态属性。
bPublic	公共区域标识。 取值范围： <ul style="list-style-type: none">• HI_TRUE：公共区域。• HI_FALSE：非公共区域。 静态属性。
stRect	区域的位置和高宽。 取值范围：叠加区域的起始点的座标必须大于 0 且为偶数，长宽也都必须为偶数，以像素为单位，最大为 2047×2047，高宽最大值都为 2047。而且对于叠加区域，起始点的 X 坐标必须为 8 的倍数。 起始位置可以动态改变。 高宽不可以改变。
enPixelFormat	像素的格式。 取值范围： <ul style="list-style-type: none">• PIXEL_FORMAT_RGB_1555。• PIXEL_FORMAT_RGB_4444。 静态属性。
u32FgAlpha	前景 Alpha 值。 取值范围：[0, 128]。 动态属性。
u32BgAlpha	背景 Alpha 值。 取值范围：[0, 128]。 动态属性。



成员名称	描述
u32BgColor	背景色。 取值范围：[0, 0x7FFF]。 动态属性。

【注意事项】

无。

【相关数据类型及接口】

[REGION_ATTR_U](#)

VI_OVERLAY_ATTR_S

【说明】

定义 VI 叠加区域属性结构体。

【定义】

```
typedef struct hiVI_OVERLAY_ATTR_S
{
    VI_DEV      ViDevId;
    VI_CHN      ViChn;
    HI_BOOL     bIsPublic;

    RECT_S      stRect;
    PIXEL_FORMAT_E enPixelFormat;
    HI_U32      u32Layer;
    HI_U32      u32BgColor;

    HI_U8        u8GlobalAlpha;
    HI_BOOL     bAlphaExt1555;
    HI_U8        u8Alpha0;
    HI_U8        u8Alpha1;
} VI_OVERLAY_ATTR_S;
```

【成员】

成员名称	描述
ViDevId	VI 设备号。
ViChn	VI 通道号。



成员名称	描述
bPublic	公共区域标识。 目前不支持配置为 TRUE。
stRect	VI 叠加区域相对 VI 图像的位置坐标和高宽。 取值范围：VI 叠加区域的起始点的坐标值必须大于 0 且为偶数，长宽都必须为大于 16 的偶数，以像素为单位。起始位置可以通过控制接口进行动态改变，高宽不可以动态改变。
enPixelFormat	像素的格式。 取值范围： • PIXEL_FORMAT_RGB_1555。
u32Layer	区域层次。 取值范围：[0, 100]。值越大，层次越高。
u32BgColor	背景色。 数据格式取决于 enPixelFormat 项。
u8GlobalAlpha	全局透明度。 取值范围：[0, 255]。取值越小，越透明。
bAlphaExt1555	是否扩展 ARGB1555 格式的透明度功能。 取值范围：TRUE or FALSE。
u8Alpha0	bAlphaExt1555 为 TRUE 时，Alpha 位 0 的像素点的透明度。 取值范围：[0, 255]。取值越小，越透明。
u8Alpha1	bAlphaExt1555 为 TRUE 时，Alpha 位 1 的像素点的透明度。 取值范围：[0, 255]。取值越小，越透明。

【注意事项】

无。

【相关数据类型及接口】

[REGION_ATTR_U](#)

COVEREX_ATTR_S

【说明】

定义扩展遮挡区域属性结构体。

【定义】

```
typedef struct hiCOVER_ATTR_S
{
```



```

VI_DEV  ViDevId;
VI_CHN  ViChn;
HI_BOOL bIsPublic;
VIDEO_FIELD_E enField;
HI_U32  u32Layer;
RECT_S  stRect;
HI_U32  u32Color;
} COVER_ATTR_S;

```

【成员】

成员名称	描述
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。 静态属性。
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。 静态属性。
bIsPublic	是否公共区域。 取值范围： <ul style="list-style-type: none"> • HI_TRUE：是公共区域。 • HI_FALSE：不是公共区域。 静态属性。
enField	遮挡的帧场属性。 取值范围：[VIDEO_FIELD_TOP, VIDEO_FIELD_BUTT)。 静态属性。 只有在 VI 图像是两场隔行格式时，此项配置才有意义。
u32Layer	层次。 取值范围：[0, 100]。 动态属性。
stRect	区域的位置和大小。 取值范围：遮挡区域起始点坐标大于等于 0，高宽大于 0，最大为 4095 × 4095，高宽最大值都为 4095。 动态属性。
u32Color	背景色。 取值范围：[0, 0xFFFFFF]。 动态属性。



【注意事项】

无。

【相关数据类型及接口】

[REGION_ATTR_U](#)

REGION_ATTR_U

【说明】

定义区域属性联合体。

【定义】

```
typedef union hiREGION_ATTR_U
{
    COVER_ATTR_S        stCover;
    OVERLAY_ATTR_S      stOverlay;
    VI_OVERLAY_ATTR_S   stViOverlay;
    COVEREX_ATTR_S      stCoverEx;
}REGION_ATTR_U;
```

【成员】

成员名称	描述
stCover	遮挡区域属性。
stOverlay	叠加区域属性。
stViOverlay	VI 叠加区域属性。
stCoverEx	扩展遮挡区域属性。

【注意事项】

无。

【相关数据类型及接口】

- [COVER_ATTR_S](#)
- [OVERLAY_ATTR_S](#)
- [VI_COVER_REGION](#)
- [COVEREX_ATTR_S](#)

REGION_ATTR_S

【说明】

定义区域类型结构体。

**【定义】**

```
typedef struct hiREGION_ATTR_S
{
    REGION_TYPE_E    enType;
    REGION_ATTR_U    unAttr;
}REGION_ATTR_S;
```

【成员】

成员名称	描述
enType	区域类型。
unAttr	区域属性。

【注意事项】

无。

【相关数据类型及接口】

- [REGION_TYPE_E](#)
- [REGION_ATTR_U](#)
- [HI_MPI_VPP_CreateRegion](#)

REGION_CTRL_CODE_E

【说明】

定义区域操作命令。

【定义】

```
typedef enum hiREGION_CTRL_CODE_E
{
    REGION_SHOW = 0,
    REGION_HIDE,
    REGION_SET_POSTION,
    REGION_SET_COLOR,
    REGION_SET_LAYER,
    REGION_SET_SIZE,
    REGION_SET_ALPHA0,
    REGION_SET_ALPHA1,
    REGION_SET_GLOBAL_ALPHA,
    REGION_SET_BITMAP,
    REGION_GET_SIGNLE_ATTR,
    REGION_GET_ALL_COVER_ATTR,
    REGION_GET_ALL_OVERLAY_ATTR,
```



```
REGION_GET_ALL_SOFT_OVERLAY_ATTR,  
}REGION_CTRL_CODE_E;
```

【成员】

成员名称	描述
REGION_SHOW	显示区域。
REGION_HIDE	隐藏区域。
REGION_SET_POSTION	改变区域位置。
REGION_SET_COLOR	改变区域背景色。
REGION_SET_LAYER	改变区域层次。
REGION_SET_SIZE	改变区域高宽。
REGION_SET_ALPHA0	改变区域背景 Alpha 值（只针对像素格式为 ARGB1555）。
REGION_SET_ALPHA1	改变区域前景 Alpha 值（只针对像素格式为 ARGB1555）。
REGION_SET_GLOBAL_ALPHA	改变区域全局 Alpha 值。
REGION_SET_BITMAP	填充区域位图。
REGION_GET_SIGNLE_ATTR	获取单个区域属性。
REGION_GET_ALL_COVER_ATTR	获取所有遮挡区域属性。 暂不支持。
REGION_GET_ALL_OVERLAY_ATTR	获取所有叠加区域属性。 暂不支持。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VPP_ControlRegion](#)

COVER_S

【说明】

定义所有遮挡区域结构体。

【定义】

```
typedef struct hiCOVER_S  
{
```



```
HI_U32          u32CoverNum;
REGION_HANDLE   aCoverHandles[VI_COVER_REGION];
COVER_ATTR_S    astAttr[VI_COVER_REGION];
}COVER_S;
```

【成员】

成员名称	描述
u32CoverNum	遮挡区域的个数。
aCoverHandles[VI_COVER_REGION]	遮挡区域的句柄。
astAttr[VI_COVER_REGION]	遮挡区域的属性。

【注意事项】

无。

【相关数据类型及接口】

- [REGION_HANDLE](#)
- [COVER_ATTR_S](#)

OVERLAY_S

【说明】

定义所有叠加区域结构体。

【定义】

```
typedef struct hiOVERLAY_S
{
    HI_U32          u32OverlayNum;
    REGION_HANDLE   aOverlayHandles[VENC_OVERLAY_REGION];
    OVERLAY_ATTR_S  astAttr[VENC_OVERLAY_REGION];
}OVERLAY_S;
```

【成员】

成员名称	描述
u32OverlayNum	叠加区域的个数。
aOverlayHandles[VENC_OVERLAY_REGION]	叠加区域的句柄。
astAttr[VENC_OVERLAY_REGION]	叠加区域的属性。

【注意事项】



无。

【相关数据类型及接口】

- [REGION_HANDLE](#)
- [COVER_ATTR_S](#)

COVEREX_S

【说明】

定义所有扩展遮挡区域结构体。

【定义】

```
typedef struct hiCOVEREX_S
{
    HI_U32 u32CoverExNum;
    REGION_HANDLE aCoverExHandles[VI_COVEREX_REGION];
    COVEREX_ATTR_S astAttr[VI_COVEREX_REGION];
} COVEREX_S;
```

【成员】

成员名称	描述
u32CoverExNum	扩展遮挡区域的个数。
aCoverExHandles[VI_COVEREX_REGION]	扩展遮挡区域的句柄。
astAttr[VI_COVEREX_REGION]	扩展遮挡区域的属性。

【注意事项】

无。

【相关数据类型及接口】

- [REGION_HANDLE](#)
- [COVEREX_ATTR_S](#)

REGION_CTRL_PARAM_U

【说明】

定义控制区域参数联合体。

【定义】

```
typedef union hiREGION_CTRL_PARAMETER_U
{
    HI_U32 u32Layer;
    HI_U32 u32Alpha;
    HI_U32 u32Color;
```



```
POINT_S      stPoint;  
DIMENSION_S  stDimension;  
BITMAP_S     stBitmap;  
REGION_ATTR_S stRegionAttr;  
COVER_S      stCovers;  
OVERLAY_S    stOverlays;  
}REGION_CTRL_PARAM_U;
```

【成员】

成员名称	描述
u32Layer	区域层次。 取值范围：[0, 100]。 动态属性。
u32Alpha	区域 Alpha 值（只针对像素格式为 ARGB1555）。 取值范围：[0, 128]。 动态属性。
u32Color	背景色。 取值范围：请参见 COVER_ATTR_S 和 OVERLAY_ATTR_S 的相应取值范围。 动态属性。
stPoint	位置。 取值范围：请参见 COVER_ATTR_S 和 OVERLAY_ATTR_S 的相应取值范围。 动态属性。
stDimension	高宽。 取值范围：请参见 COVER_ATTR_S 和 OVERLAY_ATTR_S 的相应取值范围。 动态属性。
stBitmap	位图结构体。
stRegionAttr	区域属性。
stCovers	所有遮挡区域。
stOverlays	所有叠加区域。

【注意事项】

无。

【相关数据类型及接口】



HI_MPI_VPP_ControlRegion

PIC_SCALE_TASK_S

【说明】

定义缩放任务结构体。

【定义】

```
typedef struct hiPIC_SCALE_TASK_S
{
    HI_U32          u32TaskId; /* used to identify a task */
    VIDEO_FRAME_INFO_S stSrcPic; /* source picture */
    VIDEO_FRAME_INFO_S stDesPic; /* destination picture */
}PIC_SCALE_TASK_S;
```

【成员】

成员名称	描述
u32TaskId	任务 ID。
stSrcPic	源图像。
stDesPic	目标图像。

【注意事项】

无。

【相关数据类型及接口】

- [HI_MPI_VPP_CreateScaleTask](#)
- [HI_MPI_VPP_WaitScaleTask](#)

5.5 错误码

视频前处理 API 错误码如表 5-2 所示。

表5-2 视频前处理 API 错误码

错误代码	宏定义	描述
0xA0078001	HI_ERR_VPP_INVALID_DEVID	设备 ID 超出合法范围
0xA0078002	HI_ERR_VPP_INVALID_CHNID	通道组号错误或无效区域句柄
0xA0078003	HI_ERR_VPP_ILLEGAL_PARAM	参数超出合法范围



错误代码	宏定义	描述
0xA0078004	HI_ERR_VPP_EXIST	重复创建已存在的设备、通道或资源
0xA0078005	HI_ERR_VPP_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源
0xA0078006	HI_ERR_VPP_NULL_PTR	函数参数中有空指针
0xA0078007	HI_ERR_VPP_NOT_CONFIG	模块没有配置
0xA0078008	HI_ERR_VPP_NOT_SUPPORT	不支持的参数或者功能
0xA0078009	HI_ERR_VPP_NOT_PERM	该操作不允许，如试图修改静态配置参数
0xA007800C	HI_ERR_VPP_NOMEM	分配内存失败，如系统内存不足
0xA007800D	HI_ERR_VPP_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA007800E	HI_ERR_VPP_BUF_EMPTY	缓冲区中无数据
0xA007800F	HI_ERR_VPP_BUF_FULL	缓冲区中数据满
0xA0078010	HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块
0xA0078011	HI_ERR_VPP_BADADDR	地址非法
0xA0078012	HI_ERR_VPP_BUSY	系统忙



目 录

6 视频编码.....	6-1
6.1 概述.....	6-1
6.2 重要概念.....	6-1
6.3 API 参考	6-1
6.4 数据类型.....	6-57
6.5 错误码.....	6-86



表格目录

表 6-1 编码通道的部分属性的约束.....	6-11
表 6-2 视频编码 API 错误码	6-86



6 视频编码

6.1 概述

VENC 模块完成各个协议编码，协调 MD、VPP 相关模块的管理、同步和控制，配合软件调度和硬件共同完成视频编码相关功能

6.2 重要概念

- 主次码流
主次码流是指硬件逻辑单元启动一次同时产生的 2 路码流，即 1 路主码流和 1 路次码流。主码流和次码流可以为不同的编码协议，但其宽高比例都必须满足 1:1、1:2 或 1:4，次码流不能单独存在（必须和 1 路主码流在同一个通道组中）。
- 双码流
双码流是指硬件逻辑单元启动 2 次分时产生的 2 个码流，即 2 路主码流。双码流可以为不同的编码协议，双码流之间的大小比例没有约束关系。
- 通道组
通道组是指芯片能够同时处理的编码通道的集合，相当于一个容器。一个通道组最多可同时包含 1 路主码流（H.264/MJPEG）、1 路次码流（H.264/MJPEG），或者仅包含 1 路 JPEG 抓拍（即 JPEG 抓拍时，不允许包含任何其他通道），或者 1 路 MPEG4 编码通道。



注意

目前尚不支持 MPEG4 编码。

6.3 API 参考

视频编码功能实际包含 VENC（视频编码）和 GROUP（通道组管理）两个重要的部分，主要提供视频编码通道组的创建和销毁、通道组 GROUP 与视频输入通道的绑定



和解绑定、视频编码通道的创建和销毁、注册和反注册到通道组、开启和停止接收图像、设置和获取编码通道属性、获取和释放码流、设置和获取数字水印属性、启用和禁用数字水印、视频编码通道属性的设置和查询等功能。

该功能模块提供以下 MPI：

- [HI_MPI_VENC_CreateGroup](#)：创建编码通道组。
- [HI_MPI_VENC_DestroyGroup](#)：销毁编码通道组。
- [HI_MPI_VENC_BindInput](#)：绑定 VI 到通道组。
- [HI_MPI_VENC_UnbindInput](#)：解绑定 VI 到通道组。
- [HI_MPI_VENC_CreateChn](#)：创建编码通道。
- [HI_MPI_VENC_DestroyChn](#)：销毁编码通道。
- [HI_MPI_VENC_RegisterChn](#)：注册编码通道到通道组。
- [HI_MPI_VENC_UnRegisterChn](#)：反注册编码通道到通道组。
- [HI_MPI_VENC_StartRecvPic](#)：开启编码通道接收输入图像。
- [HI_MPI_VENC_StopRecvPic](#)：停止编码通道接收输入图像。
- [HI_MPI_VENC_Query](#)：查询编码通道状态。
- [HI_MPI_VENC_SetChnAttr](#)：设置编码通道的属性。
- [HI_MPI_VENC_GetChnAttr](#)：获取编码通道的属性。
- [HI_MPI_VENC_GetStream](#)：获取编码码流。
- [HI_MPI_VENC_ReleaseStream](#)：释放码流缓存。
- [HI_MPI_VENC_RequestIDR](#)：请求 I 帧。
- [HI_MPI_VENC_InsertUserData](#)：插入用户数据。
- [HI_MPI_VENC_GetCapability](#)：获取视频编码能力集。
- [HI_MPI_VENC_SendFrame](#)：支持用户发送原始图像进行编码。
- [HI_MPI_VENC_SetWmAttr](#)：设置编码数字水印的属性。
- [HI_MPI_VENC_GetWmAttr](#)：获取编码数字水印的属性。
- [HI_MPI_VENC_EnableWm](#)：启用编码数字水印。
- [HI_MPI_VENC_DisableWm](#)：禁用编码数字水印。
- [HI_MPI_VENC_SetMaxStreamCnt](#)：设置码流缓存帧数。
- [HI_MPI_VENC_GetMaxStreamCnt](#)：获取码流缓存帧数。
- [HI_MPI_VENC_SetH264eRcPara](#)：设置 H.264 编码的码率控制参数。
- [HI_MPI_VENC_GetH264eRcPara](#)：获取 H.264 编码的码率控制参数。
- [HI_MPI_VENC_GetFd](#)：获取编码通道对应的设备文件句柄。
- [HI_MPI_VENC_CfgMestPara](#)：设置编码通道运动估计参数。
- [HI_MPI_VENC_SetH264eNaluPara](#)：设置 H.264 编码的 nalu 划分参数。
- [HI_MPI_VENC_GetH264eNaluPara](#)：获取 H.264 编码的 nalu 划分参数。
- [HI_MPI_VENC_SetH264eRefMode](#)：设置跳帧参考模式。
- [HI_MPI_VENC_GetH264eRefMode](#)：获取跳帧参考模式。
- [HI_MPI_VENC_SetParamSet](#)：设置编码参数集合。



- [HI_MPI_VENC_GetParamSet](#): 获取编码参数集合。
- [HI_MPI_VENC_SetMeParam](#): 设置编码运动估计参数。
- [HI_MPI_VENC_GetMeParam](#): 获取编码运动估计参数。

HI_MPI_VENC_CreateGroup

【描述】

创建编码通道组。

【语法】

```
HI_S32 HI_MPI_VENC_CreateGroup(VENC_GRP VeGroup);
```

【参数】

参数名称	描述	输入/输出
VeGroup	编码通道组号。 取值范围: [0, VENC_MAX_GRP_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	编码通道组创建成功。
HI_ERR_VENC_EXIST	编码通道组重复创建。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_INVALID_CHNID	无效的编码通道组号。

【需求】

- 头文件: `hi_comm_venc.h`、`mpi_venc.h`
- 库文件: `libmpi.a`

【注意】



- 本文档中含有通道组号的接口的通道组号的取值范围为[0, [VENC_MAX_GRP_NUM](#)], 否则返回 [HI_ERR_VENC_INVALID_CHNID](#)。
- 编码通道组是指芯片能够同时处理的编码通道的集合, 一个通道组最多可同时包含 1 路主码流 (H.264/MJPEG) 和一路次码流 (H.264/MJPEG), 或者包含 1 路 JPEG 抓拍, 或者仅包含 1 路 MPEG4 通道。
- 如果指定的通道组已经存在, 则返回错误码 [HI_ERR_VENC_EXIST](#)。

【举例】

```
HI_S32 StartVenc(HI_VOID)
{
    HI_S32 s32Ret;
    VI_DEV ViDev = 0;
    VI_CHN ViChn = 0;
    VENC_GRP VeGroup = 0;
    VENC_CHN VeChn = 0;
    VENC\_CHN\_ATTR\_S stAttr;
    VENC\_ATTR\_H264\_S stH264Attr;

    /* set 264 channel attribute */
    stH264Attr.u32PicWidth = 720;
    stH264Attr.u32PicHeight = 576;
    ..... // omit other assignments here.

    stAttr.enType = PT_H264;
    stAttr.pValue = (HI_VOID *)&stH264Attr;

    s32Ret = HI_MPI_VENC_CreateGroup(VeGroup);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_CreateGroup err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_CreateChn(VeChn, &stAttr, HI_NULL);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_CreateChn err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_RegisterChn(VeGroup, VeChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_RegisterChn err 0x%x\n", s32Ret);
```



```
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_StartRecvPic(VeChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_StartRecvPic err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_BindInput(VeGroup, ViDev, ViChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_BindInput err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
}
```

更详细的过程，请参考Sample代码。

【相关主题】

无。

HI_MPI_VENC_DestroyGroup

【描述】

销毁编码通道组。

【语法】

```
HI_S32 HI_MPI_VENC_DestroyGroup(VENC_GRP VeGroup);
```

【参数】

参数名称	描述	输入/输出
VeGroup	编码通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。



【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	无效的编码通道组号。
HI_ERR_VENC_UNEXIST	通道组不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 销毁通道组时，必须保证通道组为空，即没有任何通道在通道组中注册，否则会返回错误码 HI_ERR_VENC_NOT_PERM。
- 销毁并不存在的通道组，返回错误码 HI_ERR_VENC_UNEXIST。

【举例】

```
HI_S32 StopVenc(HI_VOID)
{
    HI_S32 s32Ret;
    VENC_CHN VeChn = 0;
    VENC_GRP VeGroup = 0;

    s32Ret = HI_MPI_VENC_UnbindInput(VeGroup);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_UnbindInput err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32Ret =HI_MPI_VENC_StopRecvPic(VeChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_StopRecvPic err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_UnRegisterChn(VeChn);
```



```
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_UnRegisterChn err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_DestroyChn(VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_DestroyChn err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret =HI_MPI_VENC_DestroyGroup(VeGroup);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_DestroyGroup err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

return HI_SUCCESS;
}
```

【相关主题】

无。

HI_MPI_VENC_BindInput

【描述】

绑定 VI 到通道组。

【语法】

```
HI_S32 HI_MPI_VENC_BindInput(VENC_GRP VeGroup, VI_DEV ViDevId, VI_CHN
ViChn);
```

【参数】

参数名称	描述	输入/输出
VeGroup	编码通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入
ViDevId	VI 设备号。 取值范围：[0, VI_MAX_DEV_NUM)。	输入



参数名称	描述	输入/输出
ViChn	VI 通道号。 取值范围：[0, VI_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_RET_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	无效的通道号。
HI_ERR_VENC_INVALID_DEVID	无效的设备号。
HI_ERR_VENC_UNEXIST	通道组不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 绑定并不存在的通道组，则返回错误码 [HI_ERR_VENC_UNEXIST](#)。
- 如果 VI 设备或者 VI 通道超出范围，则返回 [HI_ERR_VENC_INVALID_DEVID](#) 或者 [HI_ERR_VENC_INVALID_CHNID](#)。
- 此接口并不判断 VI 的状态，ViDevId 和 ViChn 可以对应实际的 VI 设备，也可以对应虚拟的 VI 设备，对应虚拟的 VI 设备主要用于用户手动发送图像编码，[HI_MPI_VENC_SendFrame](#) 会对此作出详细的说明。
- 如果通道组已经绑定了某个 VI 通道，则返回错误码 [HI_ERR_VENC_NOT_PERM](#)。
- 一个通道组只能绑定一个 VI 通道，但一个 VI 通道可以被多个通道组绑定。
- 在编码过程中，可以动态解绑定和绑定 VI，达到编码不同图像源的目的。



【举例】

请参见 [HI_MPI_VENC_CreateGroup](#) 的举例。

【相关主题】

无。

HI_MPI_VENC_UnbindInput

【描述】

解绑定 VI 到通道组。

【语法】

```
HI_S32 HI_MPI_VENC_UnbindInput(VENC_GRP VeGroup);
```

【参数】

参数名称	描述	输入/输出
VeGroup	编码通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	无效的通道组号。
HI_ERR_VENC_UNEXIST	通道组不存在。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a



【注意】

- 解绑定并不存在的通道组，返回错误码 [HI_ERR_VENC_UNEXIST](#)。
- 解绑定之后，VI 通道如果满足条件，可以再绑定到其他任意通道组。
- 可以重复解绑定，返回 [HI_SUCCESS](#)。

【举例】

请参见 [HI_MPI_VENC_DestroyGroup](#) 的举例。

【相关主题】

无。

HI_MPI_VENC_CreateChn

【描述】

创建编码通道。

【语法】

```
HI_S32 HI_MPI_VENC_CreateChn(VENC_CHN VeChn, const VENC\_CHN\_ATTR\_S
*pstAttr, const VENC\_WM\_ATTR\_S *pstWm);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	编码通道属性指针。	输入
pstWm	数字水印属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	无效的编码通道号。



接口返回值	含义
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。
HI_ERR_VENC_EXIST	编码通道重复创建。
HI_ERR_VENC_NOBUF	内存不足导致 buffer 分配失败。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- Hi3520 支持对主次码流进行编码。在创建编码通道的时候必须指定该通道是主码流还是次码流。
- 在创建编码通道的时候，编码通道属性除需要输入各个协议的特有的编码属性之外，一般还需要输入主次码流（MPEG4 编码协议无此属性）、编码协议、编码的帧场模式、输入图像的帧场模式、获取码流的方式（按帧还是按包获取码流）、编码图像大小属性，这些属性受表 6-1 约束，并且这些属性都为静态属性，不允许动态设置。

表6-1 编码通道的部分属性的约束

编码协议	大小码流	编码方式	编码图像大小
H.264	大码流	Frame、Field	160×112 以上，2048 × 1536（3M）像素以下，高度和宽度均 8 像素对齐。
MJPEG	大码流	Frame	160×112 以上，2048 × 1536（3M）像素以下，高度和宽度均 8 像素对齐。
H.264、MJPEG	小码流	Frame	CIF <ul style="list-style-type: none">• PAL：352 × 288• NTSC：352 × 240 QCIF <ul style="list-style-type: none">• PAL：176 × 144• NTSC：176 × 120 QVGA：320 × 240 QQVGA：160 × 120
MPEG4	无	Frame	QCIF



编码协议	大小码流	编码方式	编码图像大小
			<ul style="list-style-type: none"> • PAL: 176 × 144 • NTSC: 176 × 112
JPEG 抓拍	无	Frame	80 × 64 以上, 2048 × 1536 (3M) 像素以下, 高度和宽度均 8 像素对齐。

- 若输入图像大于大码流的宽高, 但相差 16 像素以内 (含 16 像素), 则大码流编码图像通过输入图像做切边得到。
- 若输入图像小于大码流的宽高, 会丢弃这些图像, 而不会对其放大进行编码。该出错信息会在 log 中显示。
- 推荐的大码流编码宽高为: 2048×1536 (3M 像素)、1280×1024 (1.3M 像素)、1920×1080 (1080P)、1280×720 (720P)、704×576、704×480、352×288、352×240。
- 对于 H.264 主码流, 编码图像大小不为 D1 时, 其编码方式推荐使用帧编码。
- 当参数 `pstWm` 为空时, 表示该编码通道不需要使用水印, 否则认为需要使用数字水印。如果创建成功, 数字水印默认使能。目前只有 H.264 编码的大码流可以设置数字水印, 其他的情况设置数字水印时均返回错误码 `HI_ERR_VENC_NOT_SUPPORT`。
- H.264 编码支持目标帧率为分数, 可使用宏 `FRACTION32(de,nu)` 设置, 表示分数 `nu/de`。该宏包含在 `hi_math.h` 文件中。

【举例】

请参见 [HI_MPI_VENC_CreateGroup](#) 的举例。

【相关主题】

无。

HI_MPI_VENC_DestroyChn

【描述】

销毁编码通道。

【语法】

```
HI_S32 HI_MPI_VENC_DestroyChn(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
<code>VeChn</code>	编码通道号。 取值范围: [0, VENC_MAX_CHN_NUM)。	输入

【返回值】



返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 销毁并不存在的通道，返回错误码 HI_ERR_VENC_UNEXIST。
- 销毁前必须保证通道已经从通道组反注册，否则返回错误码 HI_ERR_VENC_NOT_PERM。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_RegisterChn

【描述】

注册编码通道到通道组。

【语法】

```
HI_S32 HI_MPI_VENC_RegisterChn(VENC_GRP VeGroup, VENC_CHN VeChn);
```

【参数】



参数名称	描述	输入/输出
VeGroup	通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入
VeChn	通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号/通道组号错误。
HI_ERR_VENC_UNEXIST	通道/通道组不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_NOMEM	内存分配失败。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 注册并不存在的通道，返回错误码 HI_ERR_VENC_UNEXIST。
- 注册通道到不存在的通道组，返回错误码 HI_ERR_VENC_UNEXIST。
- 同一个编码通道只能注册到一个通道组，如果该通道已经注册到某个通道组，则返回 HI_ERR_VENC_NOT_PERM。
- 主次码流注册的时候需要判定以下约束关系：
 - 主码流要先于次码流注册，否则返回 HI_ERR_VENC_NOT_PERM。
 - 如果编码通道已经注册，则在反注册前不能再进行注册，否则返回 HI_ERR_VENC_NOT_PERM。



- MD 通道注册必须在编码通道注册成功之后进行，否则返回 [HI_ERR_VENC_NOT_PERM](#)。
- 同组的主次码流若为 1:1 的关系，则编码方式必须同为帧编码，否则返回 [HI_ERR_VENC_NOT_PERM](#)。
- 同组的主次码流宽高必须符合如下约束：
 - 主码流的宽度范围[160, 2048]，高度范围[112, 1536]，且均为 2 的倍数；
 - 次码流的宽度范围[160, 352]，高度范围[112, 256]，且均为 2 的倍数；
 - 主次码流宽高比例必须为 1:1、1:2、1:4 三者之一。

【举例】

请参见 [HI_MPI_VENC_CreateGroup](#) 的举例。

【相关主题】

无。

HI_MPI_VENC_UnRegisterChn

【描述】

反注册编码通道到通道组。

【语法】

```
HI_S32 HI_MPI_VENC_UnRegisterChn(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。



接口返回值	含义
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 反注册未创建的通道，则返回错误码 HI_ERR_VENC_UNEXIST。
- 如果通道未注册，则返回错误码 HI_ERR_VENC_NOT_PERM。
- 如果编码通道未停止接收图像编码（HI_MPI_VENC_StopRecvPic 可停止接收），则返回错误码 HI_ERR_VENC_NOT_PERM。
- 反注册后会将编码通道复位，如果用户还在使用未及时释放的码流 buffer，将不能保证此 buffer 数据的正确性。用户可以使用 HI_MPI_VENC_Query 接口来查询状态，确认自己所有的操作都完成之后再反注册通道。

【举例】

请参见 HI_MPI_VENC_DestroyGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_StartRecvPic

【描述】

开启编码通道接收输入图像。

【语法】

```
HI_S32 HI_MPI_VENC_StartRecvPic(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】



返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回 [HI_ERR_VENC_UNEXIST](#)。
- 如果通道没有注册到通道组，则返回 [HI_ERR_VENC_NOT_PERM](#)。
- 此接口不判断当前是否已经开启接收，直接将状态设置为开启接收。
- 此接口用于开启编码通道接收图像来编码，请注意它和绑定通道组的区别。
- 开始接收输入是针对通道的，只有开启接收之后编码器才开始接收图像编码。

【举例】

请参见 [HI_MPI_VENC_CreateGroup](#) 的举例。

【相关主题】

无。

HI_MPI_VENC_StopRecvPic

【描述】

停止编码通道接收输入图像。

【语法】

```
HI_S32 HI_MPI_VENC_StopRecvPic(VENC_CHN VeChn);
```



【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回 HI_ERR_VENC_UNEXIST。
- 如果通道没有注册到通道组，则返回 HI_ERR_VENC_NOT_PERM。
- 此接口并不判断当前是否停止接收，直接将状态设置为停止接收。
- 此接口用于编码通道停止接收图像来编码，在编码通道反注册前必须停止接收图像。
- 调用此接口仅停止接收原始数据编码，码流 buffer 并不会被清除。

【举例】

请参见 HI_MPI_VENC_DestroyGroup 的举例。

【相关主题】



无。

HI_MPI_VENC_Query

【描述】

查询编码通道状态。

【语法】

```
HI_S32 HI_MPI_VENC_Query(VENC_CHN VeChn, VENC_CHN_STAT_S *pstStat);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstStat	编码通道的状态指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】



- 如果通道未创建，则返回 [HI_ERR_VENC_UNEXIST](#)。
- 此接口用于查询此函数调用时刻的编码器状态，pstStat 包含三个主要的信息：
 - 在编码通道状态结构体中，u32LeftPics 表示待编码的帧个数。
在反注册通道前，可以通过查询是否还有图像待编码来决定反注册时机，防止反注册时将可能需要编码的帧清理出去。
 - 在编码通道状态结构体中，u32LeftStreamBytes 表示码流 buffer 中剩余的 byte 数目。
在反注册通道前，可以通过查询是否还有码流没有被处理来决定反注册时机，防止反注册时将可能需要的码流清理出去。
 - 在编码通道状态结构体中，u32CurPacks 表示当前帧的码流包个数。
在按包获取时当前帧可能不是一个完整帧（被取走一部分），按帧获取时表示当前一个完整帧的包个数（如果没有一帧数据则为 0）。用户在需要按帧获取码流时，需要查询一个完整帧的包个数，在这种情况下，通常可以在 select 成功之后执行 query 操作，此时 u32CurPacks 是当前完整帧中包的个数。

【举例】

请参见 [HI_MPI_VENC_GetStream](#) 的举例。

【相关主题】

无。

HI_MPI_VENC_SetChnAttr

【描述】

设置编码通道属性。

【语法】

```
HI_S32 HI_MPI_VENC_SetChnAttr(VENC_CHN VeChn, const VENC_CHN_ATTR_S
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	编码通道属性指针。	输入

【返回值】

返回值	描述
0	成功。



返回值	描述
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 设置未创建的通道的属性，则返回 HI_ERR_VENC_UNEXIST。
- 如果 pstAttr 为空，则返回 HI_ERR_VENC_NULL_PTR。
- 此接口只能设置动态属性，如果设置静态属性，则返回 HI_ERR_VENC_NOT_PERM。主次码流（MPEG4 编码协议无此属性）、编码协议、编码的帧场模式、输入图像的帧场模式、获取码流的方式（按帧还是按包获取码流）、编码图像大小属性均为静态属性。另外，各个编码协议的静态属性由各个协议模块指定，具体请参见 VENC_CHN_ATTR_S。

【举例】

```
/* change target bitrate to 2Mbps */

VENC_CHN_ATTR_S stAttr;
VENC_ATTR_H264_S stH264Attr;

stAttr.pValue = (HI_VOID *)&stH264Attr;
s32Ret = HI_MPI_VENC_GetChnAttr(VeChn, &stAttr);
if (s32Ret != HI_SUCCESS)
```



```
{
    printf("HI_MPI_VENC_GetChnAttr err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

stH264Attr.u32TargetBitrate = 2000;
s32Ret = HI_MPI_VENC_SetChnAttr(VeChn, &stAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_SetChnAttr err 0x%x\n",s32Ret);
    return HI_FAILURE;
}
```

【相关主题】

无。

HI_MPI_VENC_GetChnAttr

【描述】

获取编码通道属性。

【语法】

```
HI_S32 HI_MPI_VENC_GetChnAttr(VENC_CHN VeChn, VENC_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	编码通道属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 获取未创建的通道的属性，返回 HI_ERR_VENC_UNEXIST。
- 不同的协议类型（pstAttr->enType），有不同的通道属性结构体（pstAttr->pValue），注意两者的匹配，否则可能出现无法预料的问题。
- pstAttr->enType 为不支持的类型，则返回 HI_ERR_VENC_NOT_SUPPORT。
- 如果 pstAttr 为空，则返回 HI_ERR_VENC_NULL_PTR。

【举例】

请参见 HI_MPI_VENC_SetChnAttr 的举例。

【相关主题】

无。

HI_MPI_VENC_GetStream

【描述】

获取编码的码流。

【语法】

```
HI_S32 HI_MPI_VENC_GetStream(VENC_CHN VeChn, VENC_STREAM_S *pstStream,  
HI_U32 u32BlockFlag);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入



参数名称	描述	输入/输出
pstStream	码流结构体指针。	输出
u32BlockFlag	阻塞方式。 取值范围： • HI_IO_BLOCK：阻塞。 • HI_IO_NOBLOCK：非阻塞。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_BUF_EMPTY	缓冲区中无数据。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，返回错误码 HI_ERR_VENC_UNEXIST。
- 如果 pstStream 为空，返回错误码 HI_ERR_VENC_NULL_PTR。
- 支持阻塞或非阻塞两种方式获取。同时可支持 select/poll 系统调用。
 - 非阻塞获取时，如果缓冲无数据，则返回 HI_ERR_VENC_BUF_EMPTY。
 - 阻塞时，如果缓冲无数据，则会等待有数据时才返回 HI_SUCCESS。



- 支持按包或按帧方式获取码流。如果按包获取，则：
 - 对于 H.264 编码协议，每次获取的是一个 NAL 单元。
 - 对于 JPEG 编码协议（包括 JPEG 抓拍和 MJPEG），每次获取的是一个 ECS 或图像参数码流包。
 - 对于 MPEG4 编码协议，每次获取的是一帧一个包，因此按帧获取或者按包获取，结果相同。
- 码流结构体 `VENC_STREAM_S` 包含 3 个部分：
 - 码流包信息指针 `pstPack`
指向一组 `VENC_PACK_S` 的内存空间，该空间由调用者分配。如果是按包获取，则此空间不小于 `sizeof(VENC_PACK_S)` 的大小；如果按帧获取，则此空间不小于 $N \times \text{sizeof}(VENC_PACK_S)$ 的大小，其中 N 代表当前帧之中的包的个数，可以在 `select` 之后通过查询接口获得。
 - 码流包个数 `u32PackCount`
在输入时，此值指定 `pstPack` 中 `VENC_PACK_S` 的个数。按包获取时，`u32PackCount` 必须不小于 1；按帧获取时，`u32PackCount` 必须不小于当前帧的包个数。在函数调用成功后，`u32PackCount` 返回实际填充 `pstPack` 的包的个数。
 - 序列号 `u32Seq`
按帧获取时是帧序列号；按包获取时为包序列号。
- 如果用户长时间不获取码流，那么码流缓冲区就会满。一个编码通道如果发生码流缓冲区满，就会停止该编码通道编码，等有足够的码流缓冲可以用来编码时，才开始继续编码，这种情况对于主次码流编码通道来说，相互不受影响。
- 用户应该及时获取码流，防止由于码流 `buffer` 阻塞导致编码器停止工作。

【举例】

```
HI_S32 VencGetH264Stream(HI_VOID)
{
    HI_S32 i;
    HI_S32 s32Ret;
    HI_S32 s32VencFd;
    HI_U32 u32FrameIdx = 0;
    VENC_CHN VeChn = 0;
    VENC_CHN_STAT_S stStat;
    VENC_STREAM_S stStream;
    fd_set read_fds;
    FILE *pFile = NULL;

    pFile = fopen("stream.h264", "wb");
    if(pFile == NULL)
    {
        return HI_FAILURE;
    }
}
```



```
s32VencFd = HI_MPI_VENC_GetFd(VeChn);
do{
    FD_ZERO(&read_fds);
    FD_SET(s32VencFd,&read_fds);

    s32Ret = select(s32VencFd+1, &read_fds, NULL, NULL, NULL);
    if (s32Ret < 0)
    {
        printf("select err\n");
        return HI_FAILURE;
    }
    else if (0 == s32Ret)
    {
        printf("time out\n");
        return HI_FAILURE;
    }
    else
    {
        if (FD_ISSET(s32VencFd, &read_fds))
        {
            s32Ret = HI_MPI_VENC_Query(VeChn, &stStat);
            if (s32Ret != HI_SUCCESS)
            {
                return HI_FAILURE;
            }

            stStream.pstPack = (VENC_PACK_S*)
                malloc(sizeof(VENC_PACK_S)*stStat.u32CurPacks);
            if (NULL == stStream.pstPack)
            {
                return HI_FAILURE;
            }

            stStream.u32PackCount = stStat.u32CurPacks;
            s32Ret = HI_MPI_VENC_GetStream(VeChn, &stStream, HI_TRUE);
            if (HI_SUCCESS != s32Ret)
            {
                free(stStream.pstPack);
                stStream.pstPack = NULL;
                return HI_FAILURE;
            }

            for (i=0; i< stStream.u32PackCount; i++)
            {
```




```
        fwrite(stStream.pstPack[i].pu8Addr[0],
               stStream.pstPack[i].u32Len[0], 1, pFile);
        if (stStream.pstPack[i].u32Len[1] > 0)
        {
            fwrite(stStream.pstPack[i].pu8Addr[1],
                   stStream.pstPack[i].u32Len[1], 1, pFile);
        }
    }

    s32Ret = HI_MPI_VENC_ReleaseStream(VeChn, &stStream);
    if (HI_SUCCESS != s32Ret)
    {
        free(stStream.pstPack);
        stStream.pstPack = NULL;
        return HI_FAILURE;
    }

    free(stStream.pstPack);
    stStream.pstPack = NULL;
}

}

u32FrameIdx++;
}while (u32FrameIdx < 0xff);

fclose(pFile);
return HI_SUCCESS;
}
```

更详细的内容，请参考sample代码。

【相关主题】

无。

HI_MPI_VENC_ReleaseStream

【描述】

释放码流缓存。

【语法】

```
HI_S32 HI_MPI_VENC_ReleaseStream(VENC_CHN VeChn, VENC_STREAM_S
*pstStream);
```

【参数】



参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstStream	码流结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回错误码 HI_ERR_VENC_UNEXIST。
- 如果 pstStream 为空，则返回错误码 HI_ERR_VENC_NULL_PTR。
- 此接口应当和 HI_MPI_VENC_GetStream 配对起来使用，用户获取码流后必须及时释放已经获取的码流缓存，否则可能会导致码流 buffer 满，影响编码器编码，并且用户必须按先获取先释放的顺序释放已经获取的码流缓存。
- 在编码通道反注册以后，所有未释放的码流包均无效，不能再使用或者释放这部分无效的码流缓存。
- 释放无效的码流会返回 HI_ERR_VENC_ILLEGAL_PARAM。

【举例】



请参见 [HI_MPI_VENC_GetStream](#) 的举例。

【相关主题】

无。

HI_MPI_VENC_RequestIDR

【描述】

请求 I 帧。

【语法】

```
HI_S32 HI_MPI_VENC_RequestIDR( VENC_CHN VeChn );
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a



【注意】

- 如果通道未创建，则返回 [HI_ERR_VENC_UNEXIST](#)。
- 接受 IDR 帧或 I 帧请求后，在尽可能短的时间内编出 IDR 帧或 I 帧。
- I 帧请求，只支持 H.264 和 MPEG4 编码协议，JPEG 和 MJPEG 编码请求会返回 [HI_ERR_VENC_NOT_SUPPORT](#)。

【举例】

