DVPP API 参考

文档版本	01
发布日期	2020-05-09

华为技术有限公司

商标声明

HUAWEI和其他华为商标均为华为技术有限公司的商标。
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

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

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。

目 录

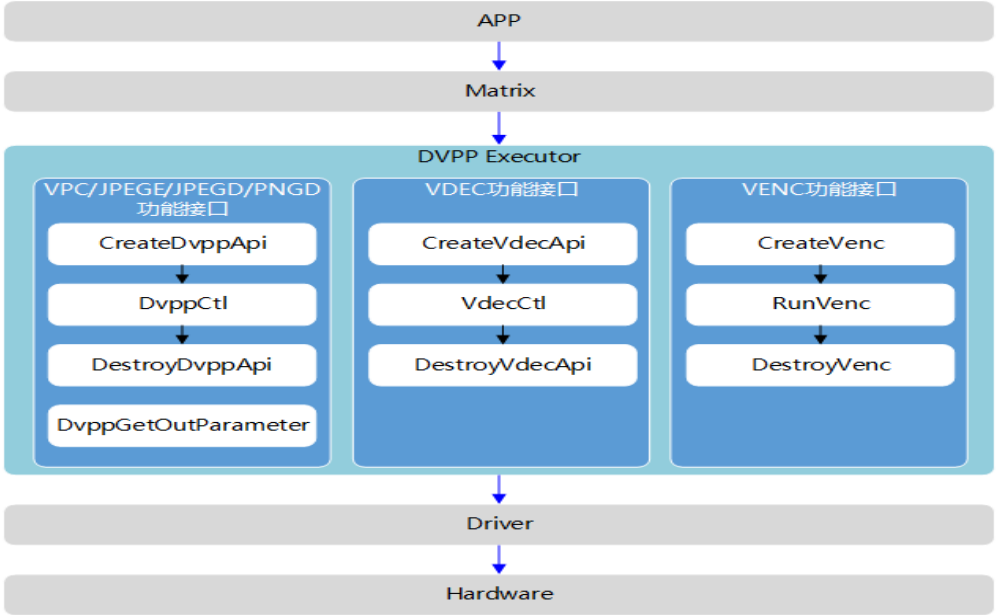
1 概述

1.1 接口简介

本文档详细描述了DVPP（Digital Vision Pre-Processing）执行器对外提供的接口，接口描述包括：接口函数描述、接口调用说明、整体示例等，适合开发人员、测试人员。

开发人员在开发APP时调用DVPP接口的流程如下：

图 1-1 流程图



- 当前版本调用**CreateDvppApi**、**DvppCtl**、**DestroyDvppApi**接口实现VPC功能有两种方式：
  - 方式一，在**DvppCtl**接口中传入入参：DVPP_CTL_VPC_PROC命令字和**VpcUserImageConfigure结构体**的参数。**推荐使用。**
  - 方式二，在**DvppCtl**接口中传入入参：DVPP_CTL_VPC_PROC命令字和**resize_param_in_msg结构体**的参数。对时延要求不高、处理的图片分辨率低、数量多的场景，可以在**DvppCtl**接口中传入入参：DVPP_CTL_CMDLIST_PROC和**IMAGE_CONFIG结构体**的参数。这种方式是为了兼容旧版本中的功能，后续版本会删除，**不推荐使用**。
- 当前版本实现VENC功能有两种方式：
  - 方式一，调用**CreateVenc**、**RunVenc**、**DestroyVenc**接口实现VENC功能，**推荐使用**。
  - 方式二，调用**CreateDvppApi**、**DvppCtl**、**DestroyDvppApi**接口实现VENC功能，在**DvppCtl**接口中传入入参：DVPP_CTL_VENC_PROC命令字和**venc_in_msg结构体**的参数，这种方式在为了兼容旧版本中的功能，后续版本会删除，**不推荐使用**。

1.2 接口列表

您可以在DDK（Device Development Kit）的安装目录下的“ddk/include/inc/dvpp/”目录下查看接口的头文件。若需要调用DVPP提供的接口，在代码中可以包含idvppapi.h、Venc.h、Vpc.h（定义数据类型的头文件，请参见**7.1 VpcUserImageConfigure中的结构体**）。

表 1-1 接口列表

分类	接口	功能说明	定义接口的头文件
实现VPC/JPEGE/JPEGD/PNGD功能	**CreateDvppApi**	创建dvppapi实例，相当于获取DVPP执行器句柄，调用方可以使用申请到的dvppapi实例调用**DvppCtl接口**处理图片，可以跨函数调用，跨线程调用。	idvppapi.h
	DvppCtl	使用**CreateDvppApi**接口创建的实例来调用**DvppCtl**接口，控制DVPP各模块执行，模块主要包括VPC（Vision Pre-processing Core）、JPEGE、JPEGD、PNGD等。	
	DestroyDvppApi	销毁由**CreateDvppApi**接口创建的dvppapi实例，关闭DVPP执行器。	
	DvppGetOutParameter	获取JPEGD/JPEGE/PNGD模块的输出内存大小。	
实现VDEC功能	**CreateVdecApi**	获取vdecapi实例，相当于vdec执行器句柄。调用方可以使用申请到的vdecapi实例调用**CreateVdecApi接口**进行视频解码，可以跨函数调用，跨线程调用。	idvppapi.h
	VdecCtl	使用**CreateVdecApi**接口创建的实例调用**VdecCtl**接口，控制DVPP执行器进行视频解码。	
	DestroyVdecApi	释放由**CreateVdecApi**接口创建的vdecapi实例，关闭VDEC执行器。	

分类	接口	功能说明	定义接口的头文件
实现VENC功能	**CreateVenc**	获取VENC编码实例，相当于获取VENC执行器句柄。调用方可以使用申请到的VENC编码实例调用**RunVenc接口**进行图片编码。	Venc.h
	RunVenc	使用**CreateVenc**接口创建的实例调用**RunVenc**接口，控制DVPP执行器进行视频编码。	
	DestroyVenc	释放由**CreateVenc**接口创建的VENC编码实例，关闭VENC执行器。	

1.3 VPC 功能

功能说明

VPC功能包括：

- **抠图**，从输入图片中抠出需要用的图片区域。
- **缩放**
  - 针对不同分辨率的图像，VPC的处理方式可分为非8K缩放、8K缩放。
    - 8K缩放，用于处理输入图像宽度在4096~8192范围内或高度在4096~8192范围内的图片。
    - 非8K缩放，用于处理输入图像分辨率在32*6~4096*4096范围内的图片。
  - 从是否抠多张图的维度，可分为单图裁剪缩放（支持非压缩格式和HFBC压缩格式）、一图多框裁剪缩放（支持非压缩格式和HFBC压缩格式）。

    HFBC，是VDEC输出的一种压缩图像格式，使用这种方式压缩图像，VDEC的处理性能更优。
  - 其它缩放方式，如：原图缩放、等比例缩放。
- **叠加**，从输入图片中抠出来的图，对抠出的图进行缩放后，放在用户输出图片的指定区域，输出图片可以是空白图片（由用户申请的空输出内存产生的），也可以是已有图片（由用户申请输出内存后将已有图片读入输出内存），只有当输出图片是已有图片时，才表示叠加。
- **拼接**，从输入图片中抠多张图片，对抠出的图进行缩放后，放到输出图片的指定区域。
- **格式转换**
  - 将RGB格式/YUV422格式/YUV444格式的图片转为YUV420格式的图片。

– 图像灰度化，对输出图像数据只取Y分量的数据。

约束说明

- 针对不同分辨率的图像，VPC的处理方式包括8K缩放、非8K缩放，如下表所示。

输入图像分辨率	输入图像格式	输入图像宽stride*高stride对齐要求	VPC功能	输出图像分辨率	输出图像格式	输出图像宽stride*高stride对齐要求
宽度在4096~8192范围内或高度在4096~8192范围内（不包括4096）	YUV420SP（NV12、NV21）	2*2对齐	8K缩放	16*16~4096*4096	请参见表2-1处的outputFormat参数	2*2对齐
32*6~4096*4096（包括4096）	请参见表2-1处的inputFormat参数	• 宽stride的对齐：请参见表2-1处的widthStride参数 • 高stride是2对齐	非8K缩放	32*6~4096*4096	请参见表2-1处的outputFormat参数	16*2对齐

- 针对缩放功能，宽高缩放比例范围：[1/32, 16]。
- 对于8K缩放功能，支持缩放、支持YUV420SP NV12与YUV420SP NV21之间的格式转换、不支持抠图。

功能示意图

图 1-2 VPC 功能示意图（抠图+缩放+叠加）

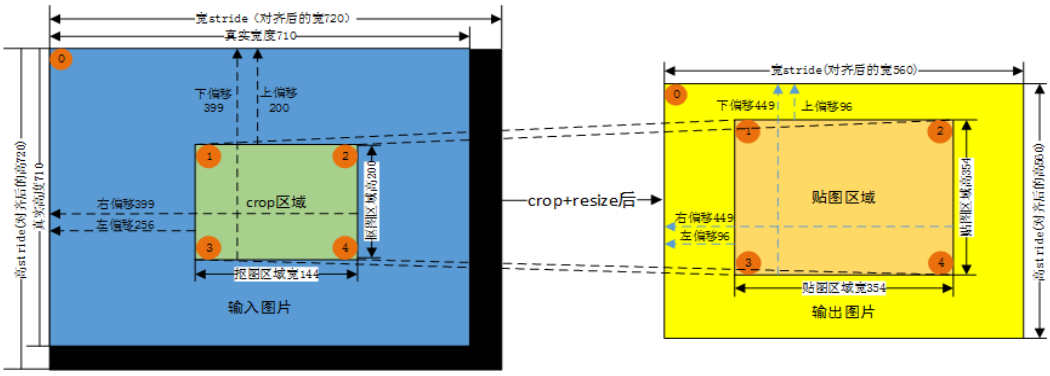


图 1-3 VPC 功能示意图（拼接）

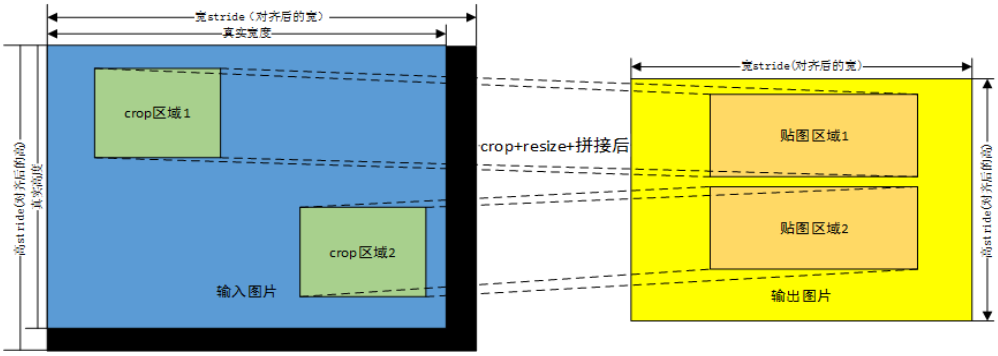


表 1-2 概念解释

概念	描述
宽stride	指一行图像步长，表示输入图片对齐后的宽，RGB格式或YUV格式的宽stride计算方式不一样。宽stride最小为32，最大为4096 * 4（即宽是4096的argb格式的图像）。 ● yuv400sp、yuv420sp、yuv422sp、yuv444sp：输入图像的宽对齐到16。 ● yuv422packed：输入图像的宽*2后再对齐到16后。 ● yuv444packed、rgb888：输入图像的宽对齐*3后再对齐到16。 ● xrgb8888：输入图像的宽*4后再对齐到16。 ● HFBC格式：输入图像的宽。
高stride	指图像在内存中的行数，表示输入图片对齐后的高。 取值为：输入图像的高对齐到2。高stride最小为6，最大为4096。

概念	描述
上/下/左/右偏移	通过配置上偏移、下偏移、左偏移、右偏移可以实现两个功能：指定抠图区域或贴图区域的位置；控制抠图或贴图区域的宽、高，右偏移-左偏移+1=宽，下偏移-上偏移+1=高。 ● 左偏移：输入/输出图片中，抠图/贴图区域1、3两个点相对于0点水平向左偏移的值。 ● 右偏移：输入/输出图片中，抠图/贴图区域2、4两个点相对于0点水平向左偏移的值。 ● 上偏移：输入/输出图片中，抠图/贴图区域1、2两个点相对于0点垂直向上偏移的值。 ● 下偏移：输入/输出图片中，抠图/贴图区域3、4两个点相对于0点垂直向上偏移的值。
抠图区域	指用户指定的需裁剪的图片区域。 抠图区域最小分辨率为10*6，最大分辨率为4096*4096。
贴图区域	指在输出图片中用户指定的区域，贴图区域最小分辨率为10*6，最大分辨率为4096*4096。 约束如下： ● 贴图区奇数、偶数限制为：左偏移和上偏移为偶数、右偏移和下偏移为奇数。 ● 抠图区域不超出输入图片，贴图区域不超出输出图片。 ● 贴图时可直接放置在输出图片的最左侧，即相对输出图片的左偏移为0。 ● 最大贴图个数为256个。 ● 贴图区域相对输出图片的左偏移16对齐。 ● 输出图片的贴图宽度建议16对齐，如果不是16对齐，会多写一段无效数据使其16对齐。

性能指标说明

- 对于**非8K缩放**，VPC性能由于涉及到抠图、缩放等不同的场景，当处理图片的过程中分辨率改变时，以大分辨率来计算性能。例如，对于缩放，缩放后图片的分辨率大于缩放前图片的分辨率，则以缩放后图片的分辨率来计算性能指标；对于抠图，贴图区域的分辨率大于抠图区域的分辨率，则以贴图区域的分辨率来计算性能指标。对于YUV420 SP格式的图片，典型场景性能指标参考如下：

场景举例	总帧率
1080p * n路（n<4，1路对应一个线程）	n*360fps
1080p * n路（n>=4，1路对应一个线程）	1440fps
4k * n路（n<4，1路对应一个线程）	n*90fps

场景举例	总帧率
4k * n路（n>=4，1路对应一个线程）	360fps

- 对于**8K缩放**，VPC性能与输出分辨率强相关，输出分辨率越大，处理耗时越久，性能越低，典型场景（输出分辨率为1080p、4k的场景，图片格式为YUV420 SP）性能指标参考如下：

场景举例	总帧率
1080p * n路（n=1，1路对应一个线程）	4fps
1080p * n路（n>=4，1路对应一个线程）	16fps
4k * n路（n=1，1路对应一个线程）	1fps
4k * n路（n>=4，1路对应一个线程）	4fps

参考说明

RBG、YUV格式图像的各分量排布示意图。示例：sp图像以yuv420sp为例，packed和rgb图像以argb图像为例。

格式	分辨率	宽stride	高stride	buffer大小					
yuv420sp	4*4	4	4	24					
内存排列									
y11	y12	y13	y14						
y21	y22	y23	y24						
y31	y32	y33	y34						
y41	y42	y43	y44						
u11	v11	u13	v13						
u31	v31	u33	v33						
格式	分辨率	宽stride	高stride	buffer大小					
yuv420sp	4*4	6	6	54					
内存排列	x为无效数据								
y11	y12	y13	y14	x	x				
y21	y22	y23	y24	x	x				
y31	y32	y33	y34	x	x				
y41	y42	y43	y44	x	x				
x	x	x	x	x	x				
x	x	x	x	x	x				
u11	v11	u13	v13	x	x				
u31	v31	u33	v33	x	x				
x	x	x	x	x	x				
格式	分辨率	宽stride	高stride	buffer大小					
argb	2*2	10	4	40					
内存排列	x为无效数据								
a11	r11	g11	b11	a12	r12	g12	b12	x	x
a21	r21	g21	b21	a22	r22	g22	b22	x	x
x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x

1.4 JPEGE 功能

功能及约束说明

JPEGE功能：实现编码生成.jpg图片。

- JPEGE支持以下格式输入：
  - YUV422 packed（yuyv,yvyu,uyvy,vyuy）
  - YUV420 Semi-planar（NV12，NV21）
- JPEGE支持的分辨率为：

  最大分辨率：8192 * 8192，最小分辨率：32 * 32
- JPEGE输出格式为jpeg，只支持huffman编码，不支持算术编码，不支持渐进编码。

性能指标说明

场景举例	总帧率
1080p * n路（n>=1）	64fps
4k * n路（n>=1）	16fps

1.5 JPEGD 功能

功能说明

实现.jpg、.jpeg、.JPG、.JPEG图片的解码，对于硬件不支持的格式，会使用软件解码。

解码后，输出如下格式的图片：

jpeg(444) -> yuv444 / yuv420半平面V在前U在后。

jpeg(422) -> yuv422 / yuv420半平面V在前U在后。

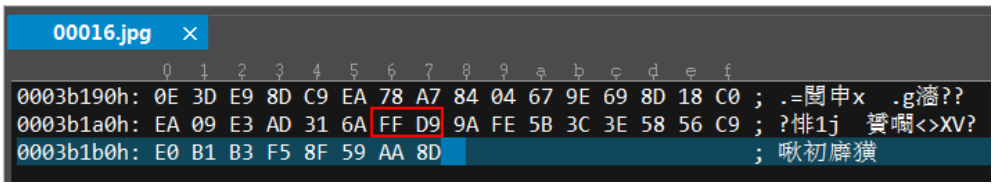
jpeg(420) -> yuv420半平面V在前U在后。

jpeg(400) -> yuv420，uv数据采用0 x 80填充。

须知

- JPEGD由于编程规范要求驼峰风格命名方式，原内核风格的入参和出参继续支持，同时提供新的驼峰风格参数供调用方使用，所以JPEGD暂时支持两套参数供用户使用，推荐使用驼峰风格参数。
- 若图片内EOI（End Of Image，标记代码为0XFFD9）之后，还有用户自定义的数据，则JPEGD在对图片进行解码时，会直接清零EOI之后的8字节数据，若用户需要保留这些自定义的数据，则将图片数据读入内存之后，需要提前备份这部分数据，再传给JPEGD处理。

  若需要查看图片内EOI之后是否存在自定义数据，可以使用二进制查看工具打开图片查看，例如下图中的FFD9标记符之后就存在自定义数据。



约束说明

- 关于输入图片的约束：
  - 最大分辨率：8192 * 8192，最小分辨率：32 * 32
  - 只支持Huffman编码，码流的colorspace为YUV，码流的subsample为444/422/420/400；
  - 不支持算术编码；
  - 不支持渐进JPEG格式；
  - 不支持JPEG2000格式；
- 关于硬件约束：
  - 最多支持4张Huffman表，其中包括2 张DC（直流）表和2 张AC（交流）表；

  - 最多支持3张量化表；
  - 只支持8bit采样精度；
  - 只支持对顺序式编码的图片进行解码；
  - 只支持基于DCT（Discrete Cosine Transform）变换的JPEG 格式解码；
  - 只支持一个SOS（Start of Scan）标志的图片解码。
- 关于软件约束：
  - 支持3个SOS标志的图片解码；
  - 支持mcu（Minimum Coded Unit）数据不足的异常图片解码。

性能指标说明

JPEGD性能指标是基于硬件解码的性能，JPEGD硬件解码不支持3个SOS的图片解码，对于硬件不支持的格式，会使用软件解码，软件解码性能参考为1080P*1路 15fps。

场景举例	总帧率
1080p * 1路	128fps
1080p * n路（n>=2）	256fps
4k * 1路	32fps
4k * n路（n>=2）	64fps

1.6 PNGD 功能

功能及约束说明

PNGD功能：实现PNG格式图片的硬件解码。

- PNGD支持以下两种输入格式：
  RGBA、RGB
- PNGD输出格式为：
  RGBA、RGB
- PNGD支持的分辨率
  最大分辨率4096 * 4096，最小分辨率32 * 32。

性能指标说明

场景举例	总帧率
1080p * 1路	4fps
1080p * 2路	8fps
1080p * 3路	12fps
1080p * 4路	16fps

场景举例	总帧率
1080p * 5路	20fps
1080p * n路（n>= 6）	24fps
4k * 1路	1fps
4k * 2路	2fps
4k * 3路	3fps
4k * 4路	4fps
4k * 5路	5fps
4k * n路（n>= 6）	6fps

1.7 VDEC 功能

功能及约束说明

实现视频的解码。

- VDEC支持两种输入格式：

  H264 bp/mp/hp level5.1 yuv420sp编码的码流。

  H265 8/10bit level5.1 yuv420sp编码的码流。
- VDEC支持的分辨率

  最大分辨率4096 * 4096，最小分辨率128 * 128。
- VDEC输出格式为：yuv420sp压缩后的HFBC数据。

  HFBC，是VDEC输出的一种压缩图像格式，使用这种方式压缩图像，VDEC的处理性能更优。
- 若码流中有坏帧、缺帧等情况，解码器VDEC可能会丢帧。
- 通过隔行扫描方式编码出来的码流，VDEC不支持解码。
- VDEC比较特殊，由于当前内部不是单例模式，所以其对外提供C接口函数与其他不同。

性能指标说明

场景举例	总帧率
1080p n路（n>=2且n<=16）	480fps
1080p *1路	240fps
4k * n路（n>=2且n<=16）	120fps
4k * 1路	60fps

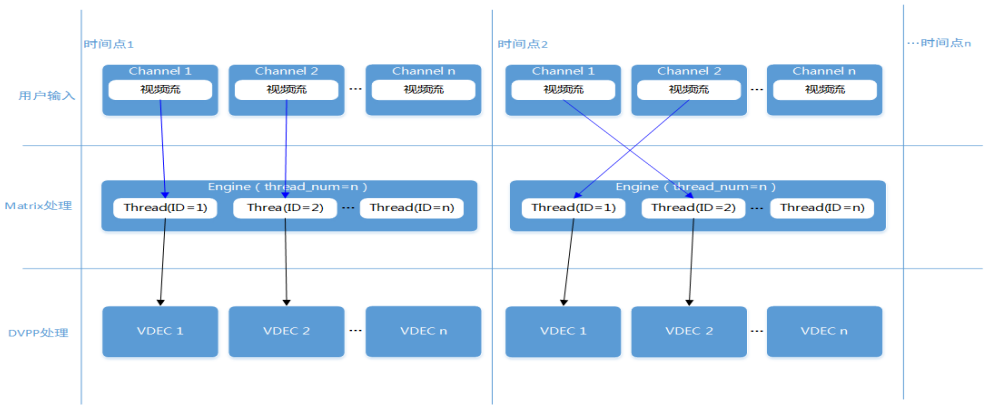
如果用户使用Matrix模块编排应用的流程，那么在使用DVPP对多路（Channel）输入的视频流进行解码时，由于视频流中每帧的数据之间有关联关系，因此为保证同一个视频流中每帧数据的顺序，要求每个VDEC固定对应一路视频流，否则DVPP中的VDEC无法解码视频流。

- 多路视频流解码时，为保证视频流中每帧数据的顺序，可能存在多种实现方式，推荐使用以下配置：
  - 在Graph配置文件中，在graphs配置段内，配置多个进行视频解码的Engine，一路视频流对应一个Engine。
  - 在Graph配置文件中，在视频解码的engine配置段内，将thread_num配置为1，一个Engine对应一个线程。

  说明

  - 关于Matrix的功能及其Graph配置文件的配置，请参见《 Matrix API参考 》。
  - 一个Graph（一个进程）最大可以对应16路的视频流。
- 如果在多路视频流解码时，在Graph配置文件中，仅配置一个视频解码的Engine，并将thread_num配置为n（n的值就是视频流channel的数量），在不同时刻，在Matrix中各路视频流可能会在不同的Thread处理，但是在DVPP中，一个VDEC需要固定一路视频流数据，一路视频流数据对应一个线程，这样可能导致VDEC解码时无法保证视频流中每帧数据的顺序，如图所示。

图 1-4 VDEC 处理流程



1.8 VENC 功能

VENC将编码分成实例化、编码、释放资源三小步。

功能及约束说明

实现YUV420/YVU420图片数据的编码。

- VENC支持以下格式输入：
  YUV420 semi-planar NV12/NV21-8bit
- VENC支持的分辨率
  最大分辨率1920 * 1920，最小分辨率128 * 128。
- VENC输出格式为：
  H264 BP/MP/HP

H265 MP（仅支持Slice码流）

性能指标说明

场景举例	总帧率
1080p * n路（一个进程对应一路）	30fps

注：其它分辨率可以等量估算。

1.9 关于输入输出内存的说明

关于HIAI_DMalloc/HIAI_DFree接口、HIAI_DVPP_DMalloc/HIAI_DVPP_DFree接口的说明，以及Graph配置文件（receive_memory_without_dvpp参数）的说明，请参见《Matrix API参考》。

表 1-3 内存要求说明

功能模块	输入内存	输出内存
VPC	请使用Matrix提供的接口申请/释放内存： ● 使用HIAI_DVPP_DMalloc接口申请内存，内存满足DVPP的要求（内存起始地址16对齐）； ● 使用HIAI_DVPP_DFree接口释放内存。	请使用Matrix提供的接口申请/释放内存： ● 使用HIAI_DVPP_DMalloc接口申请内存，内存满足DVPP的要求（内存起始地址16对齐）； ● 使用HIAI_DVPP_DFree接口释放内存。
JPEGE和JPEGD	请使用Matrix提供的接口申请/释放内存： ● 使用HIAI_DVPP_DMalloc接口申请内存，内存满足DVPP的要求（内存起始地址128对齐）； ● 使用HIAI_DVPP_DFree接口释放内存。	● 由用户指定输出内存时，由用户自行释放内存 。请使用Matrix提供的接口申请/释放内存： – 使用HIAI_DVPP_DMalloc接口申请内存，内存满足DVPP的要求（内存起始地址128对齐）。在申请内存前，可以调用**DvppGetOutParameter**接口获取输出内存大小。 – 使用HIAI_DVPP_DFree接口释放内存。 ● 不由用户指定输出内存时，DVPP内部申请内存，需由用户调用JPEGE/JPEGD出参结构体中的cbFree()回调函数释放内存，并将内存地址指针置为空。

功能模块	输入内存	输出内存
PNGD	请使用Matrix提供的接口申请/释放内存： ● 使用HIAI_DVPP_DMalloc接口申请内存，内存满足DVPP的要求（内存起始地址128对齐）； ● 使用HIAI_DVPP_DFree接口释放内存。	由用户指定输出内存时，由用户自行释放内存 。请使用Matrix提供的接口申请/释放内存： ● 使用HIAI_DVPP_DMalloc接口申请内存，内存满足DVPP的要求（内存起始地址128对齐）。在申请内存前，可以调用**DvppGetOutParameter**接口获取输出内存大小。 ● 使用HIAI_DVPP_DFree接口释放内存。
VDEC和VENC	对内存无要求，支持调用malloc/free、new/delete等原生接口申请/释放内存，也支持调用Matrix提供的HIAI_DMalloc/HIAI_DFree接口、HIAI_DVPP_DMalloc/HIAI_DVPP_DFree接口申请/释放内存，还支持通过Matrix提供的receive_memory_without_dvpp参数控制内存是否给DVPP使用。	VDEC输出的HFBC格式数据直接作为VPC的输入。 VENC输出内存是DVPP内部管理，用户在使用时可以拷贝输出内存中的数据。