```
HI_S32 s32Ret;
VENC_CHN VeChn = 0;
s32Ret = HI_MPI_VENC_RequestIDR(VeChn);
if(HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_RequestIDR err 0x%xn", s32Ret);
}
```

【相关主题】

无。

HI_MPI_VENC_InsertUserData

【描述】

插入用户数据。

【语法】

```
HI_S32 HI_MPI_VENC_InsertUserData(VENC_CHN VeChn, HI_U8 *pu8Data, HI_U32
u32Len);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pu8Data	用户数据指针。	输入
u32Len	用户数据长度。 取值范围：[0, 1024]，以 byte 为单位。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。



【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回错误码 HI_ERR_VENC_UNEXIST。
- 如果 pu8Data 为空，则返回错误码 HI_ERR_VENC_NULL_PTR。
- 用户数据大于 1KB，则返回错误码 HI_ERR_VENC_ILLEGAL_PARAM。
- 插入用户数据，只支持 H.264 和 MJPEG/JPEG 编码协议，MPEG4 编码请求会返回 HI_ERR_VENC_NOT_SUPPORT。

【举例】

```
HI_U8 au8UserData[] = "hisilicon2009";
s32Ret = HI_MPI_VENC_InsertUserData(VeChn, au8UserData,
                                     sizeof(au8UserData));
if(HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_InsertUserData err 0x%xn",s32Ret);
}
```

【相关主题】

无。

HI_MPI_VENC_GetCapability

【描述】

获取视频编码能力集。

**【语法】**

```
HI_S32 HI_MPI_VENC_GetCapability(VENC_CAPABILITY_S *pstCap);
```

【参数】

参数名称	描述	输入/输出
pstCap	编码能力集指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 无需创建通道即可获取编码能力集。
- 各个协议编码能力集不同，请参照各个协议的编码能力集描述数据结构。
pstCap->pCapability 指向各个协议编码能力集空间，如：H.264 指向
H264_VENC_CAPABILITY_S 空间，JPEG 或者 MJPEG 指向
JPEG_VENC_CAPABILITY_S 空间。

【举例】

```
HI_S32 VencGetCapability(HI_VOID)
{
    HI_S32 s32Ret;
    VENC_CAPABILITY_S stCapAbility;
```



```
H264_VENC_CAPABILITY_S stH264CapAbility;
JPEG_VENC_CAPABILITY_S stJpegCapAbility;

stCapAbility.enType = PT_H264;
stCapAbility.pCapability = (HI_VOID *)&stH264CapAbility;

s32Ret = HI_MPI_VENC_GetCapability(&stCapAbility);
if(HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_GetCapability err 0x%xn",s32Ret);
    return HI_FAILURE;
}

stCapAbility.enType = PT_MJPEG;
stCapAbility.pCapability = (HI_VOID *)&stJpegCapAbility;

s32Ret = HI_MPI_VENC_GetCapability(&stCapAbility);
if(HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_GetCapability err 0x%xn",s32Ret);
    return HI_FAILURE;
}
return HI_SUCCESS;
}
```

【相关主题】

无。

HI_MPI_VENC_SendFrame

【描述】

支持用户发送原始图像进行编码。

【语法】

```
HI_S32 HI_MPI_VENC_SendFrame(VI_DEV ViDevId, VI_CHN ViChn, HI_U32
u32PoolId, VIDEO_FRAME_S *pstFrame, HI_BOOL bIDR);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VI_MAX_DEV_NUM)。	输入



参数名称	描述	输入/输出
ViChn	VI 通道号。 取值范围：[0, VI_MAX_CHN_NUM)。	输入
u32PoolId	内存池句柄。 取值范围：[0, VB_MAX_POOLS)。	输入
pstFrame	原始图像信息结构指针。	输入
bIDR	是否编为 IDR 或者 I 帧（当前版本会忽略此标志）。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_INVALID_CHNID	无效的通道组号。
HI_ERR_VENC_INVALID_DEVID	无效的设备号。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 用户发送原始图像必须为 Semi-planar YUV 4:2:0 格式，如果是场图像，必须是 2 场 Interlaced 的一帧图像。
- ViDevId 和 ViChn 指定一个虚拟的 VI 通道，在编码器接收原始图像进行编码前，用户需要把对应的通道组绑定到该虚拟 VI，VI 设备或者通道超出会返回 HI_ERR_VENC_INVALID_DEVID 或者 HI_ERR_VENC_INVALID_CHNID。



- 当开启 Deinterlace 功能时，只有输入的原始图像超过 2 帧才能进行编码。如果开启时域滤波功能，则需要超过 3 帧才能进行编码。因此对视频输入的原始图像开启 Deinterlace 功能时，编码器不会对最后一帧原始图像进行编码，而开启时域滤波后，第一帧和最后一帧都不会进行编码。
- 视频输入的原始图像大小必须大于或者等于与该视频输入通道绑定的编码通道的编码图像大小，否则编码器不会编码。
- 此版本 bIDR 无效，设置为 HI_TRUE 或者 HI_FALSE 结果一样。

【举例】

请参见 [HI_MPI_VENC_CreateGroup](#) 的举例。

【相关主题】

无。

HI_MPI_VENC_SetWmAttr

【描述】

设置编码数字水印的属性。

【语法】

```
HI_S32 HI_MPI_VENC_SetWmAttr(VENC_CHN VeChn, VENC_WM_ATTR_S *pstWm);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstWm	编码数字水印属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。



接口返回值	含义
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回 HI_ERR_VENC_UNEXIST。
- 如果 pstWm 为空，则返回 HI_ERR_VENC_NULL_PTR。
- 必须在水印禁用的情况下才能设置水印属性，否则返回 HI_ERR_VENC_NOT_PERM。
- 数字水印的密钥必须为 8 字节，数字水印字符必须为 16 字节，否则会在提取的时候出现乱码，对于不足的情况可以用空格补足。

【举例】

```
VENC_WM_ATTR_S stWm;
memcpy(stWm.au8Key, "hisilicon      ", strlen("hisilicon      "));
memcpy(stWm.au8User, "sdk        ", strlen("sdk        "));
s32Ret = HI_MPI_VENC_SetWmAttr(VeChn, &stWm);
if (HI_SUCCESS != s32Ret)
{
    Return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_EnableWm(VeChn);
if (HI_SUCCESS != s32Ret)
{
    Return HI_FAILURE;
}
```

【相关主题】

无。



HI_MPI_VENC_GetWmAttr

【描述】

获取编码数字水印的属性。

【语法】

```
HI_S32 HI_MPI_VENC_GetWmAttr(VENC_CHN VeChn, VENC_WM_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstWm	编码数字水印属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回 HI_ERR_VENC_UNEXIST。



- 如果 pstWm 为空，则返回 [HI_ERR_VENC_NULL_PTR](#)。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_EnableWm

【描述】

启用编码数字水印。

【语法】

```
HI_S32 HI_MPI_VENC_EnableWm(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】



- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回 [HI_ERR_VENC_UNEXIST](#)。
- 仅支持大码流 H.264 编码数字水印功能，否则返回 [HI_ERR_VENC_NOT_SUPPORT](#)。
- 可以在存储码流中间启用水印。如果开启水印，从开启时到码流文件结束前，一旦发现水印信息错误或者没有水印，解码时均会反馈水印被篡改。因此在需要检测水印的码流存储结束前，不要禁用水印，否则会引起误报。

【举例】

请参考 [HI_MPI_VENC_SetWmAttr](#) 的举例。

【相关主题】

无。

HI_MPI_VENC_DisableWm

【描述】

禁用编码数字水印。

【语法】

```
HI_S32 HI_MPI_VENC_DisableWm(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_SetMaxStreamCnt

【描述】

设置码流缓存帧数。

【语法】

```
HI_S32 HI_MPI_VENC_SetMaxStreamCnt(VENC_CHN VeChn, HI_U32 u32MaxStrmCnt);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
u32MaxStrmCnt	最大码流缓存帧数。	输入

【返回值】

返回值	描述
0	成功。



返回值	描述
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 设置成功后，Buffer 中缓存的码流帧数将不会超过设置值。若缓存码流帧数已达到设置值，将会丢弃当前待编码图像。
- 建议在启动编码前进行设置，不建议在编码过程中动态调整。
- 该接口允许多次调用。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_GetMaxStreamCnt

【描述】

获取码流缓存帧数。

【语法】

```
HI_S32 HI_MPI_VENC_GetMaxStreamCnt(VENC_CHN VeChn, HI_U32 *pu32MaxStrmCnt);
```

【参数】



参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pu32MaxStrmCnt	最大码流缓存帧数的指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_SetH264eRcPara

【描述】

设置 H.264 编码的码率控制参数。



【语法】

```
HI_S32 HI_MPI_VENC_SetH264eRcPara(VENC_CHN VeChn, VENC_ATTR_H264_RC_S  
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	H264 编码的码率控制参数指针	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_ILLEGAL_PARAM	参数错误
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 此接口只对 H.264 协议有效，是高级功能接口，一般情况下无需使用。
- 最大 QP、最小 QP 仅对 VBR 有效。



- 若设置最大 QP 或最小 QP，实际码率可能无法满足用户设置的码率要求。OSD 区域的 QP 不受用户设置值的影响。
- 若设置码率控制不丢帧，实际码率可能会超出目标码率却无法得到控制。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_GetH264eRcPara

【描述】

获取 H.264 编码的码率控制参数。

【语法】

```
HI_S32 HI_MPI_VENC_GetH264eRcPara(VENC_CHN VeChn, VENC_ATTR_H264_RC_S
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	H264 编码的码率控制参数指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。



接口返回值	含义
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

若获取到的最大 QP、最小 QP 值为-1，则表示用户没有设置该参数。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_GetFd

【描述】

获取编码通道对应的设备文件句柄。

【语法】

```
HI_S32 HI_MPI_VENC_GetFd(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】



接口返回值	含义
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_CfgMestPara

【描述】

设置编码通道运动估计参数。

【语法】

```
HI_S32 HI_MPI_VENC_CfgMestPara(VENC_CHN VeChn, VENC\_ATTR\_MEPARA\_S *  
pstParam );
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstParam	运动估计参数。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。



【错误码】

接口返回值	含义
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

此接口只对 H.264 协议有效，是高级功能接口，一般情况下无需使用。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_SetH264eNaluPara

【描述】

设置 H.264 编码的 nalu 划分参数。

【语法】

```
HI_S32 HI_MPI_VENC_SetH264eNaluPara( VENC_CHN VeChn,  
VENC_ATTR_H264_NALU_S *pstH264eNalu);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstH264eNalu	H.264 编码的 nalu 划分指针。	输入

**【返回值】**

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_ILLEGAL_PARAM	参数错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 默认编码方式为一帧一个 Slice，可通过该接口设置为一帧多个 Slice。
- 此接口只对 H.264 协议有效，是高级功能接口，一般情况下无需使用。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_GetH264eNaluPara

【描述】

获取 H.264 编码的 nalu 划分参数。

【语法】



```
HI_S32 HI_MPI_VENC_GetH264eNaluPara( VENC_CHN VeChn,  
VENC_ATTR_H264_NALU_S *pstH264eNalu);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstH264eNalu	H.264 编码的 nalu 划分指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】



无。

HI_MPI_VENC_SetH264eRefMode

【描述】

设置 H.264 编码的跳帧参考模式。

【语法】

```
HI_S32 HI_MPI_VENC_SetH264eRefMode(VENC_CHN VeChn,  
VENC_ATTR_H264_REF_MODE_E enRefMode);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
enRefMode	H.264 编码的跳帧参考模式。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_ILLEGAL_PARAM	参数错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a



【注意】

此接口只对 H.264 协议有效，是高级功能接口，一般情况下无需使用。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_GetH264eRefMode

【描述】

获取 H.264 编码的跳帧参考模式。

【语法】

```
HI_S32 HI_MPI_VENC_GetH264eRefMode(VENC_CHN VeChn,  
VENC_ATTR_H264_REF_MODE_E *penRefMode);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
penRefMode	H.264 编码的跳帧参考模式指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。



接口返回值	含义
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_SetParamSet

【描述】

设置编码参数集合。

【语法】

```
HI_S32 HI_MPI_VENC_SetParamSet(VENC_CHN VeChn, VENC\_PARAM\_SET\_S *  
pstParamSet);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstParamSet	编码参数集合。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。



【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 用户可以通过该接口设置特定的编码参数。
- 系统有默认的编码参数，一般情况下无需调用该接口进行设置。
- 必须在创建通道之后、启动编码之前设置。若编码启动，该接口将不再支持。
- 注销通道的操作，不会影响已设置的参数集。
- 建议使用方式：先调用 HI_MPI_VENC_GetParamSet 获取系统默认编码参数集合，然后更改用户希望修改的参数，再调用 HI_MPI_VENC_SetParamSet 接口设置。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_GetParamSet

【描述】

获取编码参数集合。

【语法】

```
HI_S32 HI_MPI_VENC_GetParamSet(VENC_CHN VeChn, VENC_PARAM_SET_S *  
pstParamSet);
```

【参数】



参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstParamSet	编码参数集合	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 若用户未设置编码参数集合，则返回系统默认的参数集合。

【举例】

无。

【相关主题】

无。



HI_MPI_VENC_SetMeParam

【描述】

设置编码运动估计参数。

【语法】

```
HI_S32 HI_MPI_VENC_SetMeParam(VENC_CHN VeChn, VENC_ME_PARAM_S*  
pstMeParam);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstMeParam	编码运动估计参数。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

**【注意】**

无。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_GetMeParam

【描述】

获取编码运动估计参数。

【语法】

```
HI_S32 HI_MPI_VENC_GetMeParam(VENC_CHN VeChn, VENC_ME_PARAM_S*  
pstMeParam);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstMeParam	编码运动估计参数。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。



接口返回值	含义
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 若用户未设置编码运动估计参数，则返回系统默认参数。

【举例】

无。

【相关主题】

无。

6.4 数据类型

相关数据类型、数据结构定义如下：

- [H264E_NALU_TYPE_E](#)：定义 H.264 码流 NALU 类型。
- [JPEG_E_PACK_TYPE_E](#)：定义 JPEG 码流的 PACK 类型。
- [MPEG4E_PACK_TYPE_E](#)：定义 MPEG4 码流的 PACK 类型。
- [VENC_DATA_TYPE_U](#)：定义码流结果联合体。
- [VENC_PACK_S](#)：定义帧码流包结构体。
- [VENC_STREAM_S](#)：定义帧码流类型结构体。
- [VENC_ATTR_H264_S](#)：定义 H.264 编码属性结构体。
- [VENC_ATTR_MJPEG_S](#)：定义 MJPEG 编码属性结构体。
- [VENC_ATTR_JPEG_S](#)：定义 JPEG 抓拍编码属性结构体。
- [VENC_ATTR_MPEG4_S](#)：定义 MPEG4 编码通道属性结构体。
- [VENC_ATTR_MEpara_S](#)：定义视频编码搜索窗设置结构体。
- [H264_VENC_CAPABILITY_S](#)：定义 H.264 私有能力集描述结构体。
- [JPEG_VENC_CAPABILITY_S](#)：定义 JPEG 私有能力集描述结构体。
- [VENC_CHN_ATTR_S](#)：定义编码通道属性结构体。
- [VENC_CHN_STAT_S](#)：定义编码通道的状态结构体。
- [VENC_CAPABILITY_S](#)：定义编码通道编码能力集结构体。



- [VENC_WM_ATTR_S](#): 定义编码的数字水印的结构体。
- [VENC_ATTR_H264_RC_S](#): 定义 H264 编码的码率控制参数的结构体。
- [VENC_ATTR_H264_NALU_S](#): 定义 H.264 编码的 nalu 大小设置结构体。
- [VENC_ATTR_H264_REF_MODE_E](#): 定义 H.264 编码的跳帧参考模式。
- [RC_MODE_E](#): 定义 H.264 编码的码率控制模式。
- [VENC_PARAM_SET_H264_S](#): 定义 H.264 编码参数集合。
- [VENC_PARAM_SET_JPEG_S](#): 定义 JPEG 编码参数集合。
- [VENC_PARAM_SET_MJPEG_S](#): 定义 MJPEG 编码参数集合。
- [VENC_PARAM_SET_S](#): 定义编码参数集合。
- [VENC_ME_PARAM_H264_S](#): 定义 H.264 编码运动估计参数。
- [VENC_ME_PARAM_S](#): 定义编码运动估计参数。

H264E_NALU_TYPE_E

【说明】

定义 H.264 码流 NALU 类型。

【定义】

```
typedef enum hiH264E_NALU_TYPE_E
{
    H264E_NALU_PSLICE    = 1,
    H264E_NALU_ISLICE    = 5,
    H264E_NALU_SEI       = 6,
    H264E_NALU_SPS       = 7,
    H264E_NALU_PPS       = 8,
    H264E_NALU_BUTT
} H264E_NALU_TYPE_E;
```

【成员】

成员名称	描述
H264E_NALU_PSLICE	PSLICE 类型。
H264E_NALU_ISLICE	ISLICE 类型。
H264E_NALU_SEI	SEI 类型。
H264E_NALU_SPS	SPS 类型。
H264E_NALU_PPS	PPS 类型。

【注意事项】

无。



【相关数据类型及接口】

无。

JPEGE_PACK_TYPE_E

【说明】

定义 JPEG 码流的 PACK 类型。

【定义】

```
typedef enum hiJPEGE_PACK_TYPE_E
{
    JPEGE_PACK_ECS = 5,
    JPEGE_PACK_APP = 6,
    JPEGE_PACK_VDO = 7,
    JPEGE_PACK_PIC = 8,
    JPEGE_PACK_BUTT
} JPEGE_PACK_TYPE_E;
```

【成员】

成员名称	描述
JPEGE_PACK_ECS	ECS 类型。
JPEGE_PACK_APP	APP 类型。
JPEGE_PACK_VDO	VDO 类型。
JPEGE_PACK_PIC	PIC 类型。

【注意事项】

无。

【相关数据类型及接口】

无。

MPEG4E_PACK_TYPE_E

【说明】

定义 MPEG4 码流的 PACK 类型。

【定义】

```
typedef enum hiMPEG4E_PACK_TYPE_E
{
    MPEG4E_PACK_VO = 1,
    MPEG4E_PACK_VOS = 2,
```



```
MPEG4E_PACK_VOL = 3,  
MPEG4E_PACK_VOP = 4,  
MPEG4E_PACK_SLICE = 5  
} MPEG4E_PACK_TYPE_E;
```

【成员】

成员名称	描述
MPEG4E_PACK_VO	VO 类型。
MPEG4E_PACK_VOS	VOS 类型。
MPEG4E_PACK_VOL	VOL 类型。
MPEG4E_PACK_VOP	VOP 类型。

【注意事项】

此版本暂不支持 MPEG4 编码。

【相关数据类型及接口】

无。

VENC_DATA_TYPE_U

【说明】

定义码流结果类型。

【定义】

```
typedef union hiVENC_DATA_TYPE_U  
{  
    H264E_NALU_TYPE_E    enH264EType;  
    JPEG_E_PACK_TYPE_E    enJPEGType;  
    MPEG4E_PACK_TYPE_E    enMPEG4EType;  
} VENC_DATA_TYPE_U;
```

【成员】

成员名称	描述
enH264EType	H.264 码流包类型。
enJPEGType	JPEG 码流包类型。
enMPEG4EType	MPEG4 码流包类型。

【注意事项】



此版本暂不支持 MPEG4 编码。

【相关数据类型及接口】

- [H264E_NALU_TYPE_E](#)
- [JPEGE_PACK_TYPE_E](#)
- [MPEG4E_PACK_TYPE_E](#)

VENC_PACK_S

【说明】

定义帧码流包结构体。

【定义】

```
typedef struct hiVENC_PACK_S
{
    HI_U32          u32PhyAddr[2];
    HI_U8           *pu8Addr[2];
    HI_U32          u32Len[2];
    VENC_DATA_TYPE_U  DataType;

    HI_U64          u64PTS;
    HI_BOOL         bFieldEnd;
    HI_BOOL         bFrameEnd;
} VENC_PACK_S;
```

【成员】

成员名称	描述
pu8Addr[2]	码流包首地址。
u32PhyAddr[2]	码流包物理地址。
u32Len[2]	码流包长度。
DataType	码流类型。
u64PTS	时间戳。单位：us。
bFieldEnd	场结束标识。 取值范围： HI_TRUE：该码流包是该场的最后一个包。 HI_FALSE：该码流包不是该场的最后一个包。



成员名称	描述
bFrameEnd	帧结束标识。 取值范围： HI_TRUE：该码流包是该帧的最后一个包。 HI_FALSE：该码流包不是该场的最后一个包。

【注意事项】

无。

【相关数据类型及接口】

[VENC_DATA_TYPE_U](#)

VENC_STREAM_S

【说明】

定义帧码流类型结构体。

【定义】

```
typedef struct hiVENC_STREAM_S
{
    VENC\_PACK\_S *pstPack;
    HI_U32      u32PackCount;
    HI_U32      u32Seq;
}VENC_STREAM_S;
```

【成员】

成员名称	描述
pstPack	帧码流包结构。
u32PackCount	一帧码流的所有包的个数。
u32Seq	码流序列号。 按帧获取帧序号；按包获取包序号。

【注意事项】

无。

【相关数据类型及接口】

- [VENC_PACK_S](#)
- [HI_MPI_VENC_GetStream](#)



VENC_ATTR_H264_S

【说明】

定义 H.264 编码属性结构体。

【定义】

```
typedef struct hiVENC_ATTR_H264_S
{
    HI_U32  u32Priority;
    HI_U32  u32PicWidth;
    HI_U32  u32PicHeight;
    HI_U32  u32ViFramerate;
    HI_BOOL bMainStream;
    HI_BOOL bVIField;
    HI_U32  u32BufSize;
    HI_BOOL bByFrame;
    HI_BOOL bField;
    HI_U32  u32TargetFramerate;
    HI_U32  u32Gop;
    HI_U32  u32MaxDelay;
    RC_MODE_E enRcMode;
    HI_U32  u32Bitrate;
    HI_U32  u32PicLevel;
    HI_S32  s32QpI;
    HI_S32  s32QpP;
    HI_S32  s32Minutes;
}VENC_ATTR_H264_S;
```

【成员】

成员名称	描述
u32Priority	通道优先级。 目前未使用，取值不限。
u32PicWidth	编码图像宽度。 取值范围：[160, 2048]，以像素为单位。 静态属性。
u32PicHeight	编码图像宽度。 取值范围：[112, 1536]，以像素为单位。 静态属性。



成员名称	描述
u32ViFramerate	VI 输入的帧率（原始帧率）。 取值范围：(0, 60]，以帧/秒为单位。 静态属性。
bMainStream	主次码流标识。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：主码流。 • HI_FALSE：次码流。 静态属性。
bViField	输入图像的帧场标志。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：场。 • HI_FALSE：帧。 静态属性。目前未使用。
u32BufSize	码流 buffer 大小。 取值范围：[Min, Max]，以 byte 为单位。 推荐值：一幅 YUV420 编码图像大小。以编码 D1 图像为例，推荐值为 704×576×1.5 byte。 最小值：一幅 YUV420 编码图像大小的 1/2。 最大值：无限制，但是会消耗更多的内存。 静态属性。
bByFrame	帧/包模式获取码流。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：按帧获取。 • HI_FALSE：按包获取。 静态属性。
bField	帧场编码模式。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：场编码。 • HI_FALSE：帧编码。 静态属性。
u32TargetFramerate	目标帧率。 取值范围：[1/16, 60]，以帧/秒位单位。 整数：高 16bit 为 0。 分数：高 16bit 为分母，低 16bit 为分子。 动态属性。



成员名称	描述
u32Gop	I 帧间隔。 取值范围：[0, 1000]，以帧为单位。 动态属性。
u32MaxDelay	最大延迟。目前未使用。 取值范围：最大延迟，以帧为单位。 动态属性。
enRcMode	码率控制模式。 取值范围：[0, 3]。 0：VBR 模式。 1：CBR 模式。 2：ABR 模式。 3：FIXQP。 动态属性。
u32Bitrate	CBR/ABR 模式，表示平均码率。 VBR 模式，表示最大码率。 FIXQP 模式，该值无效。 取值范围：[1, 20000]，单位 Kbps。 动态属性。
u32PicLevel	图像等级，仅 VBR/CBR 模式下有效。 VBR 模式下，表示图像的质量等级。 取值范围：[0, 5]，值越小，图像质量越好。 CBR 模式下，表示码率波动范围。 取值范围：[0, 5]。 0：由 SDK 软件自行控制码率，推荐使用。 1~5：对应码率波动范围分别为±10%~±50%。
s32QpI	I 帧 QP。FIXQP 模式下有效。 取值范围：[10, 50]。
s32QpP	P 帧 QP。FIXQP 模式下有效。 取值范围：[10, 50]。
s32Minutes	码率统计时段。ABR 模式下有效。 ABR，即码率短时间波动，长时间平稳。 长时间码率的统计，以此时间为准。

【注意事项】



无。

【相关数据类型及接口】

[RC_MODE_E](#)

VENC_ATTR_MJPEG_S

【说明】

定义 MJPEG 编码属性结构体。

【定义】

```
typedef struct hiVENC_ATTR_MJPEG_S
{
    HI_U32  u32Priority;
    HI_U32  u32BufSize;
    HI_U32  u32PicWidth;
    HI_U32  u32PicHeight;
    HI_BOOL bByFrame;
    HI_U32  u32ViFramerate;
    HI_BOOL bMainStream;
    HI_BOOL bVIField;
    HI_U32  u32TargetBitrate;
    HI_U32  u32TargetFramerate;
    HI_U32  u32MCUPerECS;
}VENC_ATTR_MJPEG_S;
```

【成员】

成员名称	描述
u32Priority	通道优先级。 目前未使用，取值不限
u32BufSize	配置 buffer 大小。 取值范围：不小于图像宽高乘积的 1.5 倍。 静态属性。
u32PicWidth	编码图像宽度。 取值范围：[160, 2048]，以像素为单位。 静态属性。
u32PicHeight	编码图像高度。 取值范围：[112, 1536]，以像素为单位。 静态属性。



成员名称	描述
bByFrame	获取码流模式。 取值范围：{HI_TRUE, HI_FALSE}。 <ul style="list-style-type: none">• HI_TRUE：按帧获取。• HI_FALSE：按包获取。 静态属性。
u32ViFramerate	原始帧率。 取值范围： <ul style="list-style-type: none">• PAL：(0, 25]，以帧为单位。• NTSC：(0, 30]，以帧为单位。 静态属性。
bMainStream	主次码流标志。 取值范围：{HI_TRUE, HI_FALSE}。 <ul style="list-style-type: none">• HI_TRUE：主码流。• HI_FALSE：次码流。 静态属性。
bVIField	输入图像的帧场标志。 取值范围：{HI_TRUE, HI_FALSE}。 <ul style="list-style-type: none">• HI_TRUE：场。• HI_FALSE：帧。 静态属性。
u32TargetBitrate	目标码率。 取值范围：[1, 20480]，单位 Kbps。 动态属性。
u32TargetFramerate	目标帧率。 取值范围： P 制 (0, 25]，以帧为单位。 N 制 (0, 30]，以帧为单位。 动态属性。
u32MCUPerECS	每个 ECS 中 MCU 个数。 取值范围：[0, MCU 总数]。建议不小于 16 个。 MCU 总数即图像中宏块（16 像素×16 像素）个数。 动态属性。 0 表示不划分 ECS，为推荐设置。与设置为 (u32PicWidth×u32PicHeight) / 256 等效。



【注意事项】

无。

【相关数据类型及接口】

无。

VENC_ATTR_JPEG_S

【说明】

定义 JPEG 抓拍属性结构体。

【定义】

```
typedef struct hiVENC_ATTR_JPEG_S
{
    HI_U32  u32Priority;
    HI_U32  u32BufSize;
    HI_U32  u32PicWidth;
    HI_U32  u32PicHeight;
    HI_BOOL bVIField;
    HI_BOOL bByFrame;
    HI_U32  u32MCUPerECS;
    HI_U32  u32ImageQuality;
}VENC_ATTR_JPEG_S;
```

【成员】

成员名称	描述
u32Priority	通道优先级。 目前未使用，取值不限
u32BufSize	配置 buffer 大小。 取值范围：不小于图像宽高乘积的 1.5 倍。 静态属性。
u32PicWidth	编码图像宽度。 取值范围：[80, 2048]。 静态属性。
u32PicHeight	编码图像高度。 取值范围：[64, 1536]。 静态属性。



成员名称	描述
bVIField	输入图像的帧场标志。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：场。 • HI_FALSE：帧。 静态属性。
bByFrame	获取码流模式,帧或包。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：按帧获取。 • HI_FALSE：按包获取。 静态属性。
u32MCUPerECS	每个 ECS 中 MCU 个数。 取值范围：[0, MCU 总数]。 建议不小于 16 个。 MCU 总数即图像中宏块（16 像素×16 像素）个数。 动态属性。 0 表示不划分 ECS，为推荐设置。与设置为(u32PicWidth×u32PicHeight) / 256 等效。
u32ImageQuality	抓拍图像质量。 取值范围：[0, 5]。 数字越小，图像质量越好。 动态属性。

【注意事项】

无。

【相关数据类型及接口】

无。

VENC_ATTR_MPEG4_S

【说明】

MPEG4 编码属性结构体。

【定义】

```
typedef struct hiVENC_ATTR_MPEG4_S
{
    HI_U32  u32Priority;
```



```

HI_U32  u32PicWidth;
HI_U32  u32PicHeight;
HI_U32  u32TargetBitrate;
HI_U32  u32ViFramerate;
HI_U32  u32TargetFramerate;
HI_U32  u32Gop;
HI_U32  u32MaxDelay;
M4Qtype enQuantType;
HI_U32  u32BufSize;
HI_BOOL bVIField;
HI_BOOL bByFrame;
}VENC_ATTR_MPEG4_S;

```

【成员】

成员名称	描述
u32Priority	通道优先级。 取值范围：(0, 7)。0 最高，7 最低。 静态属性。 目前无效。
u32PicWidth	图像宽度。 取值范围：QCIF（PAL：176×144，NTSC：176×112）。 静态属性。
u32PicHeight	图像高度。 取值范围：QCIF（PAL：176×144，NTSC：176×112）。 静态属性。
u32TargetBitrate	目标码率。 取值范围：[1, 64]，以 kbit/s 为单位。 动态属性。
u32ViFramerate	原始帧率。 取值范围：[1, 15]。以帧每秒（f/s）为单位。 静态属性。
u32TargetFramerate	目标帧率。 取值范围：[1, 15]。以帧每秒（f/s）为单位。 动态属性。
u32Gop	I 帧间隔。 动态属性。



成员名称	描述
u32MaxDelay	最大延迟。 取值范围：[5, 20]。以帧为单位。 静态属性。
enQuantType	MPEG4 编码量化方式。 动态属性。
u32BufSize	配置码流 buffer 大小。 取值范围：应大于图像大小。 静态属性。
bVIField	输入图像的帧场标志。 静态属性。
bByFrame	帧/包模式获取码流方式。 静态属性。 目前只支持按帧获取。

【注意事项】

本版本不支持 MPEG4 编码。

【相关数据类型及接口】

无。

VENC_ATTR_MEPARA_S

【说明】

定义视频编码运动估计参数的结构体。

【定义】

```
typedef struct hiVENC_ATTR_MEPARA_S
{
    HI_S32 s32HWSize;
    HI_S32 s32VWSize;

    HI_S32 s32IterNum[8];
    HI_S32 s32Denoise[2];
    HI_S32 s32RefPicNum;

} VENC_ATTR_MEPARA_S;
```

【成员】



成员名称	描述
s32HWSize	水平搜索窗。 取值范围：[0, 2]。 0：[-16, +15]。 1：[-32, +31]。 2：[-64, +63]。 默认值为 1。
s32VWSize	垂直搜索窗。 取值范围：[0, 1]。 0：[-16, +15]。 1：[-32, 31]。 默认值为 0。
s32IterNum[8]	保留。
s32Denoise[2]	保留。
s32RefPicNum	保留。

【注意事项】

- 不推荐将 s32HWSize 和 s32VWSize 值设置为 0。
- 搜索窗越大，消耗的带宽越多，可能会对软件性能造成影响。
- 接口中其他参数保留，暂不使用。

【相关数据类型及接口】

[HI_MPI_VENC_CfgMestPara](#)

H264_VENC_CAPABILITY_S

【说明】

H.264 私有能力集描述结构体。

【定义】

```
typedef struct hiH264_VENC_CAPABILITY_S
{
    HI_U8    u8Profile;
    HI_U8    u8Level;
    HI_U8    u8BaseAttr;
    HI_U8    u8ViFormat;
    HI_U8    u8MaxWInMb;
    HI_U8    u8MaxHInMb;
```



```
HI_U16  u16MaxCifNum;  
HI_U16  u16MaxBitrate;  
HI_U16  upperbandwidth;  
HI_U16  lowerbandwidth;  
HI_U8   palfps;  
HI_U8   ntscfps;  
}H264_VENC_CAPABILITY_S;
```

【成员】

成员名称	描述
u8Profile	编码 profile。 0: baseline。 1: mainprofile。 2: extened profile。
u8Level	编码 level。 例如 22 表示 level2.2。
u8BaseAttr	Bit[0:5]分别表示 MBAFF、PAFF、B SLICE、FMO、ASO 和 PARTITION。
u8ViFormat	支持输入制式。 bit0: PAL(25)。 bit1: NTSC(30)。
u8MaxWInMb	输入图像宽度的最大尺寸。 以 16 像素为单位。
u8MaxHInMb	输入图像高度的最大尺寸。 以 16 像素为单位。
u16MaxCifNum	最大编码能力，多通道总和。 以 CIF 为单位。
u16MaxBitrate	最大单通道输出码率能力。 以 kbit/s 为单位。
upperbandwidth	带宽上限。 以 kbit/s 为单位。
lowerbandwidth	带宽下限。 以 kbit/s 为单位。
palfps	PAL 制式帧率。 以帧每秒（f/s）为单位。



成员名称	描述
ntscfps	NTSC 制式帧率。 以帧每秒（f/s）为单位。

【注意事项】

无。

【相关数据类型及接口】

无。

JPEG_VENC_CAPABILITY_S

【说明】

JPEG 私有能力集描述结构体。

【定义】

```
typedef struct hiJPEG_VENC_CAPABILITY_S
{
    HI_U8    u8Profile;
    HI_U8    u8ViFormat;
    HI_U8    u8MaxWInMb;
    HI_U8    u8MaxHInMb;
    HI_U16   u16MaxCifNum;
    HI_U16   u16MaxBitrate;
    HI_U16   upperbandwidth;
    HI_U16   lowerbandwidth;
    HI_U8    palfps;
    HI_U8    ntscfps;
}JPEG_VENC_CAPABILITY_S;
```

【成员】

成员名称	描述
u8Profile	编码 profile。 0: baseline。 1: extened profile。 2: loseless profile。 3: hierarchical profile。



成员名称	描述
u8ViFormat	支持输入制式。 bit0: PAL(25)。 bit1: NTSC(30)。
u8MaxWInMb	输入图像宽度的最大尺寸。 以 16 像素为单位。
u8MaxHInMb	输入图像高度的最大尺寸。 以 16 像素为单位。
u16MaxCifNum	最大编码能力，多通道总和。 以 CIF 为单位。
u16MaxBitrate	最大单通道输出码率能力。 以 kbit/s 为单位。
upperbandwidth	带宽上限。 以 kbit/s 为单位。
lowerbandwidth	带宽下限。 以 kbit/s 为单位。
palfps	PAL 制式帧率。 以帧每秒 (f/s) 为单位。
ntscfps	NTSC 制式帧率。 以帧每秒 (f/s) 为单位。

【注意事项】

无。

【相关数据类型及接口】

无。

VENC_CHN_ATTR_S

【说明】

定义编码通道属性结构体。

【定义】

```
typedef struct hiVENC_CHN_ATTR_S
{
    PAYLOAD_TYPE_E  enType;
    HI_VOID          *pValue;
```



```
}VENC_CHN_ATTR_S;
```

【成员】

成员名称	描述
enType	编码协议类型。
pValue	编码属性指针。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VENC_CreateChn](#)

VENC_CHN_STAT_S

【说明】

定义编码通道的状态结构体。

【定义】

```
typedef struct hiVENC_CHN_STAT_S
{
    HI_BOOL bRegistered;
    HI_U32  u32LeftPics;
    HI_U32  u32LeftStreamBytes;
    HI_U32  u32CurPacks;
}VENC_CHN_STAT_S;
```

【成员】

成员名称	描述
bRegistered	注册到通道组标志。 取值范围：{HI_TRUE, HI_FALSE}。 <ul style="list-style-type: none">• HI_TRUE：注册。• HI_FALSE：未注册。
u32LeftPics	待编码的图像数。
u32LeftStreamBytes	码流 buffer 剩余的 byte 数。
u32CurPacks	当前帧的码流包个数。

【注意事项】



无。

【相关数据类型及接口】

[HI_MPI_VENC_Query](#)

VENC_CAPABILITY_S

【说明】

定义编码通道编码能力集结构体。

【定义】

```
typedef struct hiVENC_CAPABILITY_S
{
    PAYLOAD_TYPE_E  enType;
    HI_VOID          *pCapability;
}VENC_CAPABILITY_S;
```

【成员】

成员名称	描述
enType	编码协议类型。
pCapability	能力集。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VENC_GetCapability](#)

VENC_WM_ATTR_S

【说明】

定义编码的数字水印的结构体。

【定义】

```
#define    DWM_KEY_LEN    8
#define    DWM_CHAR_LEN    16
typedef struct hiVENC_WM_ATTR_S
{
    HI_U8    au8Key[DWM_KEY_LEN];
    HI_U8    au8User[DWM_CHAR_LEN];
}VENC_WM_ATTR_S;
```

【成员】



成员名称	描述
DWM_KEY_LEN	密钥字符的最大个数。
DWM_CHAR_LEN	水印字符个数。
au8Key	数字水印的密钥字符串。最多 8 个字符，不满 8 个字符填充 0。
au8User	数字水印用户字符。个数最多不超过 DWM_CHAR_LEN。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VENC_SetWmAttr](#)

VENC_ATTR_H264_RC_S

【说明】

定义 H.264 编码的码率控制参数的结构体。

【定义】

```
typedef struct hiVENC_ATTR_H264_RC_S
{
    HI_S32 s32MinQP;
    HI_S32 s32MaxQP;
    HI_BOOL bFrameFixQP;
    HI_BOOL bFrameLostAllow;
} VENC_ATTR_H264_RC_S;
```

【成员】

成员名称	描述
s32MinQP	H.264 编码的最小量化系数。取值范围：[4, 50]。 仅 VBR 模式时生效。 默认值：-1。表示用户没有设置。一旦设置，I、P 帧使用的 QP 不会小于该值。 OSD 区域不受该设置值的影响。



成员名称	描述
s32MaxQP	H.264 编码的最大量化系数。取值范围：[4, 50]。 仅 VBR 模式时生效。 默认值：-1。表示用户没有设置。一旦设置，I、P 帧使用的 QP 不会大于该值。 OSD 区域不受该设置值的影响。
bFrameFixQP	帧固定 QP。 0：每一帧的所有宏块可以使用不同量化系数。 1：每一帧的所有宏块使用相同的量化系数。包括 OSD 区域。
bFrameLostAllow	当码率超出太多时，是否允许通过丢帧来控制码率。 0：不允许。 1：允许。 默认值：1。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VENC_SetH264RcPara](#)

VENC_ATTR_H264_NALU_S

【说明】

定义 H.264 编码的 NALU 大小设置结构体。

【定义】

```
typedef struct hiVENC_ATTR_H264_NALU_S
{
    HI_BOOL bNaluSplitEnable;
    HI_U32 u32NaluSize;
} VENC_ATTR_H264_NALU_S;
```

【成员】

成员名称	描述
bNaluSplitEnable	是否打开 NALU 划分。 HI_TRUE：打开。 HI_FALSE：关闭。



成员名称	描述
u32NaluSize	NALU 划分使能的情况下指定 NALU 的大小，以字节为单位，在关闭使能的情况下，此参数无效。 必须满足 $128 \leq \text{u32NaluSize} \leq \text{图象大小(包括色度)}$ 。

【注意事项】

无。

【相关数据类型及接口】

- [HI_MPI_VENC_SetH264eNaluPara](#)
- [HI_MPI_VENC_GetH264eNaluPara](#)

VENC_ATTR_H264_REF_MODE_E

【说明】

定义 H.264 编码的跳帧参考模式。

【定义】

```
typedef enum hiVENC_ATTR_H264_REF_MODE_E
{
    H264E_REF_MODE_1X = 1,
    H264E_REF_MODE_2X = 2,
    H264E_REF_MODE_4X = 5,
    H264E_REF_MODE_BUTT,
}VENC_ATTR_H264_REF_MODE_E;
```

【成员】

成员名称	描述
H264E_REF_MODE_1X	正常参考模式。
H264E_REF_MODE_2X	间隔 2 的跳帧参考模式。
H264E_REF_MODE_4X	间隔 4 的跳帧参考模式。

【注意事项】

无。

【相关数据类型及接口】

- [HI_MPI_VENC_GetH264eRefMode](#)
- [HI_MPI_VENC_SetH264eRefMode](#)



RC_MODE_E

【说明】

定义 H.264 编码的码率控制模式。

【定义】

```
typedef enum hiRC_MODE_E
{
    RC_MODE_VBR = 0,
    RC_MODE_CBR,
    RC_MODE_ABR,
    RC_MODE_FIXQP,
    RC_MODE_BUTT,
} RC_MODE_E;
```

【成员】

成员名称	描述
RC_MODE_VBR	可变码率模式。 该模式下，码率波动大，图像质量稳定。
RC_MODE_CBR	恒定码率模式。 该模式下，码率始终保持平稳。
RC_MODE_ABR	平均码率模式。 该模式下，码率长时间平稳，短时间内波动。
RC_MODE_FIXQP	固定 QP 模式。 该模式下，使用固定的 QP 分别编码 I 帧和 P 帧。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VENC_CreateChn](#)

VENC_PARAM_SET_H264_S

【说明】

定义 H.264 编码参数集合。

【定义】

```
typedef struct hiVENC_PARAM_SET_H264_S
{
```



```
HI_S32 chroma_qp_index_offset;
HI_S32 timing_info_present_flag;
HI_S32 num_units_in_tick;
HI_S32 time_scale;
HI_S32 fixed_frame_rate_flag;
HI_S32 pic_order_cnt_type;
HI_S32 deblocking_filter_control_present_flag;
HI_S32 disable_deblocking_filter_idc;
HI_S32 slice_alpha_c0_offset_div2;
HI_S32 slice_beta_offset_div2;
} VENC_PARAM_SET_H264_S;
```

【成员】

成员名称	描述
chroma_qp_index_offset	H.264 协议语法元素。色度 QP 索引偏移。
timing_info_present_flag	H.264 协议语法元素。时间信息存在标识。
num_units_in_tick	H.264 协议语法元素。
time_scale	H.264 协议语法元素。
fixed_frame_rate_flag	H.264 协议语法元素。
pic_order_cnt_type	H.264 协议语法元素。
deblocking_filter_control_present_flag	H.264 协议语法元素。
disable_deblocking_filter_idc	H.264 协议语法元素。
slice_alpha_c0_offset_div2	H.264 协议语法元素。
slice_beta_offset_div2	H.264 协议语法元素。

【注意事项】

编码参数的默认值，未在本文档描述。请通过接口 [HI_MPI_VENC_GetParamSet\(\)](#) 查询最新默认参数。

【相关数据类型及接口】

- [HI_MPI_VENC_GetParamSet](#)
- [HI_MPI_VENC_SetParamSet](#)

VENC_PARAM_SET_JPEG_S

【说明】

定义 JPEG 编码参数集合。



【定义】

```
typedef struct hiVENC_PARAM_SET_JPEG_S
{
    HI_S32 as32QpOfImgQuality[6];
} VENC_PARAM_SET_JPEG_S;
```

【成员】

成员名称	描述
as32QpOfImgQuality	不同抓拍图像质量对应的 QP 值。尚不支持。

【注意事项】

无。

【相关数据类型及接口】

- [HI_MPI_VENC_GetParamSet](#)
- [HI_MPI_VENC_SetParamSet](#)

VENC_PARAM_SET_MJPEG_S

【说明】

定义 MJPEG 编码参数集合。

【定义】

```
typedef struct hiVENC_PARAM_SET_MJPEG_S
{
    HI_S32 s32LeftForExtend;
} VENC_PARAM_SET_MJPEG_S;
```

【成员】

成员名称	描述
s32LeftForExtend	保留，供扩展。

【注意事项】

无。

【相关数据类型及接口】

- [HI_MPI_VENC_GetParamSet](#)
- [HI_MPI_VENC_SetParamSet](#)



VENC_PARAM_SET_S

【说明】

定义编码参数集合。

【定义】

```
typedef struct hiVENC_PARAM_SET_S
{
    PAYLOAD_TYPE_E enPayload;

    union
    {
        VENC_PARAM_SET_H264_S stParamSetH264;

        VENC_PARAM_SET_JPEG_S stParamSetJpeg;

        VENC_PARAM_SET_MJPEG_S stParamSetMjpeg;
    };
}VENC_PARAM_SET_S;
```

【成员】

成员名称	描述
enPayload	协议类型。
stParamSetH264	H.264 编码参数集合。
stParamSetJpeg	JPEG 编码参数集合。
stParamSetMjpeg	MJPEG 编码参数集合。

【注意事项】

无。

【相关数据类型及接口】

- [HI_MPI_VENC_GetParamSet](#)
- [HI_MPI_VENC_SetParamSet](#)

VENC_ME_PARAM_H264_S

【说明】

定义 H.264 运动估计参数。

【定义】

```
typedef struct hiVENC_ME_PARAM_S
```



```
{  
    HI_S32 s32VWSize;  
    HI_S32 s32HWSize;  
    HI_S32 s32InterPredType;  
    HI_S32 s32IntraPredType;  
} VENC_ME_PARAM_S;
```

【成员】

成员名称	描述
s32VWSize	垂直搜索窗大小。取值范围：{0,1}。单位：像素。 0：[-16, +15]。 1：[-32, +31]。
s32HWSize	水平搜索窗大小。取值范围：{0,1,2}。单位：像素。 0：[-16, +15]。 1：[-32, +31]。 2：[-64, +63]。
s32InterPredType	帧间预测类型。取值范围：{0,1,2,3}。 0：支持 16x16 16x8 8x16 8x8 模式。 1：支持 16x16 16x8 8x16 模式。 2：支持 16x16 8x8 模式。 3：支持 16x16 模式。
s32IntraPredType	帧内预测类型。取值范围：{0,1}。 0：支持 16x16 4x4 模式。 1：支持 4x4 模式。

【注意事项】

无。

【相关数据类型及接口】

- [HI_MPI_VENC_GetMeParam](#)
- [HI_MPI_VENC_SetMeParam](#)

VENC_ME_PARAM_S

【说明】

定义编码运动估计参数。

【定义】

```
typedef struct hiVENC_ME_PARAM_S
```



```
{
    PAYLOAD_TYPE_E enPayload;
    union
    {
        VENC_ME_PARAM_H264_S stMeParamH264;
    };
}VENC_ME_PARAM_S;
```

【成员】

成员名称	描述
enPayload	协议类型。
stMeParamH264	H.264 运动估计参数。

【注意事项】

无。

【相关数据类型及接口】

- [HI_MPI_VENC_GetMeParam](#)
- [HI_MPI_VENC_SetMeParam](#)

6.5 错误码

视频编码 API 错误码如表 6-2 所示。

表6-2 视频编码 API 错误码

错误代码	宏定义	描述
0xA0068001	HI_ERR_VENC_INVALID_DEVID	设备 ID 超出合法范围
0xA0068002	HI_ERR_VENC_INVALID_CHNID	通道 ID 超出合法范围
0xA0068003	HI_ERR_VENC_ILLEGAL_PARAM	参数超出合法范围
0xA0068004	HI_ERR_VENC_EXIST	试图申请或者创建已经存在的设备、通道或者资源
0xA0068005	HI_ERR_VENC_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源
0xA0068006	HI_ERR_VENC_NULL_PTR	函数参数中有空指针
0xA0068007	HI_ERR_VENC_NOT_CONFIG	使用前未配置
0xA0068008	HI_ERR_VENC_NOT_SUPPORT	不支持的参数或者功能



错误代码	宏定义	描述
0xA0068009	HI_ERR_VENC_NOT_PERM	该操作不允许，如试图修改静态配置参数
0xA006800C	HI_ERR_VENC_NOMEM	分配内存失败，如系统内存不足
0xA006800D	HI_ERR_VENC_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA006800E	HI_ERR_VENC_BUF_EMPTY	缓冲区中无数据
0xA006800F	HI_ERR_VENC_BUF_FULL	缓冲区中数据满
0xA0068010	HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块



目 录

7 运动侦测.....	7-1
7.1 概述.....	7-1
7.2 重要概念.....	7-1
7.3 API 参考	7-2
7.4 数据类型.....	7-19
7.5 错误码.....	7-26



7 运动侦测

7.1 概述

移动侦测是检测正在视频编码的图像是否发生亮度变化以及相应的运动向量。移动侦测通道就是视频编码通道，最大支持运动侦测路数与编码路数相同。

Hi3520/Hi3515 提供的移动侦测功能以宏块为最小单位，计算指定图像的宏块在指定图像间隔内的亮度变化和运动向量。如需要获取移动侦测的结果，则启用某一视频编码通道的移动侦测功能。移动侦测的结果包括宏块 SAD、宏块运动向量 MV、宏块报警信息、宏块报警像素的个数。

MD (Move Detect) 模块支持 H.264 编码和 MJPEG 编码时进行移动侦测，针对同一 VI 通道的主次码流，二者只能有一个作 MD。

7.2 重要概念

- MD 图像与 MD 参考图像
比较 MD 图像与 MD 参考图像，来进行移动侦测。两者均可以随时间而不断更新。
- MD 间隔
MD 图像与 MD 参考图像之间的间隔，单位为帧。通俗地讲，就是几帧做一次 MD。
- MD 数据
也称 MD 结果。
Hi3520/Hi3515 可以输出四种 MD 数据：宏块 SAD，宏块 MV，宏块报警信息，宏块报警像素个数。
每一种 MD 数据是否输出，均可以通过开关进行控制。
SDK 仅提供 MD 数据，而不进行 MD 判决。客户可根据不同的应用选择合适的 MD 数据进行 MD 判决，推荐使用宏块 SAD 作为 MD 判决依据。
- 宏块
将图像划分为 16×16(以像素为单位)大小的块，每一块称为一个宏块。
- 宏块 SAD



当前帧与参考帧相应宏块之间的亮度绝对差之和。

SAD(Sum of Absolute Difference): 绝对差之和。

宏块 SAD 越大, 说明两帧对应宏块间的差别越大。

- 宏块 MV

参考帧至当前帧的各帧中相应宏块的运动向量的累加值, 表示该宏块运动的方向。不推荐 MV 作为移动侦测的判断依据。

MV (Motion Vector) 运动向量。

- 宏块报警信息

宏块是否报警。

在去光照效应不打开的情况下, MD 模块通过比较宏块 SAD 与用户设置的 SAD 报警阈值决定是否报警。

在去光照效应打开的情况下, MD 模块通过宏块 SAD, 用户设置的 SAD 报警阈值, 结合用户设置的像素报警阈值、像素报警个数阈值, 通过一定去光照效应运算后得到的结果。Hi3520/Hi3515 芯片暂不支持去光照效应。

只有设定宏块报警 SAD 阈值且启用宏块报警信息输出时, 才会输出该信息。

- 宏块报警像素个数

宏块中报警的像素总数。

MD 模块通过比较像素亮度差(当前帧和参考帧之间)与用户设置的像素报警阈值决定像素是否报警。

只有设定宏块报警像素阈值且启用宏块报警像素个数输出时, 才会输出该数据。

7.3 API 参考

该功能模块提供以下 MPI:

- [HI_MPI_MD_EnableChn](#): 启用某一路视频编码通道的运动侦测功能。
- [HI_MPI_MD_DisableChn](#): 禁用某一路视频编码通道的运动侦测功能。
- [HI_MPI_MD_SetChnAttr](#): 设置运动侦测的属性。
- [HI_MPI_MD_GetChnAttr](#): 获取运动侦测的属性。
- [HI_MPI_MD_SetRefFrame](#): 设置运动侦测的参考图像属性。
- [HI_MPI_MD_GetRefFrame](#): 获取运动侦测的参考图像属性。
- [HI_MPI_MD_GetData](#): 获取运动侦测结果。
- [HI_MPI_MD_ReleaseData](#): 释放运动侦测结果。
- [HI_MPI_MD_GetFd](#): 获取运动侦测通道对应的设备文件句柄。

HI_MPI_MD_EnableChn

【描述】

启用某一路视频编码通道的运动侦测功能。

【语法】



```
HI_S32 HI_MPI_MD_EnableChn(VEnc_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_NOT_CONFIG	没有配置属性。
HI_ERR_MD_NOMEM	模块内部分配内存失败。
HI_ERR_MD_NOT_SUPPORT	不支持（只有 H264 和 MJPEG 的编码通道支持 MD，其他协议的编码通道不支持）。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

- 在启用该编码通道的运动侦测前，相对应的编码通道必须已经创建，且注册到编码通道组 GROUP 中，否则启用失败。运动侦测与编码同时实现，若对应的编码通道没有启动编码，无论运动侦测是否启用，都不会进行运动侦测。
- 在启用前，必须设置该视频编码通道的运动侦测属性。



- 如果需要获取宏块 SAD、宏块报警信息、宏块报警像素个数，还必须设置参考图像属性。
- 多次启用某一视频编码通道的运动侦测功能，和启用一次效果相同，都会返回成功。
- 目前支持 H.264 编码和 MJPEG 编码时，进行运动侦测。对于大小码流编码通道，只支持其中一个码流编码通道进行运动侦测。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_sys.h"
#include "hi_comm_md.h"
#include "mpi_md.h"

HI_S32 MdStart(HI_VOID)
{
    HI_S32 i;
    HI_S32 s32Ret;
    VENC_CHN VeChn = 0;
    MD_CHN_ATTR_S stMdAttr;
    MD_REF_ATTR_S stRefAttr;

    /*set MD attribute*/
    stMdAttr.stMBMode.bMBSADMode = HI_TRUE;
    stMdAttr.stMBMode.bMBMVMode = HI_FALSE;
    stMdAttr.stMBMode.bMBPelNumMode = HI_FALSE;
    stMdAttr.stMBMode.bMBALARMMode = HI_FALSE;
    stMdAttr.u16MBALSADTh = 1000;
    stMdAttr.u8MBPelALTh = 20;
    stMdAttr.u8MBPerALNumTh = 20;
    stMdAttr.enSADBits = MD_SAD_8BIT;
    stMdAttr.stDlight.bEnable = HI_FALSE;
```



```
stMdAttr.u32MDInternal = 0;
stMdAttr.u32MDBufNum = 16;

/*set MD frame*/
stRefAttr.enRefFrameMode = MD_REF_AUTO;
stRefAttr.enRefFrameStat = MD_REF_DYNAMIC;

s32Ret = HI_MPI_MD_SetChnAttr(VeChn, &stMdAttr);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_MD_SetChnAttr Err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_MD_SetRefFrame(VeChn, &stRefAttr);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_MD_SetRefFrame Err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

memset(&stMdAttr, 0, sizeof(MD_CHN_ATTR_S));
s32Ret = HI_MPI_MD_GetChnAttr(VeChn, &stMdAttr);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_MD_GetChnAttr Err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

memset(&stRefAttr, 0, sizeof(MD_REF_ATTR_S));
s32Ret = HI_MPI_MD_GetRefFrame(VeChn, &stRefAttr);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_MD_GetRefFrame Err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_MD_EnableChn(VeChn);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_MD_EnableChn Err 0x%x\n", s32Ret);
    return HI_FAILURE;
}
```



```
        return HI_SUCCESS;
    }
}
```

【相关主题】

无。

HI_MPI_MD_DisableChn

【描述】

禁用某一路视频编码通道的运动侦测功能。

【语法】

```
HI_S32 HI_MPI_MD_DisableChn(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_UNEXIST	通道不存在。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】



多次禁用某一视频编码通道的运动侦测功能，和禁用一次效果相同，都会返回成功。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_sys.h"
#include "hi_comm_md.h"
#include "mpi_md.h"

HI_S32 MdStop(HI_VOID)
{
    HI_S32 s32Ret;
    VENC_CHN VeChn = 0;

    s32Ret = HI_MPI_MD_DisableChn(VeChn);

    if(HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_MD_DisableChn Err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```

【相关主题】

无。

HI_MPI_MD_SetChnAttr

【描述】



设置运动侦测的属性。

【语法】

```
HI_S32 HI_MPI_MD_SetChnAttr(VENC_CHN VeChn, const MD_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	运动侦测属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_NULL_PTR	空指针。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_NOT_SUPPORT	不支持。
HI_ERR_MD_ILLEGAL_PARAM	参数错误。
HI_ERR_MD_NOT_PERM	操作不允许，例如没有禁用 MD 通道

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 在启用运动侦测功能前，需要设置某一路视频编码通道的运动侦测属性。



- 在设置运动侦测的属性前，必须保证运动侦测处于禁用状态。
- 如果要获取宏块报警信息，在去光照效应不打开时，需设置宏块报警 SAD 阈值；在去光照效应打开时，需要设置三个阈值：宏块的 SAD 阈值、宏块内像素报警阈值和宏块内像素报警个数阈值（Hi3520/Hi3515 芯片暂不支持去光照效应）。当发现宏块阈值的变化超过设定的阈值，就认为该宏块是运动的，然后给出宏块的报警信息。
- 如果要获取宏块报警信息，需设置宏块报警 SAD 阈值。
- 如果要获取宏块像素报警个数时，需设置宏块报警的像素阈值。

【举例】

请参见 [HI_MPI_MD_EnableChn](#) 的举例。

【相关主题】

无。

HI_MPI_MD_GetChnAttr

【描述】

获取运动侦测的属性。

【语法】

```
HI_S32 HI_MPI_MD_GetChnAttr(VENC_CHN VeChn, MD_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	运动侦测属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_NULL_PTR	空指针。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或已经去初始化
HI_ERR_MD_NOT_CONFIG	没有配置。

【需求】

- 头文件：[hi_comm_md.h](#)、[mpi_md.h](#)
- 库文件：[libmpi.a](#)

【注意】

若该编码通道的运动侦测的属性没有设置，则获取属性失败。

【举例】

请参见 [HI_MPI_MD_EnableChn](#) 的举例。

【相关主题】

无。

HI_MPI_MD_SetRefFrame

【描述】

设置运动侦测的参考图像属性。

【语法】

```
HI_S32 HI_MPI_MD_SetRefFrame(VENC_CHN VeChn, const MD\_REF\_ATTR\_S
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	参考图像属性指针。	输入

【返回值】



返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_NULL_PTR	空指针。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_NOT_SUPPORT	不支持。
HI_ERR_MD_ILLEGAL_PARAM	非法参数。
HI_ERR_MD_NOMEM	分配内存失败（系统内存不足）。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

- 如果配置的某一视频编码通道运动侦测属性的运动侦测模式中包含宏块 SAD、宏块报警信息、宏块像素报警个数模式中的其中一项或几项，那么必须设置该视频编码通道运动侦测参考图像属性信息。
- 自动设置参考图像时，用户可以根据需要，设置参考图像是否更新。
 - 如果不更新，就按照启用运动侦测后第一帧的编码图像作为参考。
 - 如果更新，就按照运动侦测属性中的运动侦测间隔进行更新。
- 在设置运动侦测的参考图像属性之前必须保证运动侦测处于禁用状态，如果运动侦测为启用状态，则必须首先禁用运动侦测。
- Hi3520/Hi3515 均不支持用户输入参考图像模式。

【举例】

请参见 [HI_MPI_MD_EnableChn](#) 的举例。

【相关主题】



无。

HI_MPI_MD_GetRefFrame

【描述】

获取运动侦测的参考图像属性。

【语法】

```
HI_S32 HI_MPI_MD_GetRefFrame(VENC_CHN VeChn, MD_REF_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	参考图像属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_NULL_PTR	空指针。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_NOT_CONFIG	没有配置。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a



【注意】

如果该通道的运动侦测参考图像属性没有配置，返回失败。

【举例】

请参见 [HI_MPI_MD_EnableChn](#) 的举例。

【相关主题】

无。

HI_MPI_MD_GetData

【描述】

获取运动侦测结果。

【语法】

```
HI_S32 HI_MPI_MD_GetData(VEnc_CHN VeChn, MD_DATA_S *pstMdData, HI_U32  
u32BlockFlag);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstMdData	运动侦测结果指针。	输出
u32BlockFlag	阻塞标志。 取值范围： <ul style="list-style-type: none">• HI_IO_BLOCK：阻塞。• HI_IO_NOBLOCK：非阻塞。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_NULL_PTR	空指针。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_BUF_EMPTY	BUF 空，没有数据可获取（采用非阻塞方式获取数据时）

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

- 如果运动侦测已经禁用，返回失败。
- 支持阻塞和非阻塞接口。
- 如果在获取过程中禁用 MD 通道，则立即返回失败。
- 虽然 MD 可以提供四种 MD 数据，但只有用户打开该数据输出的开关，才会输出该数据。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_sys.h"
#include "hi_comm_md.h"
#include "mpi_md.h"
HI_S32 MdGetData(HI_VOID)
{
    HI_S32 i;
```



```
HI_S32 j;
HI_S32 s32Ret;
HI_S32 s32MdFd;
HI_S32 s32Cnt = 0;
HI_U16* pTmp = NULL;
MD_DATA_S stMdData;
VENC_CHN VeChn = 0;
fd_set read_fds;

s32MdFd = HI_MPI_MD_GetFd(VeChn);

do{
    FD_ZERO(&read_fds);
    FD_SET(s32MdFd,&read_fds);

    s32Ret = select(s32MdFd+1, &read_fds, NULL, NULL, NULL);

    if (s32Ret < 0)
    {
        printf("select err\n");
        return HI_FAILURE;
    }
    else if (0 == s32Ret)
    {
        printf("time out\n");
        return HI_FAILURE;
    }
    else
    {
        sleep(1);
        memset(&stMdData, 0, sizeof(MD_DATA_S));

        if (FD_ISSET(s32MdFd, &read_fds))
        {
            s32Ret = HI_MPI_MD_GetData(VeChn, &stMdData, HI_IO_BLOCK);
            if(s32Ret != HI_SUCCESS)
            {
                printf("HI_MPI_MD_GetData err 0x%x\n",s32Ret);
                return HI_FAILURE;
            }
        }

        s32Cnt++;
    }
}
```



```
/*get MD SAD data*/
if(stMdData.stMBMode.bMBSADMode)
{
    HI_U16* pTmp = NULL;
    for(i=0; i<stMdData.u16MBHeight; i++)
    {
        pTmp = (HI_U16 *) ((HI_U32)stMdData.stMBSAD.pu32Addr+
                           i*stMdData.stMBSAD.u32Stride);
        for(j=0; j<stMdData.u16MBWidth; j++)
        {
            printf("%2d", *pTmp);
            pTmp++;
        }

        printf("\n");
    }
}

s32Ret = HI_MPI_MD_ReleaseData(VeChn, &stMdData);
if(s32Ret != HI_SUCCESS)
{
    printf("Md Release Data Err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

}

}while (s32Cnt < 0x1f);

return HI_SUCCESS;
}
```

【相关主题】

无。

HI_MPI_MD_ReleaseData

【描述】

释放运动侦测缓存。

【语法】

```
HI_S32 HI_MPI_MD_ReleaseData(VEnc_CHN VeChn, const MD_DATA_S *pstMdData);
```

【参数】



参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstMdData	运动侦测数据指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_NULL_PTR	空指针。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_ILLEGAL_PARAM	非法参数。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

- 此接口需与 [HI_MPI_MD_GetData](#) 配对使用。
- 运动侦测结果需要在使用完之后立即释放，如果不及时释放会导致无运动侦测结果缓存而不能进行运动侦测。
- 释放的运动侦测结果必须是从该通道获取的运动侦测结果，不得对运动侦测结果结构体进行任何修改，也不允许释放从别的通道获取的运动侦测结果，否则会导致运动侦测结果不能释放，使此运动侦测结果缓存丢失，甚至导致程序异常。
- 如果在释放运动侦测结果缓存过程中销毁通道，则立刻返回失败。

【举例】

请参见 [HI_MPI_MD_GetData](#) 的举例。

**【相关主题】**

无。

HI_MPI_MD_GetFd**【描述】**

获取运动侦测通道对应的设备文件句柄。

【语法】

```
HI_S32 HI_MPI_MD_GetFd(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_ERR_MD_INVALID_CHNID	通道号错误。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

无。

【举例】

请参见 [HI_MPI_MD_GetData](#) 的举例。

【相关主题】

无。



7.4 数据类型

相关数据类型、数据结构定义如下：

- **MD_MB_MODE_S**：定义宏块模式结构体。
- **MD_DLIGHT_S**：定义去光照效应属性结构体。
- **MD_SADBITS_E**：定义运动侦测的 SAD 的精度。
- **MD_REF_MODE_E**：定义参考图像模式。
- **MD_REF_STATUS_E**：定义参考图像更新状态。
- **MD_REF_ATTR_S**：定义参考图像属性结构体。
- **MD_MB_DATA_S**：定义宏块结果结构体。
- **MD_DATA_S**：定义运动侦测结果结构体。
- **MD_CHN_ATTR_S**：定义运动侦测属性结构体。

MD_MB_MODE_S

【说明】

定义宏块模式结构体。

【定义】

```
typedef struct hiMD_MB_MODE_S
{
    HI_BOOL bMBSADMode;
    HI_BOOL bMBMVMode;
    HI_BOOL bMBALARMMode;
    HI_BOOL bMBPelNumMode;
} MD_MB_MODE_S;
```

【成员】

成员名称	描述
bMBSADMode	宏块 SAD 模式开关。 取值范围：{HI_TRUE, HI_FALSE}。 <ul style="list-style-type: none">• HI_TRUE：打开。• HI_FALSE：关闭。
bMBMVMode	宏块 MV 模式开关。 取值范围：{HI_TRUE, HI_FALSE}。 <ul style="list-style-type: none">• HI_TRUE：打开。• HI_FALSE：关闭。



成员名称	描述
bMBALARMMode	宏块报警模式开关。 取值范围：{HI_TRUE, HI_FALSE}。 <ul style="list-style-type: none">• HI_TRUE：打开。• HI_FALSE：关闭。
bMBPelNumMode	宏块报警像素个数模式开关。 取值范围：{HI_TRUE, HI_FALSE}。 <ul style="list-style-type: none">• HI_TRUE：打开。• HI_FALSE：关闭。

【注意事项】

四种 MD 数据是否输出，即通过该结构体的开关进行控制。

【相关数据类型及接口】

无。

MD_DLIGHT_S

【说明】

定义去光照效应属性结构体。

【定义】

```
typedef struct hiMD_DLIGHT_S
{
    HI_BOOL bEnable;
    HI_U8   u8DlBeta;
    HI_U8   u8DlAlpha;
    HI_U16  Reserved;
} MD_DLIGHT_S;
```

【成员】

成员名称	描述
bEnable	去光照效应开关。 取值范围：{HI_TRUE, HI_FALSE}。 <ul style="list-style-type: none">• HI_TRUE：打开。• HI_FALSE：关闭。
u8DlBeta	去光照效应 Beta 值。 取值范围：[0, 7]。



成员名称	描述
u8DIAlpha	去光照效应 Alpha 值。 取值范围：[0, 7]。
Reserved	保留。

【注意事项】

目前此结构体设置无效。

【相关数据类型及接口】

无。

MD_SADBITS_E

【说明】

定义运动侦测的 SAD 的精度。

【定义】

```
typedef enum hiMD_SADBITS_E
{
    MD_SAD_8BIT = 0,
    MD_SAD_16BIT,
    MD_SAD_BUTT
} MD_SADBITS_E;
```

【成员】

成员名称	描述
MD_SAD_8BIT	SAD 精度为 8bit。
MD_SAD_16BIT	SAD 精度为 16bit。

【注意事项】

不管精度值如何，在运动侦测结果中，每个宏块 SAD 都要占用 2byte 内存。

【相关数据类型及接口】

无。

MD_REF_MODE_E

【说明】

定义参考图像模式。

**【定义】**

```
typedef enum hiMD_REF_MODE_E
{
    MD_REF_AUTO = 0,
    MD_REF_USER,
    MD_REF_MODE_BUTT
} MD_REF_MODE_E;
```

【成员】

成员名称	描述
MD_REF_AUTO	自动设置参考图像模式。
MD_REF_USER	用户输入参考图像模式。

【注意事项】

无。

【相关数据类型及接口】

无。

MD_REF_STATUS_E

【说明】

定义参考图像更新状态。

【定义】

```
typedef enum hiMD_REF_STATUS_E
{
    MD_REF_STATIC = 0,
    MD_REF_DYNAMIC,
    MD_REF_STATUS_BUTT
} MD_REF_STATUS_E;
```

【成员】

成员名称	描述
MD_REF_STATIC	参考图像不更新。
MD_REF_DYNAMIC	参考图像更新。

【注意事项】



参考图像是否更新均是在参考图像模式为自动模式的情况下。参考图像为用户设置模式时，此参考图像状态无效。

【相关数据类型及接口】

无。

MD_REF_ATTR_S

【说明】

定义参考图像属性结构体。

【定义】

```
typedef struct hiMD_REF_ATTR_S
{
    MD_REF_MODE_E    enRefFrameMode;
    MD_REF_STATUS_E  enRefFrameStat;
    VIDEO_FRAME_S    stUserRefFrame;
} MD_REF_ATTR_S;
```

【成员】

成员名称	描述
enRefFrameMode	参考图像模式。
enRefFrameStat	参考图像更新状态。
stUserRefFrame	用户输入模式，参考图像的信息结构体。

【注意事项】

参考图像是否更新均是在参考图像模式为自动模式的情况下。参考图像为用户设置模式时，此参考图像更新状态无效，在此模式下用户还需配置参考图像。

【相关数据类型及接口】

- [MD_REF_MODE_E](#)
- [MD_REF_STATUS_E](#)
- [HI_MPI_MD_SetRefFrame](#)
- [HI_MPI_MD_GetRefFrame](#)

MD_MB_DATA_S

【说明】

定义宏块结果结构体。

【定义】

```
typedef struct hiMD_MB_DATA_S
```



```
{
    HI_U32* pu32Addr;
    HI_U32  u32Stride;
} MD_MB_DATA_S;
```

【成员】

成员名称	描述
pu32Addr	宏块数据指针。
u32Stride	宏块数据以行为单位的内存宽度。

【注意事项】

无。

【相关数据类型及接口】

无。

MD_DATA_S

【说明】

定义运动侦测结果结构体。

【定义】

```
typedef struct hiMD_DATA_S
{
    HI_U32*      pu32Addr;
    HI_U16       u16MBWidth;
    HI_U16       u16MBHeight;
    HI_U64       u64Pts;

    MD_MB_MODE_S stMBMode;
    MD_MB_DATA_S stMBSAD;
    MD_MB_DATA_S stMBMV;
    MD_MB_DATA_S stMBAlarm;
    MD_MB_DATA_S stMBPelAlarmNum;
} MD_DATA_S;
```

【成员】

成员名称	描述
pu32Addr	MD 结果地址信息，释放结果时需用到。 不允许修改。



成员名称	描述
u16MBWidth	图像的宽度。以宏块为单位。
u16MBHeight	图像的高度。以宏块为单位。
u64Pts	时间戳。
stMBMode	宏块模式。
stMBSAD	宏块 SAD。
stMBMV	宏块 MV。
stMBAAlarm	宏块报警信息。
stMBPelAlarmNum	宏块报警像素个数。

【注意事项】

无。

【相关数据类型及接口】

- [MD_MB_MODE_S](#)
- [MD_MB_DATA_S](#)
- [HI_MPI_MD_GetData](#)
- [HI_MPI_MD_ReleaseData](#)

MD_CHN_ATTR_S

【说明】

定义运动侦测属性结构体。

【定义】

```
typedef struct hiMD_CHN_ATTR_S
{
    MD_MB_MODE_S      stMBMode;
    MD_SADBITS_E      enSADBits;
    MD_DLIGHT_S       stDlight;
    HI_U8              u8MBPelALTh;
    HI_U8              u8MBPerALNumTh;
    HI_U16             u16MBALSADTh;
    HI_U32             u32MDInternal;
    HI_U32             u32MDBufNum;
} MD_CHN_ATTR_S;
```

【成员】



成员名称	描述
stMBMode	宏块模式。
enSADBits	SAD 输出精度。
stDlight	去光照效应属性。
u8MBPelALTh	像素报警阈值。 取值范围：(0, 255)。
u8MBPerALNumTh	像素报警个数阈值。 取值范围：(0, 255)。
u16MBALSADTh	宏块报警阈值。 取值范围： <ul style="list-style-type: none"> 去光照效应打开时：(0, 255)。 去光照效应不打开时：(0, 65535)。
u32MDInternal	MD 侦测间隔。 取值范围：[0, 256]，以帧为单位。
u32MDBufNum	MD 缓存大小。 取值范围：[1, 16]。

【注意事项】

无。

【相关数据类型及接口】

- MD_MB_MODE_S
- MD_SADBITS_E
- MD_DLIGHT_S
- HI_MPI_MD_SetChnAttr
- HI_MPI_MD_GetChnAttr

7.5 错误码

运动侦测 API 错误码如表 7-1 所示。

表7-1 运动侦测 API 错误码

错误代码	宏定义	描述
0xA0088001	HI_ERR_MD_INVALID_DEVID	设备 ID 超出合法范围
0xA0088002	HI_ERR_MD_INVALID_CHNID	通道 ID 超出合法范围



错误代码	宏定义	描述
0xA0088003	HI_ERR_MD_ILLEGAL_PARAM	参数超出合法范围
0xA0088004	HI_ERR_MD_EXIST	试图申请或者创建已经存在的设备、通道或者资源
0xA0088005	HI_ERR_MD_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源
0xA0088006	HI_ERR_MD_NULL_PTR	参数中有空指针
0xA0088007	HI_ERR_MD_NOT_CONFIG	使用前未配置
0xA0088008	HI_ERR_MD_NOT_SUPPORT	不支持的参数或者功能
0xA0088009	HI_ERR_MD_NOT_PERM	该操作不允许，如试图修改静态配置参数
0xA008800C	HI_ERR_MD_NOMEM	分配内存失败，如系统内存不足
0xA008800D	HI_ERR_MD_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA008800E	HI_ERR_MD_BUF_EMPTY	缓冲区中无数据
0xA008800F	HI_ERR_MD_BUF_FULL	缓冲区中数据满
0xA0088010	HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块
0xA0088012	HI_ERR_MD_BUSY	系统忙



目 录

8 视频解码.....	8-1
8.1 概述.....	8-1
8.2 重要概念.....	8-1
8.3 API 参考	8-2
8.4 数据类型.....	8-32
8.5 错误码.....	8-45



8 视频解码

8.1 概述

VDEC 模块提供驱动 Hi3520/Hi3515 芯片视频解码硬件工作的 MPI 接口，实现 H.264、MJPEG、JPEG 视频解码功能。

8.2 重要概念

- 码流发送方式

根据 H.264 协议，一帧码流的结束需要收到新一帧码流时才能识别。

Hi3520/Hi3515 解码器可以通过设置码流发送方式，让解码器快速识别一帧码流的结束。码流发送方式包括流式发送和按帧发送：

- 流式发送：用户每次可发送任意大小码流到解码器。选定流式发送方式后，由解码器根据 264 协议识别一帧码流的结束。
- 按帧发送：用户每次发送完整一帧码流到解码器。选定按帧发送方式后，每调用一次发送接口，解码器就认为该帧码流已经结束，因此用户须保证每次调用发送接口发送的码流必为一帧，否则会出现解码错误。

发送方式在通道属性中设置，两种发送方式都须保证每次调用发送接口发送的码流大小不超过通道属性中设置的 buffer 大小。

- 图象输出方式

根据 264 协议，解码图象可能不会在解码后立即输出。Hi3520/Hi3515 解码器可以通过设置不同的图象输出方式达到尽快输出的目的。图象输出方式包括以下三种：

- 普通输出：完全按照 264 协议输出图像。
- 直接输出：收到下一帧码流，输出当前帧图象。
- 按帧输出：收到当前帧码流，输出当前帧图象。

以上三种输出方式，自上而下，输出速度越来越快。其中直接输出和按帧输出方式都不符合 H.264 协议，因此需用户正确配置和使用解码器，才可实现，详细请参见《Hi3520/Hi3515 媒体处理软件 FAQ》。

- 超大帧



当待解码图像中的一帧码流过大（目前取值是用户设置码流 Buffer 的 1/2），超出解码器的处理能力时，即认为其是超大帧。

超大帧会被丢弃，从而导致解码图像出现问题。在某些极特殊场景（如噪声测试序列等）中才可能出现超大帧，自然视频极少出现。

可适当加大码流 Buffer 来处理超大帧。

8.3 API 参考

视频解码模块实现创建解码通道、绑定到视频输出、发送视频码流、获取解码后图像等功能。

该功能模块提供以下 MPI：

- [HI_MPI_VDEC_CreateChn](#)：创建视频解码通道。
- [HI_MPI_VDEC_DestroyChn](#)：销毁视频解码通道。
- [HI_MPI_VDEC_StartRecvStream](#)：解码器开始接收用户发送的码流。
- [HI_MPI_VDEC_StopRecvStream](#)：解码器停止接收用户发送的码流。
- [HI_MPI_VDEC_Query](#)：查询解码通道状态。
- [HI_MPI_VDEC_BindOutput](#)：绑定视频解码通道到视频输出。
- [HI_MPI_VDEC_UnbindOutput](#)：解绑定视频解码通道到视频输出。
- [HI_MPI_VDEC_SetChnAttr](#)：设置视频解码通道属性。
- [HI_MPI_VDEC_GetChnAttr](#)：获取视频解码通道属性。
- [HI_MPI_VDEC_SendStream](#)：向视频解码通道发送码流数据。
- [HI_MPI_VDEC_GetData](#)：获取视频解码通道的解码数据。
- [HI_MPI_VDEC_ReleaseData](#)：释放视频解码通道的图像缓存。
- [HI_MPI_VDEC_GetCapability](#)：获取视频解码能力集。
- [HI_MPI_VDEC_GetFd](#)：获取视频解码通道的设备文件句柄。
- [HI_MPI_VDEC_ResetChn](#)：复位解码通道。
- [HI_MPI_VDEC_UnbindOutputChn](#)：解除某视频输出通道与解码通道的绑定。
- [HI_MPI_VDEC_QueryData](#)：查询是否有解码数据。
- [HI_MPI_VDEC_SetChnParam](#)：设置解码通道参数。
- [HI_MPI_VDEC_GetChnParam](#)：获取解码通道参数。

HI_MPI_VDEC_CreateChn

【描述】

创建视频解码通道。

【语法】

```
HI_S32 HI_MPI_VDEC_CreateChn(VDEC_CHN VdChn, const VDEC_CHN_ATTR_S
*pstAttr, const VDEC_WM_ATTR_S *pstWm);
```



【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstAttr	解码属性指针。	输入
pstWm	数字水印属性指针（暂不支持该功能，必须置为空指针）。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者相关依赖的模块没有加载（如进行 jpeg 解码时 Hi3520_jpegd.ko 未加载）。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_ILLEGAL_PARAM	参数错或超出合法范围。
HI_ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能。解码协议不支持，或者数字水印不支持。
HI_ERR_VDEC_EXIST	试图申请或者创建已经存在的设备、通道或者资源。
HI_ERR_VDEC_NOMEM	分配内存失败（如系统内存不足）。
HI_ERR_VDEC_NOBUF	分配缓存失败（如申请的数据缓冲区太大）。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h



- 库文件: libmpi.a

【注意】

- 如果参数 `pstAttr` 错误或者为空会返回错误码 `HI_ERR_VDEC_ILLEGAL_PARAM` 或 `HI_ERR_VDEC_NULL_PTR`。参数 `pstAttr` 主要包括如下内容:
 - 协议类型
 - 码流 `buffer` 大小
修改码流 `buffer` 允许设置的最小值为宽高乘积的 3/4, 而不是宽高乘积的 1.5 倍。
 - 各个协议属性
各个协议的属性由各个协议指定, 目前 H.264 和 JPEG 协议属性包括优先级、最大解码图像大小、参考图像个数等, H.264 还需要设置码流发送方式 (按帧发送或流式发送)。
- 在创建视频解码通道之前必须保证通道未创建 (或者已经销毁), 否则会直接返回通道已存在错误。
- 目前版本不支持获取数字水印解码, 因此参数 `pstWm` 必须置为空, 否则返回错误码 `HI_ERR_VDEC_NOT_SUPPORT`。

【举例】

- H.264 解码

```
HI_S32 s32ret;
VDEC_CHN VdChn = 0;
VDEC_ATTR_H264_S    stH264Attr;
VDEC_CHN_ATTR_S     stAttr;
VDEC_STREAM_S       stStream;
VDEC_DATA_S         stVdecData;

stH264Attr.u32Priority = 0;
stH264Attr.u32PicHeight = 576;
stH264Attr.u32PicWidth = 720;
stH264Attr.u32RefFrameNum = 3;
stH264Attr.enMode = H264D_MODE_STREAM;

stAttr.enType = PT_H264;
stAttr.u32BufSize = (((stH264Attr.u32PicWidth) *
(stH264Attr.u32PicHeight))<<1);
stAttr.pValue = (void*)&stH264Attr;

/* create vdec chn*/
s32ret = HI_MPI_VDEC_CreateChn(VdChn, &stAttr, NULL);
if (HI_SUCCESS != s32ret)
{
    printf("create vdec chn failed, errno 0x%x \n", s32ret);
    return s32ret;
}
```



```
    }

    /*start recv stream*/
    s32ret = HI_MPI_VDEC_StartRecvStream(VdChn);
    if (HI_SUCCESS != s32ret)
    {
        printf("start recv stream failed, errno 0x%x \n", s32ret);
        return s32ret;
    }

    /* send stream to vdec chn*/
    s32ret = HI_MPI_VDEC_SendStream(VdChn, &stStream, HI_IO_BLOCK);
    if (HI_SUCCESS != s32ret)
    {
        printf("send stream to vdec chn fail,errno 0x%x \n", s32ret);
        return s32ret;
    }

    /* get video frame from vdec chn*/
    s32ret = HI_MPI_VDEC_GetData(VdChn, &stVdecData, HI_IO_BLOCK);
    if (HI_SUCCESS != s32ret)
    {
        printf("get video frame from vdec chn fail,errno 0x%x \n", s32ret);
        return s32ret;
    }

    /* deal with video frame ,send to vo chn for example ... */

    /* release video frame*/
    s32ret = HI_MPI_VDEC_ReleaseData(VdChn, &stVdecData);
    if (HI_SUCCESS != s32ret)
    {
        return s32ret;
    }
    /* ...*/

    /*stop recv stream*/
    s32ret = HI_MPI_VDEC_StopRecvStream(VdChn);
    if (HI_SUCCESS != s32ret){
        printf("stop recv stream failed errno 0x%x \n", s32ret);
        return s32ret;
    }

    /* destroy vdec chn*/
    s32ret = HI_MPI_VDEC_DestroyChn(VdChn);
```



```
if (HI_SUCCESS != s32ret)
{
    printf("destroy vdec chn failed errno 0x%x \n", s32ret);
    return s32ret;
}

•   JPEG 解码
HI_S32 s32ret;
VDEC_CHN VdChn = 0;
VDEC_ATTR_H264_S    stJpegAttr;
VDEC_CHN_ATTR_S     stAttr;
VDEC_STREAM_S       stStream;
VDEC_DATA_S         stVdecData;

stJpegAttr.u32Priority = 0;
stJpegAttr.u32PicHeight = 576;
stJpegAttr.u32PicWidth = 720;

stAttr.enType = PT_JPEG;
stAttr.u32BufSize = stJpegAttr.u32PicHeight * stJpegAttr.u32PicWidth * 2;
stAttr.pValue = (void*)&stJpegAttr;

/* create vdec chn*/
s32ret = HI_MPI_VDEC_CreateChn(VdChn, &stAttr, NULL);
if (HI_SUCCESS != s32ret)
{
    printf("create vdec chn failed, errno 0x%x \n", s32ret);
    return s32ret;
}

/*start recv stream*/
s32ret = HI_MPI_VDEC_StartRecvStream(VdChn);
if (HI_SUCCESS != s32ret)
{
    printf("start recv stream failed, errno 0x%x \n", s32ret);
    return s32ret;
}

/* send stream to vdec chn*/
s32ret = HI_MPI_VDEC_SendStream(VdChn, &stStream, HI_IO_BLOCK);
if (HI_SUCCESS != s32ret)
{
    printf("send stream to vdec chn fail,errno 0x%x \n", s32ret);
    return s32ret;
}
```




```
/* get video frame from vdec chn*/
s32ret = HI_MPI_VDEC_GetData(VdChn, &stVdecData, HI_IO_BLOCK);
if (HI_SUCCESS != s32ret)
{
    printf("get video frame from vdec chn fail,errno 0x%x \n", s32ret);
    return s32ret;
}

/* release video frame*/
s32ret = HI_MPI_VDEC_ReleaseData(VdChn, &stVdecData);
if (HI_SUCCESS != s32ret)
{
    return s32ret;
}

/*stop recv stream*/
s32ret = HI_MPI_VDEC_StopRecvStream(VdChn);
if (HI_SUCCESS != s32ret){
    printf("stop recv stream failed errno 0x%x \n", s32ret);
    return s32ret;
}

/* destroy vdec chn*/
s32ret = HI_MPI_VDEC_DestroyChn(VdChn);
if (HI_SUCCESS != s32ret)
{
    printf("destroy vdec chn failed errno 0x%x \n", s32ret);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_VDEC_DestroyChn