2 VPC/JPEGE/JPEGD/PNGD 功能接口

2.1 CreateDvppApi

函数原型	**int CreateDvppApi(IDVPPAPI*& pIDVPPAPI)**
功能	创建dvppapi实例，相当于获取DVPP执行器句柄，调用方可以使用申请到的dvppapi实例调用**DvppCtl接口**处理图片，可以跨函数调用，跨线程调用。
输入说明	输入为“IDVPPAPI”类型指针引用，输入指针必须为“NULL”。
输出说明	输出为“IDVPPAPI”类型指针引用，获取失败则为NULL，获取成功则不为NULL。
返回值说明	● 返回值“0”代表成功。 ● 返回值“-1”代表失败。
使用说明	调用方创建**“IDVPPAPI”**对象指针，初始化为NULL，调用**“CreateDvppApi”**函数将**“IDVPPAPI”**对象指针传入。如果申请成功，**“CreateDvppApi”**函数会返回dvppapi实例，否则返回NULL。调用方需要对返回值进行校验。
使用约束	调用方负责dvppapi实例的生命周期，即申请与释放，申请使用**CreateDvppApi**接口，释放使用**DestroyDvppApi接口**。

调用示例

```
IDVPPAPI *pidvppapi = NULL;
CreateDvppApi(pidvppapi);
```

2.2 DvppCtl

2.2.1 接口说明

函数原型	**int DvppCtl(IDVPPAPI*& pIDVPPAPI, int CMD, dvppapi_ctl_msg* MSG)**
功能	使用**CreateDvppApi**接口创建的实例来调用**DvppCtl**接口，控制DVPP各模块执行，模块主要包括VPC（Vision Pre-processing Core）、JPEGE、JPEGD、PNGD等。
输入说明	• **“IDVPPAPI”**类型指针引用 • **CMD控制命令字** • **dvppapi_ctl_msg**类型的DVPP执行器配置信息MSG。DVPP各个模块的配置信息不同，因此调用各模块功能时需传入相应的配置信息。
输出说明	根据DVPP各模块的功能，出参也不一样，请参见**dvppapi_ctl_msg**。
返回值说明	• 返回值“0”代表成功。 • 返回值“-1”代表失败。
使用说明	调用方调用**“DvppCtl”**函数，传入**“IDVPPAPI”**对象指针、传入正确的**CMD控制命令字**、配置好相应功能的**dvppapi_ctl_msg**。
使用约束	无。

CMD 控制命令字

CMD控制命令字在DDK安装目录下的“ddk/include/inc/dvpp/DvppCommon.h”文件中定义。

成员变量	说明	取值范围
DVPP_CTL_VPC_PROC	VPC功能控制命令字。	0
DVPP_CTL_PNGD_PROC	PNGD功能控制命令字。	1
DVPP_CTL_JPEGE_PROC	JPEGE功能控制命令字。	2
DVPP_CTL_JPEGD_PROC	JPEGD功能控制命令字。	3
DVPP_CTL_VENC_PROC	预留命令字。	5
DVPP_CTL_DVPP_CAPABILITY	查询DVPP能力控制命令字。	6
DVPP_CTL_CMDLIST_PROC	预留命令字。	7

成员变量	说明	取值范围
DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM	预留命令字。	8

dvppapi_ctl_msg

该类型在DDK安装目录下的“ddk/include/inc/dvpp/DvppCommon.h”文件中定义。

成员变量	说明	取值范围
int32_t in_size	入参大小。	-
int32_t out_size	出参大小。	-
void *in	入参。	• VPC功能入参：**VpcUserImageConfigure** • JPEGE功能入参：**sJpegeIn** • JPEGD功能入参：**JpegdIn**（Jpegd输入结构体的驼峰风格版本，推荐使用） • JPEGD功能入参：**jpegd_raw_data_info**（jpegd输入结构体的内核风格版本，不推荐使用） • PNGD功能入参：**PngInputInfoAPI** • 查询DVPP引擎入参：**device_query_req_stru**
void *out	出参。	• VPC功能出参：无 • JPEGE功能出参：**sJpegeOut** • JPEGD功能出参：**JpegdOut**（Jpegd输出结构体的驼峰风格版本，推荐使用） • JPEGD功能出参：**jpegd_yuv_data_info**（jpegd输出结构体的内核风格版本，不推荐使用） • PNGD功能出参：**PngOutputInfoAPI** • 查询DVPP引擎出参：**dvpp_engine_capability_stru**

2.2.2 VPC 参数说明

入参：VpcUserImageConfigure

表 2-1 入参 VpcUserImageConfigure

成员变量	说明	取值范围
uint8_t* bareDataAddr	非压缩格式输入图片地址，当图像为压缩格式时，不需要填，默认为NULL。 使用Matrix提供的HIAI_DVPP_DMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址16对齐）。HIAI_DVPP_DMalloc接口的说明，请参见《 Matrix API参考 》。	-
uint32_t bareDataBufferSize	非压缩格式输入图片内存大小，其中大小是根据图像的宽高stride计算出来的，bareDataAddr指向内存大小应该等于bareDataBufferSize，当图像为压缩格式时，不需要填，默认为0。	非压缩格式，对于不同图像格式，内存大小的计算公式如下： ● yuv400sp、yuv420sp：widthStride*heightStride*3/2 ● yuv422sp：widthStride*heightStride*2 ● yuv444sp：widthStride*heightStride*3 ● yuv422packed：widthStride*heightStride ● yuv444packed、rgb888：widthStride*heightStride ● xrgb8888：widthStride*heightStride

成员变量	说明	取值范围
uint32_t widthStride	图像宽度方向的步长。	宽stride最小为32，最大为4096 * 4（即宽是4096的argb格式的图像）。 对于8K缩放，要求宽stride对齐到2；对于非8K缩放，对于不同图像格式，widthStride的计算公式不同： ● yuv400sp、yuv420sp、yuv422sp、yuv444sp：输入图像的宽对齐到16。 ● YUV422packed：输入图像的宽对齐到16后，再乘以2（宽16对齐，每个像素占2个字节）。 ● YUV444packed、RGB888：输入图像的宽对齐到16，再乘以3（宽16对齐，每个像素占3个字节）。 ● XRGB8888：输入图像的宽对齐到16后，再乘以4（宽16对齐，每个像素占4个字节）。 ● HFBC格式：输入图像的宽。
uint32_t heightStride	图像高度方向的步长，对于yuv sp图像根据该参数计算uv数据的起始地址。	取值为：输入图像的高对齐到2。高stride最小为6，最大为4096。

成员变量	说明	取值范围
enum VpcInputFormat inputFormat	输入图像格式。	enum VpcInputFormat { INPUT_YUV400, // 0 INPUT_YUV420_SEMI_PLANNER_UV, // 1 INPUT_YUV420_SEMI_PLANNER_VU, // 2 INPUT_YUV422_SEMI_PLANNER_UV, // 3 INPUT_YUV422_SEMI_PLANNER_VU, // 4 INPUT_YUV444_SEMI_PLANNER_UV, // 5 INPUT_YUV444_SEMI_PLANNER_VU, // 6 INPUT_YUV422_PACKED_YUYV, // 7 INPUT_YUV422_PACKED_UYVY, // 8 INPUT_YUV422_PACKED_YVYU, // 9 INPUT_YUV422_PACKED_VYUY, // 10 INPUT_YUV444_PACKED_YUV, // 11 INPUT_RGB, // 12，输入图像格式为RGB888 INPUT_BGR, // 13，输入图像格式为BGR888 INPUT_ARGB, // 14，此格式的输入图像在存储时各分量的排列顺序类似RGB888，其中，A表示透明度 INPUT_ABGR, // 15，此格式的输入图像在存储时各分量的排列顺序类似BGR888，其中，A表示透明度 INPUT_RGBA, // 16，此格式的输入图像在存储时各分量的排列顺序类似RGB888，其中，A表示透明度 INPUT_BGRA, // 17，此格式的输入图像在存储时各分量的排列顺序类似BGR888，其中，A表示透明度

成员变量	说明	取值范围
		INPUT_YUV420_SEMI_PLANNER_UV_10BIT, // 18 INPUT_YUV420_SEMI_PLANNER_VU_10BIT, // 19 };

enum VpcOutputFormat outputFormat	输出图像格式。	enum VpcOutputFormat {
		OUTPUT_YUV420SP_UV,
		OUTPUT_YUV420SP_VU
		};

VpcUserRoiConfigure* roiConfigure	抠图区域配置，详细请参见 **•VpcUserRoiConfigure结构体**。	-
bool isCompressData	是否是视频解码输出的HFBC压缩图片数据格式。	当图片来源于VDEC时，需要配置为true，其他格式为false，默认为false。 单图裁剪缩放和一图多框裁剪缩放均支持HFBC压缩格式。
VpcCompressDataConfigure compressDataConfigure	视频解码输出的HFBC压缩图片数据配置。详细请参见 **•VpcCompressDataConfigur...**。	-
bool yuvSumEnable	是否需要计算yuvSum值，当需要计算该值时只支持一图一框。	当需要计算yuvSum值时配置true，其他为false，默认为false。 统计yuvSum值的约束条件： 1，输入图像分辨率小于等于4096*4096 2，贴图区域起始点是（0,0）位置 3，输入只有一图一框
VpcUserYuvSum yuvSum	yuvSum计算配置。详细请参见 **•VpcUserYuvSum结构体**。	-
VpcUserPerformanceTuningParameter tuningParameter	预留参数，用于参数调优，详细请参见 **•VpcUserPerformanceTunin...**。	-

成员变量	说明	取值范围
uint32_t* cmdListBufferAddr	预留参数。	-
uint32_t cmdListBufferSize	预留参数。	-
uint64_t yuvScalerParaSetAddr	预留参数，滤波参数集文件路径地址和文件名数组。 **说明** ● 参数集须在Device设备上。 ● 请确保传入的文件路径是正确路径。	参数集文件路径为device侧的绝对文件路径＋文件名，如{"/root/share/vpc/YUVScaler_pra.h"}
uint16_t yuvScalerParaSetSize	预留参数，滤波参数集地址指向的参数集数组元素个数（默认为1）。	1<=yuvscaler_paraset_size<=10
uint16_t yuvScalerParaSetIndex	预留参数，yuvScalerParaSetIndex变量对应yuvScalerParaSetAddr的索引（默认为0）。	0<=yuvScalerParaSetIndex<10
uint8_t isUseMultiCoreAccelerate	预留参数。	-
uint8_t reserve1	预留参数。	-
uint16_t reserve2;	预留参数。	-

出参说明

暂无出参。

2.2.3 JPEGE 参数说明

入参：sJpegeIn

表 2-2 入参 sJpegeIn

成员变量	说明
eEncodeFormat format	输入YUV数据的类型，支持YUV422 packed（yuyv,yvyu,uyvy,vyuy）和YUV420 Semi-planar（NV12，NV21） typedef enum { JPGENC_FORMAT_UYVY = 0x0, JPGENC_FORMAT_VYUY = 0x1, JPGENC_FORMAT_YVYU = 0x2, JPGENC_FORMAT_YUYV = 0x3, JPGENC_FORMAT_NV12 = 0x10, JPGENC_FORMAT_NV21 = 0x11, } eEncodeFormat;
unsigned char* buf	yuv输入数据，需要调用方申请，需要合理对齐，见参数stride，heightAligned。 使用Matrix提供的HIAI_DVPP_DMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。HIAI_DVPP_DMalloc接口的说明，请参见《 Matrix API参考 》。 **须知** 使用HIAI_DVPP_DMalloc接口申请内存时，则由用户保证申请的内存大小与输入参数bufSize的参数值一致。
uint32_t bufSize	输入buf长度，指宽高对齐后的数据长度。
uint32_t width	输入图片的宽度，范围[32,8192]。
uint32_t height	输入图片的高度，范围[32,8192]。
uint32_t stride	输入图片对齐后宽度，对齐到16，兼容对齐到16的倍数如128。对于YUV422packed数据，stride应该为width的两倍对齐到16。
uint32_t heightAligned	1. 支持与高度height相同，2.支持输入图片对齐后高度（Matrix异侧传输后的对齐），3.兼容高度向16对齐的数。
uint32_t level	编码质量范围[0, 100]，其中level 0编码质量与level 100差不多，而在[1, 100]内数值越小输出图片质量越差。

出参：sJpegeOut

表 2-3 出参 sJpegeOut

成员变量	说明
unsigned char* jpgData	输出缓冲区中jpg数据的起始地址。 如果由用户指定内存，使用Matrix提供的HIAI_DVPP_DMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。HIAI_DVPP_DMalloc接口的说明，请参见《 Matrix API参考 》。 **须知** 使用HIAI_DVPP_DMalloc接口申请内存时，则由用户保证申请的内存大小与输入参数jpgSize的参数值一致。
uint32_t jpgSize	编码后的jpg图片数据长度。
JpegCalBackFree cbFree	释放输出内存的回调函数。 ● 由用户指定输出内存时，由用户自行释放内存 。 ● 不由用户指定输出内存时，DVPP内部申请内存，需由用户调用cbFree()回调函数释放内存，并将jpgData置为空指针。调用示例请参见**6.2 实现JPEGE功能**。

2.2.4 JPEGD 参数说明

入参：JpegdIn

表 2-4 入参 JpegdIn

成员变量	说明
unsigned char* jpegData	输入jpg图片数据，起始地址128对齐，2M大页表方式申请。 使用Matrix提供的HIAI_DVPP_DMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。HIAI_DVPP_DMalloc接口的说明，请参见《 Matrix API参考 》。 **须知** 使用HIAI_DVPP_DMalloc接口申请内存时，则由用户保证申请的内存大小与输入参数jpegDataSize的参数值一致。

成员变量	说明
uint32_t jpegDataSize	输入内存的长度。 使用Matrix提供的HIAI_DVPP_DMalloc接口申请内存，申请到的内存大小为“实际数据大小+8 byte”，8 byte为硬件约束要求。
bool isYUV420Need	是否需要输出yuv420 semi-planar格式的数据。 ● true：是 ● false：否，保持源格式输出。 JPEGD支持raw格式（包括yuv420sp、yuv422sp、yuv444sp）和降采样为yuv420的半平面格式输出。其中灰度图片输出的yuv420为fake420形式。
bool isVBeforeU	该参数值只能配置为true，v在u前，预留字段。

出参：JpegdOut

表 2-5 出参 JpegdOut

成员变量	说明
unsigned char* yuvData	输出yuv图片数据buf，该图片的宽高为128*16对齐后的宽高。 如果由用户指定内存，使用Matrix提供的HIAI_DVPP_DMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。HIAI_DVPP_DMalloc接口的说明，请参见《 Matrix API参考 》。 **须知** 使用HIAI_DVPP_DMalloc接口申请内存时，则由用户保证申请的内存大小与输入参数yuvDataSize的参数值一致。
uint32_t yuvDataSize	输出yuv数据的长度，数据长度由对齐后的宽高计算，也可以通过调用**DvppGetOutParameter接口**获取数据长度。
uint32_t imgWidth	输出yuv图片的宽度。
uint32_t imgHeight	输出yuv图片的高度。
uint32_t imgWidthAligned	输出图片的对齐后的宽度，对齐到128。
uint32_t imgHeightAligned	输出图片的对齐后的高度，对齐到16。

成员变量	说明
JpegCalBackFree cbFree	释放输出内存的回调函数。 ● 由用户指定输出内存时，由用户自行释放内存。 ● 不由用户指定输出内存时，DVPP内部申请内存，需由用户调用cbFree()回调函数释放内存，并将yuvData置为空指针。调用示例请参见**6.3 实现JPEGD功能**。

jpegd_color_space outFormat	输出yuv数据格式：
	enum jpegd_color_space{
	DVPP_JPEG_DECODE_OUT_UNKNOWN = -1,
	DVPP_JPEG_DECODE_OUT_YUV444 = 0,
	DVPP_JPEG_DECODE_OUT_YUV422_H2V1 = 1, /* YUV422 */
	DVPP_JPEG_DECODE_OUT_YUV422_H1V2 = 2, /* YUV440 */
	DVPP_JPEG_DECODE_OUT_YUV420 = 3,
	DVPP_JPEG_DECODE_OUT_YUV400 = 4,
	DVPP_JPEG_DECODE_OUT_FORMAT_MAX,
	};

入参：jpegd_raw_data_info

表 2-6 入参 jpegd_raw_data_info

成员变量	说明
unsigned char* jpeg_data	输入jpg图片数据，起始地址128对齐，2M大页表方式申请。 使用Matrix提供的HIAI_DVPP_DMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。HIAI_DVPP_DMalloc接口的说明，请参见《 Matrix API参考 》。 **须知** 使用HIAI_DVPP_DMalloc接口申请内存时，则由用户保证申请的内存大小与输入参数jpeg_data_size的参数值一致。
uint32_t jpeg_data_size	输入内存的长度。 使用Matrix提供的HIAI_DVPP_DMalloc接口申请内存，申请到的内存大小为“实际数据大小+8 byte”，8 byte为硬件约束要求。

成员变量	说明

jpegd_raw_format in_format	输入图片中yuv的采样格式，不需要填充，默认值即可。
	enum jpegd_raw_format{
	DVPP_JPEG_DECODE_RAW_YUV_UNSUPPORT = -1,
	DVPP_JPEG_DECODE_RAW_YUV444 = 0,
	DVPP_JPEG_DECODE_RAW_YUV422_H2V1 = 1, // 422
	// yuv440 不再支持，保留字段
	DVPP_JPEG_DECODE_RAW_YUV422_H1V2 = 2, // 440
	DVPP_JPEG_DECODE_RAW_YUV420 = 3,
	DVPP_JPEG_DECODE_RAW_YUV400 = 4,
	DVPP_JPEG_DECODE_RAW_MAX,
	};

bool IsYUV420Need	是否需要输出yuv420 semi-planar格式的数据。 ● true：是 ● false：否，保持源格式输出。 JPEGD支持raw格式（包括yuv420sp、yuv422sp、yuv444sp）和降采样为yuv420的半平面格式输出。其中灰度图片输出的yuv420为fake420形式。
bool isVBeforeU	该参数值只能配置为true，v在u前，预留字段。

出参：jpegd_yuv_data_info

表 2-7 出参 jpegd_yuv_data_info

成员变量	说明
unsigned char* yuv_data	输出yuv图片数据buf，该图片的宽高为对齐后的宽高。 **须知** 该内存是由DVPP内部申请并管理，不能由用户指定。
uint32_t yuv_data_size	输出yuv数据的长度，数据长度由对齐后的宽高计算。
uint32_t img_width	输出yuv图片的宽度。
uint32_t img_height	输出yuv图片的高度。
uint32_t img_width_aligned	输出图片的对齐后的宽度，对齐到128。

成员变量	说明
uint32_t img_height_aligned	输出图片的对齐后的高度，对齐到16。
JpegCalBackFree cbFree	释放输出内存的回调函数。 DVPP内部申请内存，需由用户调用cbFree()回调函数释放内存，并将yuv_data置为空指针。调用示例请参见6.3 实现JPEGD功能。

enum jpegd_color_space out_format	输出yuv数据格式：
	enum jpegd_color_space {
	DVPP_JPEG_DECODE_OUT_UNKNOWN = -1,
	DVPP_JPEG_DECODE_OUT_YUV444 = 0,
	DVPP_JPEG_DECODE_OUT_YUV422_H2V1 = 1,// 422
	// yuv440 不再支持，保留字段
	DVPP_JPEG_DECODE_OUT_YUV422_H1V2 = 2,// 440
	DVPP_JPEG_DECODE_OUT_YUV420 = 3,
	DVPP_JPEG_DECODE_OUT_YUV400 = 4,
	DVPP_JPEG_DECODE_OUT_FORMAT_MAX,
	};

2.2.5 PNGD 参数说明

入参：PngInputInfoAPI

表 2-8 入参 PngInputInfoAPI

成员变量	说明
void* inputData	输入图像数据的地址。 使用Matrix提供的HIAI_DVPP_DMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。HIAI_DVPP_DMalloc接口的说明，请参见《 Matrix API参考 》。 **须知** 使用HIAI_DVPP_DMalloc接口申请内存时，则由用户保证申请的内存大小与输入参数inputSize的参数值一致。
uint64_t inputSize	输入内存长度，用于校验输入数据。
void* address	输入图像数据的地址。当前版本不使用该参数。

成员变量	说明
uint64_t size	输入内存长度。当前版本不使用该参数。
int32_t transformFlag	转换标志，1表示RGBA转换到RGB，0保留原格式。

出参：PngOutputInfoAPI

表 2-9 出参 PngOutputInfoAPI

成员变量	说明
void* outputData	输出图像数据的地址。当前版本不使用该参数。
uint64_t outputSize	输出图像数据的地址。当前版本不使用该参数。
void* address	输出内存地址。 如果由用户指定内存，使用Matrix提供的HIAI_DVPP_DMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。HIAI_DVPP_DMalloc接口的说明，请参见《 Matrix API参考 》。 **须知** 使用HIAI_DVPP_DMalloc接口申请内存时，则由用户保证申请的内存大小与输入参数outputSize的参数值一致。
uint64_t size	内存大小。
int32_t format	输出图像格式： ● 2表示RGB输出。 ● 6表示RGBA输出。
uint32_t width	输出图像宽度。
uint32_t high	输出图像高度。
uint32_t widthAlign	宽度内存对齐，图片一行数据占用的内存大小进行128位对齐。若输出格式为RGB，则width*3后128位对齐，若输出格式为RGBA，则width*4后128位对齐。
uint32_t highAlign	高度对齐，目前为16位对齐。

2.2.6 查询 DVPP 引擎参数说明

功能

主要用于查询DVPP引擎的能力，包括各个模块的能力，各个模块的分辨率限制及性能参数等。

入参：device_query_req_stru

表 2-10 入参 device_query_req_stru

成员变量	说明	取值范围
uint32_t module_id	模块的ID。	固定为1
uint32_t engine_type	DVPP引擎类型。	VDEC：0 JPEGD：1 PNGD：2 JPEGE：3 VPC：4 VENC：5

出参：dvpp_engine_capability_stru

表 2-11 出参 dvpp_engine_capability_stru

成员变量	说明	取值范围
int32_t engine_type	DVPP引擎类型。	VDEC：0 JPEGD：1 PNGD：2 JPEGE：3 VPC：4 VENC：5
struct dvpp_resolution_stru max_resolution	最大分辨率。	详细见**7.4 dvpp_engine_capability_stru中的结构体**。
struct dvpp_resolution_stru min_resolution;	最小分辨率。	详细见**7.4 dvpp_engine_capability_stru中的结构体**。

成员变量	说明	取值范围
uint32_t protocol_num;	引擎所支持的标准协议类型数量。	VDEC：5 JPEGD：1 PNGD：1 JPEGE：1 VPC：0 VENC：4

uint32_t protocol_type[DVPP_PROTOCOL_TYPE_MAX];	引擎所支持的标准协议类型表格。	*enum* dvpp_proto_type {
		dvpp_proto_unsupport =-1,
		dvpp_itu_t81,
		iso_iec_15948_2003,
		h265_main_profile_level_5_1_hightier,
		h265_main_10_profile_level_5_1_hightier,
		h264_main_profile_level_5_1,
		h264_baseline_profile_level_5_1,
		h264_high_profile_level_5_1,
		h264_high_profile_level_4_1,
		h264_main_profile_level_4_1,
		h264_baseline_profile_level_4_1,
		h265_main_profile_level_4_1
		};

uint32_t input_format_num;	支持的输入格式数量。	VDEC：2 JPEGD：1 PNGD：1 JPEGE：6 VPC：51 VENC：2
struct dvpp_format_unit_stru engine_input_format_table[DVPP_VADIO_FORMAT_MAX];	引擎所支持的输入格式表。	详细见**7.4 dvpp_engine_capability_stru中的结构体**。

成员变量	说明	取值范围
uint32_t output_format_num;	支持的输出格式数量。	VDEC：4 JPEGD：4 PNGD：2 JPEGE：1 VPC：2 VENC：2
struct dvpp_format_unit_stru engine_output_format_table[DVPP_VADIO_FORMAT_MAX];	引擎所支持的输出格式表结构体。	详细见**7.4 dvpp_engine_capability_stru中的结构体**。
uint32_t performance_mode_num;	性能模式的数量。	固定为1。
struct dvpp_perfomance_unit_stru performance_mode_table[DVPP_PERFOMANCE_MODE_MAX];	模块的性能结构体。	详细见**7.4 dvpp_engine_capability_stru中的结构体**。
struct dvpp_pre_contraction_stru pre_contraction;	预缩小信息结构体。	详细见**7.4 dvpp_engine_capability_stru中的结构体**。
struct dvpp_pos_scale_stru pos_scale;	后缩放信息结构体。	详细见**7.4 dvpp_engine_capability_stru中的结构体**。
uint32_t spec_input_num	输入格式所属类型的数目，目前支持两种类型： ● HFBC ● YUV或RGB	-
struct dvpp_vpc_data_spec_stru spec_input[DVPP_DATA_INPUT_SPEC_TYPE_MAX]	输入格式所属类型的描述。	详细见**•dvpp_vpc_data_spec_stru...**。

2.3 DestroyDvppApi

函数原型	**int DestroyDvppApi(IDVPPAPI*& pIDVPPAPI)**
功能	销毁由**CreateDvppApi**接口创建的dvppapi实例，关闭DVPP执行器。

输入说明	输入为“**IDVPPAPI**”类型指针引用。
输出说明	无。
返回值说明	● 返回值“0”代表成功。 ● 返回值“-1”代表失败。
使用说明	无。
使用约束	一旦调用**DestroyDvppApi**接口，如果还想在继续调用DVPP，需要调用**CreateDvppApi接口**重新创建dvppapi实例。

调用示例

```
DestroyDvppApi(pidvppapi);
```

2.4 DvppGetOutParameter

函数原型	**int32_t DvppGetOutParameter(void* in, void* out, int32_t cmd)**
功能	获取JPEGD/JPEGE/PNGD模块的输出内存大小。
输入说明	● in指针，各模块对应的指针不同，请参见**表2-12**。 ● out指针，各模块对应的指针不同，请参见**表2-12**。 ● **cmd控制命令字**
输出说明	各模块输出内存的大小： ● JPEGE功能，取**sJpegeOut**结构体中的jpgSize参数值。 ● JPEGD功能，取**JpegdOut**结构体中的yuvDataSize参数值。 ● PNGD功能，取**PngOutputInfoAPI**结构体中的outputSize参数值。
返回值说明	● 返回值“0”代表成功。 ● 返回值“-1”代表失败。
使用说明	调用方调用“**DvppGetOutParameter**”函数，传入in和out指针，并传入正确的**cmd控制命令字**。 调用示例如下： ● JPEGD模块，请参见**6.3 实现JPEGD功能**。 ● JPEGE模块，请参见**6.2 实现JPEGE功能**。 ● PNGD模块，请参见**6.4 实现PNGD功能**。
使用限制	无。

表 2-12 入参说明

入参	说明	取值范围
void* in	输入结构体指针	● JPEGE功能入参：**sJpegeIn** ● JPEGD功能入参：**JpegdIn** ● PNGD功能入参：**PngInputInfoAPI** **须知** 该函数不支持JPEGD 的**jpegd_raw_data_info**结构体。
void* out	输出结构体指针	● JPEGE功能出参：**sJpegeOut** ● JPEGD功能出参：**JpegdOut** ● PNGD功能出参：**PngOutputInfoAPI** **须知** 该函数不支持JPEGD 的**jpegd_yuv_data_info**结构体。
int32_t cmd	获取输出参数控制命令字	● JPEGE功能的命令字：GET_JPEGE_OUT_PARAMETER ● JPEGD功能的命令字：GET_JPEGD_OUT_PARAMETER ● PNGD功能的命令字：GET_PNGD_OUT_PARAMETER

3 VDEC 功能接口

3.1 总体说明

对于一个视频码流，调用一次**CreateVdecApi**接口创建实例后，必须使用同一个实例调用**VdecCtl**接口进行视频解码，最后再调用一次**DestroyVdecApi**接口释放实例。

在视频解码时，如果需要从一个视频码流切换到另一个视频码流，则需要先调用**DestroyVdecApi**接口释放前一个码流的实例，再调用**CreateVdecApi**接口创建新的实例，用于处理新的视频码流。

3.2 CreateVdecApi

函数原型	**int CreateVdecApi(IDVPPAPI*& pIDVPPAPI, int singleton)**
功能	获取vdecapi实例，相当于vdec执行器句柄。调用方可以使用申请到的vdecapi实例调用**CreateVdecApi接口**进行视频解码，可以跨函数调用，跨线程调用。
输入说明	• **IDVPPAPI**类型指针引用，输入指针必须为NULL。 • singleton为内部保留使用，为以后实现pIDVPPAPI单例预留，建议调用方当前设置为0。
输出说明	输出为**IDVPPAPI**类型指针引用，输出可能为NULL，可能不为NULL，获取失败则为NULL，成功则不为NULL。
返回值说明	• 返回值“0”代表成功。 • 返回值“-1”代表失败。

使用说明	调用方创建**IDVPPAPI**对象指针，初始化为NULL，调用**CreateVdecApi**函数将**IDVPPAPI**对象指针传入。如果申请成功，**CreateVdecApi**接口会返回DvppApi实例，否则返回NULL。调用方需要对返回值进行校验。
使用约束	调用方负责vdecapi实例的生命周期，即申请与释放，申请使用**CreateVdecApi**接口，释放使用**DestroyVdecApi接口**。

3.3 VdecCtl

函数原型	**int VdecCtl(IDVPPAPI*& pIDVPPAPI, int CMD, dvppapi_ctl_msg* MSG, int singleton)**
功能	使用**CreateVdecApi**接口创建的实例调用**VdecCtl**接口，控制DVPP执行器进行视频解码。
输入说明	• **IDVPPAPI**类型指针引用。 • 控制命令字CMD(VDEC为DVPP_CTL_VDEC_PROC)。 • **dvppapi_ctl_msg**类型的vdec执行器配置信息MSG。其中此结构体中的in请见**入参：vdec_in_msg**。 • 输入singleton为内部保留使用，为以后实现pIDVPPAPI单例预留，建议调用方当前设置为0。
输出说明	输出为MSG的配置信息中的输出buffer，以及VDEC的输出状态信息(都存储于MSG中)。
返回值说明	• 返回值“0”代表接口调用成功，并不代表解码成功（由于VDEC解码是异步方式）。 • 返回值“-1”代表接口调用失败。
使用说明	调用方调用**VdecCtl**接口，传入**IDVPPAPI**对象指针，配置好相应功能的**dvppapi_ctl_msg**，并传入正确的控制命令字CMD。VDEC解码为异步方式，调用**VdecCtl**接口会将用户码流buffer中的数据拷贝到VDEC内部输入buffer后即返回，故调用**VdecCtl**接口后并不代表解码成功（仅表示数据拷贝成功），并且此接口返回后，用户码流buffer即可释放。
使用约束	对于一个视频码流，调用一次**CreateVdecApi接口**创建实例后，必须使用同一个实例调用**VdecCtl**接口进行视频解码，最后再调用一次**DestroyVdecApi接口**释放实例。

入参：vdec_in_msg

表 3-1 入参 vdec_in_msg

成员变量	说明
char video_format[10]	输入视频格式，“h264”或者“h265”，默认是“h264”， 仅支持yuv420sp(NV12、NV21)编码后的h264和h265码流。
char image_format[10]	输出帧格式，“nv12”或者“nv21”，默认是“nv12”。
void (*call_back)(FRAME* frame,void * hiai_data)	调用方回调函数，FRAME为vdec解码后的输出结构体，详见**7.2 vdec_in_msg中的结构体和类 FRAME结构体**。用户可以根据该指针获取输出结果。 建议用户在回调函数内仅调用**DvppCtl**接口输出yuv格式的图片数据，其它功能不建议放在回调函数内实现，因为回调函数内实现的功能太多，可能会耗时过长，导致解码过程阻塞等待资源。回调函数允许的最大耗时和帧率相关，计算公式为：最大耗时=1/帧率，例如帧率=30fps，则最大耗时=1/（30fps）=0.033s；帧率=25fps，则允许最大耗时=1/（25fps）=0.04s。
char* in_buffer	输入视频码流内存，此码流为h264或h265裸码流。 用户将存放解码前码流的内存（由用户自行申请）赋值给in_buffer，调用**VdecCtl**接口有返回后，就可以释放存放解码前码流的buffer。
int32_t in_buffer_size	输入视频码流内存大小。
void * hiai_data	解码后输出结果帧回调函数的形参指针，指针指向对象由调用方定义具体的结构。 **须知** VDEC内部仅调用第一次传给VDEC的hiai_data，若要想多次使用hiai_data传送不同对象，请用下面智能指针对象。

成员变量	说明
std::shared_ptr<HIAI_DATA_SP> hiai_data_sp	如果不涉及帧序号的配置，可不设置该参数，默认为NULL。如果需要设置帧序号，使用方法如下。 调用方回调函数形参指针，其中HIAI_DATA_SP为VDEC内部定义的父类，具体类HIAI_DATA_SP结构请见**7.2 vdec_in_msg中的结构体和类 HIAI_DATA_SP类**，用户可以继承衍生子类，使用方式可以参考**6.5 实现VDEC功能**。此指针指向的类对象必须设置帧序号信息，只支持一帧（必须包含I帧或P帧或B帧）对应一个帧序号，不支持多帧对应同一个帧序号，并且帧序号需以等差数列方式设置，从1开始，间隔大小为1。 **使用注意点：** 1. 在用户自定义子类hiai_data_sp对象中，不能包含需申请过大内存空间的成员变量。因为针对每路视频码流的解码，VDEC内部最多支持100个hiai_data_sp对象，如果成员变量申请的内存空间过大，可能会导致内存消耗过大； 2. 用户自定义子类hiai_data_sp对象中若有需要申请内存空间的成员变量，在申请内存并使用完成后，需在析构函数中释放内存，避免内存泄露； 3. 使用VDEC进行视频码流解码时，如果使用了hiai_data_sp对象，但视频码流中存在异常帧，VDEC会直接将异常帧的hiai_data_sp对象丢弃，因此建议用户在析构函数的实现代码中对这个异常情况做处理。 4. 使用VDEC进行视频码流解码时，如果使用了hiai_data_sp对象，仅支持码流参考帧间距不超出30帧，例如解码第30帧时可以参考第1帧，但解码第31帧时不能再参考第1帧。 5. VDEC会将用户传入的hiai_data_sp对象缓存至内部队列，队列最大长度为100。若某帧解码失败则对应的hiai_data_sp对象最快会在之后30帧解码全部返回的时刻被丢弃，例如第1帧解码失败，第1帧对应的hiai_data_sp对象会在第31帧解码结束后被丢弃；若码流中全是异常帧，则第1帧对应的hiai_data_sp对象会在第101帧开始解码时被丢弃。 6. hiai_data_sp和hiai_data只能选其一，如果使用hiai_data_sp，则必须与channelId参数配合使用。 7. 若码流中含有B帧，帧序号只支持按显示顺序输出，不支持按解码顺序输出。
int32_t channelId	输入码流对应的解码通道的ID，不同码流同时解码需设置不同的值，取值范围0~15。

成员变量	说明
void (*err_report) (VDECERR* vdecErr)	错误上报回调函数，用于将解码过程中出现的异常通知用户，比如码流错误、硬件问题、解码器状态错误等，以便用户决定如何处理。其中形参VDECERR为VDEC内部定义结构体，具体请见**7.2 vdec_in_msg中的结构体和类 VDECERR结构体**。
bool isEOS	码流结束标志，标识本路解码结束。用户可以不用关心此标志，默认在**DestroyVdecApi**接口中会将isEOS设置true，并在内部实现结束本路解码同时释放资源。若用户主动配置isEOS为true，则在调用VdecCtl接口中优先使用用户配置，结束本路解码并释放资源。具体使用请见**6.5 实现VDEC功能**。
int32_t isOneInOneOutMode	预留参数，请保持默认值0。

3.4 DestroyVdecApi

函数原型	**int DestroyVdecApi(IDVPPAPI*& pIDVPPAPI, int singleton)**
功能	释放由**CreateVdecApi**接口创建的vdecapi实例，关闭VDEC执行器。
输入说明	输入为**IDVPPAPI**类型指针引用。 输入singleton为内部保留使用，为以后实现pIDVPPAPI单例预留，建议调用方当前设置为0。
输出说明	无输出。
返回值说明	● 返回值“0”代表成功。 ● 返回值“-1”代表失败。
使用说明	无特殊说明。
使用约束	一旦调用**DestroyVdecApi**接口，如果还想在继续调用VDEC，需要重新创建vdecapi实例。

4 VENC 功能接口

4.1 总体说明

若需要将多张图片编码成一个视频，则调用一次**CreateVenc**接口创建实例后，必须使用同一个实例调用**RunVenc**接口进行视频编码，最后再调用一次**DestroyVenc**接口释放实例。

4.2 CreateVenc

函数原型	**int32_t CreateVenc(struct VencConfig* vencConfig)**
功能	获取VENC编码实例，相当于获取VENC执行器句柄。调用方可以使用申请到的VENC编码实例调用**RunVenc接口**进行图片编码。
输入说明	输入为结构体VencConfig类型指针，VencConfig参见说明入参：**入参：VencConfig**。其中需要配置回调函数，用于处理编码结果。
输出说明	无输出。
返回值说明	● 返回值非负数代表创建VENC编码实例成功 ● 返回值“-1”代表创建VENC编码实例失败。
使用说明	调用方创建VencConfig对象指针，调用**CreateVenc**接口将VencConfig对象指针传入。如果申请成功，**CreateVenc**接口会返回VENC编码实例句柄号，否则返回-1。调用方需要对返回值进行校验。
使用约束	调用方负责VENC编码实例的生命周期，即申请与释放，申请使用CreateVenc接口，释放使用**DestroyVenc接口**。

入参：VencConfig

该入参是初始化VENC模块时使用，结构体所有成员变量必须初始化后再使用。

表 4-1 入参 VencConfig

成员变量	说明	取值范围
uint32_t width	图像宽度。	128~1920，且为偶数。
uint32_t height	图像高度。	128~1920，且为偶数。
uint32_t codingType	视频编码协议H265-main level（0）、H264-baseline level（1）、H264-main level(2)、H264-high level（3）	0~3 ● 0：H265 main level（仅支持Slice码流）。 ● 1：H264 baseline level。 ● 2：H264 main level。 ● 3：H264 high level。
uint32_t yuvStoreType	YUV图像存储格式。	0或者1 ● 0：YUV420 semi-planner（nv12） ● 1：YVU420 semi-planner（nv21）
uint32_t keyFrameInterval	I帧间隔	大于0小于65535
VencOutMsgCallBack vencOutMsgCallBack	回调函数，用于处理编码结果。 DVPP（VENC模块）处理每一路图片数据的编码时，对于首帧图片数据，会调用两次回调函数，第一次调用回调函数处理文件头信息，第二次调用回调函数处理本帧的图片数据	typedef void (*VencOutMsgCallBack) (struct VencOutMsg* vencOutMsg, void* userData);不可为空
void* userData	用户记录想要传递的信息，随回调函数返回给用户	可以为NULL

4.3 RunVenc

函数原型	**int32_t RunVenc(int32_t vencHandle, struct VencInMsg* vencInMsg)**

功能	使用**CreateVenc**接口创建的实例调用**RunVenc**接口，控制DVPP执行器进行视频编码。
输入说明	输入为int32_t句柄号和VencInMsg指针。vencHandle为CreateVenc返回值。VencInMsg参见入参：**入参：VencInMsg**。VENC执行器配置信息VencInMsg，该结构体用于将要编码的视频信息传递给执行器进行编码。
输出说明	无输出。
返回值说明	● 返回值“0”代表成功。 ● 返回值“-1”代表失败。
使用说明	调用方调用RunVenc函数，传入编码实例句柄号和VencInMsg对象指针，配置好相应功能的vencInMsg。
使用约束	若需要将多张图片编码成一个视频，则调用一次**CreateVenc接口**创建实例后，必须使用同一个实例调用**RunVenc**接口进行视频编码，最后再调用一次**DestroyVenc**接口释放实例。

入参：VencInMsg

该入参是调用VENC模块执行编码时使用，所有结构体成员变量必须初始化后再使用。

表 4-2 入参 VencInMsg

成员变量	说明	取值范围
void* inputData	输入数据地址	非空
uint32_t inputDataSize	输入数据大小	不能大于输入数据buffer大小，推荐值和输入数据buffer大小一样
uint32_t keyFrameInterval	I帧间隔	大于等于0小于65535，0表示这个参数不起作用
uint32_t forceIFrame	强制重新开始I帧间隔，0：不强制；1：强制重新开始I帧	0或者1
uint32_t eos	是否为结束帧，0：不是；1：是结束帧	0或者1

出参：VencOutMsg

表 4-3 出参 VencOutMsg

成员变量	说明	取值范围
void* outputData	输出数据地址	由VENC内部申请

成员变量	说明	取值范围
uint32_t outputDataSize	输出数据大小	由VENC内部填写
uint32_t timeStamp	记录调用回调函数的时序	由VENC内部填写

4.4 DestroyVenc

函数原型	**int32_t DestroyVenc(int32_t vencHandle)**
功能	释放由**CreateVenc**接口创建的VENC编码实例，关闭VENC执行器。
输入说明	输入为int32_t类型VENC编码实例句柄号，为调用函数CreateVenc返回值。
输出说明	无输出。
返回值说明	• 返回值“0”代表成功。 • 返回值“-1”代表失败。
使用说明	无特殊说明。
使用约束	一旦调用**DestroyVenc**接口，如果还想在继续调用VENC，需要重新创建VENC编码实例。

5 兼容旧版本的功能

5.1 兼容性说明

为了兼容旧版本中的功能，当前版本支持以下方式实现VPC功能和VENC功能：

- 实现VPC功能：
  - 在**DvppCtl**接口中传入入参：DVPP_CTL_VPC_PROC命令字和**resize_param_in_msg结构体**的参数。
  - 对时延要求不高、处理的图片分辨率低、数量多的场景，可以在**DvppCtl**接口中传入入参：DVPP_CTL_CMDLIST_PROC和**IMAGE_CONFIG结构体**的参数。

  这种方式是为了兼容旧版本中的功能，后续版本会删除，不推荐使用。如果使用了不推荐使用的功能，建议迁移到推荐的VPC功能，请参见**2 VPC/JPEGE/JPEGD/PNGD功能接口**。
- 实现VENC功能：调用**CreateDvppApi**、**DvppCtl**、**DestroyDvppApi**接口实现VENC功能，在**DvppCtl**接口中传入入参：DVPP_CTL_VENC_PROC命令字和**venc_in_msg结构体**的参数。

  这种方式在为了兼容旧版本中的功能，后续版本会删除，不推荐使用。如果使用了不推荐使用的功能，建议迁移到推荐的VENC功能，请参见**4 VENC功能接口**。

5.2 VPC 功能及参数说明

不推荐使用旧版本中的VPC功能，后续版本会删除。旧版本中的VPC功能和CMDLIST功能均迁移到VPC功能，请参见2.2.2 VPC参数说明处的参数说明进行开发。

旧版本中的VPC功能：主要用于处理媒体图像转换，包括缩放、色域转换、降bit数处理、格式转换、区块切割，叠加拼接，CMDLIST。

- VPC路数无限制，其性能指标如下表。

  VPC性能由于涉及到抠图、缩放等不同的场景，当处理图片过程中的分辨率改变时，性能会对应改变。当处理过程中图像分辨率都没有比原图分辨率大时，典型场景性能指标如下：

场景	总帧率
1080p*32路	1440fps
4k*4路	360fps

  当处理过程中图像分辨率大于原图分辨率时，总帧率计算公式参考如下：

  总帧率=(3840*2160*360)/(W*H) fps

  其中：W和H为VPC处理过程中最大的宽度和高度，例如1080p抠图到960*540，再缩放到3840*2160，此时W为3840，H为2160。

  在处理过程中，缩小比例的增大会导致VPC处理性能有略微下降。

- VPC输入输出图像宽高限制：
  - 输入图像宽高都需要2对齐（偶数）。
  - 输出图像宽高都需要2对齐（偶数）。

须知

此处所说的图像宽高是指图片的有效区域，实际上对于送进VPC的内存需要128*16对齐。

- VPC输入输出图像内存限制：
  - VPC输入内存限制
    - 输入数据地址（os虚拟地址）：128字节对齐。
    - 输入图像宽度内存限制：128字节对齐。
    - 输入图像高度内存限制：16字节对齐。

说明

对于yuv400sp格式图片，需要调用方申请和yuv420sp格式相同大小的空间，即：width_align_128*high_align_16*1.5。

  - VPC输出内存限制
    - 输出数据地址（os虚拟地址）：128字节对齐。
    - 输出图像宽内存限制：128字节对齐。
    - 输出图像高内存限制：16字节对齐。

说明

在使用VPC实现图像裁剪与缩放时，共需要调用两次DvppCtl，第一次调用的输入参数为resize_param_in_msg，输出参数为resize_param_out_msg，第二次调用的输入参数为vpc_in_msg，详细信息请参见**入参：resize_param_in_msg**~**入参：vpc_in_msg**。

入参：resize_param_in_msg

成员变量	说明	取值范围
int src_width	原图的宽度（2对齐）。 **须知** 原图指图片的有效区域，实际上对于送进VPC的内存需要128*16对齐。	最小分辨率： ● 128（来源VDEC） ● 16（来源非VDEC） 最大分辨率： 4096 必填
int src_high	原图的高度（2对齐）。 **须知** 原图指图片的有效区域，实际上对于送进VPC的内存需要128*16对齐。	最小分辨率： ● 128（来源VDEC） ● 16（来源非VDEC） 最大分辨率： 4096 必填
int hmax	与原点在水平方向的最大偏移。	奇数，具体参见**入参：vpc_in_msg**。
int hmin	与原点在水平方向的最小偏移。	偶数，具体参见**入参：vpc_in_msg**。
int vmax	与原点在垂直方向的最大偏移。	奇数，具体参见**入参：vpc_in_msg**。
int vmin	与原点在垂直方向的最小偏移。	偶数，具体参见**入参：vpc_in_msg**。
int dest_width	输出图像宽度。	偶数。
int dest_high	输出图像高度。	偶数。

出参：resize_param_out_msg

成员变量	说明	取值范围
int dest_width_stride	输出图像的宽度对齐值。	128
int dest_high_stride	输出图像的高度对齐值。	16

成员变量	说明	取值范围
int hmax	与原点在水平方向的最大偏移。	奇数，具体参见**入参：vpc_in_msg**。
int hmin	与原点在水平方向的最小偏移。	偶数，具体参见**入参：vpc_in_msg**。
int vmax	与原点在垂直方向的最大偏移。	奇数，具体参见**入参：vpc_in_msg**。
int vmin	与原点在垂直方向的最小偏移。	偶数，具体参见**入参：vpc_in_msg**。
double hinc	水平缩放系数。	具体参见**入参：vpc_in_msg**。
double vinc	垂直缩放系数。	具体参见**入参：vpc_in_msg**。

入参：vpc_in_msg

成员变量	说明	取值范围
int format	图像格式，调用方可以自行定义，只要后面的数字对应正确即可。例如：yuv420_semi_plannar对应的是0，调用方声明为yuv420sp，那么也必须对应为0。	yuv420_semi_plannar = 0,//0 yuv400_semi_plannar = 0,//0 yuv422_semi_plannar,//1 yuv444_semi_plannar,//2 yuv422_packed,//3 yuv444_packed,//4 rgb888_packed,//5 xrgb8888_packed,//6 必填
int rank	图像排列方式，调用方可以自行定义，只要后面的数字对应正确即可。	具体参见**表5-1** 必填
int bitwidth	位深，默认是8bit。10bit只有在图像格式为YUV/YVU420 Semi-Planar且为HFBC压缩场景下使用。	8或者10 选填
int cvdr_or_rdma	输入图像走CVDR通道还是RDMA，默认走CVDR通道，RDMA通道的输入仅为VDEC输出的HFBC格式数据。	CVDR：1，RDMA：0。 选填

成员变量	说明	取值范围
int width	输入原图像宽度，2对齐（偶数）。 **须知** 原图指图片的有效区域，实际上对于送进VPC的内存需要128*16对齐。	最小分辨率： ● 128（来源VDEC）。 ● 16（来源非VDEC）。 最大分辨率： 4096 必填
int high	输入原图像高度，2对齐（偶数）。 **须知** 原图指图片的有效区域，实际上对于送进VPC的内存需要128*16对齐。	最小分辨率： ● 128（来源VDEC）。 ● 16（来源非VDEC）。 最大分辨率： 4096 必填
int stride	图像步长。 **须知** 对于不同输入格式，其stride不同。	yuv400sp、yuv420sp、yuv422sp、yuv444sp：width对齐到128。 yuv422packed：width * 2后对齐到128 yuv444packed、rgb888：width * 3后对齐到128 xrgb8888：width * 4后对齐到128 必填
double hinc	水平放大倍数，aicore输出通道配置。	[0.03125, 1) or (1, 4] 缩放系数配置，超出配置范围，VPC会提示出错，如果不要缩放填1。 必填
double vinc	垂直放大倍数，aicore输出通道配置。	[0.03125, 1) or (1, 4] 缩放系数配置，超出配置范围，VPC会提示出错，如果不要缩放填1。 必填
double jpeg_hinc	水平放大倍数，jpege输出通道配置。	[0.03125, 1) or (1, 4] 缩放系数配置，超出配置范围，VPC会提示出错。 选填(目前不适用)
double jpeg_vinc	垂直放大倍数，jpege输出通道配置。	[0.03125, 1) or (1, 4] 缩放系数配置，超出配置范围，VPC会提示出错。 选填(目前不适用)

成员变量	说明	取值范围
int hmax	与原点在水平方向的最大偏移。	此值必须为奇数,如果不用裁剪,此值配置成输入宽度-1。 必填
int hmin	与原点在水平方向的最小偏移。	此值必须为偶数。 如果不用裁剪，此值配置成0。 必填
int vmax	与原点在垂直方向的最大偏移。	此值必须为奇数。 如果不用裁剪，此值配置成输入宽度-1。 必填
int vmin	与原点在垂直方向的最小偏移。	此值必须为偶数。 如果不用裁剪，此值配置成0。
int out_width	输出图像宽度。	此值必须为偶数(2对齐)。 当使用智能指针申请输出buffer时，不需要填，当使用自己申请的内存作为输出buffer时，必填。
int out_high	输出图像高度。	此值必须为偶数(2对齐)。 当使用智能指针申请输出buffer时，不需要填，当使用自己申请的内存作为输出buffer时，必填。
int h_stride	图像高度步长，同int: stride描述。此值无需配置。	此值默认为16，即默认输入高度为16对齐。 选填
char* in_buffer	输入buffer。	调用方需保证输入buffer的大小与in message中配置的长度与宽度一致，即：如果输入是yuv420 semi planner nv12格式的图片，其图像大小为：宽*高*1.5，而输入的buffer大小与如上计算结果不符，则会导致VPC调用失败。 必填
int in_buffer_size	输入buffer大小。	此值用来进行输入buffer校验。 必填
shared_ptr<AutoBuffer> auto_out_buffer_1	输出buffer1。	使用VPC基础功能（裁剪，缩放）需要配置此buffer，此buffer为调用方申请智能指针当做参数传入DVPP，由DVPP内部申请输出buffer。用户可读取该buffer获取输出图片。 与下一个参数out_buffer选填一个即可。

成员变量	说明	取值范围
char* out_buffer	输出buffer。	使用VPC基础功能（裁剪，缩放）需要配置此buffer，，此buffer为调用方申请的内存，需保证首地址128对齐。用户可读取该buffer获取输出图片。 与上一个参数auto_out_buffer_1选填一个即可。
int out_buffer_1_size	输出buffer1大小。	此值用来进行输出buffer校验。 如果采用shared_ptr<AutoBuffer> auto_out_buffer_1申请输出内存则选填，如果采用 char* out_buffer申请输出内存则必填。
shared_ptr<AutoBuffer> auto_out_buffer_2	此智能指针为vpc双通道的另外一个通道输出，为预留接口	--
int out_buffer_2_size	此buffersize为auto_out_buffer_2智能指针输出buffer2大小，为预留接口	--
int use_flag	使用vpc功能的标志。	● 0：vpc基础功能。 ● 1：Rawdata8k缩放。 ● 2：Rawdata叠加基础功能，预留功能。 ● 3：Rawdata拼接基础功能，预留功能。 默认为0。 选填
VpcOverLayReverse overlay	叠加结构体，预留。	预留
VpcCollageReverse collage	拼接结构体，预留。	预留
RDMACHANNEL rdma	rdma参数为HFBC数据专用配置结构体。	具体请参见**7.5.1 Rdma通道结构体RDMACHANNEL**。
VpcTurningReverse turningReverse1	Vpc优化预留接口。	具体请参见**7.5.2 Vpc内部优化结构体VpcTurningReverse**。
bool isVpcUseTurning1	标志是否使用Vpc优化，预留接口，此标志与turningReverse1配合使用。	--