【描述】

销毁视频解码通道。

【语法】

```
HI_S32 HI_MPI_VDEC_DestroyChn(VDEC_CHN VdChn);
```

【参数】



参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_NOT_PERM	操作不允许，在进行此操作前必须停止接收码流。
HI_ERR_VDEC_UNEXIST	通道已经不存在。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 销毁前必须保证通道已创建，否则会返回通道未创建的错误。
- 销毁前必须停止接收码流（或者尚未开始接收码流），否则返回错误码 [HI_ERR_VDEC_NOT_PERM](#)。

【举例】

请参见 [HI_MPI_VDEC_CreateChn](#) 的举例。

【相关主题】

无。



HI_MPI_VDEC_StartRecvStream

【描述】

解码器开始接收用户发送的码流。

【语法】

```
HI_S32 HI_MPI_VDEC_StartRecvStream(VDEC_CHN VdChn);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_UNEXIST	通道尚未创建或者不存在。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 启动接收码流前必须保证通道已创建，否则会返回通道未创建的错误。
- 启动接收码流之后，才能调用 [HI_MPI_VDEC_SendStream](#) 发送码流成功。

【举例】

请参见 [HI_MPI_VDEC_CreateChn](#) 的举例。

**【相关主题】**

无。

HI_MPI_VDEC_StopRecvStream

【描述】

解码器停止接收用户发送的码流。

【语法】

```
HI_S32 HI_MPI_VDEC_StopRecvStream(VDEC_CHN VdChn);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_UNEXIST	通道尚未创建或者不存在。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 停止接收码流前必须保证通道已创建，否则会返回通道未创建的错误。
- 调用此接口后，调用发送码流接口 [HI_MPI_VDEC_SendStream](#) 会返回失败。



【举例】

请参见 [HI_MPI_VDEC_CreateChn](#) 的举例。

【相关主题】

无。

HI_MPI_VDEC_Query

【描述】

查询解码通道状态。

【语法】

```
HI_S32 HI_MPI_VDEC_Query(VDEC_CHN VdChn, VDEC_CHN_STAT_S *pstStat);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstStat	视频解码通道状态结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_UNEXIST	通道尚未创建或者不存在。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。
HI_ERR_VDEC_NULL_PTR	视频解码通道状态结构体指针为空。

【需求】



- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

查询通道状态前必须保证通道已创建，否则会返回通道未创建的错误。

【举例】

请参见 [HI_MPI_VDEC_CreateChn](#) 的举例。

【相关主题】

无。

HI_MPI_VDEC_BindOutput

【描述】

绑定视频解码通道到视频输出通道。

【语法】

```
HI_S32 HI_MPI_VDEC_BindOutput(VDEC_CHN VdChn, VO_DEV VoDev, VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM]。	输入
VoChn	视频输出 VO 通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_VDEC_INVALID_CHNID	解码通道 ID 或者 VO 通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。
HI_ERR_VDEC_NOT_PERM	操作不允许。
HI_ERR_VDEC_INVALID_DEVID	VO 设备号超出合法范围。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 绑定指将视频解码通道和显示输出通道做关联，并不关心实际的解码或者显示输出通道是否已经创建。用户需保证 VO 设备与 VO 通道是可用的。
- 绑定之后，相应的视频解码通道的数据将直接通过相应的显示输出通道输出，输出帧率由 VO 决定，请参见 [HI_MPI_VO_SetChnFrameRate](#)。
- 可以将一个解码通道绑定到多个 VO 通道，最多支持 32 个，若超过，则返回 [HI_ERR_VDEC_NOT_PERM](#)。
- 如果重复绑定 VO 通道，则返回 [HI_ERR_VDEC_NOT_PERM](#)。

【举例】

```
HI_S32 s32ret;
VDEC_CHN VdChn = 0;
VO_CHN VoChn = 0;
VO_DEV VoDev = 0;

/* first create vdec chn*/

/* bind to vo chn */
s32ret = HI_MPI_VDEC_BindOutput(VdChn, VoDev, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("bind vdec output to vo failed!");
    return s32ret;
}

/* ... .. */

/* unbind vo chn*/
s32ret = HI_MPI_VDEC_UnbindOutput(VdChn);
```



```
if (HI_SUCCESS != s32ret)
{
    printf("unbind vdec output to vo failed!");
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_VDEC_UnbindOutput

【描述】

解绑定视频解码通道。

【语法】

```
HI_S32 HI_MPI_VDEC_UnbindOutput(VDEC_CHN VdChn);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a



【注意】

- 解绑定所有的 VO 通道。如果需要解绑定某 VO 通道，请使用 [HI_MPI_VDEC_UnbindOutputChn](#)。
- 可以重复解绑定所有通道，不返回错误。

【举例】

请参见 [HI_MPI_VDEC_BindOutput](#) 的举例。

【相关主题】

无。

HI_MPI_VDEC_SetChnAttr

【描述】

设置视频解码通道属性。

【语法】

```
HI_S32 HI_MPI_VDEC_SetChnAttr(VDEC_CHN VdChn, const VDEC_CHN_ATTR_S  
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstAttr	解码属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化。



接口返回值	含义
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_NOT_PERM	该操作不允许（如试图修改静态配置参数）。
HI_ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 先创建完通道，再设置属性。如果通道未被创建，则返回失败。
- 该接口仅用于设置通道动态属性，不支持对静态属性的修改，如试图修改静态属性，将返回操作不允许的错误码。由于目前版本没有动态属性，所以此接口无实际意义，仅为了兼容可能存在的动态属性。

【举例】

- 设置 H.264 通道属性

```
HI_S32          s32ret;
VDEC_CHN        VdChn = 0;
VDEC_ATTR_H264_S stH264Attr;
VDEC_CHN_ATTR_S  stAttr;

stH264Attr.u32Priority = 0;
stH264Attr.u32PicHeight = 576;
stH264Attr.u32PicWidth = 720;
stH264Attr.u32RefFrameNum = 16;

stAttr.enType = PT_H264;
stAttr.u32BufSize = (((stH264Attr.u32PicWidth) *
(stH264Attr.u32PicHeight))<<1);
stAttr.pValue = (void*)&stH264Attr;

s32ret = HI_MPI_VDEC_SetChnAttr(VdChn, &stAttr);
if (HI_SUCCESS != s32ret)
{
    printf("set vdec chn attr failed \n");
    return s32ret;
}
```

- 设置 JPEG 通道属性



```
HI_S32          s32ret;
VDEC_CHN        VdChn = 0;
VDEC_ATTR_JPEG_S stJpegAttr;
VDEC_CHN_ATTR_S  stAttr;

stJpegAttr.u32Priority = 0;
stJpegAttr.u32PicHeight = 576;
stJpegAttr.u32PicWidth = 720;

stAttr.enType = PT_JPEG;
stAttr.u32BufSize = stJpegAttr.u32PicHeight * stJpegAttr.u32PicWidth * 2;
stAttr.pValue = (void*)&stJpegAttr;

s32ret = HI_MPI_VDEC_SetChnAttr(VdChn, &stAttr);
if (HI_SUCCESS != s32ret)
{
    printf("set vdec chn attr failed \n");
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_VDEC_GetChnAttr

【描述】

获取视频解码通道属性。

【语法】

```
HI_S32 HI_MPI_VDEC_GetChnAttr(VDEC_CHN VdChn, VDEC_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstAttr	解码属性指针。	输出

【返回值】

返回值	描述
0	成功。



返回值	描述
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统未初始化。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_UNEXIST	试图使用不存在的通道。
HI_ERR_VDEC_NOT_PERM	操作不允许（如 reset 操作执行期间，不允许获取通道属性）

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

获取属性前必须保证通道已创建，否则会返回通道未创建的错误。

【举例】

```
VDEC_CHN    VdChn = 0;
VDEC_CHN_ATTR_S    stAttr;
if (HI_SUCCESS != HI_MPI_VDEC_GetChnAttr (VdChn, &stAttr) )
{
    printf("get vdec chn attr failed!");
    return -1;
}
```

【相关主题】

无。

HI_MPI_VDEC_SendStream

【描述】

向视频解码通道发送码流数据。

【语法】



```
HI_S32 HI_MPI_VDEC_SendStream(VDEC_CHN VdChn, const VDEC_STREAM_S  
*pstStream, HI_U32 u32BlockFlag);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstStream	解码码流数据指针。	输入
u32BlockFlag	阻塞非阻塞标志。 取值范围： <ul style="list-style-type: none">HI_IO_BLOCK：阻塞。HI_IO_NOBLOCK：非阻塞。 动态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统未初始化。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_ILLEGAL_PARAM	参数超出合法范围。
HI_ERR_VDEC_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源。
HI_ERR_VDEC_NOT_PERM	操作不允许，在通道未准备好接收码流的情况下向此通道发送码流。
HI_ERR_VDEC_BUF_FULL	缓冲区满，非阻塞接口如果发送失败则返回此错误。
HI_ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能。



【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 此接口通过改变 **Flag** 值支持阻塞方式和非阻塞方式工作。
- 发送数据前必须保证通道已经启动接收码流，否则直接返回 **HI_ERR_VDEC_NOT_PERM**。如果在发送数据过程中停止接收码流，就会立刻返回 **HI_ERR_VDEC_NOT_PERM**。
- 发送数据前必须保证通道已经被创建，否则直接返回 **HI_ERR_VDEC_UNEXIST**。如果在发送数据过程中销毁通道，就会立刻返回 **HI_ERR_VDEC_UNEXIST**。
- 发送码流时需要按照创建解码通道时设置的发送方式（按帧发送或流式发送）进行发送。按帧发送时，调用此接口一次，必须发送完整的一帧码流，否则，解码会出现错误。流式发送则无此限制。
- 若发现码流中有超大帧时，会立即返回 **HI_ERR_VDEC_NOT_SUPPORT**。

【举例】

请参见 [HI_MPI_VDEC_CreateChn](#) 的举例。

【相关主题】

无。

HI_MPI_VDEC_GetData

【描述】

获取视频解码通道的解码数据。

【语法】

```
HI_S32 HI_MPI_VDEC_GetData(VDEC_CHN VdChn, VDEC_DATA_S *pstData, HI_U32 u32BlockFlag);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
u32BlockFlag	阻塞非阻塞标志。 取值范围： <ul style="list-style-type: none"> • HI_IO_BLOCK：阻塞。 • HI_IO_NOBLOCK：非阻塞。 动态属性。	输入



参数名称	描述	输入/输出
pstData	获取的解码数据，包括解码后的图像信息和用户数据。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统未初始化。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源。
HI_ERR_VDEC_BUF_EMPTY	非阻塞获取时，缓冲区中无数据。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 解码数据包括图像信息和用户数据。
- 此接口支持阻塞方式和非阻塞方式工作。
- 如果获取成功返回，要根据图像信息和用户数据结构体中的判断位来判断，判断获取的数据是图像信息，还是用户数据中的一项或者多项。
- 通过 [HI_MPI_VDEC_GetData](#) 获取解码图像数据后，需要通过 [HI_MPI_VDEC_ReleaseData](#) 来释放。
- 获取解码后数据时必须保证通道已经被创建，否则直接返回失败，如果在获取解码后数据的过程中销毁通道，则会立刻返回失败。

【举例】



请参见 [HI_MPI_VDEC_CreateChn](#) 的举例。

【相关主题】

无。

HI_MPI_VDEC_ReleaseData

【描述】

释放视频解码通道的图像缓存。

【语法】

```
HI_S32 HI_MPI_VDEC_ReleaseData(VDEC_CHN VdChn, VDEC_DATA_S *pstData);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstData	解码后的图像信息指针，由 HI_MPI_VDEC_GetData 接口获取。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统未初始化。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源。
HI_ERR_VDEC_ILLEGAL_PARAM	参数超出合法范围。



【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 此接口需与 [HI_MPI_VDEC_GetData](#) 配对使用，获取的数据应当在使用完之后立即释放，如果不及时释放，会导致解码过程阻塞等待资源。
- 释放的数据必须是 [HI_MPI_VDEC_GetData](#) 从该通道获取的数据，不得对数据信息结构体进行任何修改，也不允许释放从其他的通道获取的数据，否则会导致数据不能释放，使此数据 buffer 丢失，甚至导致程序异常。
- 释放数据时必须保证通道已经被创建，否则直接返回 [HI_ERR_VDEC_UNEXIST](#)。如果在释放数据过程中销毁通道，则会立即返回 [HI_ERR_VDEC_UNEXIST](#)。

【举例】

请参见 [HI_MPI_VDEC_CreateChn](#) 的举例。

【相关主题】

无。

HI_MPI_VDEC_GetCapability

【描述】

获取视频解码能力集。

【语法】

```
HI_S32 HI_MPI_VDEC_GetCapability (VDEC_CAPABILITY_S*pstCap);
```

【参数】

参数名称	描述	输入/输出
pstCap	解码能力集指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能。
HI_ERR_VDEC_SYS_NOTREADY	系统未初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 无需创建通道即可获取解码能力集。
- 各个协议编码能力集不同。

【举例】

```
VDEC_CAPABILITY_S    scapability;
H264_VDEC_CAPABILITY_S sh264dcapability;

scapability.enType = PT_H264;
scapability.pCapability = &sh264dcapability;
if (HI_MPI_VDEC_GetCapability(&scapability) != HI_SUCCESS )
{
    printf("get vdec capability failed!\n");
}
```

【相关主题】

无。

HI_MPI_VDEC_GetFd

【描述】

获取视频解码通道的设备文件句柄。

【语法】

```
HI_S32 HI_MPI_VDEC_GetFd(VDEC_CHN VdChn);
```

【参数】



参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

无。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

无。

【举例】

```
int fd;
VDEC_CHN VdChn = 0;

fd = HI_MPI_VDEC_GetFd(VdChn);
if (fd <= 0)
{
    printf("get vdec chn fd err \n!");
    return -1;
}
```

【相关主题】

无。

HI_MPI_VDEC_ResetChn

【描述】

复位视频解码通道。

【语法】



```
HI_S32 HI_MPI_VDEC_ResetChn(VDEC_CHN VdChn);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_UNEXIST	通道不存在。
HI_ERR_VDEC_NOT_PERM	操作不允许。在进行此操作前必须停止接收码流。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 复位前必须保证通道已创建，否则会返回错误码 HI_ERR_VDEC_UNEXIST。
- 复位前必须停止接收码流，否则返回错误码 HI_ERR_VDEC_NOT_PERM。
- 复位操作未完成前，不响应用户的任何操作，如发送码流、销毁通道等，均会返回错误码 HI_ERR_VDEC_NOT_PERM。
- JPEG/MJPEG 不支持 reset 功能。

【举例】

```
/* stop recv stream */  
s32ret = HI_MPI_VDEC_StopRecvStream (VdChn);  
if (HI_SUCCESS != s32ret){
```



```
        printf("stop recv stream failed errno 0x%x \n", s32ret);
        return s32ret;
    }

    /* reset vdec chn*/
    s32ret = HI_MPI_VDEC_ResetChn(VdChn);
    if (HI_SUCCESS != s32ret)
    {
        printf("reset vdec chn failed errno 0x%x \n", s32ret);
        return s32ret;
    }

    /* re-start vdec to receive stream */
    s32ret = HI_MPI_VDEC_StartRecvStream(VdChn);
    if (HI_SUCCESS != s32ret)
    {
        printf("start recv stream failed, errno 0x%x \n", s32ret);
        return s32ret;
    }
}
```

【相关主题】

无。

HI_MPI_VDEC_UnbindOutputChn

【描述】

解除视频输出与某解码通道的绑定关系。

【语法】

```
HI_S32 HI_MPI_VDEC_UnbindOutputChn(VDEC_CHN VdChn, VO_DEV VoDev, VO_CHN
VoChn);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
VoDev	视频输出设备号。 取值范围：[0, VO_MAX_DEV_NUM)。	输入
VoChn	视频输出 VO 通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

**【返回值】**

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_INVALID_DEVID	VO 设备号超出范围。
HI_ERR_VDEC_UNEXIST	通道不存在。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。
HI_ERR_VDEC_NOT_PERM	操作不允许。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 解绑定某 VO 通道。如果需要解绑定所有 VO 通道，建议使用 [HI_MPI_VDEC_UnbindOutput](#)。
- 如果重复解绑定，则返回 [HI_ERR_VDEC_NOT_PERM](#)。

【举例】

无。

【相关主题】

无。

HI_MPI_VDEC_QueryData

【描述】

查询是否有解码数据。

【语法】



```
HI_S32 HI_MPI_VDEC_QueryData(VDEC_CHN VdChn, HI_BOOL *pbIsData);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pbIsData	是否有解码数据。解码数据包括解码图像、用户数据等。 取值范围：{HI_TRUE, HI_FALSE}。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_UNEXIST	通道不存在。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

如果有解码图像或者用户数据，则输出 HI_TRUE，否则输出 HI_FALSE。

【举例】

无。

【相关主题】

无。



HI_MPI_VDEC_SetChnParam

【描述】

设置解码通道参数。

【语法】

```
HI_S32 HI_MPI_VDEC_SetChnParam(VDEC_CHN VdChn, VDEC_CHN_PARAM_S*  
pstChnParam);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstChnParam	通道参数。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_UNEXIST	通道不存在。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 一般情况下，不需要调用该接口。
- 必须停止接收码流，且码流 Buffer 中的码流全部解完之后，才能调用该接口。



- 若发现超大帧，建议增大码流 Buffer 来解决。用户也可以利用该接口增加帧最大值来尝试解决。
- 只有在某些极特殊场景（如噪声测试序列等）中才可能出现超大帧，自然视频极少出现。若码流 Buffer 足够大，如设为（图像宽×图像高×3），也不会出现超大帧的情况。

【举例】

无。

【相关主题】

无。

HI_MPI_VDEC_GetChnParam

【描述】

获取解码通道参数。

【语法】

```
HI_S32 HI_MPI_VDEC_GetChnParam(VDEC_CHN VdChn, VDEC_CHN_PARAM_S*  
pstChnParam);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstChnParam	通道参数。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_UNEXIST	通道不存在。



接口返回值	含义
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。

8.4 数据类型

视频解码相关数据类型、数据结构定义如下：

- VDEC_CHN_ATTR_S：定义视频解码通道属性。
- VDEC_ATTR_JPEG_S：定义使用 JPEG 解码协议的视频解码通道属性结构体。
- VDEC_ATTR_H264_S：定义使用 H.264 解码协议的视频解码通道属性结构体。
- VDEC_STREAM_S：定义解码码流结构体。
- VDEC_USERDATA_S：定义用户私有数据结构体。
- VDEC_CHN_STAT_S：定义通道状态结构体。
- VDEC_FRAME_INFO_S：定义解码帧信息结构体。
- VDEC_DATA_S：定义解码数据结构体。
- VDEC_WM_ATTR_S：定义解码数字水印设置结构体。
- H264_VDEC_CAPABILITY_S：定义 H.264 解码私有能力集描述结构体。
- JPEG_VDEC_CAPABILITY_S：定义 JPEG 解码私有能力集描述结构体。
- VDEC_CAPABILITY_S：定义解码通道能力集。
- H264D_MODE_E：定义码流发送方式。
- VDEC_CHN_PARAM_S：定义解码通道参数。

VDEC_CHN_ATTR_S

【说明】

定义视频解码通道属性结构体。



【定义】

```
typedef struct hiVDEC_CHN_ATTR_S
{
    PAYLOAD_TYPE_E  enType;
    HI_U32           u32BufSize;
    HI_VOID          *pValue;
}VDEC_CHN_ATTR_S;
```

【成员】

成员名称	描述
enType	解码协议类型枚举值。目前仅支持 JPEG 和 H.264。 静态属性。
u32BufSize	码流缓存的大小。 取值范围：大于等于解码通道大小（宽×高）的 3/4 倍，即 420 图像大小的一半（宽×高×3/2×1/2），以 byte 为单位。 推荐值：一幅 YUV420 解码图像大小。即：宽×高×1.5。 静态属性。
pValue	void*类型，指向解码协议的属性的指针。

【描述】

无。

【相关数据类型及接口】

[HI_MPI_VDEC_CreateChn](#)

VDEC_ATTR_JPEG_S

【说明】

定义使用 JPEG 解码协议的视频解码通道属性。

【定义】

```
typedef struct hiVDEC_ATTR_JPEG_S
{
    HI_U32  u32Priority;
    HI_U32  u32PicWidth;
    HI_U32  u32PicHeight;
}VDEC_ATTR_JPEG_S;
```

【成员】



成员名称	描述
u32Priority	通道优先级。 该属性项目目前无用，且无取值范围限制。 静态属性。
u32PicWidth	解码图像宽度。 取值范围：[64, 4096]。 静态属性。
u32PicHeight	解码图像高度。 取值范围：[32, 4096]。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VDEC_CreateChn](#)

VDEC_ATTR_H264_S

【说明】

定义使用 H.264 解码协议的视频解码通道属性。

【定义】

```
typedef struct hiVDEC_ATTR_H264_S
{
    HI_U32  u32Priority;
    HI_U32  u32PicWidth;
    HI_U32  u32PicHeight;
    HI_U32  u32RefFrameNum;
    H264D\_MODE\_E enMode;
}VDEC_ATTR_H264_S;
```

【成员】

成员名称	描述
u32Priority	通道优先级。 该属性项目目前无用，且无取值范围限制。 静态属性。



成员名称	描述
u32PicWidth	解码图像宽度。 取值范围：[64,4096]。 静态属性。
u32PicHeight	解码图像高度。 取值范围：[64,4096]。 静态属性。
u32RefFrameNum	参考帧的数目。 取值范围：[1, 16]，以帧为单位。 参考帧的数目决定解码时需要的参考帧个数，会较大的影响内存占用，根据实际情况设置合适的值。 <ul style="list-style-type: none">• 海思自编码流：推荐设为 3。• 其他监控码流：推荐设为 5。• 测试码流：推荐设为 16。 静态属性。
enMode	码流发送方式。 支持按帧或者按流。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VDEC_CreateChn](#)

VDEC_STREAM_S

【说明】

定义视频解码的码流结构体。

【定义】

```
typedef struct hiVDEC_STREAM_S
{
    HI_U8*   pu8Addr;
    HI_U32   u32Len;
    HI_U64   u64PTS;
}VDEC_STREAM_S;
```

【成员】



成员名称	描述
pu8Addr	码流包的地址。
u32Len	码流包的长度。 以 byte 为单位。
u64PTS	码流包的时间戳。 以 μs 为单位。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_USERDATA_S

【说明】

定义用户私有数据结构体。

【定义】

```
typedef struct hiVDEC_PRIDATA_S
{
    HI_U8*   pu8Addr;
    HI_U32   u32Len;
    HI_BOOL  bValid;
}VDEC_USERDATA_S;
```

【成员】

成员名称	描述
pu8Addr	私有数据的地址。
u32BufSize	私有数据的长度。 以 byte 为单位。
bValid	当前私有数据的有效标识。 取值范围：{HI_TRUE, HI_FALSE}。 <ul style="list-style-type: none">• HI_TRUE：有效。• HI_FALSE：无效。

【注意事项】



无。

【相关数据类型及接口】

无。

VDEC_CHN_STAT_S

【说明】

定义通道状态结构体。

【定义】

```
typedef struct hiVDEC_CHN_STAT_S
{
    HI_U32  u32LeftStreamBytes; /*待解码byte数*/
    HI_U32  u32LeftStreamFrames; /*待解码的frame数 */
    HI_U32  u32LeftPics;          /*待获取的pic数*/
    HI_BOOL bStartRecvStream;    /*是否允许接收码流*/
    HI_U32  u32RecvStreamFrames; /*已接收的frame数*/
    HI_U32  u32DecodeStreamFrames; /*已解码的frame数*/
}VDEC_CHN_STAT_S;
```

【成员】

成员名称	描述
u32LeftStreamBytes	码流 buffer 中待解码的 byte 数。
u32LeftStreamFrames	码流 buffer 中待解码的帧数。 -1 表示无效。 仅 H.264 解码且按帧发送时有效。
u32LeftPics	图像 buffer 中剩余的 pic 数目。
bStartRecvStream	解码器是否已经启动接收码流。
u32RecvStreamFrames	码流 buffer 中已接收帧数。 -1 表示无效。 仅 H.264 解码且按帧发送时有效。
u32DecodeStreamFrames	码流 buffer 中已解码帧数。 -1 表示无效。 仅 H.264 解码且按帧发送时有效。 $u32RecvStreamFrames = u32DecodeStreamFrames + u32LeftStreamFrames$ 。



【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_FRAME_INFO_S

【说明】

定义视频解码帧信息结构体。

【定义】

```
typedef struct hiVDEC_FRAME_S
{
    VIDEO_FRAME_INFO_S  stVideoFrameInfo;
    HI_BOOL              bValid;
}VDEC_FRAME_INFO_S;
```

【成员】

成员名称	描述
stVideoFrameInfo	解码视频帧信息结构体。
bValid	当前解码视频帧的有效标识。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：有效。 • HI_FALSE：无效。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_DATA_S

【说明】

定义视频解码数据结构体。

【定义】

```
typedef struct hiVDEC_DATA_S
{
    VDEC_FRAME_INFO_S  stFrameInfo;
```




```
VDEC_WATERMARK_S    stWaterMark;  
VDEC_USERDATA_S      stUserData;  
} VDEC_DATA_S;
```

【成员】

成员名称	描述
stFrameInfo	视频解码帧信息。
stWaterMark	数字水印（本版本不支持）。
stUserData	用户私有数据。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_WM_ATTR_S

【说明】

定义解码数字水印设置结构体。

【定义】

```
typedef struct hiVDEC_WM_ATTR_S  
{  
    HI_U8          u8Key[DWM_KEY_LEN];  
}VDEC_WM_ATTR_S;
```

【成员】

成员名称	描述
u8Key[DWM_KEY_LEN]	数字水印字符串。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_VDEC_CreateChn](#)

H264_VDEC_CAPABILITY_S

【说明】



定义 H.264 解码私有能力集描述结构体。

【定义】

```
typedef struct hiH264_VDEC_CAPABILITY_S
{
    HI_U8 profile;
    HI_U8 level;
    HI_U8 FMO;
    HI_U8 ASO;
    HI_U8 MBAFF;
    HI_U8 PAF;
    HI_U8 BSlice;
    HI_U8 subqcif;
    HI_U8 qcif;
    HI_U8 cif;
    HI_U8 fourcif;
    HI_U8 sixteencif;
    HI_U8 lostpacket;
    HI_U16 upperbandwidth;
    HI_U16 lowerbandwidth;
    HI_U8 palfps;
    HI_U8 ntscfps;
}H264_VDEC_CAPABILITY_S;;
```

【成员】

成员名称	描述
profile	0: baseline。 1: mainprofile
level	4:4，高 4 位表示整数位，低 4 位表示小数位。 例如 0x22 表示 level2.2。
FMO	Flexible Macroblock Ordering。 0: 不支持。 1: 支持。
ASO	Arbitrary Slice Ordering。 0: 不支持。 1: 支持。
MBAFF	macroblock-adaptive frame/field。 0: 不支持。 1: 支持。



成员名称	描述
PAFF	picture-adaptive frame/field。 0: 不支持。 1: 支持。
BSlice	B slice。 0: 不支持。 1: 支持。
subqcif	SubQCIF 图像。 0: 不支持。 1: 支持。
qcif	QCIF 图像。 0: 不支持。 1: 支持。
cif	CIF 图像。 0: 不支持。 1: 支持。
fourcif	4CIF 图像。 0: 不支持。 1: 支持。
sixteencif	16CIF 图像。 0: 不支持。 1: 支持。
lostpacket	丢包。 0: 不支持。 1: 支持。
upperbandwidth	带宽上限。 以 kbit/s 为单位。
lowerbandwidth	带宽下限。 以 kbit/s 为单位。
palfps	pal 制式帧率。 以帧/秒单位。
ntscfps	ntsc 制式帧率。 以帧/秒单位。



【注意事项】

无。

【相关数据类型及接口】

无。

JPEG_VDEC_CAPABILITY_S

【说明】

定义 JPEG 解码私有能力集描述结构体。

【定义】

```
typedef struct hiJPEG_VDEC_CAPABILITY_S
{
    HI_U32  enProcess;          /* 支持的JPEG编码过程 0:baseline 1:extened
    profile 2:loseless profile 3:hierarchical profile*/
    HI_U32  u32ComponentNum; /* 支持的分量个数 */
    HI_U32  u32QTNum;         /* 支持的量化表数目 */
    HI_BOOL  bYUV420;          /* 是否支持4:2:0采样格式 */
    HI_BOOL  bYUV422;          /* 是否支持4:2:2采样格式 */
    HI_BOOL  bYUV444;          /* 是否支持4:4:4采样格式 */
    HI_U8    lostpacket;        /* 丢包: 1支持; 0不支持*/
    HI_U16   upperbandwidth; /*带宽上限 单位kbps*/
    HI_U16   lowerbandwidth; /*带宽下限 单位kbps*/
    HI_U8    palfps;           /*pal制式fps 单位帧/秒*/
    HI_U8    ntscfps;          /*ntsc制式fps 单位帧/秒*/
}JPEG_VDEC_CAPABILITY_S;
```

【成员】

成员名称	描述
enProcess	支持的 JPEG 编码过程。 0: baseline。 1: extened profile。 2: loseless profile。 3: hierarchical profile。
u32ComponentNum	支持的分量个数。
u32QTNum	支持的量化表数目。
bYUV420	是否支持 4:2:0 采样格式。



成员名称	描述
bYUV422	是否支持 4:2:2 采样格式。
bYUV444	是否支持 4:4:4 采样格式。
lostpacket	丢包。 0：不支持。 1：支持。
upperbandwidth	带宽上限。 以 kbit/s 为单位。
lowerbandwidth	带宽下限。 以 kbit/s 为单位。
palfps	pal 制式帧率。 以帧/秒单位。
ntscfps	ntsc 制式帧率。 以帧/秒单位。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_CAPABILITY_S

【说明】

定义解码通道能力集结构体。

【定义】

```
typedef struct hiVDEC_CAPABILITY_S
{
    PAYLOAD_TYPE_E enType;
    HI_VOID *pCapability;
} VDEC_CAPABILITY_S;
```

【成员】

成员名称	描述
enType	解码协议类型。
pCapability	能力集。



【注意事项】

无。

【相关数据类型及接口】

无。

H264D_MODE_E

【说明】

定义码流发送方式。

【定义】

```
typedef enum hiH264D_MODE_E
{
    H264D_MODE_STREAM = 0,
    H264D_MODE_FRAME,
    H264D_MODE_BUTT
}H264D_MODE_E;
```

【成员】

成员名称	描述
H264D_MODE_STREAM	按流方式发送码流。
H264D_MODE_FRAME	发送码流。 以帧为单位。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_CHN_PARAM_S

【说明】

定义解码通道参数。

【定义】

```
typedef struct hiVDEC_CHN_PARAM_S
{
    HI_U32 u32FrameSizeMax;
```



```
    HI_BOOL bDirectOut;  
} VDEC_CHN_PARAM_S;
```

【成员】

成员名称	描述
u32FrameSizeMax	帧最大值。 取值范围：[u32BufSize / 2, u32BufSize × 3 / 4]。 目前，仅对 H.264 解码通道有效。
bDirectOut	0：普通输出模式。当码流的解码顺序与输出顺序不一致时，请使用该方式。 1：直接输出模式。默认值。 仅对 H.264 解码通道有效。

【注意事项】

无。

【相关数据类型及接口】

无。

8.5 错误码

错误代码	宏定义	描述
0xA00C8001	HI_ERR_VDEC_INVALID_DEVID	设备 ID 超出合法范围
0xA00C8002	HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围
0xA00C8003	HI_ERR_VDEC_ILLEGAL_PARAM	参数超出合法范围
0xA00C8004	HI_ERR_VDEC_EXIST	试图申请或者创建已经存在的设备、通道或者资源
0xA00C8005	HI_ERR_VDEC_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源
0xA00C8006	HI_ERR_VDEC_NULL_PTR	函数参数中有空指针
0xA00C8007	HI_ERR_VDEC_NOT_CONFIG	使用前未配置
0xA00C8008	HI_ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能
0xA00C8009	HI_ERR_VDEC_NOT_PERM	该操作不允许，如试图修改静态配置参数



错误代码	宏定义	描述
0xA00C800C	HI_ERR_VDEC_NOMEM	分配内存失败，如系统内存不足
0xA00C800D	HI_ERR_VDEC_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA00C800E	HI_ERR_VDEC_BUF_EMPTY	缓冲区中无数据
0xA00C800F	HI_ERR_VDEC_BUF_FULL	缓冲区中数据满
0xA00C8010	HI_ERR_VDEC_SYS_NOTREADY	系统未初始化



目 录

9 音频.....	9-1
9.1 概述.....	9-1
9.2 重要概念.....	9-1
9.3 API 参考	9-6
9.3.1 音频输入	9-6
9.3.2 音频输出	9-23
9.3.3 音频编码	9-44
9.3.4 音频解码	9-50
9.4 数据类型.....	9-57
9.4.1 音频输入输出.....	9-57
9.4.2 音频编码	9-71
9.4.3 音频解码	9-75
9.5 错误码.....	9-80



插图目录

图 9-1 Hi3520 SIO 与 AI、AO 关系示意	9-2
图 9-2 I ² S 2/4/8/16 路接收	9-3



表格目录

表 9-1 音频编解码协议说明.....	9-3
表 9-2 AAC Encoder 各协议情况下支持的码率设置（码率单位为 kbps）	9-5
表 9-3 AAC Encoder 各协议情况下推荐的码率设置（码率单位为 kbps）	9-5
表 9-4 海思语音帧结构	9-6
表 9-5 音频输入 API 错误码	9-80
表 9-6 音频输出 API 错误码	9-81
表 9-7 音频编码 API 错误码	9-81
表 9-8 音频解码 API 错误码	9-82



9 音频

9.1 概述

AUDIO 模块包括音频输入、音频输出、音频编码、音频解码四个子模块。音频输入和输出模块通过对 Hi3520/Hi3515 芯片 SIO 设备的控制实现相应的音频输入输出功能；音频编码和解码模块则提供对 ADPCM、G726、G711、AMR 格式的音频编解码功能

9.2 重要概念

- SIO 及 AI、AO 设备

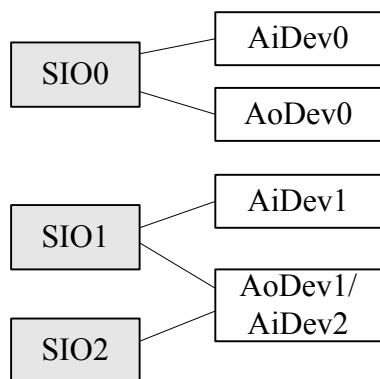
音频输入输出接口 SIO (Sonic Input/Output)，用于和片外 Audio CODEC 芯片连接，完成音乐（语音）的播放及录制。Hi3520 芯片提供 3 个 SIO 接口，Hi3515 芯片提供 2 个 SIO 接口，依次标示为 SIO0、SIO1、SIO2。每个 SIO 接口都能同时提供音频输入和音频输出功能（但由于芯片管脚的限制，各个 SIO 支持的情况是不一样的）。

SIO0 能同时提供音频输入和输出功能，SIO1 能提供音频输入功能；Hi3520 相对 Hi3515 而言，增加了 SIO1 的输出功能和 SIO2 的输入功能，但这两者是复用关系，即只能选择其中一种功能，并且芯片管脚也是复用的。

软件 SDK 将音频输入和输出功能分别用 AI 和 AO 两个模块来管理，AI 和 AO 又按照 SIO 序号来区分不同的 SIO 接口。例如 SIO0 的输入接口称为 AI0 设备，SIO0 的输出接口称为 AO0 设备，SIO1 的输入接口称为 AI1 设备，如图 9-1 所示。



图9-1 Hi3520 SIO 与 AI、AO 关系示意



注：Hi3515 没有 AoDev1/AiDev2

- 音频接口时序

SIO 接口支持标准的 I²S 接口时序模式和 PCM 接口时序模式，并提供灵活的配置以支持与多种 AUDIO CODEC 对接。详细的时序支持情况请参考《Hi3520/Hi3515 H.264 编解码处理器用户指南》。

用户需要对 I²S 或者 PCM 协议以及对接的 CODEC 时序支持情况有足够了解，这里只简单介绍下 Hi3520/Hi3515 I²S 及 PCM 接口时序的几个特性：

- 按照标准 I²S 或 PCM 协议，总是先传送 MSB，后传送 LSB，即按照从高位到低位的顺序传输串行数据。
- Hi3520/Hi3515 SIO 支持扩展的多路接收的 I²S 及 PCM 接口时序，对接 CODEC 的时序支持情况、时钟、位宽等配置必须与 Hi3520/Hi3515 的配置保持一致，否则可能采集不到正确的数据。
- Hi3520/Hi3515 SIO 支持 SIO 主模式和 SIO 从模式，主模式即 SIO 提供时钟，从模式即 CODEC 提供时钟；主模式时 SIO 提供的时钟供输入和输出共同使用，而从模式时的输入输出可以分别由外围 CODEC 提供时钟。
- Hi3520/Hi3515 SIO 支持标准 PCM 模式和自定义 PCM 模式。

- AI、AO 通道

标准的 I²S 协议只有左右声道这两个通道的概念，Hi3520/Hi3515 SIO 同时能够支持多路复用的接收模式，最大支持 16bit 16 通道的 I²S 数据接收。SIO 本身并不关心具体的通道分布，通道由软件 SDK 来管理；根据输入和输出的不同，分为 AI 通道和 AO 通道，通道又隶属于设备，例如 AiDev0 下可支持多个通道 AiChn0、AiChn1、...、AiChn15，具体支持的通道个数由用户根据对接时序要求来做相应的配置。

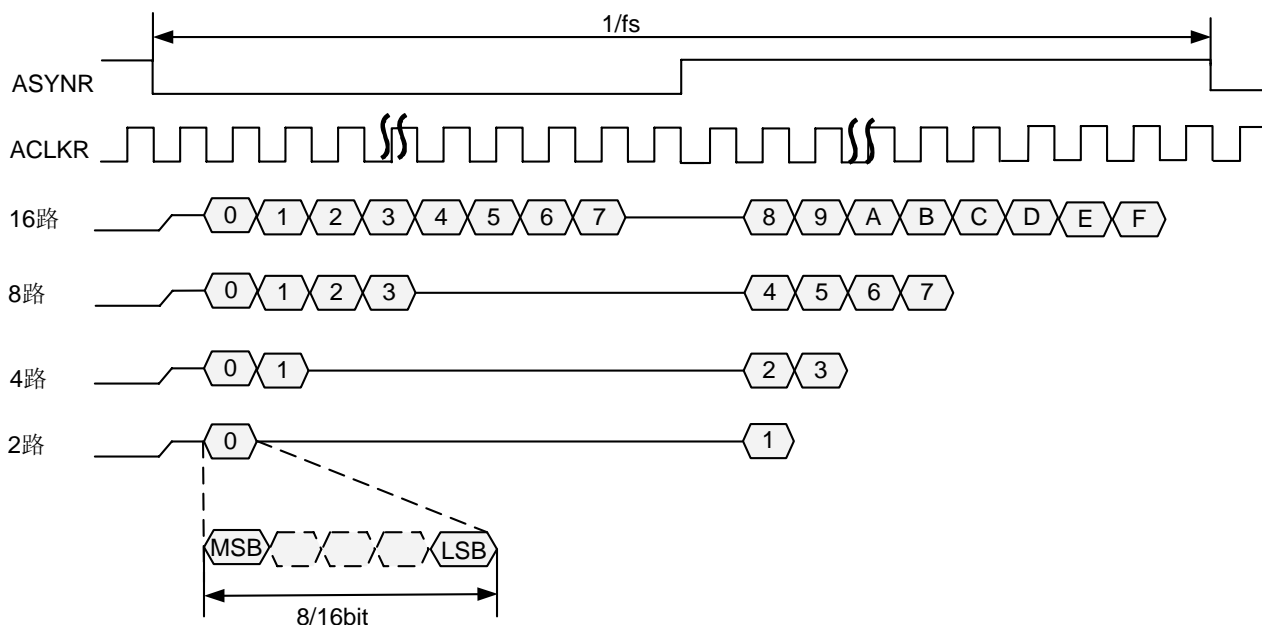
- AI、AO 通道排列

必须理解对接 CODEC 通道和 AI、AO 通道的对应关系，才能从正确的通道获取数据。

SIO 音频数据接收方面（即 AI 功能），Hi3520/Hi3515 SIO 在一个帧同步时钟内，最大能接收 16bit×16chn=256bit 的音频串行数据（I²S 和 PCM 都支持此规格）；如图 9-2 所示，通道的排列顺序与 SIO 取数据的顺序一致，即从高位到低位排列；SIO 实际取的数据多少由配置的通道路数和采样精度决定。



图9-2 I²S 2/4/8/16 路接收



SIO 音频数据发送方面（即 AO 功能），I2S 接口模式时，通道位宽支持配置为 8bit 或 16bit，左右声道最大总位宽为 64bit，例如可配置为 8bit*2chn、16bit*2chn、16bit*4chn 等模式；PCM 接口模式时，只支持单声道 8bit 或 16bit，但软件支持将 16bit 虚拟配置为 2 通道*8bit，因此软件可配置为 8bit*1chn、16bit*1chn 及 8bit*2chn 等模式。

- 音频编解码协议

Hi3520、Hi3515 SDK 音频部分的编解码功能基于独立封装的海思音频编解码库，核心编解码器工作在用户态，使用 CPU 软件编解码。

Hi3520、Hi3515 支持的音频编解码协议说明如表 9-1 所示。

表9-1 音频编解码协议说明

协议	采样率	帧长 (采样点)	码率 (kbps)	压缩率	CPU 消耗	描述
G711	8kHz	80/160/ 240/320/ 480	64	2	1/1 MHz	优点：语音质量最好；CPU 消耗小；支持广泛，协议免费。 缺点：压缩效率低。 G.711 提供 A 律与 μ 律压缩编码，适用于综合业务网和大多数数字电话链路。北美与日本通常采用 μ 律编码，欧洲和其他地区大都采用 A 律编码。
G726	8kHz	80/160/ 240/320/	16 、 24 、	8~3.2	5/5 MHz	优点：算法简单；语音质量高，多次转换后语音质量有保证，能够在低码率上达到网络等级的话



协议	采样率	帧长 (采 样 点)	码 率 (kbps)	压缩率	CPU 消耗	描述
		480	32、40			音质量。 缺点：压缩效率较低。 G726_16KBPS 与 MEDIA_G726_16KBPS 两种编码器区别在于编码输出的打包格式。G726_16KBPS 适用于网络传输；MEDIA_G726_16KBPS 适用于 ASF 存储。请参考 RFC3551.pdf。
ADPCM	8kHz	80/160/ 240/320/ 480	32	4	2/2 MHz	优点：算法简单；语音质量高，多次转换后语音质量有保证，能够在低码率上达到网络等级的语音质量。 缺点：压缩效率较低。 ADPCM_IMA 编码器需要多输入一个样点作为每帧编码的参考电平，IMA 编码每帧输入样点个数为 81/161/241/321/481。DVI 编码每帧输入样点个数为 80/160/240/320/480。
AMR-NB	8kHz	160	4.75 、 5.15 、 5.9 、 6.7 、 7.4 、 7.95 、 10.2 、 12.2	26.9~10 .5	40/9 MHz(VAD 1)	优点：语音质量好；压缩率高；支持广泛。 缺点：运算复杂，CPU 消耗大；需要收专利费。 打开 DTX，在静音情况下，可进一步降低采样率。
AAC Encoder	16kHz,22 .05kHz,2 4kHz,32k HZ, 44.1kHz, 48kHz	AACLC 支 持 1024 ； EAAC 和 EAACP LUS 支 持 2048	-	-	50 MHz	AAC 有两次突破性的技术升级： • aacPlus1（即 EAAC），增加 SBR(带宽扩展)技术，使得编解码器可以在比原来少一半的码率的条件下达到相同的音质。 • aacPlus2（即 EAACPLUS），增加 PS(参数立体声)技术，在低码率情况下得到极佳的音质效果，aacPlus2 可以在 48kbit/s 的速率下得到 CD 音质。 推荐码率设置如表 9-3 所示。



协议	采样率	帧长 (采样点)	码率 (kbps)	压缩率	CPU 消耗	描述
AAC Decoder	兼容全部速率	1024、2048	-	-	25 MHz	后向兼容。传统 AAC 解码器，仅解码 aac Plus v1 码流低频信息，而 aacPlus 解码器则可以同时还原高频信息。不支持 PS 的 AAC 解码器，解码 aac Plus v2 码流时，仅能得到单声道信息，而 aacPlus2 解码器则可以得到立体声声音。
AEC	8KHz	80、160	-	-	50 MHz	2 路语音输入，分别为远端语音和本地 MIC 采集语音，经过处理后可消除本地语音里的回声。 最大尾长为 960 个采样点 (120ms)，推荐打开 NLP 开关。

注：“cpu 消耗”的结果值基于 ARM9 288MHz 环境，2/2 MHz 表示编码和解码分别占有 2M 和 2M CPU。

表9-2 AAC Encoder 各协议情况下支持的码率设置（码率单位为 kbps）

采样率	LC BitRate	Plus v1 BitRate	Plus v2 BitRate
16kHz	16~96	Not supported	Not supported
22.05kHz	16~128	Not supported	Not supported
24kHz	16~128	Not supported	Not supported
32kHz	24~256	18~23	10~17
44.1kHz	32~320	24~51	12~35
48kHz	32~320	24~51	12~35

表9-3 AAC Encoder 各协议情况下推荐的码率设置（码率单位为 kbps）

采样率	LC BitRate	Plus v1 BitRate	Plus v2 BitRate
16kHz	48	Not supported	Not supported
22.05kHz	64	Not supported	Not supported
24kHz	64	Not supported	Not supported
32kHz	96	22	16
44.1kHz	128	48	24、32



采样率	LC BitRate	Plus v1 BitRate	Plus v2 BitRate
48kHz	128	48	24、32

- 海思语音帧结构

使用海思语音编解码库进行 G711、G726、ADPCM 格式的编码，编码后的码流遵循以下表格中描述的帧结构，即在每帧码流数据的净荷数据之前填充有 4 个字节的帧头；使用语音编解码库进行以上格式的解码时，需要读取相应的帧头信息。

表9-4 海思语音帧结构

参数位置(单位: HI_S16)	参数比特位说明	参数含义
0	[15:8]	数据帧类型标志位。 01: 语音帧; 其他: 保留。
	[7:0]	保留。
1	[15:8]	帧循环计数器: 0~255。
	[7:0]	数据净荷长度(单位: HI_S16)。
2	[15:0]	净荷数据。
3	[15:0]	净荷数据。
.....	[15:0]	净荷数据。
2+n-1	[15:0]	净荷数据。
2+n	[15:0]	净荷数据。

9.3 API 参考

9.3.1 音频输入

音频输入 (AI) 主要实现配置及启用音频输入设备、获取音频帧数据等功能。

该功能模块提供以下 MPI:

- [HI_MPI_AI_SetPubAttr](#): 设置 AI 设备属性。
- [HI_MPI_AI_GetPubAttr](#): 获取 AI 设备属性。
- [HI_MPI_AI_Enable](#): 启用 AI 设备。
- [HI_MPI_AI_Disable](#): 禁用 AI 设备。
- [HI_MPI_AI_EnableChn](#): 启用 AI 通道。



- [HI_MPI_AI_DisableChn](#): 禁用 AI 通道。
- [HI_MPI_AI_EnableAec](#): 启用回声抵消功能。
- [HI_MPI_AI_DisableAec](#): 禁用回声抵消功能。
- [HI_MPI_AI_EnableReSmp](#): 启用 AI 重采样。
- [HI_MPI_AI_DisableReSmp](#): 禁用 AI 重采样。
- [HI_MPI_AI_GetFd](#): 获取 AI 通道对应设备文件句柄。

HI_MPI_AI_SetPubAttr

【描述】

设置 AI 设备属性。

【语法】

```
HI_S32 HI_MPI_AI_SetPubAttr(AUDIO_DEV AudioDevId, const AIO\_ATTR\_S  
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围: [0, SIO_MAX_NUM)。	输入
pstAttr	AI 设备属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败, 其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_ILLEGAL_PARAM	输入参数无效。
HI_ERR_AI_NULL_PTR	空指针错误。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。



接口返回值	含义
HI_ERR_AI_NOT_PERM	操作非法。
HI_ERR_AI_NOT_SUPPORT	操作不支持。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 音频输入设备的属性决定了输入数据的格式，输入设备属性包括 SIO 工作模式、采样率、采样精度、buffer 大小、每帧的采样点数、扩展标志、时钟选择和通道数目。
 - 工作模式
SIO 输入输出目前支持 I²S 主模式、I²S 从模式、标准 PCM 主模式、标准 PCM 从模式、自定义 PCM 主模式以及自定义 PCM 从模式。
 - 采样率
采样率指一秒中内的采样点数，采样率越高表明失真度越小，处理的数据量也就随之增加。主模式下目前支持 8k、16k 以及 32k 的采样率配置，一般来说语音使用 8k 采样率，音频使用 32k 或以上的采样率；在从模式下，采样率由 codec 芯片决定。
 - 采样精度
采样精度指某个通道的采样点数据宽度，同时决定整个设备的通道分布。采样位宽可以设置为 8bit 或 16bit。
 - buffer 大小
AIO_ATTR_S 中的 u32FrmNum 项用于配置 AI 音频帧 Buffer 的大小，以帧个数为单位，建议配置为 5 以上，否则可能出现采集丢帧等异常。
 - 扩展标志
扩展标志表明在 8bit 采样精度的条件下是否需要将 8bit 数据进行 8bit 到 16bit 带符号扩展，扩展后获取的数据就为 16bit，以满足编码器需求。在设置非 8bit 采样精度的情况下，此标志是无效的。
 - 通道数目
通道数目指当前 SIO 设备的 AI 功能支持的最大通道数目，需与对接的 AUDIO CODEC 的配置保持一致；支持 2 路、4 路、8 路和 16 路。
 - 时钟选择
此项在 SIO 从模式和 SIO 主模式时的意义不同。AUDIO CODEC 提供时钟（即 SIO 从模式）时，如果能提供独立的 AD 和 DA 功能信号（即 RCK/RFS/XCK/XFS），那么将此项配置为 0，否则配置为 1；Hi3520/Hi3515 提供时钟（即 SIO 主模式）时，如果 ACKOUT 时钟来自于 SIO 0，则将此项配置为 0，如果 ACKOUT 时钟来自于 SIO 1 或 2，则将此项配置为 1。



- 在设置属性之前需要保证 AI 处于非启用状态，如果处于启用状态则需要首先禁用 AI 设备。
- 同一 SIO 下的 AI 和 AO 设备的主从模式、时钟选择应该一致，否则设置属性时会返回错误。
- 在从模式下，采样率的设置不起作用。
- SIO 主模式时，决定 SIO 输出时钟的关键配置项是采样率、采样精度以及通道数目，采样精度乘以通道数目即为 SIO 时序一次采样的位宽；
- SIO 主模式时，同一 SIO 下的 AI 和 AO 设备的时钟配置必须相同。
- AI 必须和 AD 配合起来才能正常工作，用户必须清楚 AD 采集的数据分布和通道的关系才能从正确的通道取得数据。

【举例】

下面的代码实现设置 AI 设备属性及启用 AI 设备。

```
HI_S32 s32ret;
AIO_ATTR_S stAttr;
AUDIO_DEV AiDevId = 1;

stAttr.enBitwidth = AUDIO_BIT_WIDTH_16;
stAttr.enSamplerate = AUDIO_SAMPLE_RATE_8000;
stAttr.enSoundmode = AUDIO_SOUND_MODE_MOMO;
stAttr.enWorkmode = AIO_MODE_I2S_SLAVE;
stAttr.u32EXFlag = 0;
stAttr.u32FrmNum = 5;
stAttr.u32PtNumPerFrm = 160;
stAttr.u32ChnCnt = 16;
stAttr.u32ClkSel = 0;

/* set public attribute of AI device*/
s32ret = HI_MPI_AI_SetPubAttr(AiDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
    printf("set ai %d attr err:0x%x\n", AiDevId,s32ret);
    return s32ret;
}
/* enable AI device */
s32ret = HI_MPI_AI_Enable(AiDevId);
if(HI_SUCCESS != s32ret)
{
    printf("enable ai dev %d err:0x%x\n", AiDevId, s32ret);
    return s32ret;
}
```

【相关主题】



无。

HI_MPI_AI_GetPubAttr

【描述】

获取 AI 设备属性。

【语法】

```
HI_S32 HI_MPI_AI_GetPubAttr(AUDIO_DEV AudioDevId, AIO_ATTR_S*pstAttr);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
pstAttr	AI 设备公共属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_NOT_CONFIG	音频输入设备属性未配置。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 获取的属性为前一次配置的属性。
- 如果从来没有配置过属性，则返回属性未配置的错误。



【举例】

```
HI_S32 s32ret;
AUDIO_DEV AiDevId = 1;
AIO_ATTR_S stAttr;

s32ret = HI_MPI_AI_GetPubAttr(AiDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
    printf("get ai %d attr err:0x%x\n", AiDevId,s32ret);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_AI_Enable

【描述】

启用 AI 设备。

【语法】

```
HI_S32 HI_MPI_AI_Enable(AUDIO_DEV AudioDevId);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。



接口返回值	含义
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AI_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 必须在启用前配置 AI 设备属性，否则返回属性未配置错误。
- 如果 AI 设备已经处于运行状态，则直接返回成功。

【举例】

请参见 [HI_MPI_AI_SetPubAttr](#) 的举例。

【相关主题】

无。

HI_MPI_AI_Disable

【描述】

禁用 AI 设备。

【语法】

```
HI_S32 HI_MPI_AI_Disable(AUDIO_DEV AudioDevId);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AI_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 如果 AI 设备已经处于非运行状态，则直接返回成功。
- 禁用 AI 设备前必须先禁用该设备下已启用的所有 AI 通道。

【举例】

```
HI_S32 s32ret;
AUDIO_DEV AiDevId = 1;

s32ret = HI_MPI_AI_Disable(AiDevId);
if(HI_SUCCESS != s32ret)
{
    printf("disable ai %d err:0x%x\n", AiDevId);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_AI_EnableChn

【描述】

启用 AI 通道。

【语法】

```
HI_S32 HI_MPI_AI_EnableChn(AUDIO_DEV AudioDevId, AI_CHN AiChn);
```

【参数】



参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	音频输入通道号。 支持的通道范围由 AI 设备属性中的最大通道个数 u32ChnCnt 决定。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_INVALID_CHNID	音频输入通道号无效。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AI_NOT_ENABLED	音频设备未启用。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

启用 AI 通道前，必须先启用其所属的 AI 设备，否则返回设备未启动的错误码。

【举例】

无。

【相关主题】

无。



HI_MPI_AI_DisableChn

【描述】

禁用 AI 通道。

【语法】

```
HI_S32 HI_MPI_AI_DisableChn(AUDIO_DEV AudioDevId, AI_CHN AiChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	音频输入通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_INVALID_CHNID	音频输入通道号无效。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AI_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

无。



【举例】

无。

HI_MPI_AI_EnableAec

【描述】

启用指定 AI 及 AO 的回声抵消功能。

【语法】

```
HI_S32 HI_MPI_AI_EnableAec(AUDIO_DEV AiDevId,  
AI_CHN AiChn, AUDIO_DEV AoDevId, AO_CHN AoChn);
```

【参数】

参数名称	描述	输入/输出
AiDevId	需要进行回声抵消的 AI 设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	需要进行回声抵消的 AI 通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入
AoDevId	用于回声抵消的 AO 设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	用于回声抵消的 AO 通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_INVALID_CHNID	音频输入通道号无效。
HI_ERR_AI_NOT_PERM	操作非法。



接口返回值	含义
HI_ERR_AI_NOT_SUPPORT	操作不支持。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 启用回声抵消前必须先启用相对应的 AI 设备。
- 成功启用回声抵消需要具备一定的条件：单声道模式，采样率为 8kHz，采样精度为 16bit，帧长为 80 或 160 个采样点，且 AI 和 AO 帧长必须相同。以上条件 AI 和 AO 都必须满足（但实际上本接口只检查 AI 的属性）。
- 多次启用相同 AI、AO 的回声抵消，则返回成功。

【举例】

```
HI_S32 s32ret;
AUDIO_DEV AiDevId = 1;
AI_CHN AiChn = 0;
AUDIO_DEV AoDevId = 0;
AO_CHN AoChn = 0;

s32ret = HI_MPI_AI_EnableAec(AiDevId, AiChn, AUDIO_DEV AoDevId, AO_CHN
AoChn);
if(HI_SUCCESS != s32ret)
{
    printf("enable aec err:0x%x\n", s32ret);
    return s32ret;
}

s32ret = HI_MPI_AI_DisableAec(AiDevId, AiChn)
if(HI_SUCCESS != s32ret)
{
    printf("disable aec err:0x%x\n", s32ret);
    return s32ret;
}
```

【相关主题】

无。



HI_MPI_AI_DisableAec

【描述】

禁用回声抵消功能。

【语法】

```
HI_S32 HI_MPI_AI_DisableAec(AUDIO_DEV AiDevId, AI_CHN AiChn);
```

【参数】

参数名称	描述	输入/输出
AiDevId	AI 设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	AI 通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_INVALID_CHNID	音频输入通道号无效。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AI_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h
- 库文件：libmpi.a

【注意】

重复调用本接口则返回成功。



【举例】

参考 [HI_MPI_AI_EnableAec](#) 的举例。

【相关主题】

无。

HI_MPI_AI_EnableReSmp

【描述】

启用 AI 重采样。

【语法】

```
HI_S32 HI_MPI_AI_EnableReSmp(AUDIO_DEV AudioDevId, AI_CHN AiChn,  
AUDIO\_RESAMPLE\_ATTR\_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	音频输入通道号。 实际支持的通道范围由采样精度决定。	输入
pstAttr	音频重采样属性结构体指针	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_INVALID_CHNID	音频输入通道号无效。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AI_ILLEGAL_PARAM	输入参数非法。



接口返回值	含义
HI_ERR_AI_NOMEM	内存不足。
HI_ERR_AI_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a libsample.a

【注意】

- 在启用 AI 通道之后，绑定 AI 通道之前，调用此接口启用重采样功能。
- 如果启用 AI 重采样功能，则在 AI 通道输出音频原始数据之前，内部将会先执行重采样处理，处理完后的数据再输出给绑定的 AO 或 AENC。
- 音频重采样属性包含以下项：
 - u32InPointNum：输入音频帧的每帧采样点个数。
 - enInSampleRate：输入音频帧的采样率。
 - enReSampleType：重采样类型，支持 2 到 1 倍、4 到 1 倍及 6 到 1 倍的重采样，例如 32kHz 降采样到 8kHz 或 16kHz 降采样到 8kHz。
- 重采样处理后音频帧的每帧采样点数目会发生变化，调用 HI_MPI_AI_SetPubAttr 接口时，音频公共属性中的 32PtNumPerFrm 项应该配置为重采样之前的值，本接口中重采样属性中的 u32InPointNum 项也配置为重采样之前的值。

例如 AI 从 32K 到 8K 重采样，重采样完成后输出给 AO 或 AENC 的每帧采样点个数将会是重采样前的四分之一，如果期望输出的采样点数为 320，那么公共属性中的 u32PtNumPerFrm 项则应配置为 $320 \times 4 = 1280$ ，重采样属性中的 u32InPointNum 项也配置成相同值。
- 不允许重复启用重采样功能，即在再次启用之前需要先将其禁用。

【举例】

以AI从32K到8K的重采样为例，配置如下：

```
/* dev attr of ai */
stAioAttr.u32ChnCnt = 2;
stAioAttr.enBitwidth = AUDIO_BIT_WIDTH_16;
stAioAttr.enSamplerate = AUDIO_SAMPLE_RATE_32000;
stAioAttr.enSoundmode = AUDIO_SOUND_MODE_MOMO;
stAioAttr.u32EXFlag = 1;
stAioAttr.u32FrmNum = 30;
stAioAttr.u32PtNumPerFrm = 320*4;

/* attr of resample */
stReSampleAttr.u32InPointNum = 320 * 4;
stReSampleAttr.enInSampleRate = AUDIO_SAMPLE_RATE_32000;
stReSampleAttr.enReSampleType = AUDIO_RESAMPLE_4X1;
```



HI_MPI_AI_DisableReSmp

【描述】

禁用 AI 重采样。

【语法】

```
HI_S32 HI_MPI_AI_DisableReSmp(AUDIO_DEV AudioDevId, AI_CHN AiChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	音频输入通道号。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_INVALID_CHNID	音频输入通道号无效。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a libsample.a

【注意】

不再使用 AI 重采样功能的话，应该调用此接口将其禁用。



HI_MPI_AI_GetFd

【描述】

获取音频输入通道号对应的设备文件句柄。

【语法】

```
HI_S32 HI_MPI_AI_GetFd(AUDIO_DEV AudioDevId ,AI_CHN AiChn)
```

【参数】

参数名称	描述	输入/输出
AiDevId	AI 设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	AI 通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

无。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h
- 库文件：libmpi.a

【注意】

无。

【举例】

```
HI_S32 s32AiFd;  
HI_S32 s32ret;  
AUDIO_DEV AiDevId = 1;  
AI_CHN AiChn = 0;  
  
s32AiFd = HI_MPI_AI_GetFd(AiDevId, AiChn);  
if(s32AiFd <= 0)  
{
```



```
        return HI_FAILURE;  
    }
```

【相关主题】

无。

9.3.2 音频输出

音频输出（AO）主要实现启用音频输出设备、发送音频帧到输出通道等功能。

该功能模块提供以下 MPI：

- [HI_MPI_AO_SetPubAttr](#)：设置 AO 设备属性。
- [HI_MPI_AO_GetPubAttr](#)：获取 AO 设备属性。
- [HI_MPI_AO_Enable](#)：启用 AO 设备。
- [HI_MPI_AO_Disable](#)：禁用 AO 设备。
- [HI_MPI_AO_EnableChn](#)：启用 AO 通道。
- [HI_MPI_AO_DisableChn](#)：禁用 AO 通道。
- [HI_MPI_AO_EnableReSmp](#)：启用 AO 重采样。
- [HI_MPI_AO_DisableReSmp](#)：禁用 AO 重采样。
- [HI_MPI_AO_BindAdec](#)：绑定 AO 通道和 ADEC 通道。
- [HI_MPI_AO_UnBindAdec](#)：解绑定 AO 通道和 ADEC 通道。
- [HI_MPI_AO_BindAi](#)：绑定 AO 通道和 AI 通道。
- [HI_MPI_AO_UnBindAi](#)：解绑定 AO 通道和 AI 通道。
- [HI_MPI_AO_PauseChn](#)：暂停 AO 通道。
- [HI_MPI_AO_ResumeChn](#)：恢复 AO 通道。
- [HI_MPI_AO_ClearChnBuf](#)：清除 ADEC 通道中当前的音频数据缓存。
- [HI_MPI_AO_GetFd](#)：获取 AO 通道对应的设备文件句柄。

HI_MPI_AO_SetPubAttr

【描述】

设置 AO 设备属性。

【语法】

```
HI_S32 HI_MPI_AO_SetPubAttr(AUDIO_DEV AudioDevId ,const AIO_ATTR_S  
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入



参数名称	描述	输入/输出
pstAttr	音频输出设备属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_ILLEGAL_PARAM	输出参数无效。
HI_ERR_AO_NULL_PTR	空指针错误。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ao.h
- 库文件：libmpi.a

【注意】

- 在设置属性之前需要保证 AO 处于非启用状态，如果处于启用状态则需要首先禁用 AO 设备。
- 同一 SIO 下的 AI 和 AO 设备的主从模式、时钟选择应该一致，否则设置属性时会返回错误。
- 扩展标志对 AO 设备无效。
- 在从模式下，采样率由 codec 决定，采样率的设置不起作用。
- SIO 主模式时，决定 SIO 输出时钟的关键配置项是采样率、采样精度以及通道数目，采样精度乘以通道数目即为 SIO 时序一次采样的位宽。
- SIO 主模式时，同一 SIO 下的 AI 和 AO 设备的时钟选择配置（即 AIO_ATTR_S 中的 u32ClkSel 项）必须相同。
- AO 必须和 DA 配合起来才能正常工作，用户必须清楚 DA 发送的数据分布和通道的关系才能从正确的通道发送数据。



- AO 设备属性结构体中其他项请参见 AI 模块中相关接口的描述。

【举例】

```
HI_S32 s32ret;
AIO_ATTR_S stAttr;
AUDIO_DEV AoDevId = 0;

stAttr.enBitwidth = AUDIO_BIT_WIDTH_16;
stAttr.enSamplerate = AUDIO_SAMPLE_RATE_8000;
stAttr.enSoundmode = AUDIO_SOUND_MODE_MOMO;
stAttr.enWorkmode = AIO_MODE_I2S_SLAVE;
stAttr.u32EXFlag = 0;
stAttr.u32FrmNum = 5;
stAttr.u32PtNumPerFrm = 160;
stAttr.u32ChnCnt = 2;
stAttr.u32ClkSel = 0;

/* set ao public attr*/
s32ret = HI_MPI_AO_SetPubAttr(AoDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
    printf("set ao %d attr err:0x%x\n", AoDevId,s32ret);
    return s32ret;
}
/* enable ao device*/
s32ret = HI_MPI_AO_Enable(AoDevId);
if(HI_SUCCESS != s32ret)
{
    printf("enable ao dev %d err:0x%x\n", AoDevId, s32ret);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_AO_GetPubAttr

【描述】

获取 AO 设备属性。

【语法】

```
HI_S32 HI_MPI_AO_GetPubAttr(AUDIO_DEV AudioDevId , AIO_ATTR_S *pstAttr);
```

【参数】



参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
pstAttr	音频输出设备属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_NOT_CONFIG	音频输出设备属性未配置。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_aio.h、mpi_ao.h
- 库文件：libmpi.a

【注意】

- 获取的属性为前一次配置的属性。
- 如果从未配置过属性，则返回属性未配置的错误。

【举例】

```
HI_S32 s32ret;  
AUDIO_DEV AoDevId = 0;  
AIO_ATTR_S stAttr;  
  
/* first enable ao device*/  
  
s32ret = HI_MPI_AO_GetPubAttr(AoDevId, &stAttr);  
if(HI_SUCCESS != s32ret)  
{
```



```
printf("get ao %d attr err:0x%x\n", AoDevId,s32ret);  
return s32ret;  
}
```

【相关主题】

无。

HI_MPI_AO_Enable

【描述】

启用 AO 设备。

【语法】

```
HI_S32 HI_MPI_AO_Enable(AUDIO_DEV AudioDevId);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ao.h
- 库文件：libmpi.a



【注意】

- 要求在启用前配置 AO 设备属性，否则会返回属性未配置的错误。
- 如果 AO 设备已经处于运行状态，则直接返回成功。

【举例】

请参见 [HI_MPI_AO_SetPubAttr](#) 的举例。

【相关主题】

无。

HI_MPI_AO_Disable

【描述】

禁用 AO 设备。

【语法】

```
HI_S32 HI_MPI_AO_Disable(AUDIO_DEV AudioDevId);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】



- 头文件：hi_comm_aio.h、mpi_ao.h
- 库文件：libmpi.a

【注意】

- 如果 AO 设备已经处于非运行状态，则直接返回成功。
- 禁用 AO 设备前必须先禁用设备下所有 AO 通道。

【举例】

无。

【相关主题】

无。

HI_MPI_AO_EnableChn

【描述】

启用 AO 通道。

【语法】

```
HI_S32 HI_MPI_AO_EnableChn(AUDIO_DEV AudioDevId, AI_CHN AoChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 支持的通道范围由 AO 设备属性中的最大通道个数 u32ChnCnt 决定。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。



接口返回值	含义
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_ENABLED	音频输出设备未启用。

【需求】

- 头文件：hi_comm_aio.h、mpi_ao.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

启用 AO 通道前，必须先启用其所属的 AO 设备，否则返回设备未启动的错误码。

【举例】

请参见 [HI_MPI_AI_SetPubAttr](#) 的举例。

【相关主题】

无。

HI_MPI_AO_DisableChn

【描述】

禁用 AO 通道。

【语法】

```
HI_S32 HI_MPI_AO_DisableChn(AUDIO_DEV AudioDevId, AI_CHN AoChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 实际支持的通道范围由采样精度决定。	输入

【返回值】

返回值	描述
0	成功。



返回值	描述
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_aio.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

HI_MPI_AO_EnableReSmp

【描述】

启用 AO 重采样。

【语法】

```
HI_S32 HI_MPI_AO_EnableReSmp(AUDIO_DEV AudioDevId, AO_CHN AoChn,  
AUDIO_RESAMPLE_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 实际支持的通道范围由采样精度决定。	输入



参数名称	描述	输入/输出
pstAttr	音频重采样属性结构体指针	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_ILLEGAL_PARAM	输入参数非法。
HI_ERR_AO_NOMEM	内存不足。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_aio.h、hi_comm_aio.h
- 库文件：libmpi.a libsample.a

【注意】

- 应该在启用 AO 通道之后，绑定 AO 通道之前，调用此接口启用重采样功能。
- 如果启用了 AO 重采样功能，则音频数据在发送给 AO 之前，内部先执行重采样处理，处理完成后再发送给 AO 通道进行播放。
- 音频重采样属性包含以下项：
 - u32InPointNum：输入音频帧的每帧采样点个数
 - enInSampleRate：输入音频帧的采样率
 - enReSampleType：重采样类型，支持 1 到 2 倍、1 到 4 倍及 1 到 6 倍的重采样
- 重采样处理会使音频数据的帧长（即每帧采样点个数）发生变化，音频重采样属性中的 u32InPointNum 项应该配置为重采样前的值，而音频公共属性中的 u32PtNumPerFrm 项应该配置为重采样后的值；例如绑定的 ADEC 通道中的解码



码流的帧长为 320，4X 重采样后帧长则为 $320 \times 4 = 1280$ ，所以公共属性中的 u32PtNumPerFrm 项配置为 320×4 ，而重采样属性中的 u32InPointNum 项配置为 320。

- 不允许重复启用重采样功能，即在再次启用之前需要先将其禁用。

【举例】

以ADEC到AO的解码回放8K到32K重采样为例，配置如下：

```
/* dev attr of ao */
stAioAttr.u32ChnCnt = 2;
stAioAttr.enBitwidth = AUDIO_BIT_WIDTH_16;
stAioAttr.enSamplerate = AUDIO_SAMPLE_RATE_32000;
stAioAttr.enSoundmode = AUDIO_SOUND_MODE_MOMO;
stAioAttr.u32EXFlag = 1;
stAioAttr.u32FrmNum = 30;
stAioAttr.u32PtNumPerFrm = 320*4;

/* attr of resample */
stReSampleAttr.u32InPointNum = 320;
stReSampleAttr.enInSampleRate = AUDIO_SAMPLE_RATE_8000;
stReSampleAttr.enReSampleType = AUDIO_RESAMPLE_1X4;
```

HI_MPI_AO_DisableReSmp

【描述】

禁用 AO 重采样。

【语法】

```
HI_S32 HI_MPI_AO_DisableReSmp(AUDIO_DEV AudioDevId, AO_CHN AoChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 实际支持的通道范围由采样精度决定。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。



【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_aio.h、mpi_aio.h、hi_comm_aio.h
- 库文件：libmpi.a libsample.a

【注意】

- 不再使用 AO 重采样功能的话，应该调用此接口将其禁用。

HI_MPI_AO_BindAdec

【描述】

绑定 AO 通道和 ADEC 通道。

【语法】

```
HI_S32 HI_MPI_AO_BindAdec(AUDIO_DEV AoDev, AO_CHN AoChn, ADEC_CHN AdChn);
```

【参数】

参数名称	描述	输入/输出
AoDev	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入
AdChn	音频解码通道号。 取值范围：[0, ADEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。



返回值	描述
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_aio.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 绑定 ADEC 和 AO 后，ADEC 的解码音频帧数据将自动发送到绑定的 AO 通道进行播放（当然前提是 ADEC 和 AO 通道都已经启用，并向 ADEC 通道送音频码流解码）。
- 一个 AO 通道只能绑定一个 ADEC 通道，并且一个 ADEC 通道也只能绑定一个 AO 通道。
- AO 通道既可以与 ADEC 通道绑定，也可以与 AI 通道绑定，但同一时刻只能绑定其中一个。

【举例】

无。

HI_MPI_AO_UnBindAdec

【描述】

解绑定 AO 通道和 ADEC 通道。

【语法】

```
HI_S32 HI_MPI_AO_UnBindAdec(AUDIO_DEV AoDev, AO_CHN AoChn, ADEC_CHN AdChn);
```

【参数】



参数名称	描述	输入/输出
AoDev	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入
AdChn	音频解码通道号。 取值范围：[0, ADEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ao.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

HI_MPI_AO_BindAi

【描述】



绑定 AO 通道和 AI 通道。

【语法】

```
HI_S32 HI_MPI_AO_BindAi(AUDIO_DEV AoDev, AO_CHN AoChn, AUDIO_DEV AiDev,  
AI_CHN AiChn);
```

【参数】

参数名称	描述	输入/输出
AoDev	音频输出设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入
AiDev	音频输入设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	音频输入通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ao.h、hi_comm_aio.h
- 库文件：libmpi.a



【注意】

- 绑定 AI 和 AO 后，AI 中采集的音频帧数据将自动发送到绑定的 AO 通道进行播放（当然前提是 AI 和 AO 通道都已经启用并正常工作）。
- 一个 AO 通道只能绑定一个 AI 通道，并且一个 AI 通道也只能绑定一个 AO 通道。
- AO 通道既可以与 ADEC 通道绑定，也可以与 AI 通道绑定，但同一时刻只能绑定其中一个。

【举例】

无。

HI_MPI_AO_UnBindAi

【描述】

解绑定 AO 通道和 AI 通道。

【语法】

```
HI_S32 HI_MPI_AO_UnBindAi(AUDIO_DEV AoDev, AO_CHN AoChn, AUDIO_DEV AiDev,
AI_CHN AiChn);
```

【参数】

参数名称	描述	输入/输出
AoDev	音频输出设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入
AiDev	音频输入设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	音频输入通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】



接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_aio.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

HI_MPI_AO_PauseChn

【描述】

暂停 AO 通道。

【语法】

```
HI_S32 HI_MPI_AO_PauseChn(AUDIO_DEV AudioDevId, AI_CHN AoChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 实际支持的通道范围由采样精度决定。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。



【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_ENABLED	音频输出设备未启用。

【需求】

- 头文件：hi_comm_aio.h、mpi_ao.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- AO 通道暂停后，如果继续向此通道发送音频帧数据，将会被阻塞。
- AO 通道为禁用状态时，不允许调用此接口暂停 AO 通道。

【举例】

无。

【相关主题】

无。

HI_MPI_AO_ResumeChn

【描述】

恢复 AO 通道。

【语法】

```
HI_S32 HI_MPI_AO_ResumeChn(AUDIO_DEV AudioDevId, AI_CHN AoChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 实际支持的通道范围由采样精度决定。	输入



【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ao.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- AO 通道暂停后可以通过调用此接口重新恢复，
- AO 通道为非暂停状态时不应该调用此接口。

【举例】

无。

HI_MPI_AO_ClearChnBuf

【描述】

清除 ADEC 通道中当前的音频数据缓存。

【语法】

```
HI_S32 HI_MPI_AO_ClearChnBuf(AUDIO_DEV AudioDevId ,AO_CHN AoChn);
```

【参数】



参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_ENABLED	音频输出通道未启用。

【需求】

- 头文件：hi_comm_aio.h、mpi_aio.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 在 AO 通道成功启用后再调用此接口。
- 为完全清除解码回放通路上所有缓存数据，此接口还应该与 [HI_MPI_ADEC_ClearChnBuf](#) 接口配合使用。

【举例】

无。

HI_MPI_AO_GetFd

【描述】

获取音频输出通道号对应的设备文件句柄。



【语法】

```
HI_S32 HI_MPI_AO_GetFd(AUDIO_DEV AudioDevId ,AO_CHN AoChn)
```

【参数】

参数名称	描述	输入/输出
AudioDevId	AO 设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	AO 通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

无。

【需求】

- 头文件：hi_comm_aio.h、mpi_ao.h
- 库文件：libmpi.a

【注意】

无。

【举例】

```
HI_S32 s32AoFd;
HI_S32 s32ret;
AUDIO_DEV AoDevId = 0;
AO_CHN AoChn = 0;

/* first enable ao device */

s32AoFd = HI_MPI_AO_GetFd(AoDevId, AoChn);
if(s32AoFd <= 0)
{
    return HI_FAILURE;
}
```



【相关主题】

无。

9.3.3 音频编码

音频编码主要实现创建编码通道、发送音频帧编码及获取编码码流等功能。

该功能模块提供以下 MPI：

- [HI_MPI_AENC_CreateChn](#)：创建音频编码通道。
- [HI_MPI_AENC_DestroyChn](#)：销毁音频编码通道。
- [HI_MPI_AENC_GetStream](#)：获取音频编码码流。
- [HI_MPI_AENC_ReleaseStream](#)：释放音频编码码流。

HI_MPI_AENC_CreateChn

【描述】

创建音频编码通道。

【语法】

```
HI_S32 HI_MPI_AENC_CreateChn(AENC_CHN AeChn, const AENC_CHN_ATTR_S
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
AeChn	通道号。 取值范围：[0, AENC_MAX_CHN_NUM)。	输入
pstAttr	音频编码通道属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效。
HI_ERR_AENC_NULL_PTR	空指针错误。



接口返回值	含义
HI_ERR_AENC_NOT_SUPPORT	不支持的属性设置或操作。
HI_ERR_AENC_EXIST	音频编码通道已经创建。
HI_ERR_AENC_NOMEM	系统内存不足。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_aenc.h、mpi_aenc.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 协议类型指定该通道的编码协议，目前支持 G711、G726、ADPCM 和 AMR 和 AAC，具体内容如表 9-1 所示。
- 表 9-1 中列举的编解码协议只支持 16bit 线性 PCM 音频数据处理，如果输入的是 8bit 采样精度的数据，AENC 内部会将其扩展为 16bit；另外，使用 Hi3520 AI 时，建议将扩展标志置为 1，使得 AI 数据由 8bit 自动扩展到 16bit。
- 海思语音帧结构如表 9-4 所示。
- 音频编码的部分属性需要与输入的音频数据属性相匹配，例如采样率、帧长（每帧采样点数目）等。
- buffer 大小以帧为单位，取值范围是[1, MAX_AUDIO_FRAME_NUM]，建议配置为 10 以上，过小的 buffer 配置可能导致丢帧等异常。
- 在通道闲置时才能使用此接口，如果通道已经被创建，则返回通道已经创建的错误。

【举例】

```
HI_S32 s32ret;
AENC_CHN_ATTR_S stAencAttr;
ADEC_ATTR_ADPCM_S stAdpcmAenc;
AENC_CHN AencChn = 0;
AUDIO_FRAME_S stAudioFrm;
AUDIO_STREAM_S stAudioStream;

stAencAttr.enType = PT_ADPCMA; /* ADPCM */
stAencAttr.u32BufSize = 8;
stAencAttr.pValue = &stAdpcmAenc;
stAdpcmAenc.enADPCMTType = ADPCM_TYPE_DVI4;

/* create aenc chn*/
s32ret = HI_MPI_AENC_CreateChn(AencChn, &stAencAttr);
if (HI_SUCCESS != s32ret)
{
    printf("create aenc chn %d err:0x%x\n", AencChn,s32ret);
}
```




```
        return s32ret;
    }

    /* bind AENC to AI channel */
    s32ret = HI_MPI_AENC_BindAi(AencChn, AiDev, AiChn);
    if (s32ret != HI_SUCCESS)
    {
        return s32ret;
    }

    /* get stream from aenc chn */
    s32ret = HI_MPI_AENC_GetStream(AencChn, &stAudioStream);
    if (HI_SUCCESS != s32ret )
    {
        printf("get stream from aenc chn %d fail \n", AencChn);
        return s32ret;
    }

    /* deal with audio stream */

    /* release audio stream */
    s32ret = HI_MPI_AENC_ReleaseStream(AencChn, &stAudioStream);
    if (HI_SUCCESS != s32ret )
    {
        return s32ret;
    }

    /* destroy aenc chn */
    s32ret = HI_MPI_AENC_DestroyChn(AencChn);
    if (HI_SUCCESS != s32ret )
    {
        return s32ret;
    }
}
```

【相关主题】

无。

HI_MPI_AENC_DestroyChn

【描述】

销毁音频编码通道。

【语法】

```
HI_S32 HI_MPI_AENC_DestroyChn(AENC_CHN AeChn);
```



【参数】

参数名称	描述	输入/输出
AeChn	通道号。 取值范围：[0, AENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效。
HI_ERR_AENC_UNEXIST	编码通道不存在。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_aenc.h、mpi_aenc.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 先创建完编码通道，再调用此接口，否则返回通道未创建。
- 如果正在获取/释放码流或者发送帧时销毁该通道，则会返回失败。

【举例】

请参见 [HI_MPI_AENC_CreateChn](#) 的举例。

【相关主题】

无。

HI_MPI_AENC_GetStream

【描述】

获取编码后码流。

【语法】



```
HI_S32 HI_MPI_AENC_GetStream(AENC_CHN AeChn, AUDIO_STREAM_S*pstStream ,
HI_U32 u32BlockFlag);
```

【参数】

参数名称	描述	输入/输出
AeChn	通道号。 取值范围：[0, AENC_MAX_CHN_NUM)。	输入
pstStream	获取的音频码流。	输出
u32BlockFlag	阻塞标识。 取值范围： HI_IO_BLOCK：阻塞。 HI_IO_NOBLOCK：非阻塞。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效。
HI_ERR_AENC_UNEXIST	音频编码通道不存在。
HI_ERR_AENC_NULL_PTR	空指针错误。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_aenc.h、mpi_aenc.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 必须创建通道后才可能获取码流，否则直接返回失败，如果在获取码流过程中销毁通道则会立刻返回失败。
- 支持阻塞或非阻塞方式获取码流，并且支持标准的 select 系统调用。



- 当阻塞方式获取码流时，如果音频数据 Buffer 空则此接口调用会被阻塞，直至 Buffer 中有新的数据或销毁 AENC 通道。
- 直接获取 AI 原始音频数据的方法
创建一路 AENC 通道，编码协议类型设置为 PT_LPCM，绑定 AI 通道后，从此 AENC 通道获取的音频数据即 AI 原始数据。

【举例】

请参见 [HI_MPI_AENC_CreateChn](#) 的举例。

【相关主题】

无。

HI_MPI_AENC_ReleaseStream

【描述】

释放从音频编码通道获取的码流。

【语法】

```
HI_S32 HI_MPI_AENC_ReleaseStream(AENC_CHN AeChn, const AUDIO_STREAM_S  
*pstStream);
```

【参数】

参数名称	描述	输入/输出
AeChn	通道号。 取值范围：[0, AENC_MAX_CHN_NUM)。	输入
pstStream	获取的码流指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效。
HI_ERR_AENC_UNEXIST	音频编码通道不存在。



接口返回值	含义
HI_ERR_AENC_NULL_PTR	空指针错误。

【需求】

- 头文件：[hi_comm_aio.h](#)、[hi_comm_aenc.h](#)、[mpi_aenc.h](#)
- 库文件：[libmpi.a](#) [lib_VoiceEngine.a](#) [lib_amr_spc.a](#) [lib_amr_fipop.a](#) [lib_aec.a](#)

【注意】

- 码流最好能够在使用完之后立即释放，如果不及时释放，会导致编码过程阻塞等待码流释放。
- 释放的码流必须是从该通道获取的码流，不得对码流信息结构体进行任何修改，否则会导致码流不能释放，使此码流 `buffer` 丢失，甚至导致程序异常。
- 释放码流时必须保证通道已经被创建，否则直接返回失败，如果在释放码流过程中销毁通道则会立刻返回失败。

【举例】

请参见 [HI_MPI_AENC_CreateChn](#) 的举例。

【相关主题】

无。

9.3.4 音频解码

音频解码主要实现创建解码通道、发送音频码流解码及获取解码后音频帧等功能。

该功能模块提供以下 MPI：

- [HI_MPI_ADEC_CreateChn](#)：创建音频解码通道。
- [HI_MPI_ADEC_DestroyChn](#)：销毁音频解码通道。
- [HI_MPI_ADEC_SendStream](#)：发送音频码流到音频解码通道。
- [HI_MPI_ADEC_ClearChnBuf](#)：清除 ADEC 通道中当前的音频数据缓存。

HI_MPI_ADEC_CreateChn

【描述】

创建音频解码通道。

【语法】

```
HI_S32 HI_MPI_ADEC_CreateChn(ADEC_CHN AdChn, ADEC\_CHN\_ATTR\_S *pstAttr);
```

【参数】



参数名称	描述	输入/输出
AdChn	通道号。 取值范围：[0, ADEC_MAX_CHN_NUM)。	输入
pstAttr	通道属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效。
HI_ERR_ADEC_NULL_PTR	空指针错误。
HI_ERR_ADEC_NOT_SUPPORT	不支持的属性设置或操作。
HI_ERR_ADEC_EXIST	音频解码通道已经创建。
HI_ERR_ADEC_NOMEM	系统内存不足。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_adec.h、mpi_adec.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 协议类型指定了该通道的解码协议，目前支持 G711、G726、ADPCM 和 AMR 和 AAC。
- 音频解码的部分属性需要与输出设备属性相匹配，例如采样率、帧长（每帧采样点数目）等。
- buffer 大小以帧为单位，取值范围是[0, MAX_AUDIO_FRAME_NUM]，建议配置为 10 以上，过小的 buffer 配置可能导致丢帧等异常。
- 在通道未创建前（或销毁后）才能使用此接口，如果通道已经被创建，则返回通道已经创建。

【举例】