成员变量	说明	取值范围
VpcTurningReverse turningReverse2	Vpc优化预留接口。	具体请参见**7.5.2 Vpc内部优化结构体VpcTurningReverse**。
bool isVpcUseTurning2	标志是否使用Vpc优化，预留接口，此标志与turningReverse2配合使用。	--
string *yuvscaler_paraset	yuvscaler_paraset指针接收VPC滤波参数集文件的路径和文件名数组。 **说明** • 参数集须在Device设备上。 • 请确保传入的文件路径是正确路径。	参数集文件路径为device侧的绝对文件路径＋文件名，如{"/home/HwHiAiUser/matrix/YUVScaler_pra.h"}。
unsigned int yuvscaler_paraset_size	yuvscaler_paraset指针指向的string数组元素个数（默认为1）。	yuvscaler_paraset_size<=10
unsigned int index	index变量对应yuvscaler_paraset的数组索引。	0<=index<10

表 5-1 输入图片格式与输出图片格式对应 rank 配置表

输入图片格式	VPC的rank参数配置	输出图片格式
YUV420sp NV12	NV12 = 0	YUV420sp NV21
YUV420sp NV21	NV21 = 1	YUV420sp NV21
YUV420sp NV12	NV12 = 1	YUV420sp NV12
YUV420sp NV21	NV21 = 0	YUV420sp NV12
YUV422sp NV16	NV16 = 1	YUV420sp NV12
YUV422sp NV61	NV61 = 0	YUV420sp NV12
YUV422sp NV16	NV16 = 0	YUV420sp NV21
YUV422sp NV61	NV61 = 1	YUV420sp NV21
YUV444sp 444spUV	444spUV = 1	YUV420sp NV12
YUV444sp 444spVU	444spVU= 0	YUV420sp NV12
YUV444sp 444spUV	444spUV = 0	YUV420sp NV21
YUV444sp 444spVU	444spVU= 1	YUV420sp NV21

输入图片格式	VPC的rank参数配置	输出图片格式
YUV422packet YUYV	YUYV = 2	YUV420sp NV12
YUV422packet YVYU	YVYU = 3	YUV420sp NV21
YUV422packet UYVY	UYVY = 4	YUV420sp NV12
YUV444packet YUV	YUV = 5	YUV420sp NV12
RGB888 RGB	RGB = 6	YUV420sp NV12
RGB888 BGR	BGR = 7	YUV420sp NV21
RGB888 RGB	BGR = 7	YUV420sp NV21
RGB888 BGR	RGB = 6	YUV420sp NV12
XRGB8888 RGBA	RGBA = 8	YUV420sp NV12
XRGB8888 RGBA	BGRA = 9	YUV420sp NV21

调用示例

- VPC基础功能调用示例

  gXxxxx为配置参数，参数配置与VPC调用分为两个阶段。

  Step1：通过DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM命令字调用DvppCtl，获取配置参数，DvppCtl的入参为resize_paramo_in_msg，出参为resize_param_out_msg。即通过resize_param_in_msg将原始图片的宽高，裁剪区域(hmax,hmin,vamx,vmin)，以及输出宽高等参数传给DVPP，DVPP通过计算将重新生成的裁剪区域(hmax,hmin,vamx,vmin)以及缩放比(hinc,vinc)返回给调用方。

  Step2：调用方将第一步返回的参数通过vpc_in_msg结构体传递给DVPP，并将DvppCtl调用命令字改为DVPP_CTL_VPC_PROC，执行并获得处理结果。

```
dvppapi_ctl_msg dvppApiCtlMsg;
vpc_in_msg vpcInMsg;
resize_param_in_msg resize_in_param;
resize_param_out_msg resize_out_param;
resize_in_param.src_width = gWidth;
resize_in_param.src_high = gHigh;
resize_in_param.hmax = gHmax;
resize_in_param.hmin = gHmin;
resize_in_param.vmax = gVmax;
resize_in_param.vmin = gVmin;
resize_in_param.dest_width = floor(gHinc * (gHmax - gHmin + 1) + 0.5);
resize_in_param.dest_high = floor(gVinc * (gVmax - gVmin + 1) + 0.5);
printf("width  =%d\n", gWidth);
printf("high   =%d\n", gHigh);
printf("hmax   =%d\n", gHmax);
printf("hmin   =%d\n", gHmin);
printf("vmax   =%d\n", gVmax);
printf("vmin   =%d\n", gVmin);
printf("out width   =%d\n", resize_in_param.dest_width);
printf("out hight   =%d\n", resize_in_param.dest_high );

dvppApiCtlMsg.in = (void *)(&resize_in_param);
dvppApiCtlMsg.out = (void *)(&resize_out_param);
IDVPPAPI *pidvppapi = NULL;
CreateDvppApi(pidvppapi);
```

```
if (DvppCtl(pidvppapi, DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM, &dvppApiCtlMsg) != 0) {
    printf("call DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM process faild!\n");
    DestroyDvppApi(pidvppapi);
    return;
}

int bufferSize        = 0;
vpcInMsg.format       = gFormat;
vpcInMsg.cvdr_or_rdma = gcvdr_or_rdma;
vpcInMsg.bitwidth = gBitwidth;
vpcInMsg.rank = gRank;
vpcInMsg.width = gWidth;
vpcInMsg.high = gHigh;
vpcInMsg.stride = gStride;
vpcInMsg.hmax = resize_out_param.hmax;
vpcInMsg.hmin = resize_out_param.hmin;
vpcInMsg.vmax = resize_out_param.vmax;
vpcInMsg.vmin = resize_out_param.vmin;
vpcInMsg.vinc = resize_out_param.vinc;
vpcInMsg.hinc = resize_out_param.hinc;
printf("resize_out_param.hinc = %lf\n", resize_out_param.hinc);
printf("resize_out_param.vinc = %lf\n", resize_out_param.vinc);
printf("resize_out_param.hmax = %d\n", resize_out_param.hmax);
printf("resize_out_param.hmin = %d\n", resize_out_param.hmin);
printf("resize_out_param.vmax = %d\n", resize_out_param.vmax);
printf("resize_out_param.vmin = %d\n", resize_out_param.vmin);

printf("format =%d\n", vpcInMsg.format);
printf("rank   =%d\n", vpcInMsg.rank);
printf("width  =%d\n", vpcInMsg.width);
printf("high   =%d\n", vpcInMsg.high);
printf("stride =%d\n", vpcInMsg.stride);
printf("hmax   =%d\n", vpcInMsg.hmax);
printf("hmin   =%d\n", vpcInMsg.hmin);
printf("vmax   =%d\n", vpcInMsg.vmax);
printf("vmin   =%d\n", vpcInMsg.vmin);
printf("vinc   =%F\n", vpcInMsg.vinc);
printf("hinc   =%F\n", vpcInMsg.hinc);
if (vpcInMsg.cvdr_or_rdma == 0) {
    printf("rdma channel\n");
    int buffersize ;
    char * payloadY ;
    char * payloadC ;
    char * headY ;
    char * headC ;
    FILE * rdmaimage = fopen(in_file_name, "rb");

    if (vpcInMsg.bitwidth == 10) {
        buffersize                    = vpcInMsg.width  * vpcInMsg.high * 2 + 512 * 1024;
        payloadY                       = (char*)memalign(128, buffersize);
        headY                         = payloadY + vpcInMsg.width  * vpcInMsg.high * 2;
        headC                         = headY + 256 * 1024;
        vpcInMsg.rdma.luma_payload_addr   = (long)payloadY;
        vpcInMsg.rdma.chroma_payload_addr = (long)payloadY + 640;
    } else {
        buffersize                    = vpcInMsg.width * vpcInMsg.high * 1.5 + 512 * 1024;
        payloadY                       = (char*)memalign(128, buffersize);
        payloadC                       = payloadY + vpcInMsg.width * vpcInMsg.high;
        headY                         = payloadC + vpcInMsg.width * vpcInMsg.high / 2;
        headC                         = headY + 256 * 1024;
        vpcInMsg.rdma.luma_payload_addr   = (long)payloadY;
        vpcInMsg.rdma.chroma_payload_addr = (long)payloadC;
    }

    fread(payloadY, 1, buffersize, rdmaimage);
    vpcInMsg.rdma.luma_head_addr        = (long)headY;
    vpcInMsg.rdma.chroma_head_addr      = (long)headC;
    vpcInMsg.rdma.luma_head_stride      = vpcInMsg.stride;
    vpcInMsg.rdma.chroma_head_stride    = vpcInMsg.stride;
```

```
        vpcInMsg.rdma.luma_payload_stride   = gpYStride;
        vpcInMsg.rdma.chroma_payload_stride = gpUVStride;
        fclose(rdmaimage);
        printf("luma payload stride= %d\n", vpcInMsg.rdma.luma_payload_stride );
        printf("chroma payload stride= %d\n", vpcInMsg.rdma.chroma_payload_stride );
    } else {
        alloc_buffer(vpcInMsg.width, vpcInMsg.high, vpcInMsg.stride, bufferSize);
        if (open_image(in_file_name, vpcInMsg.width, vpcInMsg.high, vpcInMsg.stride) != 0) {
            printf("open file failed~\n");
            free(in_buffer);
            return;
        }
        vpcInMsg.in_buffer      = in_buffer;
        vpcInMsg.in_buffer_size = bufferSize;
    }
    //alloc in and out buffer
    shared_ptr<AutoBuffer> auto_out_buffer = make_shared<AutoBuffer>();
    vpcInMsg.auto_out_buffer_1              = auto_out_buffer;
    dvppApiCtlMsg.in                        = (void *)(&vpcInMsg);
    dvppApiCtlMsg.in_size                   = sizeof(vpc_in_msg);

    if (pidvppapi != NULL) {

        for (int i = 0; i < gLoop; i++) {
            if (DvppCtl(pidvppapi, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg) != 0) {
                printf("call dvppctl process faild!\n");
                DestroyDvppApi(pidvppapi);
                free(in_buffer);
                return;
            }
        }

    } else {
        printf("pidvppapi is null!\n");
    }

    free(in_buffer);

    FILE *fp = NULL;
    fp = fopen(out_file_name, "wb+");
    if (fp == NULL) {
        printf("open file %s failed~\n", out_file_name);
        free(in_buffer);
        return;
    }

    fwrite(vpcInMsg.auto_out_buffer_1->getBuffer(), 1, vpcInMsg.auto_out_buffer_1->getBufferSize(),
fp);
    fflush(fp);
    fclose(fp);
    DestroyDvppApi(pidvppapi);
    return;
```

- RawData8k基础功能调用示例

  参数配置：

```
    dvppapi_ctl_msg dvppApiCtlMsg;
    vpc_in_msg vpcInMsg;
    vpcInMsg.width = gWidth;
    vpcInMsg.high = gHigh;
    vpcInMsg.stride = gStride;
    vpcInMsg.out_width = gOutWidth;
    vpcInMsg.out_high = gOutHigh;
    printf("width      =%d\n", vpcInMsg.width);
    printf("high       =%d\n", vpcInMsg.high);
    printf("stride     =%d\n", vpcInMsg.stride);
    printf("out_width =%d\n", vpcInMsg.out_width);
    printf("out_high  =%d\n", vpcInMsg.out_high);
    //alloc in and out buffer
    char *in_buffer = (char *)malloc(vpcInMsg.width * vpcInMsg.high * 1.5);
```

```
    if (in_buffer = NULL) {
        printf("malloc fail\n");
        return;
    }

    shared_ptr<AutoBuffer> auto_out_buffer = make_shared<AutoBuffer>();
    FILE *yuvimage = fopen(in_file_name, "rb");
    if (yuvimage == NULL) {
        printf("no such file!,file_name=[%s]\n", in_file_name);
        free(in_buffer);
        return;
    } else {
        fread(in_buffer, 1, vpcInMsg.width * vpcInMsg.high * 3 / 2, yuvimage);
    }

    vpcInMsg.rank = gRank;
    vpcInMsg.in_buffer = in_buffer;
    vpcInMsg.in_buffer_size = vpcInMsg.width * vpcInMsg.high * 1.5;
    vpcInMsg.auto_out_buffer_1 = auto_out_buffer;
    vpcInMsg.use_flag = 1;
    dvppApiCtlMsg.in = (void *)(&vpcInMsg);
    dvppApiCtlMsg.in_size = sizeof(vpc_in_msg);
    IDVPPAPI *pidvppapi = NULL;
    CreateDvppApi(pidvppapi);
    if (pidvppapi != NULL) {
        if (DvppCtl(pidvppapi, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg) != 0) {
            printf("call dvppctl process faild!\n");
            DestroyDvppApi(pidvppapi);
            free(in_buffer);
            fclose(yuvimage);
            return;
        }
    } else {
        printf("pidvppapi is null!\n");
    }

    DestroyDvppApi(pidvppapi);
    FILE *fp = NULL;
    fp = fopen(out_file_name, "wb+");
    if (fp != NULL) {
        fwrite(vpcInMsg.auto_out_buffer_1->getBuffer(), 1, vpcInMsg.auto_out_buffer_1-
>getBufferSize(), fp);
        fflush(fp);
        fclose(fp);
    } else {
        fclose(yuvimage);
        return;
    }

    free(in_buffer);
    fclose(yuvimage);
    return;
```

- Rawdata叠加基础功能调用实例

```
    dvppapi_ctl_msg dvppApiCtlMsg;
    vpc_in_msg vpcInMsg;
    //配置RawData叠加参数
    vpcInMsg.use_flag = 2;
    vpcInMsg.overlay.image_type = 0;
    vpcInMsg.overlay.image_rank_type = 0;
    vpcInMsg.overlay.bit_width = 8;
    vpcInMsg.overlay.in_width_image = 1000;
    vpcInMsg.overlay.in_high_image = 1000;
    vpcInMsg.overlay.in_width_text = 500;
    vpcInMsg.overlay.in_high_text = 390;
    vpcInMsg.overlay.image_width_stride = 2;
    vpcInMsg.overlay.image_high_stride = 2;
    vpcInMsg.overlay.text_width_stride = 2;
    vpcInMsg.overlay.text_high_stride = 2;
    vpcInMsg.overlay.in_buffer_image = NULL;
```

```
    vpcInMsg.overlay.in_buffer_text = NULL;
    vpcInMsg.overlay.auto_overlay_out_buffer = make_shared<AutoBuffer>();
    //初始化图像内存
    if(NULL == (vpcInMsg.overlay.in_buffer_image =
(char*)malloc(vpcInMsg.overlay.in_width_image*vpcInMsg.overlay.in_high_image*3/2)))    {
        printf("in_buffer_image alloc failed!\n");
        return ;
    }
    //打开图片至申请的内存中
    char image_file[50] = "yuv_data_0";
    FILE * yuvimage = fopen(image_file,"rb");
    if(NULL == yuvimage)    {
        printf("VPC_TEST: yuv image open faild!");
        return ;
    }    else    {
        fread(vpcInMsg.overlay.in_buffer_image,
1,vpcInMsg.overlay.in_width_image*vpcInMsg.overlay.in_high_image*3/2,yuvimage);
        fclose(yuvimage);
        yuvimage = NULL;
}

    //初始化文字内存
    if(NULL == (vpcInMsg.overlay.in_buffer_text =
(char*)malloc(vpcInMsg.overlay.in_width_text*vpcInMsg.overlay.in_high_text*3/2)))    {
        printf("in_buffer_text alloc failed!");
        free(vpcInMsg.overlay.in_buffer_image);
        return ;
    }

    //打开text图片至申请的内存中
    char text_file[50] = "text_0";
    FILE * yuvtext = fopen(text_file,"rb");
    if(NULL == yuvtext)    {
        printf("VPC_TEST: yuv text image open faild!");
        free(vpcInMsg.overlay.in_buffer_image);
        return ;
    }    else    {
        fread(vpcInMsg.overlay.in_buffer_text,
1,vpcInMsg.overlay.in_width_text*vpcInMsg.overlay.in_high_text*3/2, yuvtext);
        fclose(yuvtext);
        yuvtext = NULL;
    }

    dvppApiCtlMsg.in = (void*)(&vpcInMsg);
    dvppApiCtlMsg.in_size = sizeof(vpc_in_msg);
    IDVPPAPI *pidvppapi = NULL;
    CreateDvppApi(pidvppapi);
    if(pidvppapi!=NULL)    {
        if(DvppCtl(pidvppapi,DVPP_CTL_VPC_PROC,&dvppApiCtlMsg)!= 0)        {
            printf("call dvppctl process faild!\n");
            DestroyDvppApi(pidvppapi);
            free(vpcInMsg.overlay.in_buffer_image);
            free(vpcInMsg.overlay.in_buffer_text);
            return ;
        }
        DestroyDvppApi(pidvppapi);
        FILE * fp   = fopen("./out_overlay_image_share","wb+");
        fwrite(vpcInMsg.overlay.auto_overlay_out_buffer->getBuffer(),
1,vpcInMsg.overlay.in_width_image*vpcInMsg.overlay.in_high_image*3/2,fp);
        fflush(fp);
        fclose(fp);
        fp=NULL;
        return ;
    }    else    {
        printf("pidvppapi is null!\n");
        return ;
    }
    if(NULL != vpcInMsg.overlay.in_buffer_image)    {
        free(vpcInMsg.overlay.in_buffer_image);
```

```
        vpcInMsg.overlay.in_buffer_image = NULL;
    }
    if(NULL != vpcInMsg.overlay.in_buffer_text)    {
        free(vpcInMsg.overlay.in_buffer_text);
        vpcInMsg.overlay.in_buffer_text = NULL;
    }
```

- Rawdata拼接基础功能调用实例

```
    dvppapi_ctl_msg dvppApiCtlMsg;
    vpc_in_msg vpcInMsg;
     //配置RawData拼接参数
    vpcInMsg.use_flag = 3;
    vpcInMsg.collage.image_type = 0;
    vpcInMsg.collage.image_rank_type = 0;
    vpcInMsg.collage.bit_width = 8;
    vpcInMsg.collage.in_width = 1000;
    vpcInMsg.collage.in_high = 1000;
    vpcInMsg.collage.width_stride = 2;
    vpcInMsg.collage.high_stride = 2;
    vpcInMsg.collage.collage_type = 0;
    vpcInMsg.collage.auto_out_buffer = make_shared<AutoBuffer>();
    char image_file[4][50] = {"yuv_data_0","yuv_data_1","yuv_data_2","yuv_data_3"};
    //初始化图像内存without stride************begin************
    for(int i=0; i<4; i++) {
        if(NULL == (vpcInMsg.collage.in_buffer[i] =
(char*)malloc(vpcInMsg.collage.in_width*                                    vpcInMsg.collage.in_high*3/2)))
        {
            printf("in_buffer_image alloc failed!\n");
            for(int j=0; j<i; j++) {
                free(vpcInMsg.collage.in_buffer[j]);
            }
            return ;
        }
        //打开图片至申请的内存中
        FILE * yuvimage = fopen(image_file[i],"rb");
        if(NULL == yuvimage)        {
            printf("PVC_TEST: yuv image open faild!");
            return ;
        }        else        {
            fread(vpcInMsg.collage.in_buffer[i], 1,vpcInMsg.collage.in_width*
                  vpcInMsg.collage.in_high*3/2,yuvimage);
            fclose(yuvimage);
            yuvimage = NULL;
        }
    }

    dvppApiCtlMsg.in = (void*)(&vpcInMsg);
    dvppApiCtlMsg.in_size = sizeof(vpc_in_msg);
    IDVPPAPI *pidvppapi = NULL;
    CreateDvppApi(pidvppapi);
    if(pidvppapi!=NULL)    {
        if(DvppCtl(pidvppapi,DVPP_CTL_VPC_PROC,&dvppApiCtlMsg)!= 0)        {
            printf("call dvppctl process faild!\n");
            DestroyDvppApi(pidvppapi);
            for(int i=0; i<4; i++) {
                free(vpcInMsg.collage.in_buffer[i]);
            }
            return ;
        }
        DestroyDvppApi(pidvppapi);
        FILE * fp   = fopen("./out_collage_image_share","wb+");
        fwrite(vpcInMsg.collage.auto_out_buffer->getBuffer(),
1,vpcInMsg.collage.in_width*vpcInMsg.collage.in_high*6,fp);
        fflush(fp);
        fclose(fp);
        fp=NULL;
        return ;
    }   else    {
        printf("pidvppapi is null!\n");
        return ;
```

```
    }
    for(int i=0; i<4; i++) {
        if(NULL != vpcInMsg.collage.in_buffer[i])        {
            free(vpcInMsg.collage.in_buffer[i]);
            vpcInMsg.collage.in_buffer[i] = NULL;
        }
    }
```

- VPCAPI调用

```
IDVPPAPI *pidvppapi = NULL;
CreateDvppApi(pidvppapi);
if(pidvppapi!=NULL)
{
 if(0 != DvppCtl(pidvppapi,DVPP_CTL_VPC_PROC,&dvppApiCtlMsg))
 {
  printf("call dvppctl process faild!\n");
  DestroyDvppApi(pidvppapi);
  return -1;
 }
}
else
printf("pidvppapi is null!\n");
DestroyDvppApi(pidvppapi);
```

- VPC Resize参数配置获取功能调用示例

  gXxxxx为配置参数

```
dvppapi_ctl_msg dvppApiCtlMsg;
vpc_in_msg vpcInMsg;
resize_param_in_msg resize_in_param;
resize_param_out_msg resize_out_param;
resize_in_param.src_width = gWidth;
resize_in_param.src_high = gHigh;
resize_in_param.hmax = gHmax;
resize_in_param.hmin = gHmin;
resize_in_param.vmax = gVmax;
resize_in_param.vmin = gVmin;
resize_in_param.dest_width = floor(gHinc * (gHmax - gHmin + 1) + 0.5);
resize_in_param.dest_high = floor(gVinc * (gVmax - gVmin + 1) + 0.5);
dvppApiCtlMsg.in = (void *)(&resize_in_param);
dvppApiCtlMsg.out = (void *)(&resize_out_param);
IDVPPAPI *pidvppapi = NULL;
CreateDvppApi(pidvppapi);
if (0 != DvppCtl(pidvppapi, DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM, &dvppApiCtlMsg))
{
  printf("call dvppctl process faild!\n");
  DestroyDvppApi(pidvppapi);
  return;
}
int bufferSize = 0;
vpcInMsg.format = gFormat;
vpcInMsg.cvdr_or_rdma = gcvdr_or_rdma;
vpcInMsg.bitwidth = gBitwidth;
vpcInMsg.rank = gRank;
vpcInMsg.width = gWidth;
vpcInMsg.high = gHigh;
vpcInMsg.stride = gStride;
vpcInMsg.hmax = resize_out_param.hmax;
vpcInMsg.hmin = resize_out_param.hmin;
vpcInMsg.vmax = resize_out_param.vmax;
vpcInMsg.vmin = resize_out_param.vmin;
vpcInMsg.vinc = resize_out_param.vinc;
vpcInMsg.hinc = resize_out_param.hinc;
```

5.3 CMDLIST 功能及参数说明

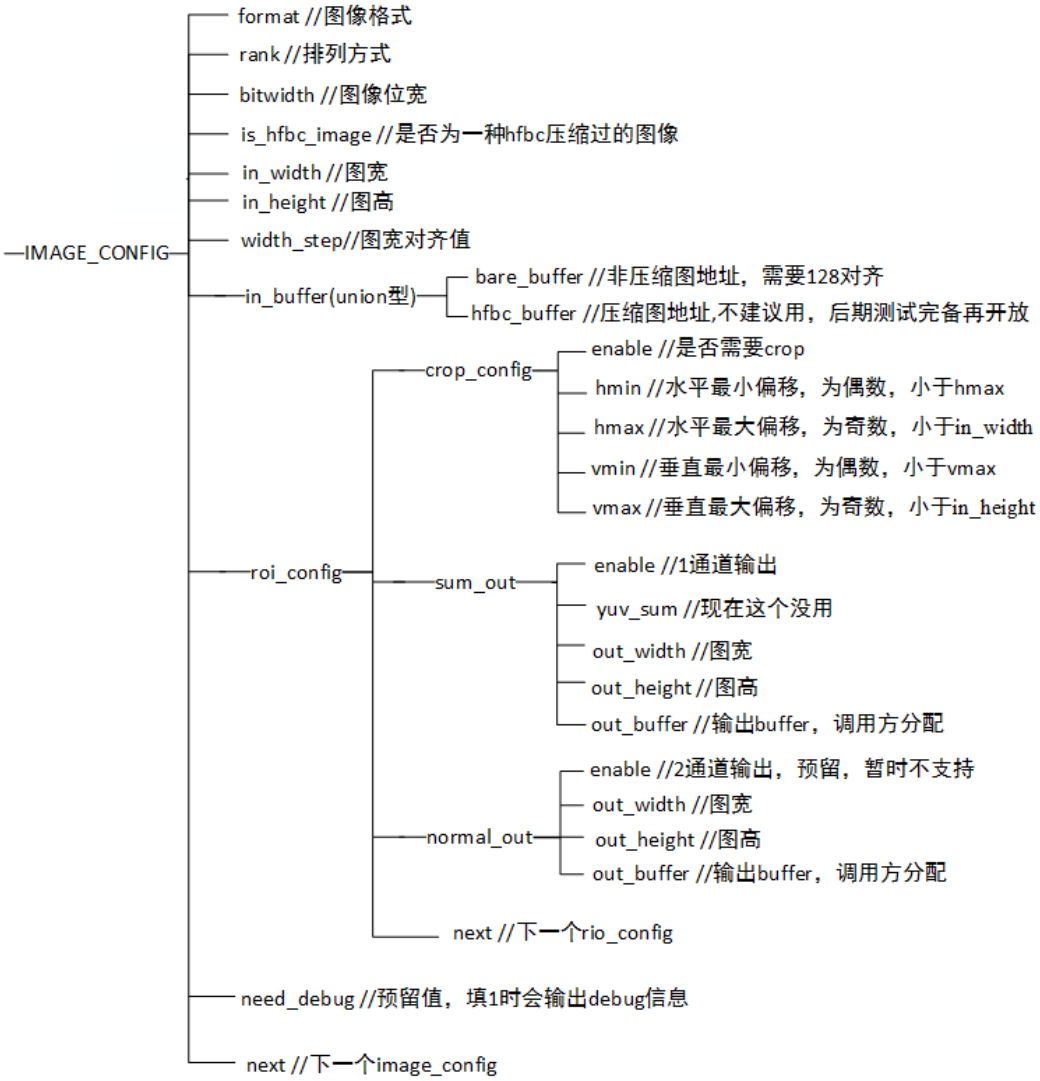
不推荐使用CMDLIST功能，后续版本会删除。旧版本中的VPC功能和CMDLIST功能均迁移到VPC功能，请参见2.2.2 VPC参数说明处的参数说明进行开发。