```
HI_S32 s32ret;
ADEC_CHN_ATTR_S stAdecAttr;
ADEC_ATTR_ADPCM_S stAdpcm;
ADEC_CHN AdChn = 0;
AUDIO_STREAM_S stAudioStream;
AUDIO_FRAME_INFO_S stAudioFrameInfo;

stAdecAttr.enType = PT_ADPCMA;
stAdecAttr.u32BufSize = 8;
stAdecAttr.enMode = ADEC_MODE_STREAM;
stAdecAttr.pValue = &stAdpcm;
stAdpcm.enADPCMType = ADPCM_TYPE_DVI4;

/* create adec chn*/
s32ret = HI_MPI_ADEC_CreateChn(AdChn, &stAdecAttr);
if (s32ret)
{
    printf("create adnc chn %d err:0x%x\n", AdChn,s32ret);
    return s32ret;
}

/* get audio stream from network or file*/

/* send audio stream to adec chn */
s32ret = HI_MPI_ADEC_SendStream(AdChn, &stAudioStream);
if (s32ret)
{
    printf("send stream to adec fail\n");
    return s32ret;
}

/* get audio frame from adec */
s32ret = HI_MPI_ADEC_GetData(AdChn, &stAudioFrameInfo);
if (HI_SUCCESS != s32ret)
{
    printf("adec get data err\n");
    return s32ret;
}

/* send audio frame to AO or others */

/* release audio frame */
s32ret = HI_MPI_ADEC_ReleaseData(AdChn, &stAudioFrameInfo);
if (HI_SUCCESS != s32ret)
```



```
{  
    printf("adec release data err\n");  
    return s32ret;  
}  
  
/* destroy adec chn */  
s32ret = HI_MPI_ADEC_DestroyChn(AdChn);  
if (HI_SUCCESS != s32ret )  
{  
    return s32ret;  
}
```

【相关主题】

无。

HI_MPI_ADEC_DestroyChn

【描述】

销毁音频解码通道。

【语法】

```
HI_S32 HI_MPI_ADEC_DestroyChn(ADEC_CHN AdChn);
```

【参数】

参数名称	描述	输入/输出
AdChn	通道号。 取值范围：[0, ADEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效。



接口返回值	含义
HI_ERR_ADEC_UNEXIST	音频解码通道不存在。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_adec.h、mpi_adec.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 要求解码通道已经被创建，如果通道未被创建则返回通道未创建。
- 如果正在获取/释放码流或者发送帧，销毁该通道则这些操作都会立即返回失败。

【举例】

请参见 [HI_MPI_ADEC_CreateChn](#) 的举例。

【相关主题】

无。

HI_MPI_ADEC_SendStream

【描述】

向音频解码通道发送码流。

【语法】

```
HI_S32 HI_MPI_ADEC_SendStream(ADEC_CHN AdChn,  
                               const AUDIO\_STREAM\_S *pstStream, HI_U32 u32BlockFlag);
```

【参数】

参数名称	描述	输入/输出
AdChn	通道号。 取值范围：[0, ADEC_MAX_CHN_NUM)。	输入
pstStream	音频码流。	输入
u32BlockFlag	阻塞标识。 取值范围： HI_IO_BLOCK：阻塞。 HI_IO_NOBLOCK：非阻塞。	输入

【返回值】



返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效。
HI_ERR_ADEC_UNEXIST	音频解码通道号不存在。
HI_ERR_ADEC_NULL_PTR	空指针错误。
HI_ERR_ADEC_ILLEGAL_PARAM	输入参数无效。
HI_ERR_ADEC_DECODER_ERR	音频解码数据错误。

【需求】

- 头文件：[hi_comm_aio.h](#)、[hi_comm_adec.h](#)、[mpi_adec.h](#)
- 库文件：[libmpi.a](#) [lib_VoiceEngine.a](#) [lib_amr_spc.a](#) [lib_amr_fipop.a](#) [lib_aec.a](#)

【注意】

- 创建解码通道时可以指定解码方式为 pack 方式或 stream 方式。
 - pack 方式用于确定码流包为一帧的情况下，比如从 AENC 直接获取的码流，从文件读取确切知道一帧边界（语音编码码流长度固定，很容易确定边界）的码流，效率较高。
 - stream 方式用于不确定码流包为一帧的情况下，效率较低，且可能会有延迟。
- 发送数据时必须保证通道已经被创建，否则直接返回失败，如果在送数据过程中销毁通道则会立刻返回失败。
- 支持阻塞或非阻塞方式发送码流。
- 当阻塞方式发送码流时，如果音频数据 Buffer 满则此接口调用会被阻塞，直至取走音频数据或销毁 ADEC 通道。
- 确保发送给 ADEC 通道的码流数据的正确性，否则可能引起解码器异常退出。

【举例】

请参见 [HI_MPI_ADEC_CreateChn](#) 的举例。

【相关主题】

无。



HI_MPI_ADEC_ClearChnBuf

【描述】

清除 ADEC 通道中当前的音频数据缓存。

【语法】

```
HI_S32 HI_MPI_ADEC_ClearChnBuf(ADEC_CHN AdChn);
```

【参数】

参数名称	描述	输入/输出
AdChn	通道号。 取值范围：[0, ADEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效。
HI_ERR_ADEC_UNEXIST	音频解码通道不存在。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_adec.h、mpi_adec.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 要求解码通道已经被创建，如果通道未被创建则返回通道不存在错误码。
- 使用本接口时，不建议使用流式解码。使用流式解码进行清除缓存操作时，用户需要确保清除完缓存后，发送给解码器的数据必须是完整的一帧码流，否则可能导致解码器不能正常工作。
- 无论是否使用流式解码，都要确保送数据解码的操作和清除缓存的操作之间的同步。



【举例】

无。

【相关主题】

无。

9.4 数据类型

9.4.1 音频输入输出

音频输入输出相关数据类型、数据结构定义如下：

- [AUDIO_SAMPLE_RATE_E](#)：定义音频采样率。
- [AUDIO_BIT_WIDTH_E](#)：定义音频采样精度。
- [AIO_MODE_E](#)：定义音频输入输出工作模式。
- [AUDIO_SOUND_MODE_E](#)：定义音频声道模式。
- [AIO_ATTR_S](#)：定义音频输入输出设备属性结构体。
- [AUDIO_FRAME_S](#)：定义音频帧数据结构体。
- [AEC_FRAME_S](#)：定义回声抵消参考帧信息结构体。
- [AUDIO_FRAME_INFO_S](#)：定义音频帧信息结构体。
- [AUDIO_STREAM_S](#)：定义音频码流结构体。
- [AUDIO_RESAMPLE_TYPE_E](#)：定义音频重采样类型。
- [AUDIO_RESAMPLE_ATTR_S](#)：定义音频重采样属性配置结构体。
- [AMR_MODE_E](#)：定义 AMR 编解码速率模式。
- [AMR_FORMAT_E](#)：定义 AMR 编解码格式。
- [G726_BPS_E](#)：定义 G.726 编解码协议速率。
- [ADPCM_TYPE_E](#)：定义 ADPCM 编解码协议类型。
- [AAC_TYPE_E](#)：定义 AAC 音频编解码协议类型。
- [AAC_BPS_E](#)：定义 AAC 音频编码码率。

AUDIO_SAMPLE_RATE_E

【说明】

定义音频采样率。

【定义】

```
typedef enum hiAUDIO_SAMPLE_RATE_E
{
    AUDIO_SAMPLE_RATE_8000 = 8000,        /* 8kHz sampling rate */
    AUDIO_SAMPLE_RATE_11025 = 11025,      /* 11.025kHz sampling rate */
    AUDIO_SAMPLE_RATE_16000 = 16000,      /* 16kHz sampling rate */
}
```



```
AUDIO_SAMPLE_RATE_22050=22050,    /* 22.050kHz sampling rate */
AUDIO_SAMPLE_RATE_24000=24000,    /* 24kHz sampling rate */
AUDIO_SAMPLE_RATE_32000=32000,    /* 32kHz sampling rate */
AUDIO_SAMPLE_RATE_44100=44100,    /* 44.1kHz sampling rate */
AUDIO_SAMPLE_RATE_48000=48000,    /* 48kHz sampling rate */
}AUDIO_SAMPLE_RATE_E;
```

【成员】

成员名称	描述
AUDIO_SAMPLE_RATE_8000	8kHz 采样率。
AUDIO_SAMPLE_RATE_11025	11.025kHz 采样率。
AUDIO_SAMPLE_RATE_16000	16kHz 采样率。
AUDIO_SAMPLE_RATE_22050	22.050kHz 采样率。
AUDIO_SAMPLE_RATE_24000	24kHz 采样率。
AUDIO_SAMPLE_RATE_32000	32kHz 采样率。
AUDIO_SAMPLE_RATE_44100	44.1kHz 采样率。
AUDIO_SAMPLE_RATE_48000	48kHz 采样率。

【注意事项】

这里枚举值不是从 0 开始，而是与实际的采样率值相同。

【相关数据类型及接口】

[AIO_ATTR_S](#)

AUDIO_BIT_WIDTH_E

【说明】

定义音频采样精度。

【定义】

```
typedef enum hiAUDIO_BIT_WIDTH_E
{
    AUDIO_BIT_WIDTH_8    =0,    /* 8bit/sample */
    AUDIO_BIT_WIDTH_16   =1,    /* 16bit/sample */
    AUDIO_BIT_WIDTH_32   =2,    /* 32bit/sample */
    AUDIO_BIT_WIDTH_BUTT,
}AUDIO_BIT_WIDTH_E;
```

【成员】



成员名称	描述
AUDIO_BIT_WIDTH_8	采样精度为 8bit 位宽。
AUDIO_BIT_WIDTH_16	采样精度为 16bit 位宽。
AUDIO_BIT_WIDTH_32	采样精度为 32bit 位宽。

【注意事项】

无。

【相关数据类型及接口】

[AIO_ATTR_S](#)

AIO_MODE_E

【说明】

定义音频输入输出设备工作模式。

【定义】

```
typedef enum hiAIO_MODE_E
{
    AIO_MODE_I2S_MASTER = 0,          /* I2S master mode */
    AIO_MODE_I2S_SLAVE = 1,           /* I2S slave mode */
    AIO_MODE_PCM_SLAVE_STD,           /* SIO PCM slave standard mode */
    AIO_MODE_PCM_SLAVE_NSTD,          /* SIO PCM slave non-standard mode */
    AIO_MODE_BUTT
}AIO_MODE_E;
```

【成员】

成员名称	描述
AIO_MODE_I2S_MASTER	I ² S 主模式。
AIO_MODE_I2S_SLAVE	I ² S 从模式。
AIO_MODE_PCM_SLAVE_STD	PCM 从模式（标准协议）
AIO_MODE_PCM_SLAVE_NSTD	PCM 从模式（非标准协议）

【注意事项】

目前只支持 I²S 从模式。

【相关数据类型及接口】

[AIO_ATTR_S](#)



AUDIO_SOUND_MODE_E

【说明】

定义音频声道模式。

【定义】

```
typedef enum hiAIO_SOUND_MODE_E
{
    AUDIO_SOUND_MODE_MOMO    =0,        /*momo*/
    AUDIO_SOUND_MODE_STEREO =1,        /*stereo*/
    AUDIO_SOUND_MODE_BUTT
}AUDIO_SOUND_MODE_E;
```

【成员】

成员名称	描述
AUDIO_SOUND_MODE_MOMO	单声道。
AUDIO_SOUND_MODE_STEREO	双声道。

【注意事项】

无。

【相关数据类型及接口】

[AIO_ATTR_S](#)

AIO_ATTR_S

【说明】

定义音频输入输出设备属性结构体。

【定义】

```
typedef struct hiAIO_ATTR_S
{
    AUDIO_SAMPLE_RATE_E enSamplerate;        /*sample rate*/
    AUDIO_BIT_WIDTH_E    enBitwidth;          /*bitwidth*/
    AIO_MODE_E           enWorkmode;          /*master or slave mode*/
    AUDIO_SOUND_MODE_E   enSoundmode;         /*momo or steror*/
    HI_U32                u32EXFlag;          /*expand 8bit to 16bit */
    HI_U32                u32FrmNum;          /*frame num in buffer*/
    HI_U32                u32PtNumPerFrm;     /*number of samples*/
    HI_U32                u32ChnCnt;
    HI_U32                u32ClkSel;
}AIO_ATTR_S;
```



【成员】

成员名称	描述
enSamplerate	音频采样率（从模式下，此参数不起作用）。 静态属性。
enBitwidth	音频采样精度（从模式下，此参数必须和音频 AD/DA 的采样精度匹配）。 静态属性。
enWorkmode	音频输入输出工作模式。 静态属性。
enSoundmode	音频声道模式。 静态属性。
u32EXFlag	8bit 到 16bit 扩展标志（8bit 精度时有效）。 取值范围：{0, 1}。 <ul style="list-style-type: none">• 0：不扩展。• 1：扩展。 静态属性。
u32FrmNum	缓存帧数目。 取值范围：[1, MAX_AUDIO_FRAME_NUM]。 静态属性。
u32PtNumPerFrm	每帧的采样点个数。 取值范围：G711、G726、ADPCM_DVI4 编码时取值为 80、160、240、320、480；ADPCM_IMA 编码时取值为 81、161、241、321、481；AMR 编码时只支持 160。 静态属性。
u32ChnCnt	支持的最大通道数目。 取值：2、4、8、16。
u32ClkSel	时钟选择。 取值：0、1。 SIO 主模式时和从模式时，配置方法不同，请参考 HI_MPI_AI_SetPubAttr 接口描述中的说明。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_AI_SetPubAttr](#)



AUDIO_FRAME_S

【说明】

定义音频帧结构体。

【定义】

```
typedef struct hiAUDIO_FRAME_S
{
    AUDIO_BIT_WIDTH_E    enBitwidth; /*audio frame bitwidth*/
    AUDIO_SOUND_MODE_E    enSoundmode; /*audio frame momo or stereo mode*/
    HI_U8                aData[MAX_AUDIO_FRAME_LEN*2];
    HI_U64                u64TimeStamp; /*audio frame timestamp*/
    HI_U32                u32Seq;      /*audio frame seq*/
    HI_U32                u32Len;      /*data lenth per channel in frame*/
}AUDIO_FRAME_S;
```

【成员】

成员名称	描述
enBitwidth	音频采样精度。
enSoundmode	音频声道模式。
aData[MAX_AUDIO_FRAME_LEN*2]	实际音频帧数据。
u64TimeStamp	音频帧时间戳。 以 μs 为单位。
u32Seq	音频帧序号。
u32Len	音频帧长度。 以 byte 为单位。

【注意事项】

- u32Len（音频帧长度）指单个声道的数据长度。
- 单声道数据直接存放，采样点数为 ptnum，长度为 len；立体声数据按左右声道分开存放，先存放采样点为 ptnum、长度为 len 的左声道数据，然后存放采样点为 ptnum，长度为 len 的右声道数据。

【相关数据类型及接口】

无。

AEC_FRAME_S

【说明】



定义音频回声抵消参考帧信息结构体。

【定义】

```
typedef struct hiAEC_FRAME_S
{
    AUDIO_FRAME_S    stRefFrame;    /* aec reference audio frame */
    HI_BOOL           bValid;        /* whether frame is valid */
}AEC_FRAME_S;
```

【成员】

成员名称	描述
stRefFrame	回声抵消参考帧结构体。
bValid	参考帧有效的标志。 取值范围： HI_TRUE：参考帧有效。 HI_FALSE：参考帧无效，无效时不能使用此参考帧进行回声抵消。

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_FRAME_INFO_S

【说明】

定义音频帧信息结构体。

【定义】

```
typedef struct hiAUDIO_FRAME_INFO_S
{
    AUDIO_FRAME_S    *pstFrame;    /*frame ptr*/
    HI_U32            u32Id;        /*frame id*/
}AUDIO_FRAME_INFO_S;
```

【成员】

成员名称	描述
pstFrame	音频帧数据结构体指针。



成员名称	描述
u32Id	音频帧序号。 内部保留字，不能更改。

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_STREAM_S

【说明】

定义音频码流结构体。

【定义】

```
typedef struct hiAUDIO_STREAM_S
{
    HI_U8    *pStream;        /*stream buffer */
    HI_U32   u32Len;          /*stream lenth*/
    HI_U64   u64TimeStamp;    /*frame time stamp*/
    HI_U32   u32Seq;          /*frame seq,if stream is not a valid frame,
                               u32Seq is 0*/
}AUDIO_STREAM_S;
```

【成员】

成员名称	描述
pStream	音频码流数据指针。
u32Len	音频码流长度。以 byte 为单位。
u64TimeStamp	音频码流时间戳。
u32Seq	音频码流序号。

【注意事项】

无。

【相关数据类型及接口】

[HI_MPI_AENC_GetStream](#)



AUDIO_RESAMPLE_TYPE_E

【说明】

定义音频重采样类型。

【定义】

```
typedef enum hiAUDIO_RESAMPLE_TYPE_E
{
    AUDIO_RESAMPLE_1X2 = 0x1,
    AUDIO_RESAMPLE_2X1 = 0x2,
    AUDIO_RESAMPLE_1X4 = 0x3,
    AUDIO_RESAMPLE_4X1 = 0x4,
    AUDIO_RESAMPLE_1X6 = 0x5,
    AUDIO_RESAMPLE_6X1 = 0x6,
    AUDIO_RESAMPLE_BUTT
} AUDIO_RESAMPLE_TYPE_E;
```

【成员】

成员名称	描述
AUDIO_RESAMPLE_1X2	2 倍重采样。
AUDIO_RESAMPLE_2X1	1/2 重采样。
AUDIO_RESAMPLE_1X4	4 倍重采样。
AUDIO_RESAMPLE_4X1	1/4 重采样。
AUDIO_RESAMPLE_1X6	6 倍重采样。
AUDIO_RESAMPLE_6X1	1/6 重采样。

【注意事项】

无。

【相关数据类型及接口】

[AUDIO_RESAMPLE_ATTR_S](#)

[HI_MPI_AO_EnableReSmp](#)

AUDIO_RESAMPLE_ATTR_S

【说明】

定义音频重采样属性配置结构体。

【定义】

```
typedef struct hiAUDIO_RESAMPLE_ATTR_S
```



```
{
    HI_U32                u32InPointNum;
    AUDIO_SAMPLE_RATE_E   enInSampleRate;
    AUDIO_RESAMPLE_TYPE_E enReSampleType;
} AUDIO_RESAMPLE_ATTR_S;
```

【成员】

成员名称	描述
u32InPointNum	输入的每帧采样点个数。
enInSampleRate	输入的采样率。
enReSampleType	重采样类型。

【注意事项】

无。

【相关数据类型及接口】

[AUDIO_SAMPLE_RATE_E](#)

[AUDIO_RESAMPLE_TYPE_E](#)

AMR_MODE_E

【说明】

定义 AMR 编解码速率模式。

【定义】

```
typedef enum hiAMR_MODE_E
{
    AMR_MODE_MR475 = 0,
    AMR_MODE_MR515,
    AMR_MODE_MR59,
    AMR_MODE_MR67,
    AMR_MODE_MR74,
    AMR_MODE_MR795,
    AMR_MODE_MR102,
    AMR_MODE_MR122,
    AMR_MODE_MRDTX,
    AMR_MODE_N_MODES,
    AMR_MODE_BUTT
}AMR_MODE_E;
```

【成员】



成员名称	描述
AMR_MODE_MR475	可选速率，4.75 kbit/s。
AMR_MODE_MR515	可选速率，5.15 kbit/s。
AMR_MODE_MR59	可选速率，5.9 kbit/s。
AMR_MODE_MR67	可选速率，6.7 kbit/s。
AMR_MODE_MR74	可选速率，7.4 kbit/s。
AMR_MODE_MR795	可选速率，7.95 kbit/s。
AMR_MODE_MR102	可选速率，10.2 kbit/s。
AMR_MODE_MR122	可选速率，12.2 kbit/s。
AMR_MODE_MRDTX	静音模式（内部模式，不可选）。
AMR_MODE_N_MODES	基本速率数目（保留）。

【注意事项】

无。

【相关数据类型及接口】

略。

AMR_FORMAT_E

【说明】

定义 AMR 编解码格式。

【定义】

```
typedef enum hiAMR_FORMAT_E
{
    AMR_FORMAT_MMS,
    AMR_FORMAT_IF1,
    AMR_FORMAT_IF2,
    AMR_FORMAT_BUTT
}AMR_FORMAT_E;
```

【成员】

成员名称	描述
AMR_FORMAT_MMS	MMS 格式（推荐采用）。
AMR_FORMAT_IF1	IF1 格式。



成员名称	描述
AMR_FORMAT_IF2	IF2 格式。

【注意事项】

无。

【相关数据类型及接口】

略。

G726_BPS_E

【说明】

定义 G.726 编解码协议速率。

【定义】

```
typedef enum hiG726_BPS_E
{
    G726_16K = 0,
    G726_24K,
    G726_32K,
    G726_40K,
    MEDIA_G726_16K,    /* G726 16kbit/s for ASF */
    MEDIA_G726_24K,    /* G726 24kbit/s for ASF */
    MEDIA_G726_32K,    /* G726 32kbit/s for ASF */
    MEDIA_G726_40K,    /* G726 40kbit/s for ASF */
    G726_BUTT,
}G726_BPS_E;
```

【成员】

成员名称	描述
G726_16K	16kbit/s G.726，请参见“RFC3551 文档 4.5.4 G72616”。
G726_24K	24kbit/s G.726，请参见“RFC3551 文档 4.5.4 G72624”。
G726_32K	32kbit/s G.726，请参见“RFC3551 文档 4.5.4 G72632”。
G726_40K	40kbit/s G.726，请参见“RFC3551 文档 4.5.4 G72640”。
MEDIA_G726_16K	G726 16kbit/s for ASF。
MEDIA_G726_24K	G726 24kbit/s for ASF。
MEDIA_G726_32K	G726 32kbit/s for ASF。
MEDIA_G726_40K	G726 40kbit/s for ASF。



【注意事项】

无。

【相关数据类型及接口】

略。

ADPCM_TYPE_E

【说明】

定义 ADPCM 编解码协议类型。

【定义】

```
typedef enum hiADPCM_TYPE_E
{
    ADPCM_TYPE_DVI4 = 0,
    ADPCM_TYPE_IMA,
    ADPCM_TYPE_BUTT,
}ADPCM_TYPE_E;
```

【成员】

成员名称	描述
ADPCM_TYPE_DVI4	32kbit/s ADPCM(DVI4)。
ADPCM_TYPE_IMA	32kbit/s ADPCM(IMA)。

【注意事项】

无。

【相关数据类型及接口】

略。

AAC_TYPE_E

【说明】

定义 AAC 音频编解码协议类型。

【定义】

```
typedef enum hiAAC_TYPE_E
{
    AAC_TYPE_AACLIC      = 0,
    AAC_TYPE_EAAC        = 1,
```




```
AAC_TYPE_EAACPLUS    = 2,
AAC_TYPE_BUTT,
}AAC_TYPE_E;
```

【成员】

成员名称	描述
AAC_TYPE_AACLC	AACLC 格式。
AAC_TYPE_EAAC	eAAC 格式（也称为 HEAAC、AAC+或 aacPlusV1）。
AAC_TYPE_EAACPLUS	eAACPLUS 格式（也称为 AAC++或 aacPlusV2）。

【注意事项】

无。

【相关数据类型及接口】

略。

AAC_BPS_E

【说明】

定义 AAC 音频编码码率。

【定义】

```
typedef enum hiAAC_BPS_E
{
    AAC_BPS_16K = 16000,
    AAC_BPS_22K = 22000,
    AAC_BPS_24K = 24000,
    AAC_BPS_32K = 32000,
    AAC_BPS_48K = 48000,
    AAC_BPS_64K = 64000,
    AAC_BPS_96K = 96000,
    AAC_BPS_128K= 128000,
    AAC_BPS_BUTT
}AAC_BPS_E;
```

【成员】

成员名称	描述
AAC_BPS_16K	16kbit/s。
AAC_BPS_22K	22kbit/s。



成员名称	描述
AAC_BPS_24K	24kbit/s。
AAC_BPS_32K	32kbit/s。
AAC_BPS_48K	48kbit/s。
AAC_BPS_64K	64kbit/s。
AAC_BPS_96K	96kbit/s。
AAC_BPS_128K	128kbit/s。

【注意事项】

无。

【相关数据类型及接口】

略。

9.4.2 音频编码

音频编码相关数据类型、数据结构定义如下：

- [AENC_ATTR_AMR_S](#)：定义 AMR 编码协议属性结构体。
- [AENC_ATTR_G711_S](#)：定义 G.711 编码协议属性结构体。
- [AENC_ATTR_G726_S](#)：定义 G.726 编码协议属性结构体。
- [AENC_ATTR_ADPCM_S](#)：定义 ADPCM 编码协议属性结构体。
- [AENC_ATTR_AAC_S](#)：定义 AAC 编码协议属性结构体。
- [AENC_CHN_ATTR_S](#)：定义音频编码通道属性结构体。

AENC_ATTR_AMR_S

【说明】

定义 AMR 编码协议属性结构体。

【定义】

```
typedef struct hiAENC_ATTR_AMR_S
{
    AMR\_MODE\_E        enMode;
    AMR\_FORMAT\_E      enFormat;
    HI_S32             s32Dtx;
}AENC_ATTR_AMR_S;
```

【成员】



成员名称	描述
enMode	AMR 编码模式。
enFormat	AMR 编码格式。
s32Dtx	Dtx 启用标志。 取值范围：{0, 1} 0：不启用。 1：启用。

【注意事项】

无。

【相关数据类型及接口】

略。

AENC_ATTR_G711_S

【说明】

定义 G.711 编码协议属性结构体。

【定义】

```
typedef struct hiAENC_ATTR_G711_S
{
    HI_U32 resv;
}AENC_ATTR_G711_S;
```

【成员】

成员名称	描述
resv	待扩展用（目前暂未使用）。

【注意事项】

无。

【相关数据类型及接口】

略。

AENC_ATTR_G726_S

【说明】

定义 G.726 编码协议属性结构体。



【定义】

```
typedef struct hiAENC_ATTR_G726_S
{
    G726_BPS_E enG726bps;
}AENC_ATTR_G726_S;
```

【成员】

成员名称	描述
enG726bps	G.726 协议码率。

【注意事项】

无。

【相关数据类型及接口】

[G726_BPS_E](#)

AENC_ATTR_ADPCM_S

【说明】

定义 ADPCM 编码协议属性结构体。

【定义】

```
typedef struct hiAENC_ATTR_ADPCM_S
{
    ADPCM_TYPE_E enADPCMType;
}AENC_ATTR_ADPCM_S;
```

【成员】

成员名称	描述
enADPCMType	ADPCM 类型。

【注意事项】

无。

【相关数据类型及接口】

[ADPCM_TYPE_E](#)

AENC_ATTR_AAC_S

【说明】



定义 AAC 编码协议属性结构体。

【定义】

```
typedef struct hiAENC_ATTR_AAC_S
{
    AAC_TYPE_E          enAACType;
    AAC_BPS_E           enBitRate;
    AUDIO_SAMPLE_RATE_E enSmpRate;
    AUDIO_BIT_WIDTH_E    enBitWidth;
    AUDIO_SOUND_MODE_E   enSoundMode;
}AENC_ATTR_AAC_S;;
```

【成员】

成员名称	描述
enAACType	AAC 编码类型（Profile）。
enBitRate	编码码率。 取值范围： LC：48～128； EAAC：22～48； EAAC+：16～32； 以 kbit/s 为单位。
enSmpRate	音频数据的采样率。 取值范围： LC：16～48； EAAC：32～48； EAAC+：32～48。 以 kHz 为单位。
enBitWidth	音频数据采样精度，只支持 16bit。
enSoundMode	输入数据的声道模式。支持输入为单声道或双声道。

【注意事项】

无。

【相关数据类型及接口】

略。

AENC_CHN_ATTR_S

【说明】



定义音频编码通道属性结构体。

【定义】

```
typedef struct hiAENC_CHN_ATTR_S
{
    PAYLOAD_TYPE_E enType;
    HI_U32 u32BufSize; /*buffer size, 以帧为单位, [1~MAX_AUDIO_FRAME_NUM]*/
    HI_VOID *pValue;
}AENC_CHN_ATTR_S;
```

【成员】

成员名称	描述
enType	音频编码协议类型。 静态属性。
u32BufSize	音频编码缓存大小。 取值范围: [1, MAX_AUDIO_FRAME_NUM], 以帧为单位。 静态属性。
pValue	具体协议属性指针。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

[AENC_ATTR_AMR_S](#)

9.4.3 音频解码

音频解码相关数据类型、数据结构定义如下:

- [ADEC_ATTR_AMR_S](#): 定义 AMR 解码协议属性结构体。
- [ADEC_ATTR_G711_S](#): 定义 G.711 解码协议属性结构体。
- [ADEC_ATTR_G726_S](#): 定义 G.726 解码协议属性结构体。
- [ADEC_ATTR_ADPCM_S](#): 定义 ADPCM 编码协议属性结构体。
- [ADEC_ATTR_AAC_S](#): 定义 AAC 解码协议属性结构体。
- [ADEC_MODE_E](#): 定义解码方式。
- [ADEC_CHN_ATTR_S](#): 定义解码通道属性结构体。

ADEC_ATTR_AMR_S

【说明】



定义 AMR 解码协议属性结构体。

【定义】

```
typedef struct hiADEC_ATTR_AMR_S
{
    AMR_FORMAT_E    enFormat;
}ADEC_ATTR_AMR_S;
```

【成员】

成员名称	描述
enFormat	AMR 编码格式。

【注意事项】

无。

【相关数据类型及接口】

略。

ADEC_ATTR_G711_S

【说明】

定义 G.711 解码协议属性结构体。

【定义】

```
typedef struct hiADEC_ATTR_G711_S
{
    HI_U32    resv;
}ADEC_ATTR_G711_S;
```

【成员】

成员名称	描述
resv	待扩展用（目前暂未使用）。

【注意事项】

无。

【相关数据类型及接口】

略。



ADEC_ATTR_G726_S

【说明】

定义 G.726 解码协议属性结构体。

【定义】

```
typedef struct hiADEC_ATTR_G726_S
{
    G726_BPS_E enG726bps;
}ADEC_ATTR_G726_S;
```

【成员】

成员名称	描述
enG726bps	G.726 协议码率。

【注意事项】

无。

【相关数据类型及接口】

[G726_BPS_E](#)

ADEC_ATTR_ADPCM_S

【说明】

定义 ADPCM 编码协议属性结构体。

【定义】

```
typedef struct hiADEC_ATTR_ADPCM_S
{
    ADPCM_TYPE_E enADPCMType;
}ADEC_ATTR_ADPCM_S;
```

【成员】

成员名称	描述
enADPCMType	ADPCM 类型。

【注意事项】

无。

【相关数据类型及接口】



ADPCM_TYPE_E

ADEC_ATTR_AAC_S

【说明】

定义 AAC 解码协议属性结构体。

【定义】

```
typedef struct hiADEC_ATTR_AAC_S
{
    HI_U32 resv;
}ADEC_ATTR_AAC_S;
```

【成员】

成员名称	描述
resv	待扩展用（目前暂未使用）。

【注意事项】

无。

【相关数据类型及接口】

略。

ADEC_MODE_E

【说明】

定义解码方式。

【定义】

```
typedef enum hiADEC_MODE_E
{
    ADEC_MODE_PACK = 0, /*require input is valid dec pack(a
                           complete frame encode result),
                           e.g.the stream get from AENC is a
                           valid dec pack, the stream know actually
                           pack len from file is also a dec pack.
                           this mode is high-performative*/
    ADEC_MODE_STREAM, /*input is stream, low-performative,
                        if you couldn't find out whether a stream is
                        valid dec pack, you could use
                        this mode*/
    ADEC_MODE_BUTT
```



```
}ADEC_MODE_E;
```

【成员】

成员名称	描述
ADEC_MODE_PACK	pack 模式解码。
ADEC_MODE_STREAM	stream 模式解码。

【注意事项】

- pack 模式用于用户确认当前码流包为一帧数据编码结果的情况下，解码器会直接进行对其解码，如果不是一帧，解码器会出错。这种模式的效率比较高，在使用 AENC 模块编码的码流包如果没有破坏，均可以使用此方式解码。
- stream 模式用于用户不能确认当前码流包是不是一帧数据的情况下，解码器需要对码流进行判断并缓存，此工作方式的效率低下，一般用于读文件码流送解码或者不确定码流包边界的情况。当然由于语音编码码流长度固定，很容易确定在码流中的帧边界，推荐使用 pack 模式解码。

【相关数据类型及接口】

略。

ADEC_CHN_ATTR_S

【说明】

定义解码通道属性结构体。

【定义】

```
typedef struct hiADEC_CH_ATTR_S
{
    PAYLOAD_TYPE_E  enType;
    HI_U32           u32BufSize;
    ADEC_MODE_E     enMode;
    HI_VOID          *pValue;
}ADEC_CHN_ATTR_S;
```

【成员】

成员名称	描述
enType	音频解码协议类型。 静态属性。
u32BufSize	音频解码缓存大小。 取值范围：[1, MAX_AUDIO_FRAME_NUM]，以帧为单位。 静态属性。



成员名称	描述
enMode	解码方式。 静态属性。
pValue	具体协议属性指针。

【注意事项】

无。

【相关数据类型及接口】

略。

9.5 错误码

音频输入错误码

音频输入 API 错误码如表 9-5 所示。

表9-5 音频输入 API 错误码

错误代码	宏定义	描述
0xA0158001	HI_ERR_AI_INVALID_DEVID	音频输入设备号无效
0xA0158002	HI_ERR_AI_INVALID_CHNID	音频输入通道号无效
0xA0158003	HI_ERR_AI_ILLEGAL_PARAM	音频输入参数设置无效
0xA0158006	HI_ERR_AI_NULL_PTR	输入参数空指针错误
0xA0158007	HI_ERR_AI_NOT_CONFIG	音频输入设备属性未设置
0xA0158008	HI_ERR_AI_NOT_SUPPORT	操作不支持
0xA0158009	HI_ERR_AI_NOT_PERM	操作不允许
0xA015800B	HI_ERR_AI_NOT_ENABLED	音频输入设备或通道未启用
0xA015800C	HI_ERR_AI_NOMEM	分配内存失败
0xA015800D	HI_ERR_AI_NOBUF	音频输入缓存不足
0xA015800E	HI_ERR_AI_BUF_EMPTY	音频输入缓存为空
0xA015800F	HI_ERR_AI_BUF_FULL	音频输入缓存为满
0xA0158010	HI_ERR_AI_SYS_NOTREADY	音频输入系统未初始化



错误代码	宏定义	描述
0xA0158012	HI_ERR_AI_BUSY	音频输入系统忙

音频输出错误码

音频输出 API 错误码如表 9-6 所示。

表9-6 音频输出 API 错误码

错误代码	宏定义	描述
0xA0168001	HI_ERR_AO_INVALID_DEVID	音频输出设备号无效
0xA0168002	HI_ERR_AO_INVALID_CHNID	音频输出通道号无效
0xA0168003	HI_ERR_AO_ILLEGAL_PARAM	音频输出参数设置无效
0xA0168006	HI_ERR_AO_NULL_PTR	输出空指针错误
0xA0168007	HI_ERR_AO_NOT_CONFIG	音频输出设备属性未设置
0xA0168008	HI_ERR_AO_NOT_SUPPORT	操作不被支持
0xA0168009	HI_ERR_AO_NOT_PERM	操作不允许
0xA016800B	HI_ERR_AO_NOT_ENABLED	音频输出设备未启用
0xA016800C	HI_ERR_AO_NOMEM	系统内存不足
0xA016800D	HI_ERR_AO_NOBUF	音频输出缓存不足
0xA016800E	HI_ERR_AO_BUF_EMPTY	音频输出缓存为空
0xA016800F	HI_ERR_AO_BUF_FULL	音频输出缓存为满
0xA0168010	HI_ERR_AO_SYS_NOTREADY	音频输出系统未初始化
0xA0168012	HI_ERR_AO_BUSY	音频输出系统忙

音频编码错误码

音频编码 API 错误码如表 9-7 所示。

表9-7 音频编码 API 错误码

错误代码	宏定义	描述
0xA0178001	HI_ERR_AENC_INVALID_DEVID	音频设备号无效
0xA0178002	HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效



错误代码	宏定义	描述
0xA0178003	HI_ERR_AENC_ILLEGAL_PARAM	音频编码参数设置无效
0xA0178004	HI_ERR_AENC_EXIST	音频编码通道已经创建
0xA0178005	HI_ERR_AENC_UNEXIST	音频编码通道未创建
0xA0178006	HI_ERR_AENC_NULL_PTR	输入参数空指针错误
0xA0178007	HI_ERR_AENC_NOT_CONFIG	编码通道未配置
0xA0178008	HI_ERR_AENC_NOT_SUPPORT	操作不被支持
0xA0178009	HI_ERR_AENC_NOT_PERM	操作不允许
0xA017800C	HI_ERR_AENC_NOMEM	系统内存不足
0xA017800D	HI_ERR_AENC_NOBUF	编码通道缓存分配失败
0xA017800E	HI_ERR_AENC_BUF_EMPTY	编码通道缓存空
0xA017800F	HI_ERR_AENC_BUF_FULL	编码通道缓存满
0xA0178010	HI_ERR_AENC_SYS_NOTREADY	系统没有初始化
0xA0178040	HI_ERR_AENC_ENCODER_ERR	音频编码数据错误

音频解码错误码

音频解码 API 错误码如表 9-8 所示。

表9-8 音频解码 API 错误码

错误代码	宏定义	描述
0xA0188001	HI_ERR_ADEC_INVALID_DEVID	音频解码设备号无效
0xA0188002	HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效
0xA0188003	HI_ERR_ADEC_ILLEGAL_PARAM	音频解码参数设置无效
0xA0188004	HI_ERR_ADEC_EXIST	音频解码通道已经创建
0xA0188005	HI_ERR_ADEC_UNEXIST	音频解码通道未创建
0xA0188006	HI_ERR_ADEC_NULL_PTR	输入参数空指针错误
0xA0188007	HI_ERR_ADEC_NOT_CONFIG	解码通道属性未配置
0xA0188008	HI_ERR_ADEC_NOT_SUPPORT	操作不被支持
0xA0188009	HI_ERR_ADEC_NOT_PERM	操作不允许
0xA018800C	HI_ERR_ADEC_NOMEM	系统内存不足



错误代码	宏定义	描述
0xA018800D	HI_ERR_ADEC_NOBUF	解码通道缓存分配失败
0xA018800E	HI_ERR_ADEC_BUF_EMPTY	解码通道缓存空
0xA018800F	HI_ERR_ADEC_BUF_FULL	解码通道缓存满
0xA0188010	HI_ERR_ADEC_SYS_NOTREADY	系统没有初始化
0xA0188040	HI_ERR_ADEC_DECODER_ERR	音频解码数据错误



目 录

10 Proc 调试信息说明	10-1
10.1 概述	10-1
10.2 SYS	10-2
10.3 VB	10-3
10.4 LOG	10-4
10.5 VI	10-6
10.6 VO	10-10
10.7 DSU	10-16
10.8 VENC	10-20
10.9 GROUP	10-23
10.10 VPP	10-26
10.11 MD	10-28
10.12 VDEC	10-32
10.13 AI	10-35
10.14 AO	10-38
10.15 AENC	10-40
10.16 ADEC	10-41
10.17 CHNL	10-41
10.18 H264E	10-44
10.19 H264D	10-49
10.20 JPEGE	10-55



10 Proc 调试信息说明

10.1 概述

调试信息采用了 Linux 下的 proc 文件系统，可实时反映当前系统的运行状态，所记录的信息可供问题定位及分析时使用。

【文件目录】

/proc/umap

【文件清单】

文件名称	描述
sys	记录当前 SYS 模块的使用情况。
vb	记录当前 VB 模块的 buffer 使用情况。
ai	音频输入通道信息。
ao	音频输出通道信息。
log	记录当前各个模块的调试级别。内部调试用。
chnl	CHNL 模块状态。
dsu	专用缩放单元信息。
vdec	视频解码器信息。
venc	视频编码器信息。
vi	视频输入模块信息。
vo	视频输出模块信息。
group	当前编码通道组的属性配置以及当前编码通道统计状态。
vpp	当前所用的遮挡区域和叠加区域的属性信息和状态信息。
md	当前开启 MD 的编码通道的使用状况及其属性配置。



文件名称	描述
H264e	H.264 编码过程中，各通道的编码属性、状态以及历史信息统计。
H264d	H.264 解码过程中，各通道的解码属性、状态以及历史信息统计。
jpege	JPEG 编码过程中，各通道的编码属性、状态以及历史信息统计。

【信息查看方法】

- 在控制台上可以使用 cat 命令查看信息，例如 cat /proc/umap/venc；也可以使用其他常用的文件操作命令，例如 cp /proc/umap/ ./ -rf，将所有 umap 下的 proc 文件拷贝到当前目录。
- 在应用程序中可以将上述文件当作普通只读文件进行读操作，例如 fopen、fread 等。

Hi3520 与 Hi3515 显示的 proc 信息格式不同，Hi3520 部分模块会额外地显示主 ARM 操作系统中的 proc 信息（以 MASTER PROC 标示开始），而其从 ARM 操作系统中的 proc 信息格式则与 Hi3515 一致。



说明

10.2 SYS ~ 10.20 JPEG 的参数在描述时有以下 2 种情况需要注意：

- 取值为 {0, 1} 的参数，如未列出具体取值和含义的对应关系，则参数为 1 时表示肯定，为 0 时表示否定。
- 取值为 {aaa, bbb, ccc} 的参数，未列出具体取值和含义的对应关系，但可直接根据取值 aaa、bbb 或 ccc 判断参数含义

10.2 SYS

【调试信息】

```
cat /proc/umap/sys
System State: 0 (0: initialized; 1: exiting; 2: exited)
MPP Configuration:
Align      =      64
```

【调试信息分析】

记录当前 SYS 模块的使用情况。

【参数说明】

参数		描述
System State	initialized	初始化状态。
	exiting	正在退出状态。
	exited	退出状态。



参数		描述
MPP Configuration	Align	内存对齐长度(以字节为单位)。

10.3 VB

【调试信息】

```
cat /proc/umap/vb
.....Configuration of Video buffer .....
max count of pools: 128
configuration of common pools:
      0      1
size   884736 663552
count      0      10
.....common.....
POOL_ID  PHYS_ADDR  VIRT_ADDR  IS_COMM  BLK_SZ  BLK_CNT  FREE
1         0xe52e5000  0xc9c80000  1         165888  80        78
BLK  VIU  VOU  DSU  VENC  VDEC  MD  H264E  JPEGE  MPEGE  H264D  JPEGD  MPEGD  VPP  GRP  MPI
43   0    0    0    0    0    0  1      0    0    0    0    0    0    0    0
22   0    1    0    0    0    0  0      0    0    0    0    0    0    0    0