CMDLIST是VPC的一个扩展功能，其概念为将原来需要多次启动VPC执行的功能，合并到一次的“启动”与“完成中断返回”间完成。简单来说，CMDLIST适用于对时延要求不高，处理的图片分辨率低，数量多的场景。

入参

CMDLIST输入为结构体，详细参数如图5-1、表5-2所示。

图 5-1 CMDLIST 结构体



详细信息以及取值范围请参见表5-2。

表 5-2 结构体说明

成员变量	说明	取值范围
unsigned int: format	输入图像类型。	typedef enum { yuv420_semi_plannar =0,//0 yuv422_semi_plannar,//1 yuv444_semi_plannar,//2 yuv422_packed,//3 yuv444_packed,//4 rgb888_packed,//5 xrgb8888_packed,//6 yuv400_semi_plannar,//7 invalid_image_type,//20 }Imge_Type;
unsigned int: rank	输出图像格式(NV12或NV21)。	详见**表5-1**。
unsigned int: bitwidth	位深，通常为8bit。10bit只有在图像格式为YUV/YVU420 Semi-Planar且为HFBC压缩场景下使用。	8：8bit 10：10bit
int: is_hfbc_image	输入图像通道，通常设置为1，走CVDR通道，仅当VDEC输出的HFBC格式数据作为输入时走RDMA通道。	0：RDMA 1：CVDR
unsigned int: in_width	输入图像宽度，必须128对齐。	128~4096
unsigned int: in_height	输入图像高度，必须16对齐。	16~4096
unsigned int: width_step	图像步长。	yuv400sp、yuv420sp、yuv422sp、yuv444sp：width对齐到128。 yuv422packed：width * 2后对齐到128。 yuv444packed、rgb888：width * 3后对齐到128。 xrgb8888：width * 4后对齐到128。
unsigned int: need_debug	内部调试预留值，通常设置为0。	0：普通模式。 1：debug模式。

成员变量	说明	取值范围

CMDLIST_IN_BUFFER: in_buffer	输入图像内存地址指针，是一个联合体，普通非压缩图像选择bare_buffer，需要128对齐。HFBC压缩图像地址选择hfbc_buffer。	union CMDLIST_IN_BUFFER
		{
		char* bare_buffer;
		RDMACHANNEL* hfbc_buffer;
		};
		与vpc_in_msg中RDMACHANNEL相同，
		详见**Rdma通道结构体RDMACHANNEL**。

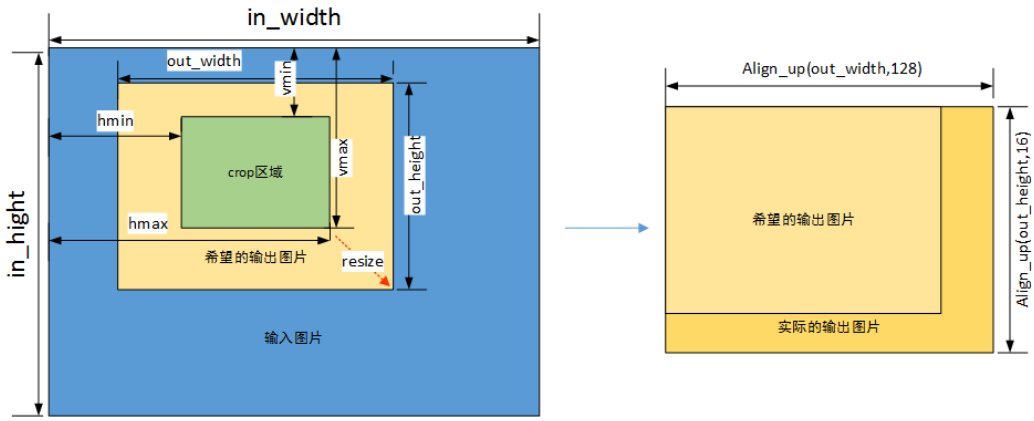
ROI_CONFIG: roi_config	输出参数配置结构体。	详见**ROI_CONFIG结构体**
IMAGE_CONFIG*: next	下一个IMAGE_CONFIG结构体指针。多图时配置，单图时配置为NULL。	--

说明

结构体详细定义请见DDK包下include/inc/dvpp/dvpp_config.h。

可以通过**图5-2**所示，直观了解输入参数信息。

图 5-2 输入参数示例说明



此图将crop和resize操作归一化到一个图中，理解如下。

- 只做crop时，相当于resize系数为1。
- 只做resize时，相当于crop大小为原图。

说明

- 输入图片的宽128对齐，高16对齐，且输入输出图片最大分辨率为4K。
- 内存地址对齐，需要用到如下接口。
  输入输出图片内存地址128对齐，需要用HIAI_DVPP_DMalloc的申请大页内存。
  申请接口：HIAI_DVPP_DMalloc(size)。
- 根据输出图片比上crop后图片计算缩放系数，需要在[0.03125,4]范围内。
- 输出内存大小需要根据输出宽128对齐，高16对齐后的分辨率计算。
- 调用一次cmdlist接口，最多支持32张图，每图最多支持256个框。

出参

暂无，其中输出图片在sum_out中的out_buffer中，用户可读取该内存获取图片。

输入图片格式与输出图片格式对应rank配置表请参见表5-2。

调用示例

```
/*****************************************************************************************/
/*****用例说明：*****************************************************************************
此用例以一张1920x1088 yuv420 nv12图片作为输入
输入图片文件名： "file1_1920x1088_nv12.yuv"
对输入图片的操作：抠出5张子图，均缩放到224x224；
 ****************************************************************************************
 *****************************************************************************************/
int main()
{
    int ret = 0;
    //读取输入文件
    char image_file[128] = "file1_1920x1088_nv12.yuv";
    ifstream in_stream(image_file);
    if (!in_stream.is_open()) {
        printf("can not open %s.\n", image_file);
        return -1;
    }
    in_stream.seekg(0, ios::end);
    int file_len = in_stream.tellg();
    char* in_buffer = (char *)HIAI_DVPP_DMalloc(file_len);
    in_stream.seekg(0, ios::beg);
    in_stream.read(in_buffer, file_len);
    in_stream.close();
    //开始添加第一个image的配置
    IMAGE_CONFIG* image_config = (IMAGE_CONFIG*)malloc(sizeof(IMAGE_CONFIG));
    image_config->in_buffer.bare_buffer = in_buffer; //目前用到的都是非压缩图
    image_config->format = 0;
    image_config->rank = 1;//输入nv12输出也为nv12，rank填1
    image_config->bitwidth = 8;//目前用到的都是8bit一个单位的
    image_config->in_width = 1920;
    image_config->in_height = 1088;
    image_config->width_step = 1920;//这个值同宽
    //开始添加第一个抠图的配置,该例子抠图区域:(0,0),(511,511)围成的区域
    ROI_CONFIG* roi_config = &image_config->roi_config;
    //开始配置crop参数
    roi_config->crop_config.enable = 1; //注：当只要resize的时候，crop参数只要这个配置为0
    roi_config->crop_config.hmin = 0;
    roi_config->crop_config.hmax = 511;
    roi_config->crop_config.vmin = 0;
    roi_config->crop_config.vmax = 511;
    //开始配置输出通道参数
    roi_config->sum_out.enable = 1; //开启第一通道输出
    roi_config->sum_out.out_width = 224;
    roi_config->sum_out.out_height = 224;
    int out_buffer_size = AlignUp(224,128)*AlignUp(224,16)*3/2;
```

```
    roi_config->sum_out.out_buffer = (char*)HIAI_DVPP_DMalloc(out_buffer_size);
    ROI_CONFIG* last_roi = roi_config;
    //开始添加第2到第5个抠图的配置
    for (int i = 0 ; i < 4; i++) {
        ROI_CONFIG* roi_config = (ROI_CONFIG*)malloc(sizeof(ROI_CONFIG));
        //开始配置crop参数
        roi_config->crop_config.enable = 1;
        roi_config->crop_config.hmin = 100*i;
        roi_config->crop_config.hmax = 299 + 100*i;
        roi_config->crop_config.vmin = 100*i;
        roi_config->crop_config.vmax = 299 + 100*i;
        //开始配置输出通道参数
        roi_config->sum_out.enable = 1;
        roi_config->sum_out.out_width = 224;
        roi_config->sum_out.out_height = 224;
        out_buffer_size = AlignUp(224,128)*AlignUp(224,16)*3/2;
        roi_config->sum_out.out_buffer = (char*)HIAI_DVPP_DMalloc(out_buffer_size);
        roi_config->next = nullptr;
        last_roi->next = roi_config;
        last_roi = roi_config;
    }
    //开始调用dvpp的cmdlist接口
    IDVPPAPI *pidvppapi = NULL;
    //无论后面调用多少次，createdvppapi只要一次调用就好
    ret = CreateDvppApi(pidvppapi);
    if (ret != 0) {
        printf("creat dvpp api faild!\n");
        return -1;
    }
    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = (void *)(image_config);
    dvppApiCtlMsg.in_size = sizeof(IMAGE_CONFIG);
    ret = DvppCtl(pidvppapi, DVPP_CTL_CMDLIST_PROC, &dvppApiCtlMsg);
    if (0 != ret) {
        printf("call cmdlist dvppctl process faild!\n");
    } else {
        printf("cmdlist success.\n");
    }
    ret += DestroyDvppApi(pidvppapi);
    roi_config = image_config->roi_config.next;
    while (roi_config != nullptr) {
        ROI_CONFIG* next_roi = roi_config->next;
        UNMAP(roi_config->sum_out.out_buffer, out_buffer_size);
        free(roi_config);
        roi_config = next_roi;
    }
    UNMAP(image_config->roi_config.sum_out.out_buffer, out_buffer_size);
    free(image_config);
    UNMAP(in_buffer, file_len);
    return ret;
}
```

5.4 VENC 功能及参数说明

不推荐使用旧版本中的VENC功能，后续版本会删除。旧版本中的VENC功能已迁移到VENC新接口，请参见4 VENC功能接口进行开发。

功能

实现YUV420/YVU420图片数据的编码。

- VENC支持以下格式输入：
  YUV420 semi-planner NV12/NV21-8bit
- VENC输出格式为：

H264 BP/MP/HP
H265 MP

VENC 性能指标

场景	总帧率
1080p * 1路（不支持多路）	30fps

入参：venc_in_msg

成员变量	说明	取值范围
Int width	图像宽度。	128~1920，且为偶数。
Int height	图像高度。	128~1920，且为偶数。
Int coding_type	视频编码协议H265-main level（0）、H264-baseline level（1）、H264-main level(2)、H264-high level（3）	0~3 ● 0：H265 main level。 ● 1：H264 baseline level。 ● 2：H264 main level。 ● 3：H264high level。
Int YUV_store_type	YUV图像存储格式。	0或者1 ● 0：YUV420 semi-planner ● 1：YVU420 semi-planner
char* input_data	输入图像数据地址。	不能为NULL。
Int input_data_size	输入图像数据大小。	正数。
shared_ptr<AutoBuffer> output_data_queue	输出编码码流数据地址。	需要配置此buffer，此buffer为调用方申请智能指针当做参数传入DVPP。用户可读取该buffer获取输出图片。 必填

出参

暂无。

调用示例

```
void TEST_3() //venc demo
{
    int read_file_size;
```

```
    int unit_file_size;
    FILE *fp = fopen(in_file_name, "rb");
    if (fp == NULL) {
        printf("open file: %s failed.\n", in_file_name);
        return;
    }
    printf("open yuv success \n");
    fseek(fp, 0L, SEEK_END);
    int file_size = ftell(fp);
    fseek(fp, 0L, SEEK_SET);

    venc_in_msg venc_msg;
    venc_msg.width = gWidth;
    venc_msg.height = gHigh;
    venc_msg.coding_type = gFormat;
    venc_msg.YUV_store_type = gBitwidth;
    venc_msg.output_data_queue = make_shared<AutoBuffer>();
    unit_file_size = gWidth * gHigh * 3 / 2 * MAX_FRAME_NUM_VENC; //单次文件大小为16帧
    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = (void *)(&venc_msg);
    dvppApiCtlMsg.in_size = sizeof(venc_in_msg);
    IDVPPAPI *pidvppapi = NULL;
    CreateDvppApi(pidvppapi);
    char out_filename[] = "venc.bin";
    FILE *outputBufferFile;
    outputBufferFile = fopen (out_filename, "wb+");

    do{
        read_file_size = file_size>unit_file_size? unit_file_size:file_size; //每次读取文件大小不能超过16帧
        venc_msg.input_data = (char *)malloc(read_file_size);
        int read_len = fread(venc_msg.input_data, 1, read_file_size, fp);
        printf("file size is %d,read len is %d.\n", read_file_size, read_len);
        venc_msg.input_data_size = read_len;

        if (pidvppapi != NULL) {
            if (DvppCtl(pidvppapi, DVPP_CTL_VENC_PROC, &dvppApiCtlMsg) != 0) {
                printf("call dvppctl process faild!\n");
                DestroyDvppApi(pidvppapi);
                fclose(fp);
                fclose(outputBufferFile);
                return;
            }
            if (venc_msg.output_data_queue->getBufferSize() > 100) { //防止编码结果只含有头部信息
                char* out_buf = venc_msg.output_data_queue->getBuffer();
                int out_buf_size = venc_msg.output_data_queue->getBufferSize();
                int write_size = fwrite(out_buf, 1, out_buf_size, outputBufferFile);
                fflush(outputBufferFile);
            } else {
                printf("venc output data is too small : %d \n", venc_msg.output_data_queue->getBufferSize());
            }

        } else {
            printf("pidvppapi is null!\n");
        }

        if (venc_msg.input_data != NULL) {
            free(venc_msg.input_data);
            venc_msg.input_data = NULL;
        }

        file_size = file_size - unit_file_size;
    } while (file_size > 0);

    DestroyDvppApi(pidvppapi);
    fclose(outputBufferFile);
    fclose(fp);
    return;
}
```

6 调用示例

6.1 实现 VPC 功能

概念说明

关于抠图、缩放、叠加、上偏移、下偏移、左偏移、右偏移等概念，请参见**1.3 VPC功能**。

示例 1：仅原图缩放

关键参数设置如下：

- 抠图区域宽高跟输入图片的真实宽高相同，抠图区域各偏移值的设置如下：
  - 左偏移leftOffset=0
  - 上偏移upOffset=0
  - 右偏移rightOffset - 左偏移leftOffset+1=输入图片真实宽
  - 下偏移downOffset - 上偏移upOffset+1=输入图片真实高
- 贴图区域宽高是缩放后图片的宽高，可指定贴图区域的位置，例如：
  若指定贴图区域的位置在输出图片的左上角，则贴图区域各偏移值的设置如下：
  - 左偏移leftOffset=0
  - 上偏移upOffset=0
  - 右偏移rightOffset - 左偏移leftOffset+1=缩放后图片的宽
  - 下偏移downOffset - 上偏移upOffset+1=缩放后图片的高

- 如果是8K的原图缩放，则inputFormat和outputFormat只支持依次设置为INPUT_YUV420_SEMI_PLANNER_UV、OUTPUT_YUV420SP_UV或设置为INPUT_YUV420_SEMI_PLANNER_VU、OUTPUT_YUV420SP_VU；如果是非8K的原图缩放，inputFormat和outputFormat的设置可参考**表2-1**。

示例代码：

本示例是将yuv420sp格式的图片从1080p缩放到720p。

```
void NewVpcTest1()
{
    uint32_t inWidthStride = 1920;
    uint32_t inHeightStride = 1080;
    uint32_t outWidthStride = 1280;
    uint32_t outHeightStride = 720;
    uint32_t inBufferSize = inWidthStride * inHeightStride * 3 / 2; // 1080P yuv420sp Image
    uint32_t outBufferSize = outWidthStride * outHeightStride * 3 / 2; // 720P yuv420sp bImage
    uint8_t* inBuffer  = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(inBufferSize)); // Construct an input picture.
    if (inBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc input buffer");
        return;
    }
    uint8_t* outBuffer = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(outBufferSize)); // Construct an output
picture.
    if (outBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc output buffer");
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        return;
    }

    FILE* fp = fopen("dvpp_vpc_1920x1080_nv12.yuv", "rb+");
    if (fp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "fopen file failed");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }

    fread(inBuffer, 1, inBufferSize, fp);
    fclose(fp);
    fp = nullptr;
    // Construct the input picture configuration.
    std::shared_ptr<VpcUserImageConfigure> imageConfigure(new VpcUserImageConfigure);
    imageConfigure->bareDataAddr = inBuffer;
    imageConfigure->bareDataBufferSize = inBufferSize;
    imageConfigure->widthStride = inWidthStride;
    imageConfigure->heightStride = inHeightStride;
    imageConfigure->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV;
    imageConfigure->outputFormat = OUTPUT_YUV420SP_UV;
    imageConfigure->yuvSumEnable = false;
    imageConfigure->cmdListBufferAddr = nullptr;
    imageConfigure->cmdListBufferSize = 0;
    std::shared_ptr<VpcUserRoiConfigure> roiConfigure(new VpcUserRoiConfigure);
    roiConfigure->next = nullptr;
    VpcUserRoiInputConfigure* inputConfigure = &roiConfigure->inputConfigure;
    // Set the drawing area, [0,0] in the upper left corner of the area,
    // and [1919,1079] in the lower right corner.
    inputConfigure->cropArea.leftOffset  = 0;
    inputConfigure->cropArea.rightOffset = inWidthStride - 1;
    inputConfigure->cropArea.upOffset    = 0;
    inputConfigure->cropArea.downOffset  = inHeightStride - 1;
    VpcUserRoiOutputConfigure* outputConfigure = &roiConfigure->outputConfigure;
    outputConfigure->addr = outBuffer;
    outputConfigure->bufferSize = outBufferSize;
    outputConfigure->widthStride = outWidthStride;
    outputConfigure->heightStride = outHeightStride;
    // Set the map area, coordinate [0,0] in the upper left corner of the map area,
    // and [1279,719] in the lower right corner.
    outputConfigure->outputArea.leftOffset  = 0;
```

```
    outputConfigure->outputArea.rightOffset = outWidthStride - 1;
    outputConfigure->outputArea.upOffset    = 0;
    outputConfigure->outputArea.downOffset  = outHeightStride - 1;

    imageConfigure->roiConfigure = roiConfigure.get();

    IDVPPAPI *pidvppapi = nullptr;
    int32_t ret = CreateDvppApi(pidvppapi);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "create dvpp api fail.");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = static_cast<void*>(imageConfigure.get());
    dvppApiCtlMsg.in_size = sizeof(VpcUserImageConfigure);
    ret = DvppCtl(pidvppapi, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "call vpc dvppctl process faild!");
        ret = DestroyDvppApi(pidvppapi);
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    } else {
        HIAI_ENGINE_LOG(HIAI_OK, "NewVpcTest1::call vpc dvppctl process success!");
    }

    FILE* outImageFp = fopen("NewVpcTest1Out.yuv", "wb+");
    if (outImageFp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "open NewVpcTest1Out.yuv faild!");
        ret = DestroyDvppApi(pidvppapi);
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }

    fwrite(outBuffer, 1, outBufferSize, outImageFp);

    ret = DestroyDvppApi(pidvppapi);
    DFreeInOutBuffer(inBuffer, outBuffer);
    fclose(outImageFp);
    outImageFp = nullptr;
    return;
}
```

示例 2：抠一张图+缩放

关键参数设置如下：

- 抠图区域宽高是缩放前图片的宽高，可指定抠图区域的位置，例如：
  若指定抠图区域的位置在输入图片的中间，则抠图区域各偏移值的设置如下：
  - 左偏移leftOffset=100
  - 右偏移rightOffset=499
  - 上偏移upOffset=100
  - 下偏移downOffset=399
  - 右偏移rightOffset - 左偏移leftOffset+1=抠图区域图片的宽
  - 下偏移downOffset - 上偏移upOffset+1=抠图区域图片的高
- 贴图区域宽高是缩放后图片的宽高，可指定贴图区域的位置，例如：
  若指定贴图区域的位置在输出图片的中间，则贴图区域各偏移值的设置如下：
  - 左偏移leftOffset=256
  - 右偏移rightOffset=399

- 上偏移upOffset=200
- 下偏移downOffset=399
- 右偏移rightOffset - 左偏移leftOffset+1=缩放后图片的宽
- 下偏移downOffset - 上偏移upOffset+1=缩放后图片的高

示例代码：

本示例是从yuv420sp格式、1080p的图片中抠出一张图，经过缩放后，将缩放后的图片贴到720p的输出图片（由用户申请的空输出内存产生的空白图片）中。如果需要将已有图片作为输出图片，用户需在申请输出内存后，将已有图片读入输出内存，代码示例请参考**示例5：叠加**。

```
void NewVpcTest2()
{
    uint32_t inWidthStride = 1920;
    uint32_t inHeightStride = 1080;
    uint32_t outWidthStride = 1280;
    uint32_t outHeightStride = 720;
    uint32_t inBufferSize = inWidthStride * inHeightStride * 3 / 2; // 1080P yuv420sp Image
    uint32_t outBufferSize = outWidthStride * outHeightStride * 3 / 2; // 720P yuv420sp Image
    uint8_t* inBuffer  = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(inBufferSize)); //  Construct an input
picture.
    if (inBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc input buffer");
        return;
    }
    uint8_t* outBuffer = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(outBufferSize)); //  Construct an output
picture.
    if (outBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc output buffer");
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        return;
    }
    (void)memset_s(outBuffer, outBufferSize, 0x80, outBufferSize);

    FILE* fp = fopen("dvpp_vpc_1920x1080_nv12.yuv", "rb+");
    if (fp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "fopen file failed");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }

    fread(inBuffer, 1, inBufferSize, fp);
    fclose(fp);
    fp = nullptr;
    // Construct the input picture configuration
    std::shared_ptr<VpcUserImageConfigure> imageConfigure(new VpcUserImageConfigure);
    imageConfigure->bareDataAddr = inBuffer;
    imageConfigure->bareDataBufferSize = inBufferSize;
    imageConfigure->widthStride = inWidthStride;
    imageConfigure->heightStride = inHeightStride;
    imageConfigure->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV;
    imageConfigure->outputFormat = OUTPUT_YUV420SP_UV;
    imageConfigure->yuvSumEnable = false;
    imageConfigure->cmdListBufferAddr = nullptr;
    imageConfigure->cmdListBufferSize = 0;
    std::shared_ptr<VpcUserRoiConfigure> roiConfigure(new VpcUserRoiConfigure);
    roiConfigure->next = nullptr;
    VpcUserRoiInputConfigure* inputConfigure = &roiConfigure->inputConfigure;
    // Set the drawing area, [100,100] in the upper left corner of the area, and [499,499] in the lower right
corner.
    inputConfigure->cropArea.leftOffset  = 100;
    inputConfigure->cropArea.rightOffset = 499;
    inputConfigure->cropArea.upOffset    = 100;
    inputConfigure->cropArea.downOffset  = 499;
```

```
    VpcUserRoiOutputConfigure* outputConfigure = &roiConfigure->outputConfigure;
    outputConfigure->addr = outBuffer;
    outputConfigure->bufferSize = outBufferSize;
    outputConfigure->widthStride = outWidthStride;
    outputConfigure->heightStride = outHeightStride;
    // Set the map area, [256,200] in the upper left corner of the map area, and [399,399] in the lower right
corner.
    outputConfigure->outputArea.leftOffset  = 256; // The offset value must be 16-pixel-aligned.
    outputConfigure->outputArea.rightOffset = 399;
    outputConfigure->outputArea.upOffset    = 200;
    outputConfigure->outputArea.downOffset  = 399;

    imageConfigure->roiConfigure = roiConfigure.get();

    IDVPPAPI *pidvppapi = nullptr;
    int32_t ret = CreateDvppApi(pidvppapi);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "create dvpp api fail.");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = static_cast<void*>(imageConfigure.get());
    dvppApiCtlMsg.in_size = sizeof(VpcUserImageConfigure);
    ret = DvppCtl(pidvppapi, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "call vpc dvppctl process faild!");
        ret = DestroyDvppApi(pidvppapi);
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    } else {
        HIAI_ENGINE_LOG(HIAI_OK, "NewVpcTest2::call vpc dvppctl process success!");
    }

    FILE* outImageFp = fopen("NewVpcTest2Out.yuv", "wb+");
    if (outImageFp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "open NewVpcTest2Out.yuv failed");
        DestroyDvppApi(pidvppapi);
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    fwrite(outBuffer, 1, outBufferSize, outImageFp);

    ret = DestroyDvppApi(pidvppapi);
    DFreeInOutBuffer(inBuffer, outBuffer);
    fclose(outImageFp);
    outImageFp = nullptr;
    return;
}
```