.....kernel.....
POOL_ID  PHYS_ADDR  VIRT_ADDR  IS_COMM  BLK_SZ  BLK_CNT  FREE
3         0xe6409000  0xcb000000  0         884736  4         3(3)
BLK  VIU  VOU  DSU  VENC  VDEC  MD  H264E  JPEGE  MPEGE  H264D  JPEGD  MPEGD  VPP  GRP  MPI
3    0    1    0    0    0    0  0      0    0    0    0    0    0    0    0
```

【调试信息分析】

记录当前 VB 模块的 buffer 使用情况。

【参数说明】

参数		描述
Configuration of Video buffer	max count of pools	最大的缓存池的个数。
	configuration of common pools:	公共缓存池的配置。
	size	缓存池内块的大小。
	count	缓存池内块的个数。
common	POOL_ID	公共缓存池的句柄。
	PHYS_ADDR	公共缓存池的开始物理地址。
	VIRT_ADDR	公共缓存池的开始逻辑地址。



参数		描述
	IS_COMM	是否公共缓存池。 取值：{0, 1}。
	BLK_SZ	公共缓存池内缓存块的大小。
	BLK_CNT	公共缓存池内缓存块的个数。
	FREE	公共缓存池空闲缓存块的个数。
	BLK	公共缓存池内缓存块的句柄。
	VIU/VOU/DSU/VENC/VDEC/MD/H264E/JPEGE/MPEGE/H264D/JPEGD/MPEGD/VPP/GRP/MPI	模块名。 下面对应的数字 0 或 1 表示是否占用相对应的公共缓存池内的该缓存块。 0: 没占用。 1: 占用。
kernel	POOL_ID	私有缓存池的句柄。
	PHYS_ADDR	私有缓存池的开始物理地址。
	VIRT_ADDR	私有缓存池的开始逻辑地址。
	IS_COMM	是否公共缓存池。 取值：{0, 1}。
	BLK_SZ	私有缓存池内缓存块的大小。
	BLK_CNT	私有缓存池内缓存块的个数。
	FREE	私有缓存池空闲缓存块的个数。
	BLK	私有缓存池内缓存块的句柄。
	VIU/VOU/DSU/VENC/VDEC/MD/H264E/JPEGE/MPEGE/H264D/JPEGD/MPEGD/VPP/GRP/MPI	模块名。 下面对应的数字 0 或 1 表示是否占用相对应的私有缓存池内的该缓存块。 0: 没占用。 1: 占用。

10.4 LOG

【调试信息】

```
~ $ cat /proc/umap/log
Log Buffer State: MaxLen ReadPos WritePos Buttpos
                64(KB)      0      7100  65536
```



Current Log Level:

```
cmpi    : 3
vb      : 3
sys     : 3
chnl    : 3
grp     : 3
venc    : 3
vpp     : 3
md      : 3
h264e   : 3
jpege   : 3
mpeg4e  : 3
vdec    : 3
h264d   : 3
jpegd   : 3
vo      : 3
vi      : 3
dsu     : 3
sio     : 3
ai      : 3
ao      : 3
aenc    : 3
adec    : 3
pciv    : 3
pcivfmw : 3
```

【调试信息分析】

记录当前各个模块的调试级别。

cat /proc/umap/log 用于获取 Hi3520 的从 ARM 或者 Hi3515 上的 log 级别信息；

cat /proc/umap/mstlog 用于获取 Hi3520 的主 ARM 上的 log 级别信息。

【参数说明】

参数		描述
Current Log Level:	cmpi	cmpi 模块名。
	vb	vb 模块名。
	sys	sys 模块名。
	chnl	chnl 模块名。
	grp	group 模块名。
	venc	venc 模块名。
	vpp	vpp 模块名。
	md	md 模块名。
	h264e	h264e 模块名。
	JPEGe	JPEGe 模块名。



参数		描述
	mpeg4e	mpeg4e 模块名。
	vdec	vdec 模块名。
	h264d	h264d 模块名。
	JPEGd	JPEGd 模块名。
	vo	vo 模块名。
	vi	vi 模块名。
	dsu	dsu 模块名。
	sio	sio 模块名。
	ai	ai 模块名。
	ao	ao 模块名。
	aenc	aenc 模块名。
	adec	adec 模块名。

10.5 VI

【调试信息】

```
~ $ cat /proc/umap/vi
-----MODULE PARAM-----
drop_err_frame
    0
-----VI DEV ATTR-----
  DevId InptMod WorkMod ViNorm ChrmChn ChrmSwp
    0   BT656   4d1
    1   BT656   4d1
    2   BT656   4d1
    3   BT656   4d1

-----VI CHN ATTR-----
Dev Chn  RectX  RectY  RectW  RectH  CapSel  DownScl  bHighP  PixFom  CrmRSmp
  0   0     8     0   704    288   both    n        n      sp420    n
  0   1     8     0   704    288   both    n        n      sp420    n
  0   2     8     0   704    288   both    n        n      sp420    n
  0   3     8     0   704    288   both    n        n      sp420    n

-----VI CHN MINOR ATTR-----
Dev Chn  RectX  RectY  RectW  RectH  CapSel  DownScl  bHighP  PixFom  CrmRSmp
  0   0     8     0   704    288   bottom  y        n      sp420    n
  0   1     8     0   704    288   bottom  y        n      sp420    n
  0   2     8     0   704    288   bottom  y        n      sp420    n
```



```
0 3 8 0 704 288 bottom y n sp420 n

-----FRAME RATE CONTROL INFO-----
Dev Chn SrcFrmR TarFrmR RelFrmR CapWid MinoEn FrmStat
0 0 25 7 7 0 y
0 1 25 7 7 0 y
0 2 25 7 7 0 y
0 3 25 7 7 0 y

-----VI CHN STATUS-----
Dev Chn IntCnt SendCnt Width Height Field NoSend VBFail BindVo
0 0 108 106 352 288 frm 2 0 (2,0)
0 1 108 106 352 288 frm 2 0 (2,1)
0 2 108 106 352 288 frm 2 0 (2,2)
0 3 108 106 352 288 frm 2 0 (2,3)

-----VI CHN OTHER INFO-----
Dev Chn BufCnt FrmTime UPicID TopLost BotLost SendTim
0 0 2 39999 -1 0 1 165
0 1 2 39996 -1 0 2 186
0 2 2 39995 -1 0 2 133
0 3 2 39996 -1 0 2 140

-----VI CHN OVERLAY INFO-----
Dev Chn RgnCnt Failcnt RunTime

-----VI CHN VBI INFO-----
Dev Chn Cascade Vbi0Loc Vbi0X Vbi0Y Vbi0L Vbi1Loc Vbi1X Vbi1Y Vbi1L
```

【调试信息分析】

记录当前视频输入设备及通道的属性配置以及状态信息。

【参数说明】

参数		描述
VI DEV ATTR 视频输入设备属性（请参考数据结构 VI_PUB_ATTR_S）	DevId	设备号。
	InptMod	输入模式。 取值：{BT656, BT601, DC, 720p}。
	WorkMod	工作模式。 取值：{1d1, 2d1, 4d1}。
	ViNorm	视频制式。 取值：{PAL, NTSC}。
	ChrmChn	是否色度通道。 取值：{Y, N}。
	ChrmSwp	是否色度翻转。 取值：{Y, N}。



参数		描述
VI CHN ATTR 视频输入通道属性（请参考数据结构 VI_CHN_ATTR_S）	Dev	VI 设备号。
	Chn	VI 通道号。
	RectX	捕获区域起始位置 X 坐标。
	RectY	捕获区域起始位置 Y 坐标。
	RectW	捕获区域宽度。
	RectH	捕获区域高度。
	CapSel	帧场选择。 取值：{top, bottom, both}。
	DownScl	是否水平压缩。 取值：{Y, N}。
	bHighP	是否高优先级。 取值：{Y, N}。
	PixFom	像素格式。 取值：{sp420, sp422}。
	CrmRSmp	是否色度重采样。 取值：{Y, N}。
VI CHN STATUS 视频输入通道基本状态	Dev	VI 设备号。
	Chn	VI 通道号。
	IntCnt	中断次数。 每次帧中断中取值增加 1。 如果 VI 工作不正常，此值可能不会增加。
	SendCnt	发送成功帧数。（发送给 VENC、VO、PCIV） 此值应该与 IntCnt 项基本匹配，否则说明有丢帧。
	Width	图像实际宽度。
	Height	图像实际高度。
	Field	图像帧场标志。 Frm：单场的逐行图像。 Intl：两场间插的隔行图像。
	NoSend	未成功发送帧数。 由 IntCnt 项减 SendCnt 项而得，取值如果为 1 或 2，属正常现象。



参数		描述
	VBFail	获取 VideoBuffer 失败次数。 VI 采集每帧图像前先获取通道图像大小的 VideoBuffer，如果失败，此值加 1；应该使此项取值为 0，否则说明相应的公共 VideoBuffer 配置不当。
	BindVo	绑定的 VO 设备和通道。
VI CHN OTHER INFO 视频输入通道其他信息	Dev	VI 设备号。
	Chn	VI 通道号。
	BufCnt	内部 buffer 个数。（内部调试用）
	FrmTime	视频帧采集时间间隔（例如 P 制为 40000us）。 以微秒为单位。 可以通过此项获知前端 AD 的采集帧率。例如 25fps PAL 制时，此值显示为 40000us 左右。
	UPicID	当前通道使用的用户图片序号。 如果未使用用户图片此值显示为 -1，使用原始用户图片则显示为 0，使用缩放后的内部用户图片则显示其他序号。
	TopLost	顶场中断丢失数目。 此值应该为 0，否则说明前端 AD 的对接有问题。
	BotLost	顶场中断丢失数目。 如果此项取值显示为 1 或 2，属正常现象，否则说明前端 AD 的对接有问题。
FRAME RATE CONTROL INFO 视频输入的帧率控制信息	SendTim	发送视频帧的时间统计（微秒为单位，内部调试用）。
	Dev	VI 设备号。
	Chn	VI 通道号。
	SrcFrmR	源帧率。 取值由接口 HI_MPI_VI_SetSrcFrameRate 配置。
	TarFrmR	目标帧率。 取值由接口 HI_MPI_VI_SetFrameRate 配置。
	MinoEn	是否启用 Minor 属性。 取值：{y, n}。



参数		描述
USER PIC INFO 用户图片信息 (设置过用户图片 才会显示)	UPicID	图片 ID。 0: 用户设置的原始图片; 其他: 内部根据通道图像大小自动缩放出来的用 户图片序号, 取值从 1 到 7。
	Width	图片宽度。
	Height	图片高度。
	Stride	图片 stride。
	PixForm	图片像素格式。 取值: {sp420, sp422}。
	PoolID	图片物理地址所在 PoolID。
	PhyAddr	图片物理地址。
	bUpdate	图片是否更新。 取值: {Y, N}。

10.6 VO

【调试信息】

```
$ cat /proc/umap/vo

[VOU] Version: [Hi3520_MPP_V1.0.0.0 Debug], Build Time[May 21 2010, 11:18:38]

-----MODULE PARAM-----
vo_power vo_2align
    0      0

-----DEV CONFIG-----
DevId  DevEn   Mux  OutMode      PipMode  BkClr  DevFrt  TgtFrt
    0      1     VGA  1280x1024@60   -1       0      60      25

-----DEV STATUS-----
DevId  VideoEn  FreeBuf  ScrnRpt  PixFmt   ResoH   ResoV   DispX   DispY   DispW   DispH
    0      1       8       1       422     1280    1152    0       0      1280    1024

-----DEV STATUS 2-----
DevId  VideoEn  EnChNum  DsuJob  LastBuf  VResoH  VResoV   Luma    Cont     Hue     Satu
    0      1       4       0       1       0       0      50      54      49      55

-----CASCADE INFO-----
DevId  CscdEn  bSlave  Pattern
    0      0       0       0
```



```
-----GRP STATUS-----
GrpId OnDev ChNum FulFr TgtFr Start Resrv GrpGap basePts

-----CHN INFO-----
DevId ChnId Prio bZoom DeFlk Field Revrs ChnGap PrePts ScalePts
  0    0    1    1    0  both    0  40000    0    0
  0    1    1    1    0  both    0  40000    0    0
  0    2    1    1    0  both    0  40000    0    0
  0    3    1    1    0  both    0  40000    0    0

-----CHN INFO 2-----
DevId ChnId bShow bSybf bSnap bCscd bZoom ZmFld ZoomX ZoomY ZoomW ZoomH SrcW SrcH
HFlt V1Flt VcFlt
0    0    1    1    0    0    0    4    0    0    0    0  720  576    0    0    0
0    1    1    1    0    0    0    4    0    0    0    0  720  576    0    0    0
0    2    1    1    0    0    0    4    0    0    0    0  720  576    0    0    0
0    3    1    1    0    0    0    4    0    0    0    0  720  576    0    0    0

-----CHN STATUS-----
DevId ChnId GetCnt FreeBuf NoBfCnt ChnRpt ChnFrt SndTyp StartX StartY Width Height
  0    0    82     6     0     2    25  both    0     0   640   576
  0    1    82     6     0     2    25  both   640    0   640   576
  0    2    82     6     0     2    25  both    0   576   640   576
  0    3    82     6     0     2    25  both   640   576   640   576
```

【调试信息分析】

记录当前 VO 的使用状况及其属性配置，包含设备状态，视频层状态，同步组状态以及通道状态。可用于动态获取当前 VO 的使用状态以便于调试或测试。

【参数说明】

参数		描述
MODULE PARAM	vo_power	功耗参数。调试使用。
	vo_2align	2 对齐开关。调试使用。
DEV CONFIG	DevId	设备 ID。 取值范围：[0, VO_MAX_DEV_NUM)
	DevEn	设备是否使能。 0：禁止；1：使能。
	Mux	接口类型。 取值范围：[0, VO_INTF_BUTT)
	OutMode	接口时序。 取值范围：[0, VO_OUTPUT_BUTT)
	PipMode	显示 buffer 模式。默认值：VO_DEFAULT_CHN (-1) 取值范围：[-1, VO_MAX_CHN_NUM)
	BkClr	设备背景色。RGB888 格式。



参数		描述
	DevFrt	设备帧率，即刷新率，与时序相关。
	TgtFrt	目标显示帧率，即 PIP 拼接帧率。 取值范围：（0, DevFrt]
DEV STATUS	DevId	设备 ID。 取值范围：[0, VO_MAX_DEV_NUM)
	VideoEn	视频层是否使能。 0：禁止；1：使能。
	FreeBuf	通道使能计数。显示 buffer 结点空闲数。
	ScrnRpt	屏幕重复次数。 该数值表示 vo 获取显示图像失败的次数。 例如：多画面输出时，该值如果一直增加，表明此时 DSU 输出较慢。这时最好关注一下 DSU 的负载，是否拼接任务或者图形处理任务过于繁重。
	PixFmt	输入图像像素格式。 只支持 PIXEL_FORMAT_YUV_SEMIPLANAR_422 和 PIXEL_FORMAT_YUV_SEMIPLANAR_420 两种。
	ResoH	图像水平分辨率。指 vo 的输入图像宽度，是静态属性。如果用户使用 PipMode 为默认模式，该参数会决定显示 buffer 的宽度。
	ResoV	图像垂直分辨率。指 vo 的输入图像高度，是静态属性。如果用户使用 PipMode 为默认模式，该参数会决定显示 buffer 的高度。
	DispX	显示区域起始点横坐标。
	DispY	显示区域起始点纵坐标。
	DispW	显示区域宽度。
	DispH	显示区域高度。
DEV STATUS 2	DevId	设备 ID。 取值范围：[0, VO_MAX_DEV_NUM)
	VideoEn	视频层是否使能。 0：禁止；1：使能。
	EnChNum	通道使能计数。即该设备有多少个通道处于使能状态。 取值范围：[0, VO_MAX_CHN_NUM)
	DsuJob	DSU 中任务计数。该数值表明 DSU 目前正在处理的 PIP 任务计数。



参数		描述
	LastBuf	last 队列长度。调试使用。
	VResoH	图像有效水平显示分辨率，是图像水平显示分辨率的子集。动态属性。该属性的设置主要是为了动态变更 vo 的输入图像大小，在多画面切换过程中不用关闭视频层。 默认值为 0。
	VResoV	图像有效垂直显示分辨率，是图像垂直显示分辨率的子集。动态属性。该属性的设置主要是为了动态变更 vo 的输入图像大小，在多画面切换过程中不用关闭视频层。 默认值为 0。
	Luma	亮度。 取值范围：[0, 100]
	Cont	对比度。 取值范围：[0, 100]
	Hue	色调。 取值范围：[0, 100]
	Satu	饱和度。 取值范围：[0, 100]
CASCAD E INFO (只有设备具有级联能力的情况下才显示该项)	DevId	设备 ID。 取值范围：[0, VO_MAX_DEV_NUM)
	CscdEn	级联使能。 0：禁止； 1：使能。
	bSlave	级联从模式。 0：主模式； 1：从模式。
	Pattern	级联布局。 取值范围：[0, 128]。
GRP STATUS	GrpId	同步组组号。 取值范围：[0, VO_SYNC_MAX_GRP)
	OnDev	同步组基准设备 ID。 取值范围：[0, VO_MAX_DEV_NUM)
	ChNum	同步组中通道个数。 取值范围：[0, VO_MAX_CHN_NUM)



参数		描述
	FulFr	同步组满帧率。和设备帧率保持一致。只读属性。
	TgtFr	同步组目标帧率。同步组的快进、慢放等都能从该数值体现出来。
	Start	同步组是否启动。 0：禁止；1：使能。
	Resrv	同步组是否反向播放。 0：禁止；1：使能。
	GrpGap	同步组帧间隔时间。与同步组目标帧率成反比。
	BasePts	同步组基准时间戳。控制同步组的同步起点，一旦设定后，同步组就从该时刻起进行同步，时间戳比它大的图像将被阻塞，比它小的图像将被跳过或显示。该值一般设置为文件中或者通道中的图像时间戳。
CHN INFO	DevId	设备 ID。 取值范围：[0, VO_MAX_DEV_NUM)
	ChnId	通道 ID。 取值范围：[0, VO_MAX_CHN_NUM)
	Prio	通道优先级。 取值范围：[0, VO_MAX_CHN_NUM)
	bZoom	通道是否缩放。目前该值无效。 0：禁止；1：使能。
	DeFlk	通道是否抗闪。隔行设备上，单场图像或者帧图像缩小时一般需要使能抗闪，否则都应设置为关闭。对于逐行输出设备该值都应该为禁止状态。 0：禁止；1：使能。
	Field	显示帧场信息。设置图像的帧场标识。 取值范围：[0, VO_FIELD_BUTT)
	bCscd	是否级联通道。 0：否；1：是。
	Revrs	是否反向播放。 0：禁止；1：使能。
	ChnGap	通道帧间隔。与通道帧率成反比。单位微秒。
	PrePts	上一帧时间戳。单位微秒。
	ScalePts	目标时间戳。下一帧要显示的时间戳。单位微秒。



参数		描述
CHN INFO 2	DevId	设备 ID。 取值范围：[0, VO_MAX_DEV_NUM)
	ChnId	通道 ID。 取值范围：[0, VO_MAX_CHN_NUM)
	bShow	通道是否显示。 0：隐藏；1：显示。
	BzBuf	busy 队列长度。通道中被占用的 buffer 结点数，不包括 LastBuf。 取值范围：[0, 8]
	bSnap	是否通道抓拍。 0：否；1：是。
	bZoom	是否局部放大。 0：否；1：是。
	ZmFld	局部放大帧场属性。主要用于级联时是否对该通道进行 DIE（去隔行）处理。 取值范围：[0, VIDEO_FIELD_BUTT)
	ZoomX	局部放大区域起始横坐标。如果是通过比例方式设置局部放大区域，那么该值为按比例处理、对齐处理后的源图像坐标。
	ZoomY	局部放大区域起始纵坐标。如果是通过比例方式设置局部放大区域，那么该值为按比例处理、对齐处理后的源图像坐标。
	ZoomW	局部放大区域宽度。如果是通过比例方式设置局部放大区域，那么该值为按比例处理、对齐处理后的源图像坐标。
	ZoomH	局部放大区域高度。如果是通过比例方式设置局部放大区域，那么该值为按比例处理、对齐处理后的源图像坐标。
	SrcW	通道源图像宽度。
	SrcH	通道源图像高度。
	FltTp	通道滤波器类型。即 DSU 滤波器类型。 取值范围：[0, FILTER_PARAM_TYPE_BUTT)
	HFlt	通道水平滤波系数。即 DSU 水平系数。 取值范围：[0, DSU_HSCALE_FILTER_BUTT)
	VIFlt	通道垂直亮度滤波系数。即 DSU 垂直亮度系数。 取值范围：[0, DSU_VSCALE_FILTER_BUTT)



参数		描述
	VcFlt	通道垂直色度滤波系数。即 DSU 垂直色度系数。 取值范围：[0, DSU_VSCALE_FILTER_BUTT)
CHN STATUS	DevId	设备 ID。 取值范围：[0, VO_MAX_DEV_NUM)
	ChnId	通道 ID。 取值范围：[0, VO_MAX_CHN_NUM)
	GetCnt	通道获取帧计数。如果该数值为 0，表示 vo 还没有收到图像数据；如果该数值一直不增加，表示 vo 接收不到图像了。
	FreeBuf	通道空闲 buffer 计数。该数值越小，说明通道图像积累越多，延时越大。解码一般通过通道 buffer 进行反压，所以解码时该数值一般较小。 取值范围：[0, 8]
	NoBfCnt	通道没 buffer 计数。通道满的情况下，该数值会增加，表示新来的图像获取不到 vo 通道 buffer。
	ChnRpt	通道重复显示帧计数。该数值越大，说明通道欠载越严重。即前端送图像较慢，常常导致通道队列中无图像。
	ChnFrt	通道帧率。通道播放控制可以通过该数值反映。
	SndTyp	通道收到的帧类型。即源的帧标识。 取值范围：[1, VIDEO_FIELD_BUTT)
	StartX	通道起始横坐标。
	StartY	通道起始纵坐标。
	Width	通道宽度。
	Height	通道高度。

10.7 DSU

【调试信息】

```
~ $ cat /proc/umap/dsu
[DSU] Version: [Hi3515_MPP_V1.0.0.0 Debug], Build Time[May 25 2010, 11:37:05]

-----MODULE PARAM-----
max_job_num
    40
-----RECENT JOB INFO-----
```



```
SeqNo  ModId  JobHdl  TaskNum  State  InSize  OutSize  CostTime
  0     VOU    13      4      ProcOk 1658880 414720  10533
  1     VOU    12      4      ProcOk 1658880 414720  10599
  2     VOU    11      4      ProcOk 1658880 414720  10583
  3     VOU    10      4      ProcOk 1658880 414720  10542
  4     VOU     9      4      ProcOk 1658880 414720  10573
  5     VOU     8      4      ProcOk 1658880 414720  10538
  6     VOU     7      4      ProcOk 1658880 414720  10605
  7     VOU     6      4      ProcOk 1658880 414720  10564

-----MAX WASTE TIME JOB INFO-----
SeqNo  ModId  JobHdl  TaskNum  State  InSize  OutSize  CostTime
  8     VOU    23      4      ProcOk 1658880 414720  11005

-----DSU JOB STATUS-----
Success    Fail    Cancel  AllJobNd    FreeNd    BeginNd    EndJobNd    ProcNd
 1294         0         0        40         39         0         0         1

-----DSU TASK STATUS-----
Success    Fail    Cancel  AllTaskNd    FreeNd    BusyNd
10349         0         0       320        312         8

-----DSU INT STATUS-----
IntNum    IntTm  UserCbTm  FifoDepth  HalProcTm
 1294        52        38         1        314

-----DSU MEM REQ STATUS -----
ReqOk    FreeOk    ReqFail    FreeFail
  0         0         0         0
```

【调试信息分析】

记录 DSU 模块最近完成的若干任务、最近耗时最大的任务、历史累计信息及中断相关信息。

【参数说明】

参数		描述
MODULE PARAM	max_job_num	最大 job 数。默认为 40。 仅当需要调整 DSU 支持的 job 数，才需要在插入模块时指定该值。
RECENT JOB INFO 最近完成的 job 的信息	SeqNo	打印序号。 取值范围：[0, 7]
	ModId	提交该 job 的模块名。
	JobHdl	该 job 的 handle 号，调试使用。 取值范围：[0, 39]
	TaskNum	该 job 包含的 task 数目。 取值范围：[0, 320)



参数		描述
	State	<p>该 job 的处理状态。</p> <p>取值范围：{ProcNok、Procok、Procing}</p> <p>ProcNok：表示 job 执行失败</p> <p>Procok：表示 job 执行成功</p> <p>Procing：表示 job 正在硬件处理</p>
	InSize	<p>该 job 下各 task 的输入图像面积之和</p> <p>单位：像素</p> <p>每向该 job 添加一个 task，此项就加上该 task 的输入面积。</p> <p>DSU 缩放性能主要受图像输入、输出面积的影响。一般，Job 的总输入/输出图像面积（InSize/OutSize）越大，处理该任务的耗时（CostTime）就越长。</p>
	OutSize	<p>该 job 下各 task 的输出图像面积之和</p> <p>单位：像素</p> <p>每向该 job 添加一个 task，此项就加上该 task 的输出面积。</p>
	CostTime	<p>该 job 从提交到完成的耗时长。</p> <p>单位：us。</p> <p>该时间包括针对该任务的软件、硬件及中断服务程序处理时间。</p> <p>如果 DSU 性能不足（如提交缩放的任务太多，超出性能限制），可能该值会超过预期。如 PAL 制预览业务，一个 Job 的完成时间超过 40ms，则会导致预览图像卡顿。此情况下，可检查业务是否超过 DSU 性能。</p>
MAX WASTE TIME JOB INFO 最近耗时最大的 job 信息	各项同 RECENT JOB INFO 的成员	<p>最近 500 个任务中耗时最长的 job 的信息。</p> <p>其各项同 RECENT JOB INFO 的成员，具体意义请参见前述。</p> <p>当出现耗时更长的任务或任务总数已超过 500 时，就更新该组值。</p> <p>通过该组值可知最近的 dsu 运行性能，以及是否出现过 DSU 处理不及时的情况。</p>
DSU JOB STATUS DSU 任务状态	Success	<p>累计成功处理的 job 数。</p> <p>当硬件处理成功时加 1。</p>
	Fail	<p>累计处理失败的 job 数。</p> <p>当 DSU 提交任务给驱动层并失败时加 1。该值增加时可通过察看日志了解失败原因。</p>



参数		描述
	Cancel	累计的用户主动取消的 job 数。 当用户调用 <code>cancelJob</code> 接口时加 1，如向 job 添加 task 时失败，用户会 <code>cancelJob</code> 。该值增加时可通过察看日志了解失败原因。
	AllJobNd	DSU 任务节点总数，同 <code>max_job_num</code> 值，一般为 40。
	FreeNd	空闲的 job 节点数。
	BeginNd	用户已创建任务但还未提交（ <code>EndJob</code> 接口）的 job 数。
	EndJobNd	用户已提交（ <code>EndJob</code> 接口）但还未提交给硬件处理的任务数。
	ProcNd	正在进行硬件处理的任务数。
DSU TASK STATUS DSU Task 状态	Success	累计成功处理的 Task 数。 一个 job 包含 1 到多个缩放 task，所以 1 个 job 成功表明其下的多个 task 都成功，故该值累加的比 job 的 <code>Success</code> 项更快。 当硬件处理一个 Job 成功时，该值累加 job 下的 task 数。
	Fail	累计处理失败的 Task 数。 一个 job 失败将导致其下的所有 task 失败。当 Job 执行失败时，该项累加其下失败的 task 数。 该值增加时可通过察看日志了解失败原因。
	Cancel	累计的用户主动 Cancel 的 Task 数。 用户调用 <code>cancelJob</code> 接口时，即取消了 job 下的所有 task 的执行，该项累加。 该值增加时可通过察看日志了解失败原因。
	AllJobNd	DSU Task 节点总数，一般为 320。
	FreeNd	空闲的 Task 节点数。
	BusyNd	已添加到 Job 下的 task 数。
DSU INT STATUS DSU 中断 状态	IntNum	DSU 中断个数。 硬件完成一个 job，该值加 1。
	IntTm	DSU 中断处理时间。 单位：us 包含 DSU 中断处理和回调用户的时间。



参数		描述
	UserCbTm	DSU 中断中回调用户的时间。 单位：us 如果该值过大，表明用户的回调处理程序执行慢，可能会影响 DSU 处理性能。
	FifoDepth	硬件处理深度，内部调试使用。
	HalProcTm	将 Job 提交给驱动层的时间，内部调试使用。 单位：us
DSU MEM REQ STATUS DSU 内存 申请状态	ReqOk	申请 VB 成功次数。 当申请成功时加 1。 注意，用户使用 DSU 的 OSD 叠加功能或将视频转换到 PACKAGE 格式时，DSU 才申请/释放内存。
	FreeOk	释放 VB 成功次数。 当释放成功时加 1。一般应与 ReqOk 项相同，表明没有内存泄露。
	ReqFail	申请 VB 失败次数。 当申请失败时加 1，一般应为 0。
	FreeFail	释放 VB 失败次数。 当释放失败时加 1，一般应为 0。

10.8 VENC

【调试信息】

```
~ $ cat /proc/umap/venc
----- info from slave arm -----
Version: [Hi3520_MPP_V1.0.0.0 Debug], Build Time[Nov 27 2009, 14:04:42]

-----VENC CHN ATTR-----
NO.  W   H   type  Fd  ViFd  Wm  Mn  SFm  Rx   Seq   Rg   LP   LB   Pk
0   352  288  96   0   0     0   1   1   1   1cce41  1   0   21261  0
1   176  144  96   0   0     0   1   1   1   1cd801  1   0   0      0
2   352  288  96   0   0     0   1   1   1   1ccf24  1   0   31431  0
3   176  144  96   0   0     0   1   1   1   1cd64d  1   0   0      0

-----VENC CHN STATE-----
NO.  GetSmFd  Notify  RlFrm  FrmF  FrmB  FrmU  UserGet  userRls  CFrmU
0    2922    3205    26    75    26    0    1887809  1887809    0
1    451     1383    25   112    1     0    1890305  1890305    0
2   2696     3199    26    80    25    0    1888036  1888036    0
3    888     1326    25   113    0     0    1889869  1889869    0
```



```
=====MASTER PROC=====
Version: [Hi3520_MPP_V1.0.0.0 Debug], Build Time[Dec 1 2009, 17:59:38]
ID      ByFrm  state  CbSize  LftPic  LftStrm
0       YES    CREATED 10816  0        0
1       YES    CREATED 128    0        0
```

【调试信息分析】

记录当前编码通道的使用状况及其属性配置，最多有 64 路编码通道。可用于检查属性配置以及当前编码通道统计状态。

【参数说明】

参数		描述
VENC CHN ATTR	NO.	编码通道号。
	W	编码通道宽度。
	H	编码通道高度。
	type	编码类型。
	Fd	帧场编码标识。 0: 帧编码; 1: 场编码。
	ViFd	输入图像的帧场标识。 0: 帧模式; 1: 场模式。
	Wm	是否开启数字水印。 取值: {0, 1}。
	Mn	主次码流标识。 0: 次码流; 1: 主码流。
	SFm	按帧获取码流标识。 0: 按包获取; 1: 按帧获取。
	Rx	阻塞或非阻塞获取码流标识。 0: 阻塞; 1: 非阻塞。
	Seq	序列号。 按帧获取时为帧序列号, 按包获取时为包序列号。



参数		描述
	Rg	是否注册到通道组中。 取值：{0, 1}。
	LP	Left Picture，待编码的图像数。
	LB	Left Byte，码流 buff 剩余的 byte 数。
	Pk	Pack，当前帧的码流包个数。
VENC CHN STATE	NO	编码通道号。
	GetSmFd	该通道获取码流失败的次数。
	Notify	暂无意义，用户不需关注。
	RlFrm	实际编码帧率。 统计一秒内编码器编了多少帧图像。 该值可用于估计编码器性能。
	FrmF	码流 buffer 中 FREE 节点的个数。
	FrmB	码流 buffer 中 BUSY 节点的个数。
	FrmU	码流 buffer 中 USER 节点的个数。 用户成功获取一帧码流后，该值加 1。 用户成功释放一帧码流后，该值减 1。
	UserGet	用户成功获取码流的码流包计数。
	userRls	用户成功释放码流的码流包计数。
MASTER PROC	CFrmU	无意义，保留。
	ID	编码通道号。
	ByFrm	是否按帧获取码流。 取值：{0, 1}。
	state	通道状态。 INIT：初始状态； CREATED：创建状态； DESTORY：已销毁状态； WTDSTY：等待销毁状态。
	CbSize	主从 ARM 共享 buffer 的使用情况。
	LftPic	等待编码图像数目。应与从 ARM LP 数据保持一致。
	LftStrm	Buffer 剩余码流 byte 数。应与从 ARM LB 数据保持一致。



10.9 GROUP

【调试信息】

```
~ $ cat /proc/umap/grp
```

```
Version: [Hi3520_MPP_V1.0.0.0 Debug], Build Time[Nov 27 2009, 14:04:11]
```

```
-----GROUP PUBLIC-----
MaxWidth    MaxHeight
      720         576
```

```
-----GROUP CHN ATTR-----
NO. num  W    H    Fd  VeS  VpS  Top  Die  St  Tim  Px  Wz  Hz  Zw  Zh  OsN  PicAdr
0  1    352  288  0   1   1   1   0   1   1   0  0  0  0  0  2    e5eeb000
1  1    176  144  0   1   1   1   0   1   1   0  0  0  0  0  2    e6075e00
```

```
-----GROUP CHN STATE-----
NO.   DsuAdd   DsuSub   DsuCrt   DsuInt   Gry  TD  Flt  SLM  HWus   Intus   SCus
0     98306   98306   49153   49153    0   1   0   1    2823   19025   44
1    4236302 4236302  2118151 2118151  0   1   0   1    21977  -1193   249
```

```
-----GROUP INFO-----
NO.   ViSend   GrpSend   GrpInq   InqOk   GrpStart   GrpCnVp   GrpInt
0     2118151  2118151  3780851  3780807  3780807   2118148   2118148
1     2118151  2118151  3780851  3780807  3780807   2118148   2118148
```

```
-----GROUP INFO 2-----
GNO.  Chn typ  SendPic  SendOk  Inq    Start   StartOk  Cfg      Int      Tr
0     0   96   2118151  2118150  3780828 3780807  2118148  2115451  2118148  0
1     1   96   2118151  2118150  3780827 3780807  2118148  2117729  2118148  411c1a
```

【调试信息分析】

记录当前编码通道组的属性配置以及当前编码通道统计状态。

【参数说明】

参数		描述
GROUP PUBLIC	Max_width	彩转灰支持的最大图像宽度。
	Max_height	彩转灰支持的最大图像高度。
GROUP CHN ATTR	NO.	GROUP 通道号。
	num	GROUP 内编码通道的个数。
	W	GROUP 通道内主码流编码通道宽度。
	H	GROUP 通道内主码流编码通道高度。



参数		描述
	Fd	GROUP 通道内主码流编码方式。 0: 帧编码; 1: 场编码。
	VeS	是否编码。 取值: {0, 1}。
	VpS	是否做视频前处理。 取值: {0, 1}。
	Top	顶底场标识。 0: 底场; 1: 顶场。主码流为帧编码时, 无效。
	Die	De-interlace 使能。 取值: {0, 1}。
	St	宏块动静判决使能。 取值: {0, 1}。
	Tim	时域滤波使能。 取值: {0, 1}。
	Px	缩放丢点使能。 取值: {0, 1}。
	Wz	水平缩放使能。 取值: {0, 1}。
	Hz	垂直缩放使能。 取值: {0, 1}。
	Zw	水平缩放倍数。
	Zh	垂直缩放倍数。
	OsN	该编码通道组里 OSD 区域的数目。
	PicAdr	编码图像的内存地址 (内部调试用)。
GROUP CHN STATE	NO.	GROUP 通道号。
	DsuAdd	内部 DSU 使用 VB 的次数 (内部调试用)。
	DsuSub	内部 DSU 释放 VB 的次数 (内部调试用)。
	DsuCrt	创建 DSU 任务的次数 (内部调试用)。
	DsuInt	DSU 完成的次数 (内部调试用)。



参数		描述
	Gry	彩转灰开关。 0: 关; 1: 开。
	TD	时域滤波开关。 0: 关; 1: 开。
	Flt	缩放系数。
	SIM	缩放模式。
	HWus	硬件时间（内部调试用）。
	Intus	中断开始时间（内部调试用）。
	SCus	中断结束时间（内部调试用）。
GROUP INFO	NO.	GROUP 通道号。
	ViSend	VI 通道发送给 GROUP 通道图像数。
	GrpSend	GROUP 通道组发送给其内的编码通道的图像数。
	GrpInq	GROUP 查询次数。
	InqOk	GROUP 查询成功次数。
	GrpStart	GROUP 启动编码次数。
	GrpCnVp	GROUP 启动视频前处理的次数。
	GrpInt	GROUP 中断次数。
GROUP INFO 2	GNO.	GROUP 通道组号。
	Chn	GROUP 通道组内编码通道号。
	Typ	GROUP 通道组内编码通道编码类型。
	SendPic	GROUP 发送图像的次数。
	SendOk	GROUP 发送图像成功的次数。
	Inq	GROUP 通道组查询的次数。
	Start	GROUP 通道组启动编码的次数。
	StartOk	GROUP 通道组启动编码成功的次数。
	Cfg	GROUP 通道组配置硬件寄存器的次数。
	Int	GROUP 通道组中断处理次数。



参数		描述
	Tr	编码通道编码图像的时间计数（内部调试用）。

10.10 VPP

【调试信息】

```

~ $ cat /proc/umap/vpp
-----VPP COVER REGION -----
Hdl Dev Chn bPub Show Lay X Y W H Color
  0  0  0  0  1  1 100 200 50 50 0
  1  0  0  0  1  2 202 205 50 50 ff00
  2  0  0  0  1  3 300 200 50 50 ff0000
  3  0  0  0  1  4 400 200 50 50 ff

-----VPP OVERLAY REGION -----
Hdl Grp bPub Show Used PiFmt X Y W H AphF AphB Color Phy Virt Stride
68  0  0  1  1  8 104 100 180 144 128 128 0 c403b000 c1340000 368
69  0  0  1  1  8 304 100 180 144 128 128 1f c4055000 c1360000 368
70  0  0  1  1  8 104 300 48 48 70 70 3e0 c406f000 c1054000 96
71  0  0  1  1  8 304 300 48 48 30 30 7c00 c4072000 c1074000 96

-----VPP VIOVERLAY REGION -----
Hdl Dev Chn bPub Show Lay X Y W H Aph bEx Aph0 Aph1 Color Len Addr
328 0 0 0 1 0 96 96 180 144 255 0 255 255 8000 51840 c3de7000

-----VPP CONFIG OF VENC -----
Grp DeNos C2G SclMode HFilt VFiltL VFiltC SrcFmR TarFmR

```

【调试信息分析】

记录当前所有 VPP 区域的属性和状态信息，以及用户更改后的 VENC 前处理配置。

【参数说明】

参数		描述
VPP COVER REGION 遮挡区域	Hdl	遮挡区域句柄。
	Dev	遮挡区域对应的 VI 设备号。
	Chn	遮挡区域对应的 VI 通道号。
	bPub	是否公共区域。 取值：{0, 1}。
	Show	是否显示。 取值：{0, 1}。
	Lay	遮挡区域层次。



参数		描述
	X	遮挡区域的起始位置 X 坐标。
	Y	遮挡区域的起始位置 Y 坐标。
	W	遮挡区域的宽度。
	H	遮挡区域的高度。
	Color	遮挡区域的背景色。
VPP OVERLAY REGION 叠加 OSD 区域	Hdl	叠加区域 OSD 句柄。
	Grp	叠加区域所属的 GROUP 通道号。
	bPub	是否公共区域。 取值：{0, 1}。
	Show	是否显示。 取值：{0, 1}。
	Uesd	正在被几个通道使用。
	PiFmt	叠加区域像素格式。
	X	叠加区域的起始位置 X 坐标。
	Y	叠加区域的起始位置 Y 坐标。
	W	叠加区域的宽度。
	H	叠加区域的高度。
	AphF	前景 alpha 值。
	AphB	背景 alpha 值。
	Color	背景色。
	Phy	叠加区域的内存物理地址（内部调试用）。
	Virt	叠加区域的内存虚拟地址（内部调试用）。
	Stride	叠加区域的行内存宽度（内部调试用）。
VPP VIOVERLAY REGION VI 叠加 OSD 区域	Hdl	遮挡区域句柄。
	Dev	遮挡区域对应的 VI 设备号。
	Chn	遮挡区域对应的 VI 通道号。
	bPub	是否公共区域。 取值：{0, 1}。
	Lay	层次。



参数		描述
	X	区域的位置 X 坐标。
	Y	区域的位置 Y 坐标。
	W	区域的宽度。
	H	区域的高度。
	Alp	全局透明度。
	bEx	是否扩展 ARGB1555 Alpha。
	Alp0	扩展 ARGB1555 Alpha 时的 Alpha 0 取值。
	Alp1	扩展 ARGB1555 Alpha 时的 Alpha 1 取值。
	Col	背景色。
	Addr	区域数据区的物理地址（内部调试用）。
	Len	区域数据区的大小（内部调试用）。
	Show	区域是否显示。 取值：{0, 1}。
VPP CONFIG OF VENC 视频编码的前处理配置（各项具体说明请参考结构体 VIDEO_PREP_ROC_CONF_S 的描述）	Grp	编码通道组号。
	DeNos	是否开启降噪。
	C2G	是否彩转灰。
	SclMode	缩放模式。
	HFilt	水平缩放系数。
	VFiltL	垂直亮度缩放系数。
	VFiltC	垂直色度缩放系数。
	SrcFmR	编码通道组源帧率。
	TarFmR	编码通道组目标帧率。

10.11 MD

【调试信息】

```
~$ cat /proc/umap/md  
  
Version: [Hi3520_MPP_V1.0.2.2 Debug], Build Time[Oct 27 2009, 19:38:20]
```



```
-----MODULE PARAM-----
    ddr16
        0
-----MD CHN ATTR-----
NO.   W   Y  type  RefM  RefS  SAD  MV  MBAl  PelAlNum  Dlight  SadB
0   352 288   96    0    1    1    0    0        0        0    0

-----MD CHN ATTR 2-----
NO.  Int  Buf  STh  PTh  PNTh  D1B  D1A      TolM      TolSz      UsePM  UseSd
0    0   16  1000 20   20    1    0      c5ea7000  240640        0    0

-----MD STATE-----
NO.   Start   Conf   End   GetD   ReaD   SADRefA  SRefSd  SADN  CSadRfA  stat
0    1779     99    99    98    93  c5ea7400  180    1    c5ec2400  0

-----MD STATE 2-----
NO.  MvRefA  MvRefSd  MvN  CMvRefA  PrLdMvA  PrStMvd  PrN    RltMvA  RltMvSd  RF  RB  RU
0    0        0        0    0        0        0        0    0        0        11  0  5