示例 3：抠多张图+缩放+拼接

关键参数设置：抠图区域上偏移、下偏移、左偏移、右偏移以及贴图区域上偏移、下偏移、左偏移、右偏移的配置，请参见**示例2：抠一张图+缩放**中的说明。

示例代码：本示例是从yuv420sp格式、1080p的图片中抠出多张图，经过缩放后，将缩放后的多张图片拼接到720p的输出图片（由用户申请的空输出内存产生的空白图片）中，贴图区域拼接的位置由上偏移、下偏移、左偏移、右偏移的值决定。如果需要将已有图片作为输出图片，用户需在申请输出内存后，将已有图片读入输出内存，代码示例请参考**示例5：叠加**。

```
void NewVpcTest3()
{
    uint32_t inWidthStride = 1920;
    uint32_t inHeightStride = 1080;
    uint32_t outWidthStride = 1280;
    uint32_t outHeightStride = 720;
```

```
    uint32_t inBufferSize = inWidthStride * inHeightStride * 3 / 2; // 1080P yuv420sp Image
    uint32_t outBufferSize = outWidthStride * outHeightStride * 3 / 2; // 720P yuv420sp Image
    uint8_t* inBuffer = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(inBufferSize)); // Construct an input picture.
    if (inBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc input buffer");
        return;
    }

    FILE* fp = fopen("dvpp_vpc_1920x1080_nv12.yuv", "rb+");
    if (fp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "fopen file failed");
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        return;
    }

    fread(inBuffer, 1, inBufferSize, fp);
    fclose(fp);
    fp == nullptr;
    // Construct the input picture configuration.
    std::shared_ptr<VpcUserImageConfigure> imageConfigure(new VpcUserImageConfigure);
    imageConfigure->bareDataAddr = inBuffer;
    imageConfigure->bareDataBufferSize = inBufferSize;
    imageConfigure->widthStride = inWidthStride;
    imageConfigure->heightStride = inHeightStride;
    imageConfigure->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV;
    imageConfigure->outputFormat = OUTPUT_YUV420SP_UV;
    imageConfigure->yuvSumEnable = false;
    imageConfigure->cmdListBufferAddr = nullptr;
    imageConfigure->cmdListBufferSize = 0;
    std::shared_ptr<VpcUserRoiConfigure> lastRoi;
    std::vector<std::shared_ptr<VpcUserRoiConfigure>> roiVector;
    for (uint32_t i = 0; i < 5; i++) {
        std::shared_ptr<VpcUserRoiConfigure> roiConfigure(new VpcUserRoiConfigure);
        roiVector.push_back(roiConfigure);
        roiConfigure->next = nullptr;
        VpcUserRoiInputConfigure* inputConfigure = &roiConfigure->inputConfigure;
        // Set the drawing area.
        inputConfigure->cropArea.leftOffset  = 100 + i * 16;
        inputConfigure->cropArea.rightOffset = 499 + i * 16;
        inputConfigure->cropArea.upOffset    = 100 + i * 16;
        inputConfigure->cropArea.downOffset  = 499 + i * 16;
        VpcUserRoiOutputConfigure* outputConfigure = &roiConfigure->outputConfigure;
        uint8_t* outBuffer = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(outBufferSize)); // Construct an input
picture
        if (outBuffer == nullptr) {
            HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc output buffer");
            HIAI_DVPP_DFree(inBuffer);
            inBuffer = nullptr;
            FreeMultipleRio(imageConfigure->roiConfigure);
            return;
        }
        (void)memset_s(outBuffer, outBufferSize, 0x80, outBufferSize);
        outputConfigure->addr = outBuffer;
        outputConfigure->bufferSize = outBufferSize;
        outputConfigure->widthStride = outWidthStride;
        outputConfigure->heightStride = outHeightStride;
        // Set the map area.
        outputConfigure->outputArea.leftOffset  = 256 + i * 16; // The offset value must be 16-pixel-aligned.
        outputConfigure->outputArea.rightOffset = 399 + i * 16;
        outputConfigure->outputArea.upOffset    = 256 + i * 16;
        outputConfigure->outputArea.downOffset  = 399 + i * 16;
        if (i == 0) {
            imageConfigure->roiConfigure = roiConfigure.get();
            lastRoi = roiConfigure;
        } else {
            lastRoi->next = roiConfigure.get();
            lastRoi = roiConfigure;
        }
```

```
    }

    IDVPPAPI *pidvppapi = nullptr;
    int32_t ret = CreateDvppApi(pidvppapi);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "create dvpp api fail.");
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        FreeMultipleRio(imageConfigure->roiConfigure);
        return;
    }
    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = static_cast<void*>(imageConfigure.get());
    dvppApiCtlMsg.in_size = sizeof(VpcUserImageConfigure);
    ret = DvppCtl(pidvppapi, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "call vpc dvppctl process faild!");
        ret = DestroyDvppApi(pidvppapi);
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        FreeMultipleRio(imageConfigure->roiConfigure);
        return;
    } else {
        HIAI_ENGINE_LOG(HIAI_OK, "NewVpcTest3::call vpc dvppctl process success!");
    }
    FILE* outImageFp = nullptr;
    uint32_t imageCount = 0;
    char fileName[50] = {0};

    while (imageConfigure->roiConfigure != nullptr) {
        int32_t safeFuncRet = sprintf_s(fileName, sizeof(fileName), "NewVpcTest3_%dOut.yuv", imageCount);
        if (safeFuncRet == -1) {
            HIAI_ENGINE_LOG(HIAI_ERROR, "sprintf_s fail, ret = %d", safeFuncRet);
            DestroyDvppApi(pidvppapi);
            HIAI_DVPP_DFree(inBuffer);
            inBuffer = nullptr;
            FreeMultipleRio(imageConfigure->roiConfigure);
            return;
        }
        outImageFp = fopen(fileName, "wb+");
        if (outImageFp == nullptr) {
            HIAI_ENGINE_LOG(HIAI_ERROR, "open %s faild!", fileName);
            DestroyDvppApi(pidvppapi);
            HIAI_DVPP_DFree(inBuffer);
            inBuffer = nullptr;
            FreeMultipleRio(imageConfigure->roiConfigure);
            return;
        }
        fwrite(imageConfigure->roiConfigure->outputConfigure.addr, 1,
            imageConfigure->roiConfigure->outputConfigure.bufferSize, outImageFp);
        fclose(outImageFp);
        outImageFp = nullptr;
        imageConfigure->roiConfigure = imageConfigure->roiConfigure->next;
        imageCount++;
    }
    ret = DestroyDvppApi(pidvppapi);
    HIAI_DVPP_DFree(inBuffer);
    inBuffer = nullptr;
    FreeMultipleRio(imageConfigure->roiConfigure);
    return;
}
```

示例 4，8K 缩放功能

对于**8K缩放**功能，支持缩放、支持YUV420SP NV12与YUV420SP NV21之间的格式转换、不支持抠图。

示例代码：本示例是将yuv420sp格式的图片从8129*8192缩放至4000*4000。

```
void NewVpcTest4()
{
    uint32_t inWidthStride = 8192; // No need for 128 byte alignment
    uint32_t inHeightStride = 8192; // No need for 16 byte alignment
    uint32_t outWidthStride = 4000; // No need for 128 byte alignment
    uint32_t outHeightStride = 4000; // No need for 16 byte alignment
    uint32_t inBufferSize = inWidthStride * inHeightStride * 3 / 2;
    uint32_t outBufferSize = outWidthStride * outHeightStride * 3 / 2; // Construct dummy data
    uint8_t* inBuffer = (uint8_t*)HIAI_DVPP_DMalloc(inBufferSize); // Construct input image
    if (inBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc roiOutput buffer");
        return;
    }
    uint8_t* outBuffer = (uint8_t*)HIAI_DVPP_DMalloc(outBufferSize); // Construct output image
    if (outBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc roiOutput buffer");
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        return;
    }

    FILE* fp = fopen("dvpp_vpc_8192x8192_nv12.yuv", "rb+");
    if (fp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "fopen file failed");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    fread(inBuffer, 1, inBufferSize, fp);
    fclose(fp);
    fp = nullptr;
    std::shared_ptr<VpcUserImageConfigure> imageConfigure(new VpcUserImageConfigure);
    imageConfigure->bareDataAddr = inBuffer;
    imageConfigure->bareDataBufferSize = inBufferSize;
    imageConfigure->isCompressData = false;
    imageConfigure->widthStride = inWidthStride;
    imageConfigure->heightStride = inHeightStride;
    imageConfigure->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV;
    imageConfigure->outputFormat = OUTPUT_YUV420SP_UV;
    imageConfigure->yuvSumEnable = false;
    imageConfigure->cmdListBufferAddr = nullptr;
    imageConfigure->cmdListBufferSize = 0;
    std::shared_ptr<VpcUserRoiConfigure> roiConfigure(new VpcUserRoiConfigure);
    roiConfigure->next = nullptr;
    VpcUserRoiInputConfigure* inputConfigure = &roiConfigure->inputConfigure; // Set the roi area
    inputConfigure->cropArea.leftOffset  = 0;
    inputConfigure->cropArea.rightOffset = inWidthStride - 1;
    inputConfigure->cropArea.upOffset = 0;
    inputConfigure->cropArea.downOffset  = inHeightStride - 1;
    VpcUserRoiOutputConfigure* outputConfigure = &roiConfigure->outputConfigure;
    outputConfigure->addr = outBuffer;
    outputConfigure->bufferSize = outBufferSize;
    outputConfigure->widthStride = outWidthStride;
    outputConfigure->heightStride = outHeightStride; // Set the map area
    outputConfigure->outputArea.leftOffset  = 0;
    outputConfigure->outputArea.rightOffset = outWidthStride - 1;
    outputConfigure->outputArea.upOffset = 0;
    outputConfigure->outputArea.downOffset  = outHeightStride - 1;
    imageConfigure->roiConfigure = roiConfigure.get();
    IDVPPAPI *pidvppapi = nullptr;
    int32_t ret = CreateDvppApi(pidvppapi);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "create dvpp api fail.");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = static_cast<void*>(imageConfigure.get());
    dvppApiCtlMsg.in_size = sizeof(VpcUserImageConfigure);
    ret = DvppCtl(pidvppapi, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg);
```

```
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "call vpc dvppctl process faild!");
        ret = DestroyDvppApi(pidvppapi);
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    } else {
        HIAI_ENGINE_LOG(HIAI_OK, "NewVpcTest4::call vpc dvppctl process success!");
    }

    FILE* outImageFp = fopen("NewVpcTest4Out.yuv", "wb+");
    if (outImageFp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "open NewVpcTest4Out.yuv faild!");
        ret = DestroyDvppApi(pidvppapi);
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    fwrite(outBuffer, 1, outBufferSize, outImageFp);
    fclose(outImageFp);
    outImageFp = nullptr;
    ret = DestroyDvppApi(pidvppapi);
    DFreeInOutBuffer(inBuffer, outBuffer);
    return;
}
```

示例 5：叠加

如果用户需要将已有图片读入内存用于存放输出图片，将贴图区域叠加在输出图片上，则需要编写代码逻辑将图片读入内存，在申请输出内存代码**uint8_t* outBuffer = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(outBufferSize));**之后，增加如下代码：

```
FILE* fpOut = fopen("vpcOut.yuv", "rb+");
if  (fpOut == nullptr) {
    HIAI_ENGINE_LOG(HIAI_ERROR, "fopen file  failed.");
    fclose(fpOut);
    return;
}
fread(outBuffer, 1,  outBufferSize, fpOut);
fclose(fpOut);
```

6.2 实现 JPEGE 功能

- 使用**DvppGetOutParameter**接口获取内存大小，由用户指定输出内存，由用户自行释放内存，调用示例如下 。

```
void TEST_JPEGE_CUSTOM_MEMORY()
{
    sJpegeIn  inData;
    sJpegeOut outData;

    inData.width         = g_width;
    inData.height        = g_high;
    inData.heightAligned = g_high; // no need to align
    inData.format        = (eEncodeFormat)g_format;
    inData.level         = 100;

    inData.stride  = ALIGN_UP(inData.width * 2, 16);
    inData.bufSize = inData.stride * inData.heightAligned;
    if (JPGENC_FORMAT_YUV420 == (inData.format & JPGENC_FORMAT_BIT)) {
        inData.stride  = ALIGN_UP(inData.width, 16);
        inData.bufSize = inData.stride * inData.heightAligned * 3 / 2;
    }

    void* addrOrig = HIAI_DVPP_DMalloc(inData.bufSize);
    if (addrOrig == nullptr) {
        HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "can not alloc input buffer");
        return;
```

```
    }
    inData.buf = reinterpret_cast<unsigned char*>(addrOrig);

    unsigned char* tmpAddr = nullptr;

    do {
        // load img file
        FILE* fpIn = fopen(g_inFileName, "rb");
        if (nullptr == fpIn) {
            HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "can not open input file");
            break;
        }
        // only copy valid image data, other part is pending
        if (JPGENC_FORMAT_YUV420 == (inData.format & JPGENC_FORMAT_BIT)) {
            // for yuv420semi-planar format, like nv12 / nv21
            HIAI_ENGINE_LOG("input yuv 420 data");
            // for y data
            for (uint32_t j = 0; j < inData.height; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width, fpIn);
            }
            // for uv data
            for (uint32_t j = inData.heightAligned; j < inData.heightAligned + inData.height / 2; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width, fpIn);
            }
        } else {
            // for yuv422packed format, like uyvy / vyuy / yuyv / yvyu
            HIAI_ENGINE_LOG("input yuv 422 data");
            for (uint32_t j = 0; j < inData.height; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width * 2, fpIn);
            }
        }
        fclose(fpIn);
        fpIn = nullptr;

        int32_t ret = DvppGetOutParameter((void*)(&inData), (void*)(&outData),
GET_JPEGE_OUT_PARAMETER);
        if (ret != 0) {
            HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "call DvppGetOutParameter process failed");
            break;
        }
        // 此处获得的jpgSize为估算值，实际数据长度可能要小于这个值
        // 最终jpgSize大小，以调用DvppCtl接口以后，此字段才为真正的编码后的jpgSize大小
        HIAI_ENGINE_LOG("outdata size is %d", outData.jpgSize);
        tmpAddr = (unsigned char*)HIAI_DVPP_DMalloc(outData.jpgSize);
        if (tmpAddr == nullptr) {
            HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "can not alloc output buffer");
            break;
        }
        uint32_t size = outData.jpgSize;

        // call jpege process
        dvppapi_ctl_msg dvppApiCtlMsg;
        dvppApiCtlMsg.in = (void *)&inData;
        dvppApiCtlMsg.in_size = sizeof(inData);
        dvppApiCtlMsg.out = (void *)&outData;
        dvppApiCtlMsg.out_size = sizeof(outData);

        if(!IsHandleJpegeCtlSucc(dvppApiCtlMsg, tmpAddr, size, &outData)) {
            break;
        }
    } while (0); // for resource inData.buf
    if (addrOrig != nullptr) {
        HIAI_DVPP_DFree(addrOrig);
        addrOrig = nullptr;
    }
    if (tmpAddr != nullptr) { // 注意：释放内存时需要使用tmpAddr，因为outData.jpgData在JPEGE处理后
会有地址偏移
        HIAI_DVPP_DFree(tmpAddr);
        tmpAddr = nullptr;
```

```
    }
}
```

- 不由用户指定输出内存时，DVPP内部申请内存，需由用户调用cbFree()回调函数释放内存，调用示例如下。

```
void JpegeProcess()
{
    sJpegeIn  inData;
    sJpegeOut outData;

    inData.width         = g_width;
    inData.height        = g_high;
    inData.heightAligned = g_high; // no need to align
    inData.format        = (eEncodeFormat)g_format;
    inData.level         = 100;

    if (JPGENC_FORMAT_YUV420 == (inData.format & JPGENC_FORMAT_BIT)) {
        inData.stride  = ALIGN_UP(inData.width, 16);
        inData.bufSize = inData.stride * inData.heightAligned * 3 / 2;
    } else {
        inData.stride  = ALIGN_UP(inData.width * 2, 16);
        inData.bufSize = inData.stride * inData.heightAligned;
    }

    void* addrOrig = HIAI_DVPP_DMalloc(inData.bufSize);

    if (addrOrig == nullptr) {
        HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "can not alloc input buffer");
        return;
    }
    inData.buf = reinterpret_cast<unsigned char*>(addrOrig);

    do {
        // load img file
        FILE* fpIn = fopen(g_inFileName, "rb");
        if (fpIn == nullptr) {
            HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "can not open input file");
            break;
        }
        // only copy valid image data, other part is pending
        if (JPGENC_FORMAT_YUV420 == (inData.format & JPGENC_FORMAT_BIT)) {
            // for yuv420semi-planar format, like nv12 / nv21
            HIAI_ENGINE_LOG("input yuv 420 data");
            // for y data
            for (uint32_t j = 0; j < inData.height; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width, fpIn);
            }
            // for uv data
            for (uint32_t j = inData.heightAligned; j < inData.heightAligned + inData.height / 2; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width, fpIn);
            }
        } else {
            // for yuv422packed format, like uyvy / vyuy / yuyv / yvyu
            HIAI_ENGINE_LOG("input yuv 422 data");
            for (uint32_t j = 0; j < inData.height; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width * 2, fpIn );
            }

        }
        fclose(fpIn);
        fpIn = nullptr;

        // call jpege process
        dvppapi_ctl_msg dvppApiCtlMsg;
        dvppApiCtlMsg.in = (void*)&inData;
        dvppApiCtlMsg.in_size = sizeof(inData);
        dvppApiCtlMsg.out = (void*)&outData;
        dvppApiCtlMsg.out_size = sizeof(outData);

        IDVPPAPI *pidvppapi = nullptr;
```

```
        CreateDvppApi(pidvppapi);
        if (pidvppapi == nullptr) {
            HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "can not open dvppapi engine");
            break;
        }

        JpegeProcessBranch(pidvppapi, dvppApiCtlMsg, outData);

        DestroyDvppApi(pidvppapi);

    } while (0); // for resource inData.buf
    if (addrOrig != nullptr) {
        HIAI_DVPP_DFree(addrOrig);
        addrOrig = nullptr;
    }
}

/*
 * only handle jpege process.
 */
void JpegeProcessBranch(IDVPPAPI*& pidvppapi, dvppapi_ctl_msg& dvppApiCtlMsg, sJpegeOut&
outData)
{
    do {
        for (int i = 0;i < g_loop; i++) { // same picture loop test
            if (DvppCtl(pidvppapi, DVPP_CTL_JPEGE_PROC, &dvppApiCtlMsg)) {
                HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "call jpeg encoder fail");
                break;
            }
            if (i < g_loop - 1) {
                outData.cbFree();
                outData.jpgData = nullptr;
            }
        }
        stringstream outFile;
        outFile << g_outFileName << "_t" << std::this_thread::get_id() << ".jpg";

        FILE* fpOut = fopen(outFile.str().c_str(), "wb");
        if (fpOut != nullptr) {
            fwrite(outData.jpgData, 1, outData.jpgSize, fpOut);
            fflush(fpOut);
            fclose(fpOut);
            fpOut = nullptr;
        } else {
            HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "call not save result file %s ", outFile.str().c_str());
        }

        outData.cbFree();
        outData.jpgData = nullptr;
        HIAI_ENGINE_LOG(HIAI_OK, "jpeg encode process completed");
    } while (0); // for resource pddvppapi
}
```

6.3 实现 JPEGD 功能

- 使用**DvppGetOutParameter**接口获取内存大小，由用户指定输出内存，由用户自行释放内存，调用示例如下 。

```
void TEST_JPEGD_CUSTOM_MEMORY()
{
    struct JpegdIn jpegdInData;
    struct JpegdOut jpegdOutData;

    if (g_rank) {
        jpegdInData.isYUV420Need = false;
    }

    FILE *fpIn = fopen(g_inFileName, "rb");
```

```
    if (fpIn == nullptr) {
        HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "can not open file %s.", g_inFileName);
        return;
    }

    do { // for resource fpIn
        fseek(fpIn, 0, SEEK_END);
        // the buf len should 8 byte larger, the driver asked
        uint32_t fileLen   = ftell(fpIn);
        jpegdInData.jpegDataSize = fileLen + 8;
        fseek(fpIn, 0, SEEK_SET);

        void* addrOrig = HIAI_DVPP_DMalloc(jpegdInData.jpegDataSize);
        if (addrOrig == nullptr) {
            HIAI_ENGINE_LOG(HIAI_JPEGD_CTL_ERROR, "can not alloc input buffer");
            fclose(fpIn);
            fpIn = nullptr;
            break;
        }

        jpegdInData.jpegData = reinterpret_cast<unsigned char*>(addrOrig);
        do { // for resource inBuf
            fread(jpegdInData.jpegData, 1, fileLen, fpIn);
            fclose(fpIn);
            fpIn = nullptr;
            int32_t ret = DvppGetOutParameter((void*)(&jpegdInData), (void*)(&jpegdOutData),
GET_JPEGD_OUT_PARAMETER);
            if (ret != 0) {
                HIAI_ENGINE_LOG(HIAI_JPEGD_CTL_ERROR, "call DvppGetOutParameter process failed");
                break;
            }
            HIAI_ENGINE_LOG("jpegd out size is %d", jpegdOutData.yuvDataSize);
            jpegdOutData.yuvData = (unsigned char*)HIAI_DVPP_DMalloc(jpegdOutData.yuvDataSize);
            if (jpegdOutData.yuvData == nullptr) {
                HIAI_ENGINE_LOG(HIAI_JPEGD_CTL_ERROR, "can not alloc output buffer");
                break;
            }
            HIAI_ENGINE_LOG("jpegdOutData.yuvData is %p", jpegdOutData.yuvData);

            dvppapi_ctl_msg dvppApiCtlMsg;
            dvppApiCtlMsg.in = (void *)&jpegdInData;
            dvppApiCtlMsg.in_size = sizeof(jpegdInData);
            dvppApiCtlMsg.out = (void *)&jpegdOutData;
            dvppApiCtlMsg.out_size = sizeof(jpegdOutData);

            IDVPPAPI *pidvppapi = nullptr;
            CreateDvppApi(pidvppapi);

            if (pidvppapi != nullptr) {
                for (int i = 0;i < g_loop; i++) {
                    if (0 != DvppCtl(pidvppapi, DVPP_CTL_JPEGD_PROC, &dvppApiCtlMsg)) {
                        HIAI_ENGINE_LOG(HIAI_JPEGD_CTL_ERROR, "call dvppctl process failed");
                        break;
                    }
                }
                DestroyDvppApi(pidvppapi);
            } else {
                HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "can not create dvpp api");
                break;
            }
            WriteTestJpegdResultToFile(&jpegdOutData);
        } while (0); // for resource inBuf

        if (addrOrig != nullptr) {
            HIAI_DVPP_DFree(addrOrig);
            addrOrig = nullptr;
        }
        if (jpegdOutData.yuvData != nullptr) {
            HIAI_DVPP_DFree(jpegdOutData.yuvData);
```

```
            jpegdOutData.yuvData = nullptr;
        }
    } while (0); // for resource fpIn
}
```

- 不由用户指定输出内存时，DVPP内部申请内存，需由用户调用cbFree()回调函数释放内存，调用示例如下。

```
void JpegdProcess()
{
    struct jpegd_raw_data_info jpegdInData;
    struct jpegd_yuv_data_info jpegdOutData;

    if (g_rank) {
        jpegdInData.IsYUV420Need = false;
    }

    FILE *fpIn = fopen(g_inFileName, "rb");
    if (fpIn == nullptr) {
        HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "can not open file %s.", g_inFileName);
        return;
    }

    do { // for resource fpIn
        fseek(fpIn, 0, SEEK_END);
        // the buf len should 8 byte larger, the driver asked
        uint32_t fileLen   = ftell(fpIn);
        jpegdInData.jpeg_data_size = fileLen + 8;
        fseek(fpIn, 0, SEEK_SET);

        void* addrOrig = HIAI_DVPP_DMalloc(jpegdInData.jpeg_data_size);
        if (addrOrig == nullptr) {
            HIAI_ENGINE_LOG(HIAI_JPEGD_CTL_ERROR, "can not alloc input buffer");
            break;
        }

        jpegdInData.jpeg_data = reinterpret_cast<unsigned char*>(addrOrig);

        do { // for resource inBuf
            fread(jpegdInData.jpeg_data, 1, fileLen, fpIn);
            dvppapi_ctl_msg dvppApiCtlMsg;
            dvppApiCtlMsg.in = (void*)&jpegdInData;
            dvppApiCtlMsg.in_size = sizeof(jpegdInData);
            dvppApiCtlMsg.out = (void*)&jpegdOutData;
            dvppApiCtlMsg.out_size = sizeof(jpegdOutData);

            IDVPPAPI *pidvppapi = nullptr;
            CreateDvppApi(pidvppapi);

            if (pidvppapi != nullptr) {
                for (int i = 0;i < g_loop; i++) { // same picture loop test
                    if (DvppCtl(pidvppapi, DVPP_CTL_JPEGD_PROC, &dvppApiCtlMsg) != 0) {
                        HIAI_ENGINE_LOG(HIAI_JPEGD_CTL_ERROR, "call dvppctl process failed");
                        break;
                    }
                    if (i < g_loop - 1) {
                        jpegdOutData.cbFree();
                        jpegdOutData.yuv_data = nullptr;
                    }
                }
                DestroyDvppApi(pidvppapi);
            } else {
                HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "can not create dvpp api");
                break;
            }

            WriteJpegdProcessResultToFile(&jpegdOutData);
            jpegdOutData.cbFree();
            jpegdOutData.yuv_data = nullptr;

        } while (0); // for resource inBuf
```

```
        if (addrOrig != nullptr) {
            HIAI_DVPP_DFree(addrOrig);
            addrOrig = nullptr;
        }

    } while (0); // for resource fpIn
    fclose(fpIn);
    fpIn = nullptr;
}
```

6.4 实现 PNGD 功能

使用**DvppGetOutParameter**接口获取内存大小，由用户指定输出内存，由用户自行释放内存，调用示例如下 。

```
void TEST_PNGD_CUSTOM_MEMORY()
{
    FILE* fpIn = fopen(g_inFileName, "rb");
    if (nullptr == fpIn) {
        HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "can not open file %s.", g_inFileName);
        return;
    }

    fseek(fpIn, 0, SEEK_END);
    uint32_t fileLen = ftell(fpIn);
    fseek(fpIn, 0, SEEK_SET);
    void* inbuf = HIAI_DVPP_DMalloc(fileLen);
    if (inbuf == nullptr) {
        HIAI_ENGINE_LOG(HIAI_PNGD_CTL_ERROR, "can not alloc input buffer");
        fclose(fpIn);
        fpIn = nullptr;
        return;
    }

    // prepare msg
    struct PngInputInfoAPI inputPngData; // input data
    inputPngData.inputData = inbuf; // input png data
    inputPngData.inputSize = fileLen; // the size of png data

    HIAI_ENGINE_LOG("inputPngData.address: %p", inputPngData.address);
    HIAI_ENGINE_LOG("inputPngData.size: %u", inputPngData.size);
    HIAI_ENGINE_LOG("inputPngData.inputData: %p", inputPngData.inputData);
    HIAI_ENGINE_LOG("inputPngData.inputSize: %u", inputPngData.inputSize);

    fread(inputPngData.inputData, 1, fileLen, fpIn);
    fclose(fpIn);
    fpIn = nullptr;

    if (g_transform == 1) { // Whether format conversion
        inputPngData.transformFlag = 1; // RGBA -> RGB
    } else {
        inputPngData.transformFlag = 0;
    }

    struct PngOutputInfoAPI outputPngData; // output data
    int32_t ret = DvppGetOutParameter((void*)(&inputPngData), (void*)(&outputPngData),
GET_PNGD_OUT_PARAMETER);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_PNGD_CTL_ERROR, "call DvppGetOutParameter process failed");
        HIAI_DVPP_DFree(inbuf);
        inbuf = nullptr;
        return;
    }
    HIAI_ENGINE_LOG("pngd out size is  %d", outputPngData.size);
    outputPngData.address = HIAI_DVPP_DMalloc(outputPngData.size);
    if (outputPngData.address == nullptr) {
```

```
        HIAI_ENGINE_LOG(HIAI_PNGD_CTL_ERROR, "can not alloc output buffer");
        HIAI_DVPP_DFree(inbuf);
        inbuf = nullptr;
        return;
    }

    dvppapi_ctl_msg dvppApiCtlMsg; // call the interface msg
    dvppApiCtlMsg.in = (void*)(&inputPngData);
    dvppApiCtlMsg.in_size = sizeof(struct PngInputInfoAPI);
    dvppApiCtlMsg.out = (void*)(&outputPngData);
    dvppApiCtlMsg.out_size = sizeof(struct PngOutputInfoAPI);

    // use interface
    IDVPPAPI *pidvppapi = nullptr;
    CreateDvppApi(pidvppapi); // create dvppapi

    if (pidvppapi != nullptr) { // use DvppCtl interface to handle DVPP_CTL_PNGD_PROC
        for (int i = 0; i < g_loop; i++) {
            if (-1 == DvppCtl(pidvppapi, DVPP_CTL_PNGD_PROC, &dvppApiCtlMsg)) {
                HIAI_ENGINE_LOG(HIAI_PNGD_CTL_ERROR, "call dvppctl process failed");
                HIAI_DVPP_DFree(inbuf);
                inbuf = nullptr;
                HIAI_DVPP_DFree(outputPngData.address);
                outputPngData.address = nullptr;
                DestroyDvppApi(pidvppapi); // destory dvppapi
                return;
            }
        }
    } else {
        HIAI_ENGINE_LOG(HIAI_PNGD_CTL_ERROR, "can not get dvpp api");
    }

    DestroyDvppApi(pidvppapi);

    HIAI_ENGINE_LOG("output result");

    char* pAddr = (char*)outputPngData.outputData;
    FILE* fpOut = fopen(g_outFileName, "wb");
    if (fpOut == nullptr) {
        HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "can not open file %s.", g_outFileName);
        HIAI_DVPP_DFree(inbuf);
        inbuf = nullptr;
        HIAI_DVPP_DFree(outputPngData.address);
        outputPngData.address = nullptr;
        return;
    }

    int size = outputPngData.width * 4;
    if (outputPngData.format == PNGD_RGB_FORMAT_NUM) {
        size = outputPngData.width * 3;
    } else if (outputPngData.format == PNGD_RGBA_FORMAT_NUM) {
        size = outputPngData.width * 4;
    }
    // copy valid image data from every line.
    for (int i = 0; i < outputPngData.high; i++) {
        fwrite(pAddr + (int)(i * outputPngData.widthAlign), size, 1, fpOut);
    }

    fclose(fpOut);
    fpOut = nullptr;
    HIAI_DVPP_DFree(inbuf);
    inbuf = nullptr;
    HIAI_DVPP_DFree(outputPngData.address);
    outputPngData.address = nullptr;
    return;
}
```