=====MASTER PROC=====

Version: [Hi3520_MPP_V1.0.2.2 Debug], Build Time[Oct 27 2009, 19:44:24]
-----MD CB INFO-----
ID    CBWh  CBWt  CBRh  CBRt  CBDl
0     2176  2176  1856  1856  320
```

【调试信息分析】

记录当前开启 MD 的编码通道的使用状况及其属性配置，最多可启用 32 路编码通道做移动侦测。可用于检查属性配置以及当前编码通道统计状态。

【参数说明】

参数		描述
MODULE PARAM	ddr16	双 DDR 场景下： 0：MD 结果在 32bit DDR 上。 1：MD 结果在 16bit DDR 上。
MD CHN ATTR	NO.	编码通道号。
	W	编码通道宽度。
	Y	编码通道高度。
	Type	编码类型。
	RefM	参考图像模式。 0：自动模式； 1：用户输入模式。



参数		描述
	RefS	参考图像状态。 0: 不更新; 1: 更新。
	SAD	宏块 SAD 是否开启。 取值: {0, 1}。
	MV	宏块 MV 是否开启。 取值: {0, 1}。
	MBAI	宏块报警信息是否开启。 取值: {0, 1}。
	PelAlNum	宏块报警像素的个数是否开启。 取值: {0, 1}。
	Dlight	去光照效应是否开启。 取值: {0, 1}。
	SadB	宏块 SAD 的精度。 0: 8bit; 1: 16bit。
MD CHN ATTR 2	NO.	编码通道号。
	Int	间隔数。
	Buf	结果缓存的个数。
	STh	SAD 阈值。
	PTh	像素报警阈值。
	PNTh	像素报警个数阈值。
	DIB	去光照效应的 Bata 值。
	DIA	去光照效应的 alpha 值。
	TolM	总内存地址 (内部调试用)。
	TolSz	总内存的大小。
	UsePM	用户提供的参考图像的内存地址。
	UseSd	用户提供的参考图像的行内存宽度。
MD state	NO.	编码通道号。
	Start	启动成功次数。



参数		描述
	Conf	配置硬件寄存器次数。
	End	结束的次数。
	GetD	用户获取 MD 结果的次数。
	ReaD	用户释放 MD 结果的次数。
	SADRefA	SAD 参考图像地址（内部调试用）。
	SRefSd	SAD 参考图像的行内存宽度（内部调试用）。
	SADN	SAD 参考图像内存倒换标识（内部调试用）。
	CSadRfA	SAD 参考图像倒换内存地址（内部调试用）。
	stat	MD 当前状态（内部调试用） 0: MD_CHN_STARTUP; 1: MD_CHN_RUNNING; 2: MD_CHN_WAITSTOP; 3: MD_CHN_STOP。
MD STATE 2	NO.	编码通道号。
	MvRefA	MV 参考图像地址（内部调试用）。
	MvRefSd	MV 参考图像的行内存宽度（内部调试用）。
	MvN	MV 参考图像内存倒换标识（内部调试用）。
	CMvRefA	MV 参考图像倒换内存地址（内部调试用）。
	PrLdMvA	MV 累加信息载入地址（内部调试用）。
	PrStMvd	MV 累加信息输出地址（内部调试用）。
	PrN	MV 累加信息内存倒换标识（内部调试用）。
	RltMvA	MV 结果内存地址（内部调试用）。
	RltMvSd	MV 结果内存行宽度（内部调试用）。
	RF	放置 MD 结果的 Free 队列元素个数（内部调试用）。
	RB	放置 MD 结果的 Busy 队列元素个数（内部调试用）。
	RU	放置 MD 结果的用户队列元素个数（内部调试用）。
MD CB INFO	ID	通道号。
	CBWh	共享内存中循环 buffer 写头（内部调试用）。
	CBWt	共享内存中循环 buffer 写尾（内部调试用）。



参数		描述
	CBRh	共享内存中循环 buffer 读头（内部调试用）。
	CBRt	共享内存中循环 buffer 读尾（内部调试用）。
	CBDl	共享内存中循环 buffer 数据长度（内部调试用）。

10.12 VDEC

【调试信息】

```

~ $ cat /proc/umap/vdec
----- info from slave arm -----
-----MODULE PARAM-----
nc_alg obey_minCR
      1      0
-----CHNL ATTR-----
ID  Type   CurNalu CurLen  FrmRate Prio   Width  Height   FrmNum  FrmMax  VoBind
0   96     c421a2be   0    25      0    720    576     3      414720  (2,2)
1   96     c561a798   0    27      0    704    576     2      414720  (2,3)

-----CHNL STATE-----
ID  SndStm  RlsStm  RlsFail SndPic  SndUdata  GetPic  GetUdata  RlsPic  RlsUdata
0   617     538    0    517     0         0       0       0       0
1   525     524    0    506     0         0       0       0       0

-----BUF STATE-----
ID  Addr      Size      Wptr      Rptr      Wable      Rable      Free      Busy
0   e7474000  829440    e748e2be  e748e80e   1360      828080     6        1
1   e75e8000  811008    e7602798  e7602740   810920     88        6        0

-----PACK SCAN STATE-----
ID  SndingLen  PackOk  PackWaitPackErr  Notify  Snd  Cost  Lefts
0   10916     618    0    0    4065    0    0    0
1    88     525    0    0    4707    0    0    0

-----CHNL OPTION-----
ID  Malloc  Write  BufBase  BufSize
0   0       0    e74740001452544
1   0       0    e75e80001420288

----- info from master arm -----

-----CHNL ATTR-----
ID  Type   CurNalu CurLen  FrmRate Prio   Width  Height   FrmNum  FrmMax  VoBind
0   96     c9c1a2be  920    0      0    720    576     3      414720  (2,2)
1   96     c9e1b178  777    0      0    704    576     2      414720  (2,2)

-----CHNL STATE-----
ID  SndStm  RlsStm  RlsFail SndPic  SndUdata  GetPic  GetUdata  RlsPic  RlsUdata

```



```
0    618    538    0    0    0    0    0    0    0
1    525    524    0    0    0    0    0    0    0
```

-----BUF STATE-----

```
ID Addr      Size      Wptr      Rptr      Wable      Rable      Free      Busy
0  e7474000  829440    e748e2be  e748e80e   1360      828080        6        0
1  e75e8000  811008    e7603178  e7602740  808392      2616        3        3
```

-----PACK SCAN STATE-----

```
ID SndingLen PackOk  PackWaitPackErr  Notify      Snd      Cost      Lefts
0  1024      618    6623      0    2053      0        0        0
1   784      528    529      0    2052      0        0        0
```

-----CHNL OPTION-----

```
ID Malloc Write  BufBase  BufSize
0     1     1      0     0
1     1     1      0     0
```

【调试信息分析】

记录当前解码通道的使用状况及其属性配置，最多有 32 路解码通道。可用于检查属性配置以及当前解码通道统计状态。

【参数说明】

参数		描述
MODULE PARAM	nc_alg	码流切分成 NALU 的算法。 0：普通方式； 1：该方式最优化，性能可能有波动；默认值。 2：该方式比较优化，性能统计较稳定。
	obey_minCR	H.264 码流是否遵守协议中 minCR 的约定。 0：不遵守。默认值； 1：遵守。此时 minCR=2； 创建通道时，会根据 obey_minCR 取值来确定帧最大值。 0：帧最大值为 u32BufSize / 2。 1：帧最大值为 u32PicWidth×u32PicHeight×1.5 / 2。
CHNL ATTR	ID	通道号。
	Type	解码通道类型。 96：PT_H264； 1002：PT_MJPEG； 26：PT_JPEG。
	CurNalu	当前正在扫描的 nalu 包地址。
	CurLen	当前已经扫描的 nalu 长度。
	FrmRate	实际解码帧率，如果码流包有错误，此数可能不准确。



参数		描述
	Prio	解码通道优先级。
	Width	配置的解码图像宽度。
	Height	配置的解码图像高度。
	FrmNum	配置的参考帧个数，H.264 有效，其他类型无此选项。
	FrmMax	帧最大值。
	VoBind	解码绑定 VO 的情况。 显示格式为(VO 设备, VO 通道)。 若解码绑定 n 个 VO，则按显示格式顺序显示 n 项。若未绑定 VO，则不显示。
CHNL STATE	ID	通道号。
	SndStm	发送码流给解码器次数。
	RlsStm	解码器释放码流次数。
	RlsFail	解码器释放失败次数。
	SndPic	解码器发送 pic 次数。
	SndUdata	解码器发送用户数据次数。
	GetPic	用户获取 pic 次数。
	GetUdata	用户获取用户数据次数。
	RlsPic	用户释放 pic 次数。
	RlsUdata	用户释放用户数据次数。
BUF STATE	ID	通道号。
	Addr	码流 buf 首地址。
	Size	码流 buf 长度。
	Wptr	写指针位置。
	Rptr	读指针位置。
	Wable	可写空间。
	Rable	可读空间。
	Free	处于 free 状态的 pack 节点。
	Busy	处于 busy 状态的 pack 节点。
PACK	ID	通道号。



参数		描述
SCAN STATE	SndingLen	用户正在发送的码流的长度。
	PackOk	成功扫描到 pack 边界的次数。
	PackWait	等待扫描 pack 边界的次数。
	PackErr	pack 扫描错误次数。
	Notify	通知发送码流次数。
	Snd	发送码流帧数。按帧发送时有效。
	Cost	解码码流帧数。按帧发送时有效。
	Lefts	码流 buffer 剩余的码流帧数。按帧发送时有效。
CHNL OPTION	ID	通道号。
	Malloc	VDEC 内部是否分配 buffer(码流 buffer 和用户数据 buffer)。
	Write	VDEC 是否解析切分 nalu 后拷贝码流。 0: 不解析, 直接拷贝; 1: 解析后拷贝。
	BufBase	Malloc=1 时, vdec 内部所分配 buffer 基址的物理地址; Malloc=0 时, 始终为 0。
	BufSize	Malloc=1 时, vdec 内部所分配 buffer 的大小; Malloc=0 时, 始终为 0。

10.13 AI

【调试信息】

```
~ $ cat /proc/umap/ai
-----MODULE PARAM-----
use_ddr16
    0
-----AI DEV ATTR-----
AiDev WorkMod SampR BitWid ChnCnt ClkSel SondMod PoiNum ExFlag FrmNum
    1 i2s_sla    8k  16bit   16    0    mono   321    1    30
-----AI DEV STATUS-----
AiDev IntCnt FrmTime DMACHn DMAReq TranLen IsrTime DMAPhy0 DMAPhy1
    1   143  40122    2    2    10272   334  e7264000 e7266820
-----AI CHN STATUS-----
AiDev AiChn State Read Write BufFul AecAo AecFail u32Data0 u32Data1
```



1	0	enable	23	23	0	(-1, -1)	0 f204f1cf f1d1f1f5
1	1	enable	23	23	0	(-1, -1)	0 f275f266 f252f267
1	2	enable	23	23	0	(-1, -1)	0 f1d0f1d9 f1c4f1d3
1	3	enable	23	23	0	(-1, -1)	0 f24df241 f246f245
1	4	enable	23	23	0	(-1, -1)	0 f353f35a f34cf357
1	5	enable	23	23	0	(-1, -1)	0 f426f41f f41af41b
1	6	enable	23	23	0	(-1, -1)	0 f4ccf4c2 f4c6f4c1
1	7	enable	23	23	0	(-1, -1)	0 f3c9f3c3 f3c4f3c3
1	8	enable	23	23	0	(-1, -1)	0 f633f63d f636f639
1	9	enable	23	23	0	(-1, -1)	0 f6e5f6d6 f6d3f6d0
1	10	enable	23	23	0	(-1, -1)	0 f6a1f6bf f6acf6c8
1	11	enable	23	23	0	(-1, -1)	0 f739f72c f735f71e
1	12	enable	23	23	0	(-1, -1)	0 f4cef4e3 f4d6f4f5
1	13	enable	23	23	0	(-1, -1)	0 f452f46e f471f472
1	14	enable	23	23	0	(-1, -1)	0 f56ff56d f579f56d
1	15	enable	23	23	0	(-1, -1)	0 f524f52d f523f530

【调试信息分析】

记录当前音频输入设备及通道的属性配置以及状态信息。

【参数说明】

参数		描述
MODULE PARAM	use_ddr16	AI 的 DMA buffer 位置。 0: buffer 在 32bit DDR 上。 1: buffer 16bit DDR 上。
AI DEV ATTR (音频输入设备 属性) (参考 数据结构 AIO_ATTR_S)	AiDev	AI 设备号。
	WorkMod	SIO 工作模式。 i2s_mas: I2S 主模式。 i2s_sla: I2S 从模式。
	SampR	采样率。 取值: {8k, 16k, 32k}。
	BitWid	采样精度。 取值: {8bit, 16bit}。
	ChnCnt	最大通道个数。 取值: {2, 4, 8, 16}。
	ClkSel	时钟选择。
	SondMod	声音模式。 mono: 单声道; stereo: 立体声。
	PoiNum	每帧的采样点个数。
	ExFlag	8bit 扩展标志。



参数		描述
	FrmNum	帧缓存数目。
AI DEV STATUS (音频输入设备信息)	AiDev	AI 设备号。
	AiChn	AI 通道号。
	IntCnt	中断计数。 完成 1 帧音频数据采集后, 此值加 1。 如果此值未递增, 说明与外围 codec 对接有问题。
	FrmTime	帧间隔时间。以毫秒为单位。 由此项可以计算出实际的音频采样率。例如当此项为 40000us 左右、每帧采样点个数配置为 320 时, 可以计算出每秒采样 8000 点, 即采样率为 8kHz。
	DMAChn	DMA 通道。
	DMAReq	DMA 请求线。
	DMAPhy0	DMA buffer0 的物理地址。
	DMAPhy1	DMA buffer1 的物理地址。
AI CHN STATUS (音频输入通道信息)	AiDev	AI 设备号。
	AiChn	AI 通道号。
	state	通道状态。 orig: 初始状态; enable: 启用; disable: 禁用。
	Read	通道 buffer 的读指针。
	Write	通道 buffer 的写指针。
	BufFul	帧 buffer 满的次数。
	AecAo	回声抵消 AEC 的 AO 设备号和通道号。(-1 表示未启用 AEC)
	AecFail	回声抵消失败次数。
	u32Data0	通道 Buffer 中第一个 32 位数据。
	u32Data1	通道 Buffer 中第二个 32 位数据。



10.14 AO

【调试信息】

```
cat /proc/umap/ao
-----MODULE PARAM-----
use_dds16
0
-----AO DEV ATTR-----
AoDev WorkMod SampR BitWid ChnCt ClkSel SndMod PoiNum ExFlag FrmNum
1 i2s_sla 8k 16bit 2 0 mono 321 1 30

-----AO DEV STATUS-----
AoDev IntCnt FrmTime DMAChn DMAReq DMAPhy0 DMAPhy1
1 19587 40121 3 3 e7396000 e7396a08

-----AO CHN STATUS-----
AoDev AoChn State Read Write BufEmp u32Data0 u32Data1
1 0 enable 27 27 0 f1c8f1b5 f1daf1c8

-----AO CHN BIND RELATION-----
AoDev AoChn AiDev AiChn AdChn
1 0 1 0 -1
```

【调试信息分析】

记录当前音频输出设备属性配置以及状态信息。

【参数说明】

参数		描述
MODULE PARAM	use_dds16	AO 的 DMA buffer 位置。 0: buffer 在 32bit DDR 上。 1: buffer 16bit DDR 上。
AO DEV ATTR (音频输出设备属性) (参考数据结构 AIO_ATTR_S)	AoDev	AO 设备号。
	WorkMod	SIO 工作模式。 i2s_mas: I2S 主模式。 i2s_sla: I2S 从模式。
	SampR	采样率。 取值: {8k, 16k, 32k}。
	BitWid	采样精度。 取值: {8bit, 16bit}。
	ChnCt	最大通道个数。 取值: {2, 4, 8, 16}。
	ClkSel	时钟选择。



参数		描述
	SondMod	声音模式。 mono: 单声道; stereo: 立体声。
	PoiNum	每帧的采样点个数。
	ExFlag	8bit 扩展标志。
	FrmNum	帧缓存数目。
AO DEV STATUS (音 频输出设备信 息)	AoDev	AO 设备号。
	AoChn	AO 通道号。
	IntCnt	中断计数。 发送 1 帧音频数据后, 此值加 1。 如果此值未递增, 说明与外围 codec 对接有问题。
	DMACHn	DMA 通道。
	DMAReq	DMA 请求线。
	DMAPhy0	DMA Buffer0 的物理地址。
	DMAPhy1	DMA Buffer1 的物理地址。
AO CHN STATUS (音 频输出通道信 息)	AoDev	AO 设备号。
	AoChn	AO 通道号。
	state	通道状态。 orig: 初始状态; enable: 启用; disable: 禁用。
	Read	通道 buffer 的读指针。
	Write	通道 buffer 的写指针。
	BufFul	帧 buffer 满的次数。
	u32Data0	通道 Buffer 中第一个 32 位数据。
	u32Data1	通道 Buffer 中第二个 32 位数据。
AO CHN BIND RELATION (音频输出通道 的绑定关系)	AoDev	AO 设备号。
	AoChn	AO 通道号。
	AiDev	绑定的 AI 设备号。(-1 表示未绑定)



参数		描述
	AiChn	绑定的 AI 通道号。（-1 表示未绑定）
	AdChn	绑定的 ADEC 通道号。（-1 表示未绑定）

10.15 AENC

【调试信息】

```
cat /proc/umap/aenc
----Attribution of AENC Channel-----
ChnId  PlType  BufSize  Attr1  Attr2  Attr3  Attr4  Attr5
    0    g726    3      m_16k

----Status of AENC Channel-----
ChnId  AiDev  AiChn  RcvFrm  EncOk  FrmErr  BufFull
    0     1     0   33969  33969    0      0
```

【调试信息分析】

记录当前音频编码属性配置以及状态信息。

【参数说明】

参数		描述
Attribution of AENC Channel (音频编码通道属性)	ChnId	AENC 通道号。
	PlType	编码协议类型。
	BufSize	帧缓存数目。
	Attr1~Attr5	协议相关属性。（具体意义由协议类型决定）
Status of AENC Channel (音频编码通道状态)	ChnId	AENC 通道号。
	AiDev	绑定的 AI 设备号。
	AiChn	绑定的 AI 通道号。
	RcvFrm	接收到的待编码的音频帧数目
	EncOk	编码成功的音频帧数目
	FrmErr	编码失败的音频帧数目
	BufFull	音频码流缓冲区满的次数（用户未及时获取音频码流数据会导致缓冲区满，进而会导致码流数据丢失）



10.16 ADEC

【调试信息】

```
cat /proc/umap/adecc
----Attribution of ADEC Channel-----
ChnId  PlType  BufSize  Attr    SendCnt  GetCnt  PutCnt
      0   adpcm    20     DVI4    188     170    169
```

【调试信息分析】

记录当前音频解码属性配置以及状态信息。

【参数说明】

参数		描述
Attribution of ADEC Channel 音频解码通道属性及状态	ChnId	通道号。
	PlType	解码码协议类型。
	BufSize	帧缓存数目。
	Attr	协议相关属性。（不同协议类型此项的意义也不同，请参考相关数据结构）
	SendCnt	成功发送解码器进行解码的音频帧数目。
	GetCnt	用户获取的码流帧数目。
	PutCnt	用户释放的码流帧数目。

10.17 CHNL

【调试信息】

```
~ $ cat /proc/umap/chnl

----- info from slave arm -----
Version: [Hi3520_MPP_V1.0.0.0 Debug], Build Time[Nov 27 2009, 14:04:08]
total VPU count:2

----- VPU 0 information-----
----- CHNL state -----
receive interrupt total cnt: 527
receive timer interrupt cnt: 470
receive vedu interrupt cnt: 57
receive error interrupt cnt: 0
channel start fail cnt: 0
channel start success cnt: 0
channel schedule cnt: 0
channel check task cnt: 1045
channel reset VEDU cnt: 0
```




```
----- CHNL current run state-----
ID  prot
no running channel here

----- CHNL state -----
ID  prot  pri      tsknum state   timeout  inqtime  startOK  startNO
1   VPP    0xd      0      2      0      1162    0        0
0   GROUP  0x0      0      2      0      924    113     0

----- CHNL performance -----
ID  prot  hw(us)  int(us)  start(us)
1   VPP    0       0       0
0   GROUP  6638    96     184

----- VPU 1 information-----
----- CHNL state -----
receive interrupt total cnt: 526
receive timer interrupt cnt: 470
receive vedu interrupt cnt: 56
receive error interrupt cnt: 0
channel start fail cnt: 0
channel start success cnt: 0
channel schedule cnt: 0
channel check task cnt: 1041
channel reset VEDU cnt: 0

----- CHNL current run state-----
ID  prot
no running channel here

----- CHNL state -----
ID  prot  pri      tsknum state   timeout  inqtime  startOK  startNO
1   VPP    0xd      0      2      0      1162    0        0
0   GROUP  0x0      0      2      0      924    113     0

----- CHNL performance -----
ID  prot  hw(us)  int(us)  start(us)
1   VPP    0       0       0
0   GROUP  6638    96     184

----- info from master arm -----
no proc from master
```

【调试信息分析】

无。

【参数说明】

参数		描述
CHNL	receive interrupt total cnt	CHNL 收到的中断总数。



参数		描述
state	receive timer interrupt cnt	CHNL 收到的定时中断数。
	receive vedu interrupt cnt	CHNL 收到的 VEDU 中断。
	receive error interrupt cnt	CHNL 收到的 VEDU 错误中断数。
	channel start fail cnt	CHNL 调度失败次数。
	channel start success cnt	CHNL 调度成功次数。
	channel schedule cnt	CHNL 调度总数。
	channel check task cnt	CHNL 查询通道次数。
	channel reset VEDU cnt	CHNL 进行 VEDUreset 次数。
CHNL current run state	ID	通道号。
	prot	协议类型。
CHNL state	ID	通道号。
	prot	协议类型。
	pri	优先级。
	tsknum	任务数。
	state	通道状态。 1: 正常; 2: 等待注销; 3: 已经注销。
	timeout	超时计数。
	inqtime	查询次数。
	startOK	启动成功次数。
	startNO	启动失败次数。
CHNL performan ce	ID	通道号。
	prot	协议类型。
	hw(us)	硬件消耗。
	int(us)	中断消耗。
	start(us)	启动消耗。



10.18 H264E

【调试信息】

```

~ $ cat /proc/umap/h264e
----- info from slave arm -----
Version: [Hi3520_MPP_V1.0.0.0 Debug], Build Time[Nov 27 2009, 14:04:25]

-----CHN ATTR-----
  ID  Pri    Width  Height  MainStr  VIField  BufSize  ByFrame  MaxStrCnt
  0    1     704    576    Yes     No       811008   Yes      0x1fffffff

-----RATE CTRL ATTR-----
  ID  PicMbs  AlignW  AlignH  InFrmRt  OutFrmRt  Field    Gop      MaxDly(ms)
  0   1584    704     576     25       25/0      No       100     100

-----RATE CTRL ATTR 2-----
  ID  RcType  BitRt(Kbps)  Level  minute  QpI    QpP    bHighBR
  0   CBR    1024( 1024)  0      0       0      0      0

-----RATE CTRL ATTR 3-----
  ID  MinQp  MaxQp  FrmFxQp  FrmLost
  0   -1    -1     0        1

-----PICTURE INFO-----
  ID  Rcv  TfErr  Encd  Disc  Skip  BufLeak  RcLost  Back  RlsStr  UnrdStr
  0   121  0     120   0     0     0        0      118   2

-----STREAM BUFFER-----
  ID  base          RdTail  RdHead  WrTail  WrHead  datalen  buffree
  0   0xc2600040   0x0     0xc5e00 0xc5e00 0xc5e00  0       448

-----CHN INFO-----
  ID  Texture  OsdPret  SlcSplT  SlcSize  NoEtrStrm  RefMode
  0   Yes     Yes     No       1024     No         1X

-----PRE PROCESS INFO-----
  ID  Die  TimeFlt  MvStl  PicTp  Repet  WaterMk
  0   Yes  Yes     Yes    420   No     No

-----VPP PARA INFO-----
  ID  DieBase  DieRate  TfBase  TfRate  DieDlt  TfdDlt  TmTrd  SadTrd
  0   8        8        9       12      0       0       3      15

-----CHN PERF INFO-----
  ID  hw(us)  int(us)  strt(us)  reg(us)  br(kbps)  fr(fps)  AvgBR  AvgFR
  0   11026   33      19        15      1038     25     1048   25/0

----- info from master arm -----
no proc from master

```

【调试信息分析】

无。



【参数说明】

参数		描述
CHN ATTR	ID	通道号。
	Pri	优先级。
	Width	宽度，以像素为单位。
	Height	高度，以像素为单位。
	MainStr	是否主码流。 取值：{0, 1}。
	VIField	输入是否场模式。 取值：{0, 1}。
	BufSize	码流 buffer 大小，以字节为单位。
	ByFrame	是否按帧获取码流。 取值：{0, 1}。
	MaxStrCnt	允许码流 Buffer 缓存的最大帧数。 缺省值：0x1FFFFFFF。
RATE CTRL ATTR	ID	通道号。
	PicMbs	图像 MB 数。
	AlignW	16 像素对齐后宽度，以像素为单位。
	AlignH	16 像素对齐后高度，以像素为单位。
	InFrmRt	图像输入帧率，以 fps 为单位。
	OutFrmRt	编码帧率，以 fps 为单位。
	Field	是否场模式。 取值：{0, 1}。
	Gop	I 帧间隔。
	MaxDly(ms)	最大延迟，以 ms 为单位。
RATE CTRL ATTR 2	ID	通道号。
	RcType	码率控制类型。 取值：{CBR, VBR, ABR, FIXQP}。
	BitRt(Kbps)	目标码率，以 kbit/s 为单位。
	Level	图像质量等级。 取值范围：[0, 5]。



参数		描述
	minute	平均码率统计时间。仅 ABR 有效。
	QpI	I 帧 QP。仅 FIXQP 有效。
	QpP	P 帧 QP。仅 FIXQP 有效。
	bHighBR	是否高码率。 取值：{0, 1}。
RATE CTRL ATTR 3	ID	通道号。
	MinQp	最小 QP。仅 VBR 有效。 取值范围[4, 50]。-1 表示未设置。
	MaxQp	最大 QP。仅 VBR 有效。 取值范围[4, 50]。-1 表示未设置。
	FrmFxQp	帧固定 QP。取值{0,1}。
	FrmLost	码率控制不丢帧。取值{0,1}。
PICTURE INFO	ID	通道号。
	Rcv	接收图像数。 接收到待编码图像并且放到图像队列后，此项 值加 1。
	TfErr	时间参考错误的图像数。
	Encd	编码图像数。 编码器编完一帧图像后，此项值加 1。
	Disc	由于图像队列满导致的丢帧个数。 编码器接收到新的待编码图像，但因图像队列 已满，无法缓存而丢弃该图像时，此项值加 1。 当编码慢时，图像队列会满，易出现此项值不 为零。
	Skip	图像队列中未编码的图像个数。 $skip = (bufLeak + rcLost + \text{帧率控制导致的丢帧个数})$ 。
	BufLeak	由于码流 buffer 不足导致的丢帧个数。 编码器启动编码前，发现码流 Buffer 可用空间 非常小而丢掉一帧图像时，此项值加 1。 当用户未能及时取走码流时，易出现此项值不 为 0。



参数		描述
	RcLost	码率控制导致的丢帧个数。 根据内部码率统计结果，若发现超出目标码率太多时，会主动丢帧以降低码率，每丢一帧，此项值加 1。
	Back	由于码流 buffer 满导致的丢帧个数。 编码器编完一帧后，发现码流 Buffer 可用空间无法容纳这一帧的码流，从而丢掉该帧时，此项值加 1。 当用户未能及时取走码流时，易出现此项值不为 0。
	RlsStr	已经释放的码流帧数。
	UnrdStr	当前 buffer 中缓存的码流帧数。
STREAM BUFFER	ID	通道号。
	base	码流 buffer 基地址。
	BufLen	码流 buffer 大小。
	RdTail	读尾指针。
	RdHead	读头指针。
	WrTail	写尾指针。
	WrHead	写头指针。
	datalen	数据长度。
	buffree	空闲长度。
CHN INFO	ID	通道号。
	Txturet	纹理检测开关。 NO: 关闭; YES: 开启。
	OsdPret	OSD 保护开关。 NO: 关闭; YES: 开启。
	SlcSplt	slice 划分。 NO: 不划分; YES: 划分。
	SlcSize	slice 大小，以字节为单位。



参数		描述
	NoOutEtrStrm	不输出超出码流。 NO: 输出; YES: 不输出。
	RefMode	跳帧参考模式。取值请参考枚举 VENC_ATTR_H264_REF_MODE_E 的定义。
PRE PROCESS INFO	ID	通道号。
	Dei	De-interlace 开关。 NO: 关闭; YES: 开启。
	EdegFlt	边缘检测开关。 NO: 关闭; YES: 开启。
	MedFlt	中值检测开关。 NO: 关闭; YES: 开启。
	TimeFlt	时域去噪开关。 NO: 关闭; YES: 开启。
	MvStl	动静判决开关。 NO: 关闭; YES: 开启。
	PicTp	图像类型。 0: 4:2:0; 1: 4:2:2。
	Repet	重复编码开关。 NO: 关闭; YES: 开启。
	WaterMk	数字水印开关。 NO: 关闭; YES: 开启。
VPP PARAM INFO	ID	通道号。
	DieBase	内部调试参数。
	DieRate	内部调试参数。



参数		描述
	TfBase	内部调试参数。
	TfRate	内部调试参数。
	DieDlt	内部调试参数。
	TfDlt	内部调试参数。
	TmTrd	内部调试参数。
	SadTrd	内部调试参数。
CHN PERF INFO	ID	通道号。
	hw(us)	硬件消耗，以 μs 为单位。
	int(us)	中断销毁，以 μs 为单位。
	strt(us)	启动消耗，以 μs 为单位。
	reg(us)	配寄存器消耗，以 μs 为单位。
	br(kbps)	瞬时码率，以 bit/s 为单位。 注意： 瞬时码率与平均码率的计算方法不同，码率控制是根据平均码率进行的，因此该值仅供参考，不宜作为评价码率控制的标准。
	fr(fps)	瞬时帧率，以 fps 为单位。
	AvgBR	平均码率，以 kbps 为单位。 此为所有历史数据的统计，仅供内部调试，不宜作为码率控制评价标准。
	AvgFR	平均帧率，以 fps 为单位。 此为所有历史数据的统计，仅供内部调试，不宜作为帧率控制评价标准。

10.19 H264D

【调试信息】

```
~ $ cat /proc/umap/h264d
----- info from slave arm -----

----- Perf(us) -----
ID Parse HWdec SWdec SWint HWrprY SWrprY SWintY HWrprC SWrprC SWintC
0 165 6051 49 81 0 0 0 0 0 0
1 94 8780 48 72 0 0 0 0 0 0
```




```
----- Const -----
ID      W      H  AlignW AlignH   Ref   DispQue  RmvQue   SendQue DescBuf FrmMax
0   720   576   720   576     3    1919    1983     63    207232 311040
1   704   576   704   576     2    1919    1983     63    202624 304128

----- Status -----
ID      W      H   Ref DispQue  RmvQue  SendQue  DescBuf  Out  Pic  10Fld State Slot
0   720   576    1     74     74      0    9984    0  0  0    1    0
1   704   576    2      0      0      0    128    0  0  0    0    0

----- HW Stat -----
ID  Start      Eop  Empty  StartY   IntY  StartC   IntC
0   7927    7927     0      0      0      0      0
1   7916    7916     0      0      0      0      0

----- Err Stat -----
ID ErrNalu LostPic  ErrSlc LostSlc Overlap   ErrW   ErrH  ErrRef ErrCncl ErrBigF
0     0      0      0      0      0       0     0    0      0      0
1     0      0      0      0      0       0     0    0      0      0

----- Flow Stat -----
ID RecvPic  DecPic  SendPic  RecvAUD  RlsAud  RecvEOS   NoFB   Rst   pps   fps
0   8002    7927    7927      0      0      0    71243  0   25   25
1   7917    7916    7916      0      0      0      0    0   25   25

----- info from master arm -----

Version: [Hi3520_MPP_V1.0.2.4 Debug], Build Time[Nov 18 2009, 15:49:09]

-----simd share cb-----
ID  addr      len    Rt     Rh     Wt     Wh   dataLen free
0c49e0000  64000   7936   7936  10464  10464   2528  61440
1c4b20000  64000   4992   4992   5024   5024    32  63936

-----simd share stat-----
ID  LftPic  DecFrm  hasData  LftStrmF  RcvStrmF  LftStrmB  bRecv
0    0     -1     0     -1x     -1x     815077x   1
1    0     -1     0     -1x     -1x     102x     1
```

【调试信息分析】

记录 H.264 解码过程中，各通道的解码属性、状态以及历史信息统计，最多有 32 路解码通道。可配合用于定位系统出现的阻塞以及图像错误等问题。

【参数说明】

参数		描述
Perf 性能 ^a	ID	通道号。
	Parse	软件解析一个 NALU 的时间（进程上下文）。
	HWdec	硬件解码一幅图像时间。
	SWdec	软件准备解码一幅图像时间（中断上下文）。
	SWint	完成图像解码后的软件处理时间（中断上下文）。



参数		描述
	HWrprY	硬件修补 Y 分量时间。
	SWrprY	软件准备修补 Y 分量时间（中断上下文）。
	SWintY	完成 Y 修补后的软件处理时间（中断上下文）。
	HWrprC	硬件修补 C 分量时间。
	SWrprC	软件准备修补 C 分量时间（中断上下文）。
	SWintC	完成 C 修补后的软件处理时间（中断上下文）。
Const 常量 ^b	ID	通道号。
	W	通道属性宽度。
	H	通道属性高度。
	AlignW	16 像素对齐后图像宽度。
	AlignH	16 像素对齐后图像高度。
	Ref	通道属性参考帧个数。
	DispQue	显示图像队列总长度。
	RmvQue	移除队列总长度。
	SendQue	发送队列总长度。
	DescBuf	硬件消息 buffer 总长度。
	FrmMax	帧最大值。表示一帧码流的最大值。
Status 状态 ^c	ID	通道号。
	W	码流当前宽度。
	H	码流当前高度。
	Ref	码流当前参考帧个数。
	DispQue	显示图像队列当前长度。
	RmvQue	移除队列当前长度。
	SendQue	发送队列当前长度。
	DescBuf	硬件消息 buffer 当前长度。
	Out	图像输出模式。 0: 快速输出; 1: DPB 满输出。



参数		描述
	Pic	当前解码的图像属性。 0: 帧; 1: 顶场; 2: 底场; 3: 场对。
	10Fld	是否 3510 私有场。 取值: {0, 1}。
	State	解码器状态。 0: 等待; 1: 解码; 2: 修补 Y; 3: 修补 C。
	Slot	图像发送缓存状态。用于场配对输出。 0: 空; 1: 缓存一个顶场。
HW Stat 硬件统计 ^d	ID	通道号。
	Start	启动解码次数。
	Eop	图像结束中断次数。
	Empty	空中断次数。
	StartY	修补 Y 次数。
	IntY	修补 Y 中断次数。
	StartC	启动修补 C 次数。
	IntC	修补 C 结束中断次数。
Err Stat 错误统计 ^e	ID	通道号。
	ErrNalu	软件解析发现的错误 NALU 个数。 因 SEI 错误不影响图像显示, 所以未包括。 码流错误时该项可能增加。 解码图象花屏时可查看该项。
	LostPic	软件发现图像丢失个数。 码流错误时该项可能增加。 解码图象花屏时可查看该项。



参数		描述
	ErrSlc	硬件发现 Slice 错误次数。 码流错误时该项可能增加。 解码图象花屏时可查看该项。
	LostSlc	硬件发现 Slice 丢失次数。 码流错误时该项可能增加。 解码图象花屏时可查看该项。
	Overlap	硬件发现 Slice 重叠次数。 码流错误时该项可能增加。 解码图象花屏时可查看该项。
	ErrW	通道宽度设置错误次数。 属性无法支持当前码流时，认为属性设置错误。 实际解码图象宽度超过通道设置宽度时该项增加。 无解码图象时可查看该项
	ErrH	通道高度设置错误次数。 属性无法支持当前码流时，认为属性设置错误。 实际解码图象高度超过通道设置高度时该项增加。 无解码图象时可查看该项
	ErrRef	通道参考帧个数设置错误次数。 属性无法支持当前码流时，认为属性设置错误。 实际参考帧个数超过用户设置参考帧个数时该项增加。 无解码图象时可查看该项。
	ErrCncl	软件解码器进行错误掩盖的次数。 软件检查参考帧错误并进行修补后该项增加。 解码图象花屏时可查看该项。
	ErrBigF	超大帧出现次数。 当码流 Buffer 中缓存的码流已经超过了设置的帧最大值，但仍收不齐一帧时，则视为出现超大帧，该项加 1。超大帧会被丢弃。 解码图象花屏时或无解码图象时可查看该项。
Flow Stat	ID	通道号。



参数		描述
流程统计 ^f	RecvPic	接收的图像个数。 在接收到新图像时该值增加。 该值结合 DecPic 可用于解码性能不足时，查看用户发送码流是否很慢。例如：当解码性能不足时，可多次查看该项与 DecPic 的差值，如果差值常为 1，说明码流发送慢，造成解码性能不足。
	DecPic	已解码的图像个数。 在 vedu 成功解码一副图象后该值增加。 该值可用于解码性能不足时，查看 vedu 性能是否不足或解码是否频繁启动失败。例如：当解码性能不足时，可多次查看该项与 RecvPic 的差值，如果差值一直较大，说明码流发送及时，可能解码启动频繁失败或者 vedu 性能不足，造成解码性能不足。
	SendPic	已发送的图像总数。 当 Out=1 时，表示帧图像个数。 若是场码流，该数值应为 DecPic/2。
	RecvAud	接收到的码流帧数。按帧发送码流时有效。
	RlsAud	已解码的码流帧数。按帧发送码流时有效。
	RecvEOS	已接收的 EOS NALU 总数。
	NoFB	无空闲帧存可用的次数。 启动解码时，若分配不到帧存，该值增加。 正常场景中，该值增加属正常现象。因解码速率一般比 vo 输出速率快，vo 占用大量解码图象帧存，造成启动解码时分配不到帧存。
	Rst	解码通道复位次数。
	pps	在相邻两次查看 proc 信息之间的时间段内，平均每秒钟解码的图象数。 通过 cat /proc/umap/h264d;sleep n;cat /proc/umap/h264d 可查看 n 秒内 h264d 平均每秒钟解码的图象数。
	fps	在相邻两次查看 proc 信息之间的时间段内，平均每秒钟解码的帧数。 通过 cat /proc/umap/h264d;sleep n;cat /proc/umap/h264d 可查看 n 秒内 h264d 平均每秒钟解码的帧数。
Simd Share Cb 共享内存中 循环 buffer	ID	通道号。
	addr	循环 buffer 的虚拟基地址。
	Len	循环 buffer 的总长度。



参数		描述
信息统计	Rt	循环 buffer 的读尾指针偏移值。
	Rh	循环 buffer 的读头指针偏移值。
	Wt	循环 buffer 的写尾指针偏移值。
	Wh	循环 buffer 的写头指针偏移值。
	dataLen	循环 buffer 内的数据长度。
	Free	循环 buffer 的剩余可用空间。
simd share stat 共享内存中 从 arm 状态 统计	ID	通道号。
	LftPic	有效数据。 从 ARM 解码器中未获取的帧数。
	DecFrm	有效数据。 按帧发送方式时，从 arm 解码器已解帧数； 按流发送方式时，始终为-1。
	hasData	有效数据。 从 ARM 解码器中是否有可获取的数据（解码图像、用户数据）。
	LftStrmF	无效数据。
	RcvStrmF	无效数据。
	LftStrmB	无效数据。
	bRecv	有效数据。 从 ARM 解码器是否开始接收码流。

- a: 软硬件解码性能，一般不需要关注。
- b: 通道创建后不再改变的参数，包括通道属性和解码内部资源，一般不需要关注。
- c: 系统当前的状态，随解码过程变化。一般不需要关注。
- d: 硬件控制统计。一般不需要关注。
- e: 解码错误统计。定位图像显示错误现象。
- f: 解码流程统计。定位系统阻塞现象。

10.20 JPEG

【调试信息】

```
~ $ cat /proc/umap/JPEG
----- info from slave arm -----
```



```
Version: [Hi3520_MPP_V1.0.0.0 Debug], Build Time[Nov 27 2009, 14:04:32]

----- Attr -----
ID  MJPG  FrmW  FrmH  AlignW  AlignH  Vi  Fld  MCU  ByFrm  [Main  TFR  TBR]  [ImgQ]
1   1     352  288   352     288    0  0   396  1      1    25   4096   0

----- Status -----
ID  BufLen  FreeLen  FQue  EQue  Q  FR  BR  PicByte  StrmCnt  MaxStrm
1   202752  0        0    26   99 36  468   2279

----- Flow Stat -----
ID  SendPic  IntEnd  IntFull  DropPic  NoBuf  NoPic  RcFail  VppFail  Rst
1   37       37     0        37      0     0     0      0

----- info from master arm -----
no proc from master
```

【调试信息分析】

记录 JPEG 编码过程中，各通道的编码属性、状态以及历史信息统计，最多有 64 路编码通道。可配合用于定位系统阻塞以及丢帧等问题。

【参数说明】

参数		描述
Attr 属性 ^a	ID	通道号。
	MJPG	是否 MJPEG 编码。 0: JPEG 抓拍。 1: MJPEG。
	FrmW	图像宽度。
	FrmH	图像高度。
	AlignW	16 像素对齐后图像宽度。
	AlignH	16 像素对齐后图像高度。
	Vi	VI 输入图像属性。 0: 帧; 1: 场。
	Fld	编码属性。 0: 帧编码; 1: 场编码。
	MCU	每个 ECS 的 MCU 个数。
	ByFrm	是否按帧获取码流。 取值: {0, 1}。



参数		描述
	Main	是否主码流。 取值：{0, 1}。
	TFR	目标帧率。
	TBR	目标码率。
	ImgQ	图像质量。
Status 状态 ^b	ID	通道号
	BufLen	码流 Buffer 总长度。
	FreeLen	空闲码流 Buffer 长度。
	FQue	可接收的图像个数。
	EQue	待编码图像个数。
	Q	量化因子。
	FR	实际帧率。（数据不精确，仅供参考）
	BR	实际码率。（数据不精确，仅供参考）
	PicByte	编码后的一帧码流 byte 数。
	StrmCnt	当前码流 buffer 缓存的帧数。
	MaxStrm	允许码流 buffer 缓存的最大帧数。 缺省值：100000。
Stat 统计 ^c	ID	通道号。
	SendPic	发送来编码的图像总数。
	IntEnd	硬件编码结束中断次数。
	IntFull	硬件 Buffer 满中断次数（会导致丢帧）。
	DropPic	归还图像帧存的次数。
	NoBuf	发现码流 Buffer 不足的次数（会导致丢帧）。
	NoPic	没有待编码图像的次数。
	RcFail	码率/帧率控制失败的次数（会导致丢帧）。
	VppFail	VPP(视频前处理)设置失败的次数。
	Rst	通道复位次数。

a: 一般不需要关注。

b: 系统当前的状态，随编码过程变化。可用于定位系统阻塞。

c: 历史统计信息。可用于定位丢帧。