6.5 实现 VDEC 功能

对于一个视频码流，调用一次**CreateVdecApi**接口创建实例后，必须使用同一个实例调用**VdecCtl**接口进行视频解码，最后再调用一次**DestroyVdecApi**接口释放实例。

在视频解码时，如果需要从一个视频码流切换到另一个视频码流，则需要先调用**DestroyVdecApi**接口释放前一个码流的实例，再调用**CreateVdecApi**接口创建新的实例，用于处理新的视频码流。

此调用示例是读取文件名为“test_file”的h264码流文件，调用VDEC功能，将解码结果存放到“output_dir”目录。

```
// 自定义子类
class HIAI_DATA_SP_SON: public HIAI_DATA_SP {
public:
    ~ HIAI_DATA_SP_SON ()
    {
        //destruct here;
    }
    // 用户自定义新增一些成员函数，以用户需求为准，下述新增成员函数仅为示例
    uint8_t GetTotalFrameNum()
    {
        return info_.totalFrameNum;
    }

private:
    // 用户自定义一些成员变量，以用户需求为准，下述新增成员变量结构体info_仅为示例
    struct INFO {
        uint8_t totalFrameNum;
        uint8_t frameRate;
    }info_;
};

IDVPPAPI * pidvppapi_vpc = NULL;
IDVPPAPI * pidvppapi_vdec = NULL;

//回调函数举例，调用方需根据自己的需求来重新定义回调函数
void FrameReturn(FRAME* frame, void* hiaiData)
{
    static int32_t imageCount = 0;
    imageCount++;
    HIAI_ENGINE_LOG("call save frame number:[%d], width:[%d], height:[%d]", imageCount, frame->width,
frame->height);
    // The image directly output by vdec is an image of hfbc compression format, which cannot be directly
displayed.
    // It is necessary to call vpc to convert hfbc to an image of uncompressed format to display.
    HIAI_ENGINE_LOG("start call vpc interface to translate hfbc.");
    IDVPPAPI* dvppHandle = nullptr;
    int32_t ret = CreateDvppApi(dvppHandle);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "creat dvpp api fail.");
        return;
    }

    // Construct vpc input configuration.
    std::shared_ptr<VpcUserImageConfigure> userImage(new VpcUserImageConfigure);
    // bareDataAddr should be null which the image is hfbc.
    userImage->bareDataAddr = nullptr;
    userImage->bareDataBufferSize = 0;
    userImage->widthStride = frame->width;
    userImage->heightStride = frame->height;
    // Configuration input format
    string imageFormat(frame->image_format);
    if (frame->bitdepth == 8) {
        if (imageFormat == "nv12") {
```

```
        userImage->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV;
    } else {
        userImage->inputFormat = INPUT_YUV420_SEMI_PLANNER_VU;
    }
} else {
    if (imageFormat == "nv12") {
        userImage->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV_10BIT;
    } else {
        userImage->inputFormat = INPUT_YUV420_SEMI_PLANNER_VU_10BIT;
    }
}
userImage->outputFormat = OUTPUT_YUV420SP_UV;
userImage->isCompressData = true;
// Configure hfbc input address
VpcCompressDataConfigure* compressDataConfigure = &userImage->compressDataConfigure;
uint64_t baseAddr = (uint64_t)frame->buffer;
compressDataConfigure->lumaHeadAddr = baseAddr + frame->offset_head_y;
compressDataConfigure->chromaHeadAddr = baseAddr + frame->offset_head_c;
compressDataConfigure->lumaPayloadAddr = baseAddr + frame->offset_payload_y;
compressDataConfigure->chromaPayloadAddr = baseAddr + frame->offset_payload_c;
compressDataConfigure->lumaHeadStride = frame->stride_head;
compressDataConfigure->chromaHeadStride = frame->stride_head;
compressDataConfigure->lumaPayloadStride = frame->stride_payload;
compressDataConfigure->chromaPayloadStride = frame->stride_payload;

userImage->yuvSumEnable = false;
userImage->cmdListBufferAddr = nullptr;
userImage->cmdListBufferSize = 0;
// Configure the roi area and output area
std::shared_ptr<VpcUserRoiConfigure> roiConfigure(new VpcUserRoiConfigure);
roiConfigure->next = nullptr;
userImage->roiConfigure = roiConfigure.get();
VpcUserRoiInputConfigure* roiInput = &roiConfigure->inputConfigure;
roiInput->cropArea.leftOffset = 0;
roiInput->cropArea.rightOffset = frame->width - 1;
roiInput->cropArea.upOffset = 0;
roiInput->cropArea.downOffset = frame->height - 1;
VpcUserRoiOutputConfigure* roiOutput = &roiConfigure->outputConfigure;
roiOutput->outputArea.leftOffset = 0;
roiOutput->outputArea.rightOffset = frame->width - 1;
roiOutput->outputArea.upOffset = 0;
roiOutput->outputArea.downOffset = frame->height - 1;
roiOutput->bufferSize = ALIGN_UP(frame->width, 16) * ALIGN_UP(frame->height, 2) * 3 / 2;
roiOutput->addr = (uint8_t*)HIAI_DVPP_DMalloc(roiOutput->bufferSize);
if (roiOutput->addr == nullptr) {
    HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc roiOutput buffer");
    DestroyDvppApi(dvppHandle);
    return;
}
roiOutput->widthStride = ALIGN_UP(frame->width, 16);
roiOutput->heightStride = ALIGN_UP(frame->height, 2);

dvppapi_ctl_msg dvppApiCtlMsg;
dvppApiCtlMsg.in = (void*)(userImage.get());
dvppApiCtlMsg.in_size = sizeof(VpcUserImageConfigure);
ret = DvppCtl(dvppHandle, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg);
if (ret != 0) {
    HIAI_ENGINE_LOG(HIAI_ERROR, "call dvppctl fail");
    HIAI_DVPP_DFree(roiOutput->addr);
    roiOutput->addr = nullptr;
    DestroyDvppApi(dvppHandle);
    return;
}
ret = FrameReturnSaveVpcResult(frame, roiOutput->addr, imageCount);
if (ret != 0) {
    HIAI_ENGINE_LOG(HIAI_ERROR, "save vpc result fail");
}
HIAI_DVPP_DFree(roiOutput->addr);
roiOutput->addr = nullptr;
```

```
    DestroyDvppApi(dvppHandle);
    return;
}

/*
*错误上报回调函数，用户若不需要错误信息，可不定义此函数
*/
void ErrReport(VDECERR* vdecErr)
{
    if (vdecErr != nullptr) {
        HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "vdec error code is %d, channelId = %u\n", vdecErr-
>errType, vdecErr->channelId);
    }
}

//测试函数主入口
void TEST_VDEC()
{
    char outputDir[20] = {0};
    int32_t safeFuncRet = memset_s(outputDir, sizeof(outputDir), 0, 20);
    if (safeFuncRet != EOK) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "memset_s fail");
        return;
    }
    safeFuncRet = strncpy_s(outputDir, sizeof(outputDir), "output_dir", strlen("output_dir"));
    if (safeFuncRet != EOK) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "strncpy_s fail");
        return;
    }
    if (access(outputDir, F_OK) == -1) {
        if (mkdir(outputDir, 0700) < 0) { // directory authority: 0700
            HIAI_ENGINE_LOG(HIAI_VDEC_CTL_ERROR, "create dir failed.");
            return;
        }
    } else if (access(outputDir, R_OK | W_OK | X_OK) == -1) {
        HIAI_ENGINE_LOG(HIAI_VDEC_CTL_ERROR, "output directory authority is not correct, use 700
instead.");
        return;
    }
    IDVPPAPI *pidvppapi = nullptr;
    int32_t ret = CreateVdecApi(pidvppapi, 0);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "create dvpp api fail.");
        return;
    }
    if (pidvppapi != nullptr) {
        FILE* fp = fopen(g_inFileName, "rb");
        if (fp == nullptr) {
            HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "open file: %s failed.", g_inFileName);
            DestroyVdecApi(pidvppapi, 0);
            return;
        }
        fseek(fp, 0L, SEEK_END);
        int file_size = ftell(fp);
        fseek(fp, 0L, SEEK_SET);
        int rest_len = file_size;
        int len = file_size;

        vdec_in_msg vdec_msg;
        vdec_msg.call_back = FrameReturn;
        vdec_msg.err_report = ErrReport; // can be unassigned, if user does not need error info
        vdec_msg.hiai_data = nullptr;
        int32_t safeFuncRet = 0;
        if (g_format == 0) {
            safeFuncRet = memcpy_s(vdec_msg.video_format, sizeof(vdec_msg.video_format), "h264", 4);
            if (safeFuncRet != EOK) {
                HIAI_ENGINE_LOG(HIAI_ERROR, "memcpy_s fail");
                DestroyVdecApi(pidvppapi, 0);
                fclose(fp);
```

```
            fp = nullptr;
            return;
        }
    } else {
        safeFuncRet = memcpy_s(vdec_msg.video_format, sizeof(vdec_msg.video_format), "h265", 4);
        if (safeFuncRet != EOK) {
            HIAI_ENGINE_LOG(HIAI_ERROR, "memcpy_s fail");
            DestroyVdecApi(pidvppapi, 0);
            fclose(fp);
            fp = nullptr;
            return;
        }
    }
    vdec_msg.in_buffer = (char*)malloc(len);
    if (vdec_msg.in_buffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "malloc fail");
        fclose(fp);
        fp = nullptr;
        DestroyVdecApi(pidvppapi, 0);
        return;
    }

    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = (void*)(&vdec_msg);
    dvppApiCtlMsg.in_size = sizeof(vdec_in_msg);

    while (rest_len > 0) {
        int read_len = fread(vdec_msg.in_buffer, 1, len, fp);
        HIAI_ENGINE_LOG("rest_len is %d,read len is %d.", rest_len, read_len);
        vdec_msg.in_buffer_size = read_len;
        fclose(fp);
        fp = nullptr;
        if (VdecCtl(pidvppapi, DVPP_CTL_VDEC_PROC, &dvppApiCtlMsg, 0) != 0) {
            HIAI_ENGINE_LOG(HIAI_VDEC_CTL_ERROR, "call dvppctl process faild!");
            free(vdec_msg.in_buffer);
            vdec_msg.in_buffer = nullptr;
            DestroyVdecApi(pidvppapi, 0);
            return;
        }
        HIAI_ENGINE_LOG(HIAI_OK, "call vdec process success!");
        rest_len = rest_len - len;
    }
    free(vdec_msg.in_buffer);
    vdec_msg.in_buffer = nullptr;
  } else {
    HIAI_ENGINE_LOG(HIAI_VDEC_CTL_ERROR, "pidvppapi is null!");
  }
  DestroyVdecApi(pidvppapi, 0);
  return;
}
```

6.6 实现 VENC 功能

若需要将多张图片编码成一个视频，则调用一次**CreateVenc**接口创建实例后，必须使用同一个实例调用**RunVenc**接口进行视频编码，最后再调用一次**DestroyVenc**接口释放实例。

本调用示例是调用VENC接口将yuv图片编码成h265或者h264格式的码流。

```
std::string vencOutFileName("venc.bin");
std::shared_ptr<FILE> vencOutFile(nullptr);
void VencCallBackDumpFile(struct VencOutMsg* vencOutMsg, void* userData)
{
  if (vencOutFile.get() == nullptr) {
    HIAI_ENGINE_LOG(HIAI_VENC_CTL_ERROR, "get venc out file fail!");
    return;
  }
```

```
    fwrite(vencOutMsg->outputData, 1, vencOutMsg->outputDataSize, vencOutFile.get());
    fflush(vencOutFile.get());
}

/*
 * venc new interface to achieve venc basic functions.
 */
void TEST_VENC()
{
    std::shared_ptr<FILE> fpIn(fopen(g_inFileName, "rb"), fclose);
    vencOutFile.reset(fopen(vencOutFileName.c_str(), "wb"), fclose);

    if (fpIn.get() == nullptr || vencOutFile.get() == nullptr) {
        HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "open open venc in/out file failed.");
        return;
    }

    fseek(fpIn.get(), 0, SEEK_END);
    uint32_t fileLen = ftell(fpIn.get());
    fseek(fpIn.get(), 0, SEEK_SET);

    struct VencConfig vencConfig;
    vencConfig.width = g_width;
    vencConfig.height = g_high;
    vencConfig.codingType = g_format;
    vencConfig.yuvStoreType = g_yuvStoreType;
    vencConfig.keyFrameInterval = 16;
    vencConfig.vencOutMsgCallBack = VencCallBackDumpFile;
    vencConfig.userData = nullptr;

    int32_t vencHandle = CreateVenc(&vencConfig);
    if (vencHandle == -1) {
        HIAI_ENGINE_LOG(HIAI_VENC_CTL_ERROR, "CreateVenc fail!");
        return;
    }

    // input 16 frames once
    uint32_t inDataLenMaxOnce = g_width * g_high * 3 / 2;
    std::shared_ptr<char> inBuffer(static_cast<char*>(malloc(inDataLenMaxOnce)), free);

    if (inBuffer.get() == nullptr) {
        HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "alloc input buffer failed");
        DestroyVenc(vencHandle);
        return;
    }

    uint32_t inDataUnhandledLen = fileLen;

    auto start = std::chrono::system_clock::now();
    uint32_t frameCount = 0;

    while (inDataUnhandledLen > 0) {
        uint32_t inDataLen      = inDataUnhandledLen;
        if (inDataUnhandledLen > inDataLenMaxOnce) {
            inDataLen           = inDataLenMaxOnce;
        }
        inDataUnhandledLen    -= inDataLen;
        uint32_t readLen        = fread(inBuffer.get(), 1, inDataLen, fpIn.get());
        if (readLen != inDataLen) {
            HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "error in read input file");
            DestroyVenc(vencHandle);
            return;
        }

        struct VencInMsg vencInMsg;
        vencInMsg.inputData = inBuffer.get();
        vencInMsg.inputDataSize = inDataLen;
        vencInMsg.keyFrameInterval = 16;
        vencInMsg.forceIFrame = 0;
```

```
        vencInMsg.eos = 0;

        if (RunVenc(vencHandle, &vencInMsg) == -1) {
            HIAI_ENGINE_LOG(HIAI_VENC_CTL_ERROR, "call video encode fail");
            break;
        }
        ++frameCount;
    }

    auto end = std::chrono::system_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end - start);
    size_t timeCount = static_cast<size_t>(duration.count());
    HIAI_ENGINE_LOG("Total frame count: %u", frameCount);
    HIAI_ENGINE_LOG("Time cost: %lu us", timeCount);
    HIAI_ENGINE_LOG("FPS: %lf", frameCount * 1.0 / (timeCount / 1000000.0));

    DestroyVenc(vencHandle);
    return;
}
```

7 数据类型

7.1 VpcUserImageConfigure 中的结构体

该结构体在DDK安装目录下的“ddk/include/inc/dvpp/Vpc.h”文件中定义。

- VpcUserRoiConfigure结构体

成员变量	说明
VpcUserRoiInputConfigure inputConfigure	用户ROI输入配置，详细见**•VpcUserRoiInputConfigur...**。若实现8K缩放功能，不用配置该参数。
VpcUserRoiOutputConfigure outputConfigure	用户ROI输出配置，详细见**•VpcUserRoiOutputConfigu...**。
VpcUserRoiConfigure* next	用户下一个ROI配置，当需要使用一图多框时配置，否则为NULL，默认为NULL。
uint64_t reserve1	预留参数。

- VpcCompressDataConfigure结构体

成员变量	说明
uint64_t lumaHeadAddr	y分量头地址。
uint64_t chromaHeadAddr	uv分量头地址。

成员变量	说明
uint32_t lumaHeadStride	y分量头stride，与 **FRAME结构体**中的stride_head参数值保持一致。
uint32_t chromaHeadStride	uv分量头stride，与 **FRAME结构体**中的stride_head参数值保持一致。
uint64_t lumaPayloadAddr	y分量数据的地址。
uint64_t chromaPayloadAddr	uv分量数据的地址。
uint32_t lumaPayloadStride	y分量数据的stride，与 **FRAME结构体**中的stride_payload参数值保持一致。
uint32_t chromaPayloadStride	uv分量数据的stride，与 **FRAME结构体**中的stride_payload参数值保持一致。

- VpcUserYuvSum结构体

成员变量	说明
uint32_t ySum	y分量总和。
uint32_t uSum	u分量总和。
uint32_t vSum	v分量总和。
uint64_t reserve1	预留参数。

- VpcUserPerformanceTuningParameter结构体

成员变量	说明
uint64_t reserve1	预留参数1。
uint64_t reserve2	预留参数2。
uint64_t reserve3	预留参数3。
uint64_t reserve4	预留参数4。
uint64_t reserve5	预留参数5。

- VpcUserRoiInputConfigure 结构体

成员变量	说明
VpcUserCropConfigure cropArea	用户抠图部分的输入数据配置，详细见**•VpcUserCropConfigure 结构...**。
uint64_t reserve1	预留参数。

- VpcUserRoiOutputConfigure结构体

成员变量	说明
uint8_t* addr	输出图片的首地址。 使用Matrix提供的HIAI_DVPP_DMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址16对齐）。HIAI_DVPP_DMalloc接口的说明，请参见《 Matrix API参考》。
uint32_t bufferSize	输出buffer的大小，根据yuv420sp计算。
uint32_t widthStride	输出图片的宽步长，需要16对齐；宽stride最小为32，最大为4096。
uint32_t heightStride	输出图片的高步长，需要2对齐；高stride最小为6，最大为4096。 输出为yuv420sp图像，需要根据heightStride计算出uv数据的起始地址。
VpcUserCropConfigure outputArea	用户指定输出区域坐标，详细见**•VpcUserCropConfigure 结构...**。 若实现8K缩放功能，不用配置该参数。
uint64_t reserve1	预留参数。

- VpcUserCropConfigure 结构体

  关于上偏移、下偏移、左偏移、右偏移各概念的解释请参见**表1-2**。

成员变量	说明
uint32_t leftOffset	左偏移，必须为偶数。 贴图区域相对输出图片的左偏移16对齐。
uint32_t rightOffset	右偏移，必须为奇数。
uint32_t upOffset	上偏移，必须为偶数。
uint32_t downOffset	下偏移，必须为奇数。
uint64_t reserve1	预留参数。

7.2 vdec_in_msg 中的结构体和类

该结构体在DDK安装目录下的“ddk/include/inc/dvpp/Vdec.h”文件中定义

- FRAME结构体

成员变量	说明
int height	输出图像的高（对齐后的值，H264为16对齐，H265为64对齐）。
int width	输出图像的宽（对齐后的值，H264为16对齐，H265为64对齐）。
int realHeight	真实图像的高。
int realWidth	真实图像的宽。
unsigned char* buffer	输出图像的内存地址。
int buffer_size	输出图像的内存大小。
unsigned int offset_payload_y	输出图像payload的Y分量偏移量，payload的Y分量地址=buffer + offset_payload_y。
unsigned int offset_payload_c;	输出图像payload的C分量偏移量，payload的C分量地址=buffer + offset_payload_c。
unsigned int offset_head_y;	输出图像head的Y分量偏移量，head的Y分量地址=buffer + offset_head_y。
unsigned int offset_head_c;	输出图像head的C分量偏移量，head的C分量地址=buffer + offset_head_c。
unsigned int stride_payload;	输出图像payload的stride。
unsigned int stride_head;	输出图像head的stride。
unsigned short bitdepth;	输出图像的位深。
char video_format[10];	输入视频的格式，为“h264”或“h265”。
char image_format[10];	输出图像的格式，为“nv12”或“nv21”。

- HIAI_DATA_SP类

成员变量或函数	说明
unsigned long long frameIndex	帧序号。
void * frameBuffer	用户申请用于存放输出帧的内存。
unsigned int frameSize	用户申请用于存放输出帧的内存大小。
void setFrameIndex(unsigned long long index)	设置帧序号函数。
unsigned long long getFrameIndex()	获取帧序号函数。

成员变量或函数	说明
void setFrameBuffer(void * frameBuff)	设置frameBuffer地址。
void * getFrameBuffer()	获取frameBuffer地址。
void setFrameSize(unsigned int size)	设置frameSize大小。
unsigned int getFrameSize()	获取frameSize大小。

- VDECERR结构体

成员变量	说明

ERRTYPE errType	错误类型。
	enum ERRTYPE{
	//VDEC解码器状态异常错误，用户需要销毁解码实例，再重新创建实例
	ERR_INVALID_STATE = 0x10001,
	//硬件错误，包含解码器启动、执行、停止等异常，用户需要销毁解码实例，再重新创建实例
	ERR_HARDWARE,
	//将视频码流分解成多帧图片异常，用户需要检查输入的视频流数据是否正确
	ERR_SCD_CUT_FAIL,
	//解码某一帧异常，用户需要检查输入的视频流数据是否正确
	ERR_VDM_DECODE_FAIL,
	//暂未使用，预留
	ERR_ALLOC_MEM_FAIL,
	//包括输入视频分辨率超范围（用户需要检查输入视频流的分辨率）、内部动态申请内存失败（用户检查系统是否有可用内存）等异常
	ERR_ALLOC_DYNAMIC_MEM_FAIL,
	//系统内部申请VDEC的输入、输出buffer异常，用户检查系统是否有可用内存
	ERR_ALLOC_IN_OR_OUT_PORT_MEM_FAIL,
	//码流错误，暂未使用，预留
	ERR_BITSTREAM,
	//输入视频格式错误，暂未使用，预留
	ERR_VIDEO_FORMAT,
	//输出格式配置错误，暂未使用，预留
	ERR_IMAGE_FORMAT,
	//回调函数为空错误，暂未使用，预留
	ERR_CALLBACK,
	//输入buffer为空错误，暂未使用，预留
	ERR_INPUT_BUFFER,
	//输入buffer大小<=0错误，暂未使用，预留
	ERR_INBUF_SIZE,
	//系统内部将解码结果通过回调函数返回给用户的线程异常，用户需要检查系统

成员变量	说明

	中资源（例如：线程、内存等）是否可用
	ERR_THREAD_CREATE_FBD_FAIL,
	//创建解码实例失败，用户需要重新创建解码实例
	ERR_CREATE_INSTANCE_FAIL,
	//初始化解码器失败，例如解码实例个数超出范围（最大16），用户需要释放部分解码实例后再重新创建实例
	ERR_INIT_DECODER_FAIL,
	//系统内部获取某路视频流的解码句柄失败，用户需要重新创建解码实例
	ERR_GET_CHANNEL_HANDLE_FAIL,
	//系统内部设置解码实例异常，用户需要检查解码的入参值是否正确，例如输入视频格式video_format、输出帧格式image_format等
	ERR_COMPONENT_SET_FAIL,
	//系统内部设置解码实例名称异常，用户需要检查解码的入参值是否正确，例如输入视频格式video_format、输出帧格式image_format等
	ERR_COMPARE_NAME_FAIL,
	//其它错误
	ERR_OTHER
	};

unsigned short channelId	解码错误的通道。

7.3 IMAGE_CONFIG 中的结构体

- ROI_CONFIG结构体

成员变量	说明	取值范围
CROP_CONFIG：crop_config	裁剪参数配置结构体。	详细见**•CROP_CONFIG结构体**。
YUV_SUM_OUT_CONFIG：sum_out	输出参数配置结构体1。	详见**•YUV_SUM_OUT_CONFIG结构体**。
NORMAL_OUT_CONFIG：normal_out	输出参数配置结构体2(预留，暂时不支持)。	详见**•NORMAL_OUT_CONFIG结构体**。

成员变量	说明	取值范围
ROI_CONFIG*：next	下一个ROI_CONFIG结构体指针。	多框时配置，单框时配置为NULL。

- CROP_CONFIG结构体

成员变量	说明	取值范围
int：enable	是否需要进行裁剪操作。	0：不需要裁剪。 1：需要裁剪。
unsigned int：hmin	水平最小偏移。	偶数，且小于hmax。
unsigned int：hmax	水平最大偏移。	奇数，且小于输入宽度in_width。
unsigned int：vmin	垂直最小偏移。	偶数，且小于vmax。
unsigned int：vmax	垂直最大偏移。	奇数，且小于输入高度in_height。

- YUV_SUM_OUT_CONFIG结构体

成员变量	说明	取值范围
int：enable	是否启用该通道进行输出。	0：不启用。 1：启用。
unsigned int：out_width	输出图像宽度。	偶数。128~4096，缩放系数需满足[0.03125, 4]。
unsigned int：out_height	输出图像高度。	偶数。16~4096，缩放系数需满足[0.03125, 4]。
char*：out_buffer	输出图像内存地址指针。	输出内存大小需要根据输出宽128对齐，高16对齐后的分辨率计算。
unsigned int：yuv_sum	输出图像所有yuv值的和。	预留，暂不支持。

- NORMAL_OUT_CONFIG结构体

成员变量	说明	取值范围
int：enable	是否启用该通道进行输出。	0：不启用。 1：启用。

成员变量	说明	取值范围
unsigned int：out_width	输出图像宽度。	偶数。128~4096，缩放系数需满足[0.03125, 4]。
unsigned int：out_height	输出图像高度。	偶数。16~4096，缩放系数需满足[0.03125, 4]。
char*：out_buffer	输出图像内存地址指针。	输出内存大小需要根据输出宽128对齐，高16对齐后的分辨率计算。

7.4 dvpp_engine_capability_stru 中的结构体

该结构体在DDK安装目录下的“ddk/include/inc/dvpp/DvppCapability.h”文件中定义

- dvpp_resolution_stru结构体

成员变量	说明	取值范围
uint32_t resolution_high;	高度分辨率。	最大值： VDEC：4096 JPEGD：8192 PNGD：4096 JPEGE：8192 VPC：4096 VENC：1920 最小值： VDEC：128 JPEGD：32 PNGD：32 JPEGE：32 VPC：16 VENC：128

成员变量	说明	取值范围
uint32_t resolution_width;	宽度分辨率。	最大值： VDEC：4096 JPEGD：8192 PNGD：4096 JPEGE：8192 VPC：4096 VENC：1920 最小值： VDEC：128 JPEGD：32 PNGD：32 JPEGE：32 VPC：16 VENC：128

- dvpp_format_unit_stru 结构体

成员变量	说明	取值范围

enum dvpp_color_format color_format;	支持图片格式	*enum* dvpp_color_format {
		//YUV444 in different ordering of YUV Semi-Planar/Packed，8 bit，Linear。
		DVPP_COLOR_YUV444_YUV_P_8BIT_LIN,
		DVPP_COLOR_YUV444_YVU_P_8BIT_LIN,
		DVPP_COLOR_YUV444_UYV_P_8BIT_LIN,
		DVPP_COLOR_YUV444_UVY_P_8BIT_LIN,
		DVPP_COLOR_YUV444_VYU_P_8BIT_LIN,
		DVPP_COLOR_YUV444_VUY_P_8BIT_LIN,
		DVPP_COLOR_YUV444_UV_SP_8BIT_LIN,
		DVPP_COLOR_YUV444_VU_SP_8BIT_LIN,
		/*422*/
		DVPP_COLOR_YUYV422_YUYV_P_8BIT_LIN,
		DVPP_COLOR_YUYV422_YVYU_P_8BIT_LIN,
		DVPP_COLOR_YUYV422_UYVY_P_8BIT_LIN,
		DVPP_COLOR_YUYV422_VYUY_P_8BIT_LIN,
		DVPP_COLOR_YUV422_UV_SP_8BIT_LIN,
		DVPP_COLOR_YUV422_VU_SP_8BIT_LIN,
		/*420*/
		DVPP_COLOR_YUV420_SP_8BIT_LIN,
		DVPP_COLOR_YVU420_SP_8BIT_LIN,
		DVPP_COLOR_YUV420_SP_8BIT_HFBC,
		DVPP_COLOR_YVU420_SP_8BIT_HFBC,
		DVPP_COLOR_YUV420_SP_10BIT_HFBC,
		DVPP_COLOR_YVU420_SP_10BIT_HFBC,
		DVPP_COLOR_YUV420_P_8BIT_LIN,

成员变量	说明	取值范围
		/*400*/ DVPP_COLOR_YVU400_SP_8BIT, /*rgb888*/ DVPP_COLOR_RGB888_RGB_P_8BIT_LIN, DVPP_COLOR_RGB888_RBG_P_8BIT_LIN, DVPP_COLOR_RGB888_GBR_P_8BIT_LIN, DVPP_COLOR_RGB888_GRB_P_8BIT_LIN, DVPP_COLOR_RGB888_BRG_P_8BIT_LIN, DVPP_COLOR_RGB888_BGR_P_8BIT_LIN, /*argb888*/ DVPP_COLOR_ARGB8888_ARGB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_ARBG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_AGBR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_AGRB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_ABRG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_ABGR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RAGB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RABG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RGBA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RGAB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RBAG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RBGA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BRGA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BRAG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BGAR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BGRA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BAR

成员变量	说明	取值范围
		G_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BAGR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GRAG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GRBA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GABR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GARB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GBRA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GBAR_P_8BIT_LIN, PIC_JPEG, PIC_PNG, VIO_H265, VIO_H264 };

uint32_t compress_type;	压缩类型	enum dvpp_compress_type {
		arithmetic_code =0,
		huffman_code
		};

uint32_t stride_size;	步长大小	VDEC：128 JPEGD：128 PNGD：128 JPEGE：0 VPC：128 VENC：0

enum dvpp_high_align_type high_alignment;	高度对齐类型	enum dvpp_high_align_type {
		pix_random = 0,
		two_pix_alignment = 2,
		four_pix_alignment = 4,
		eight_pix_alignment = 8,
		sixteen_pix_alignment = 16
		};

成员变量	说明	取值范围

enum dvpp_high_align_type width_alignment;	宽度对齐类型	enum dvpp_high_align_type {
		pix_random = 0,
		two_pix_alignment = 2,
		four_pix_alignment = 4,
		eight_pix_alignment = 8,
		sixteen_pix_alignment = 16
		};

uint32_t out_mem_alignment;	输出内存对齐参数	-

- dvpp_perfomance_unit_stru结构体

成员变量	说明	取值范围
uint32_t resolution_high;	高度分辨率	VDEC：1920 JPEGD：1920 PNGD：1920 JPEGE：1920 VPC：3840 VENC：1920
uint32_t resolution_width;	宽度分辨率	VDEC：1080 JPEGD：1080 PNGD：1080 JPEGE：1080 VPC：2160 VENC：1080
uint32_t stream_num;	流大小	VDEC：16 JPEGD：0 PNGD：0 JPEGE：0 VPC：0 VENC：1
unsigned long fps;	帧率	VDEC：30 JPEGD：256 PNGD：24 JPEGE：64 VPC：90 VENC：30

- dvpp_pre_contraction_stru结构体

成员变量	说明	取值范围

enum dvpp_support_type is_support;	预缩小是否支持	VPC：support
		others：no support
		其中，support取值范围如下：
		enum dvpp_support_type {
		no_support =0, //no support
		do_support //support
		};

uint32_t contraction_types;	缩小类型	VPC：3 others：0
uint32_t contraction_size[DVPP_PRE_CONTRATION_TYPE_MAX];	预缩小固定比例	VPC：2/4/8 others：0
enum dvpp_support_type is_horizontal_support;	是否支持水平方向预缩小	VPC：support others：no support
enum dvpp_support_type is_vertical_support;	是否支持垂直方向预缩小	VPC：support others：no support

- dvpp_pos_scale_stru结构体

成员变量	说明	取值范围
enum dvpp_support_type is_support;	后缩放是否支持	VPC：support others：no support
uint32_t min_scale;	最小缩放系数	VPC：1 others：1
uint32_t max_scale;	最大缩放系数	VPC：4 others：1
enum dvpp_support_type is_horizontal_support;	是否支持水平方向后缩放	VPC：support others：no support
enum dvpp_support_type is_vertical_support;	是否支持垂直方向后缩放	VPC：support others：no support

- dvpp_vpc_data_spec_stru结构体

成员变量	说明	取值范围
uint32_t input_type;	输入格式所属类型	包括以下两种： ● 0：表示HFBC类型 ● 1：表示YUV或RGB类型
struct dvpp_resolution_stru min_resolution;	最小分辨率。	请参见**•struct dvpp_resolution_...**
struct dvpp_resolution_stru max_resolution;	最大分辨率	请参见 **struct dvpp_resolution_...**

enum dvpp_align_type high_alignment;	高度对齐类型。	enum dvpp_high_align_type {
		//1像素对齐
		pix_random = 0,
		//2像素对齐
		two_pix_alignment = 2,
		//4像素对齐
		four_pix_alignment = 4,
		//8像素对齐
		eight_pix_alignment = 8,
		//16像素对齐
		sixteen_pix_alignment = 16
		};

成员变量	说明	取值范围

enum dvpp_align_type width_alignment;	宽度对齐类型。	enum dvpp_high_align_type {
		//1像素对齐
		pix_random = 0,
		//2像素对齐
		two_pix_alignment = 2,
		//4像素对齐
		four_pix_alignment = 4,
		//8像素对齐
		eight_pix_alignment = 8,
		//16像素对齐
		sixteen_pix_alignment = 16
		};

7.5 vpc_in_msg 中的结构体

7.5.1 Rdma 通道结构体 RDMACHANNEL

成员变量	说明	取值范围
Long luma_head_addr	Y分量的头地址。	-
Long chroma_head_addr	UV分量的头地址。	-
unsigned int luma_head_stride	Y分量的头stride。	-
nsigned int chroma_head_stride;	UV分量的头stride。	-
long luma_payload_addr	Y分量数据的地址。	-
long chroma_payload_addr	UV分量数据的地址。	-
unsigned int luma_payload_stride	Y分量数据的头stride。	-
unsigned int chroma_payload_stride;	UV分量数据的头stride。	-

7.5.2 Vpc 内部优化结构体 VpcTurningReverse

成员变量	说明	取值范围
unsigned int reverse1	内部优化预留接口1。	--
unsigned int reverse2	内部优化预留接口2。	--
unsigned int reverse3	内部优化预留接口3。	--
unsigned int reverse4	内部优化预留接口4。	--

8 附录

8.1 辅助功能接口

- JpegCalBackFree类辅助接口

class JpegCalBackFree：

JpegCalBackFree() :go

~JpegCalBackFree()

JpegCalBackFree(JpegCalBackFree& others)

void setBuf(void* addr4Free)

void setBuf(void* addr4Free, size_t size4Free)

void operator() ()

const JpegCalBackFree& operator= (JpegCalBackFree& others)

- AutoBuffer类辅助接口

class AutoBuffer：

AutoBuffer()

~AutoBuffer()

void Reset()

char* allocBuffer(int size)

char* getBuffer()

int getBufferSize()

AutoBuffer(const AutoBuffer&)

const AutoBuffer& operator= (const AutoBuffer&)

- HIAI_DATA_SP类辅助接口

class HIAI_DATA_SP：

HIAI_DATA_SP()

virtual ~HIAI_DATA_SP()

void setFrameIndex(unsigned long long index)

unsigned long long getFrameIndex()

void setFrameBuffer(void * frameBuff)

void * getFrameBuffer()

void setFrameSize(unsigned int size)

unsigned int getFrameSize()

8.2 不推荐使用接口列表

- class IDVPPCAPABILITY：

virtual ~IDVPPCAPABILITY(void)

virtual int process(dvppapi_ctl_msg *MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

- class IJPEGDAPI：

virtual ~IJPEGDAPI(void)

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

- class IJPEGEAPI：

virtual ~IJPEGEAPI(){}

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

- class IVDECAPI：

virtual ~IVDECAPI(void)

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

- class IVENCAPI：

virtual ~IVENCAPI(void)

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

- class IVPCAPI：

virtual ~IVPCAPI(void)

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

8.3 DVPP 执行器样例使用说明

DVPP执行器工具集成了DVPP的六个基础功能，包括VPC、JPEGE、JPEGD、VENC、VDEC以及PNGD。

8.3.1 环境准备

步骤1 将编译生成的可执行文件“sample_dvpp_hlt_ddk”复制到Device侧任意路径。

说明

依赖SO为： libc_sec.so， libDvpp_api.so， libDvpp_jpeg_decoder.so， libDvpp_jpeg_encoder.so， libDvpp_png_decoder.so， libDvpp_vpc.so， libOMX_hisi_video_decoder.so， libOMX_hisi_video_encoder.so， libslog.so

这些SO默认已存在deivce侧的“/usr/lib64”目录，用户无需单独拷贝。

步骤2 输入数据文件放入到可执行文件所在路径下，运行如下命令，其中，**paramList**是**8.3.2 DVPP执行器样例入参说明**中介绍的参数。

./sample_dvpp_hlt_ddk paramList。

----结束

8.3.2 DVPP 执行器样例入参说明

参数	描述
img_width	输入图片的宽度。
img_height	输入图片的高度。

参数	描述
in_format	● 对于VPC，输入图片的格式如下： INPUT_YUV400, // 0 INPUT_YUV420_SEMI_PLANNER_UV, // 1 INPUT_YUV420_SEMI_PLANNER_VU, // 2 INPUT_YUV422_SEMI_PLANNER_UV, // 3 INPUT_YUV422_SEMI_PLANNER_VU, // 4 INPUT_YUV444_SEMI_PLANNER_UV, // 5 INPUT_YUV444_SEMI_PLANNER_VU, // 6 INPUT_YUV422_PACKED_YUYV, // 7 INPUT_YUV422_PACKED_UYVY, // 8 INPUT_YUV422_PACKED_YVYU, // 9 INPUT_YUV422_PACKED_VYUY, // 10 INPUT_YUV444_PACKED_YUV, // 11 INPUT_RGB, // 12，输入图像格式为RGB888 INPUT_BGR, // 13，输入图像格式为BGR888 INPUT_ARGB, // 14，此格式的输入图像在存储时各分量的排列顺序类似RGB888，其中，A表示透明度 INPUT_ABGR, // 15，此格式的输入图像在存储时各分量的排列顺序类似BGR888，其中，A表示透明度 INPUT_RGBA, // 16，此格式的输入图像在存储时各分量的排列顺序类似RGB888，其中，A表示透明度 INPUT_BGRA, // 17，此格式的输入图像在存储时各分量的排列顺序类似BGR888，其中，A表示透明度 INPUT_YUV420_SEMI_PLANNER_UV_10BIT, // 18 INPUT_YUV420_SEMI_PLANNER_VU_10BIT, // 19 ● 对于VDEC，输入码流的格式如下： – 0：h264格式的码流 – 1：h265格式的码流 ● 对于VENC，输出码流的编码格式如下： – 0：H265 main level（仅支持Slice码流） – 1：H264 baseline level – 2：H264 main level – 3：H264 high level ● 对于JPEGE，输入图片的格式如下： – 0：JPGENC_FORMAT_UYVY – 1：JPGENC_FORMAT_VYUY – 2：JPGENC_FORMAT_YVYU – 3：JPGENC_FORMAT_YUYV – 16：JPGENC_FORMAT_NV12

参数	描述
	– 17：JPGENC_FORMAT_NV21
out_format	VPC输出图片格式如下： ● 0：OUTPUT_YUV420SP_UV ● 1：OUTPUT_YUV420SP_VU
yuvStoreType	VENC输入yuv数据的存储格式如下： ● 0：YUV420 semi-planner ● 1：YVU420 semi-planner
inWidthStride	VPC输入图像宽度方向的步长。详细见**入参：VpcUserImageConfigure**中widthStride的描述。
inHighStride	VPC输入图像高度方向的步长。详细见**入参：VpcUserImageConfigure**中heightStride的描述。
outWidthStride	VPC输出图片的宽步长，需要16对齐。
outHighStride	VPC输出图片的高步长，需要2对齐；
hmin	VPC用户指定抠图区域左偏移，必须为偶数。
hmax	VPC用户指定抠图区域右偏移，必须为奇数。
vmin	VPC用户指定抠图区域上偏移，必须为偶数。
vmax	VPC用户指定抠图区域下偏移，必须为奇数。
outLeftOffset	VPC用户指定输出区域左偏移，必须为偶数。
outRightOffset	VPC用户指定输出区域右偏移，必须为奇数。
outUpOffset	VPC用户指定输出区域上偏移，必须为偶数。
outDownOffset	VPC用户指定输出区域下偏移，必须为奇数。
out_width	输出图像宽度。
out_high	输出图像高度。
in_image_file	输入图像文件的路径，包含文件名。
out_image_file	输出图像文件的路径，包含文件名。
rank	● 0：JPEGD输出yuv420 semi-planar数据； ● 1：JPEGD输出yuv原格式数据。
transform	PNGD处理图像时的转换标志： ● 1：表示RGBA转换到RGB； ● 0：保留原格式。

参数	描述
test_type	输入数据：1、2、3、4、5、6、7、8、9、10、12。 ● 1：VPC基础功能1（抠图、缩放）。 ● 2：VPC基础功能2（原图缩放、抠图缩放、一图多框和8k图像缩放）。 ● 3：VDEC。 ● 4：VENC。 ● 5：JPEGE。 ● 6：JPEGE用户指定输出内存。 ● 7：JPEGD。 ● 8：JPEGD用户指定输出内存。 ● 9：PNGD。 ● 10：PNGD用户指定输出内存。 ● 12：JPEGD+VPC+JPEGE。

8.3.3 VPC 使用说明

8.3.3.1 VPC 基础功能 1

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_vpc_1920x1080_nv12.yuv --out_image_file vpc_out.yuv --inWidthStride 1920 --inHighStride 1080 --in_format 1 --out_format 0 --hmax 1919 --hmin 0 --vmax 999 --vmin 0 --outLeftOffset 0 --outRightOffset 719 --outUpOffset 0 --outDownOffset 719 --outWidthStride 720 --outHighStride 720 --test_type 1
```

示例描述：调用VPC实现图像的裁剪、缩放。

- 输入图像：
  - 宽1920像素，高1080像素、格式为yuv420sp、名称为"dvpp_vpc_1920x1080_nv12.yuv”的图像。
  - 与可执行文件放在同一目录下。
- 输出图像：
  - 宽720像素、高720像素、格式为yuv420sp（nv12）、名称为“vpc_out.yuv”的图像
  - 生成的图像与可执行文件在同一目录下。
- 用户可以使用图像显示软件打开vpc_out.yuv图像进行正确性验证。

8.3.3.2 VPC 基础功能 2

```
./sample_dvpp_hlt_ddk --test_type 2
```

示例描述：本命令实现4个功能，调用VPC实现原图缩放、抠图缩放、一图多框和8k图像缩放功能。

- 输入图像：
  - 宽1920像素、高1080像素、格式为yuv420sp nv12、名称为“dvpp_vpc_1920x1080_nv12.yuv”的图像。

  - 宽8192像素、高8192像素、格式为yuv420sp nv12、名称为“dvpp_vpc_8192x8192_nv12.yuv”的图像。
  - 与可执行文件放在同一目录下。
- 输出图像：
  - NewVpcTest1函数实现1080p yuv420sp图像缩小至1280*720 yuv420sp（nv12），输出文件名为“NewVpcTest1Out.yuv”。
  - NewVpcTest2函数实现1080p yuv420sp图像中抠一张子图，并贴图至输出图片的指定位置，输出文件名为“NewVpcTest2Out.yuv”，该文件像素是1280*720，格式是yuv420sp。
  - NewVpcTest3函数实现1080p yuv420sp图像中抠五张子图，并贴图至输出图片的指定位置，输出文件名为“NewVpcTest3_*Out.yuv”。该文件像素是1280*720，格式是yuv420sp。
  - NewVpcTest4函数实现8192*8192的 yuv420sp图像缩放至4000*4000的yuv420sp图像，输出文件名为“NewVpcTest4Out.yuv”。
  - 生成的图像与可执行文件在同一目录下。
- 用户可以使用图像显示软件打开图像进行正确性验证。

8.3.4 VDEC 使用说明

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_vdec_h264_1frame_bp_51_1920x1080.h264 --out_image_file h264_image --in_format 0 --test_type 3
./sample_dvpp_hlt_ddk --in_image_file dvpp_vdec_h265_1frame_mp_51_1920x1080.h265 --out_image_file h265_image --in_format 1 --test_type 3
```

示例描述：本命令用于调用VDEC解码*.h264和*.h265格式的码流，并调用VPC对解码后图像进行解压缩成yuv格式图像。

- 输入视频：
  - 宽1920像素、高1080像素、名称为“dvpp_vdec_h264_1frame_bp_51_1920x1080.h264”的码流。
  - 宽1920像素、高1080像素、名称为“dvpp_vdec_h265_1frame_mp_51_1920x1080.h265”的码流。
  - 与可执行文件放在同一目录下。
- 输出图像：
  - 宽1920像素、高1080像素、格式为YUV420sp的图像。
  - 针对h264码流的解码，在“output_dir”目录下生成名称为“h264_image*”的图片，当前视频中只有一帧，所以只输出一张图像，如果视频中有多帧，则输出多张图像。
  - 针对h265码流的解码，在“output_dir”目录下生成名称为“h265_image*”的图片，当前视频中只有一帧，所以只输出一张图像，如果视频中有多帧，则输出多张图像。
- 用户可以使用图像显示软件打开“h264_image*”、“h265_image*”图像进行正确性验证。

8.3.5 VENC 使用说明

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_venc_128x128_nv12.yuv --img_width 128 --img_height 128 --in_format 0 --yuvStoreType 0 --test_type 4
```

示例描述：本命令用于调用VENC将yuv图片编码成h265格式的码流。

- 输入图像：
  - 宽128像素、高128像素、格式为yuv420sp nv12、名称为“dvpp_venc_128x128_nv12.yuv”的图像。
  - 与可执行文件放在同一目录下。
  - in_format对应到**入参：VencConfig**中coding_type。
  - yuvStoreType对应到**入参：VencConfig**中YUV_store_type。
- 输出视频：
  - 输出名称为“venc.bin”的码流文件。
  - 与可执行文件放在同一目录下。
- 用户可以使用ffmpeg解码venc.bin文件，使用命令“ffmpeg.exe -i venc.bin venc.jpg”解码，然后使用图像显示软件打开图像进行正确性校验。ffmpeg工具的官网路径为：http://ffmpeg.org/

8.3.6 JPEGE 使用说明

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_jpege_444_1920x1080_1920_1080_nv12.yuv --img_width 1920 --img_height 1080 --in_format 16 --test_type 5
./sample_dvpp_hlt_ddk --in_image_file dvpp_jpege_444_1920x1080_1920_1080_nv12.yuv --img_width 1920 --img_height 1080 --in_format 16 --test_type 6
```

示例描述：分别使用dvpp内部申请输出内存（test_type 5）和客户申请输出内存（test_type 6）两种调用方式，将yuv格式图像编码成jpg图像。

- 输入图像：
  - 宽1920像素、高1080像素、格式为yuv420sp nv12、名称为“dvpp_jpege_444_1920x1080_1920_1080_nv12.yuv”的图像。
  - 与可执行文件放在同一目录下。
- 输出图像：
  - 宽1920像素、高1080像素、输出名称为“outfile_t*.jpg”（*为线程id）的图像。
  - 与可执行文件放在同一目录下。

8.3.7 JPEGD 使用说明

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_jpegd_decode_1920x1080.jpg --test_type 7 --rank 0
./sample_dvpp_hlt_ddk --in_image_file dvpp_jpegd_decode_1920x1080.jpg --test_type 8 --rank 0
```

示例描述：分别使用dvpp内部申请输出内存（test_type 7）和客户申请输出内存（test_type 8）两种调用方式，将jpg图像解码成yuv图像。

- 输入图像：
  - 宽1920像素、高1080像素、名称为“dvpp_jpegd_decode_1920x1080.jpg”的图像。
  - 与可执行文件放在同一目录下。
  - -- rank为0或不设置rank参数，**输出NV21格式数据**。-- rank不为0输出原格式的半平面数据。
- 输出图像：
  - 宽1920像素、高1080像素、输出名称为“dvppapi_jpegd_wxx_hxx_fx_tx.yuv”的图像。

  - 与可执行文件放在同一目录下。
- 用户可以使用图像显示软件打开输出的yuv图像进行正确性验证。

8.3.8 PNGD 使用说明

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_pngd_1920x1080_RGBA.png --out_image_file
pngd_out_RGBAtoRGB_1.rgb --test_type 9  --transform 1
./sample_dvpp_hlt_ddk --in_image_file dvpp_pngd_1920x1080_RGBA.png --out_image_file
pngd_out_RGBAtoRGB_2.rgb --test_type 10 --transform 1
```

示例描述：分别使用dvpp内部申请输出内存（test_type 9）和客户申请输出内存（test_type 10）两种调用方式，将png图像解码成rgb888格式图像。

- 输入图像：
  - 宽1920像素、高1080像素、名称为“dvpp_pngd_1920x1080_RGBA.png”的图像。
  - 与可执行文件放在同一目录下。
  - 参数transform为1时，表示将rgba格式的png图片转成rgb888格式输出。
- 输出图像：
  - 宽1920像素、高1080像素、分别输出名称为“pngd_out_RGBAtoRGB_1.rgb”（test_type 9）和“pngd_out_RGBAtoRGB_2.rgb”（test_type 10）的图像。
  - 与可执行文件放在同一目录下。
- 用户可以使用图像显示软件打开pngd_out_RGBAtoRGB_1.rgb和pngd_out_RGBAtoRGB_2.rgb图像进行正确性验证。

8.3.9 JPEGD+VPC+JPEGE 串联使用说明

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_jpegd_vpc_jpege.jpg --out_image_file
out_jpegd_vpc_jpege_new.jpg --hmax 1919 --hmin 0 --vmax 999 --vmin 0 --outLeftOffset 0 --
outRightOffset 719 --outUpOffset 0 --outDownOffset 719 --outWidthStride 720 --outHighStride 720 --
test_type 12
```

示例描述：JPEGD+VPC+JPEGE三个·模块串联使用的调用实例。其中中test_type 12使用的是客户申请输出内存的调用方式。此命令的功能是调用JPEGD将一张jpg图片进行解码生成yuv420 sp图片，送入VPC进行裁剪、缩放，然后将VPC的输出送给JPEGE进行编码，输出jpg图片。

- 输入图像：
  - 宽1920像素、高1080像素、名称为“dvpp_jpegd_vpc_jpege.jpg”的图像。
  - 与可执行文件放在同一目录下。
- 输出图像：
  - 宽720像素、高720像素、名称为“out_jpegd_vpc_jpege_new.jpg”的图像。
  - 与可执行文件放在同一目录下。

8.4 异常处理

当调用方出现调用**DvppCtl**或者**DvppGetOutParameter**失败时，也即调用该函数返回值为-1时，可通过Mind Studio界面的Log窗口查看日志，在**ModuleName**参数处选择DVPP，然后单击Search查询日志。根据Time列的时间查看最新日志，并根据日志的提示排查异常调用错误。

示例：调用方在使用VPC功能时，输入图片宽高不是128x16对齐，则在Content列的日志内容处提示如下信息。

```
The input image width should be divided by 128, high should be divided by 16, now width 512, high 456.
```

说明

关于日志查看的详细操作，可参见《Ascend 310 Mind Studio开发辅助工具》中的“日志工具 > 基本操作 > 日志查看”章节。

8.5 缩略语和术语一览

表 8-1 缩略语和术语

缩略语	英文全名
VDEC	video decoder
VENC	video encoder
JPEGD	JPEG decoder
JPEGE	JPEG encoder
PNGD	PNG decoder
VPC	vision preprocessing core
DVPP	digital vision pre-processing
HFBC	Hisi Frame Buffer Compress

8.6 修订记录

文档版本	发布日期	修改说明
01	2020-05-09	第一次正式发布。