

CANN  
5.0.4.6

# DVPP API 参考 (开放态, Ascend 310)

文档版本	01
发布日期	2023-04-10



**版权所有 © 华为技术有限公司 2023。保留一切权利。**

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

## **商标声明**



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暗示的声明或保证。

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

<b>1 使用须知</b>	<b>1</b>
<b>2 概述</b>	<b>2</b>
2.1 接口简介	2
2.2 接口列表	3
2.3 VPC 功能	5
2.4 JPEG 功能	14
2.5 JPEGD 功能	15
2.6 PNGD 功能	17
2.7 VDEC 功能	18
2.8 VENC 功能	20
2.9 关于输入输出内存的说明	21
<b>3 VPC/JPEG/JPEGD/PNGD 功能接口</b>	<b>23</b>
3.1 CreateDvppApi	23
3.2 DvppCtl	24
3.2.1 接口说明	24
3.2.2 VPC 参数说明	26
3.2.3 JPEG 参数说明	32
3.2.4 JPEGD 参数说明	34
3.2.5 PNGD 参数说明	38
3.2.6 查询 DVPP 引擎参数说明	40
3.3 DestroyDvppApi	42
3.4 DvppGetOutParameter	43
<b>4 VDEC 功能接口</b>	<b>45</b>
4.1 总体说明	45
4.2 CreateVdecApi	45
4.3 VdecCtl	46
4.4 DestroyVdecApi	53
<b>5 VENC 功能接口</b>	<b>54</b>
5.1 总体说明	54
5.2 CreateVenc	54
5.3 SetVencParam	56
5.4 RunVenc	57

5.5 DestroyVenc.....	58
<b>6 数据类型.....</b>	<b>59</b>
6.1 VpcUserImageConfigure 中的结构体.....	59
6.2 VdecInMsg 中的类.....	62
6.3 vdec_in_msg 中的结构体和类.....	65
6.4 dvpp_engine_capability_stru 中的结构体.....	69
<b>7 返回码列表.....</b>	<b>78</b>
7.1 公共返回码.....	78
7.2 VPC 返回码.....	79
7.3 JPEGD 返回码.....	80
7.4 JPEGG 返回码.....	81
7.5 VDEC 返回码.....	81
7.6 VENC 返回码.....	82
7.7 PNGD 返回码.....	83
<b>8 调用示例.....</b>	<b>85</b>
8.1 获取示例代码.....	85
8.2 实现 VPC 功能.....	85
8.3 实现 JPEGG 功能.....	93
8.4 实现 JPEGD 功能.....	96
8.5 实现 PNGD 功能.....	98
8.6 实现 VDEC 功能.....	102
8.7 实现 VENC 功能.....	106
<b>9 样例使用指导.....</b>	<b>108</b>
<b>10 附录.....</b>	<b>109</b>
10.1 辅助功能接口.....	109

# 1 使用须知

---

《[应用软件开发指南 \(C&C++, 开放态\)](#)》中的媒体数据处理和《[DVPP API参考 \(开放态, Ascend310\)](#)》都是DVPP对外提供的接口，支持的接口功能范围相同，但是《[应用软件开发指南 \(C&C++, 开放态\)](#)》中描述的接口支持在标准形态和Control CPU开放形态下调用，《[DVPP API参考 \(开放态, Ascend310\)](#)》中描述的接口仅支持在Control CPU开放形态下调用。

如果您是首次使用DVPP接口，建议使用《[应用软件开发指南 \(C&C++, 开放态\)](#)》中描述的接口，保证后续版本接口功能以及业务的连续演进。《[DVPP API参考 \(开放态, Ascend310\)](#)》中描述的接口是为了兼容旧版本，保证使用该部分接口的用户能继续使用。

# 2 概述

---

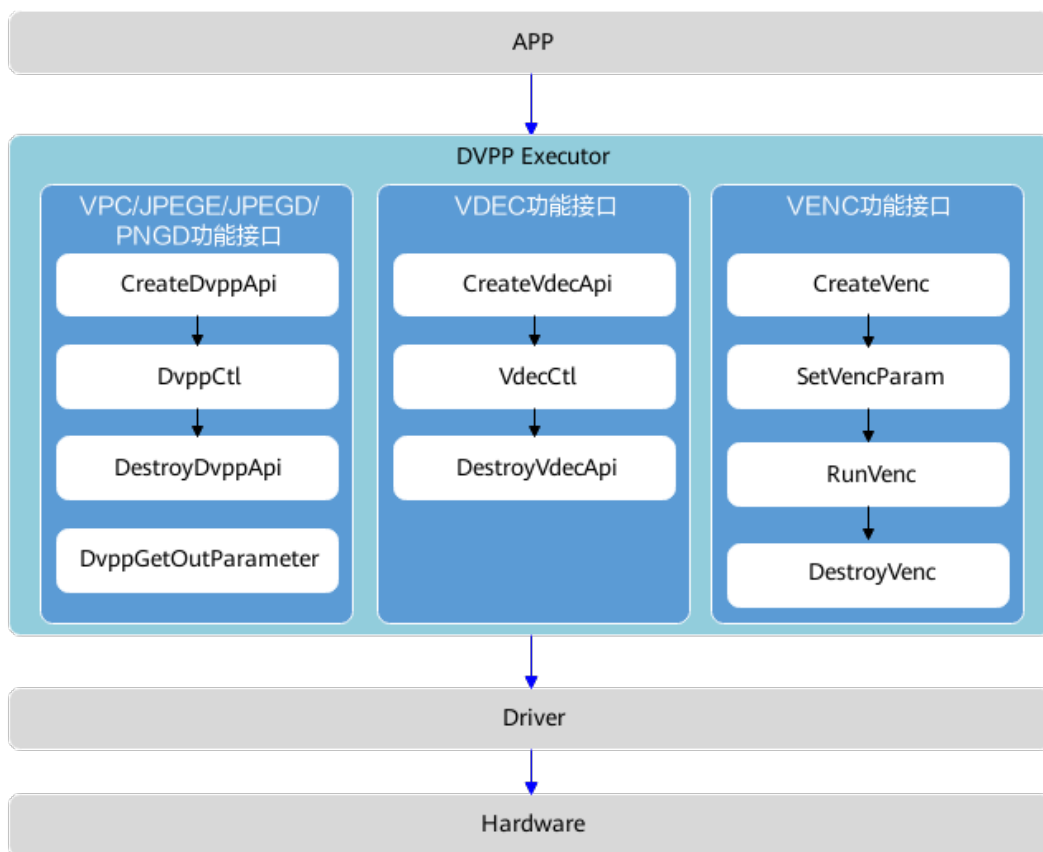
- [2.1 接口简介](#)
- [2.2 接口列表](#)
- [2.3 VPC功能](#)
- [2.4 JPEG功能](#)
- [2.5 JPEGD功能](#)
- [2.6 PNGD功能](#)
- [2.7 VDEC功能](#)
- [2.8 VENC功能](#)
- [2.9 关于输入输出内存的说明](#)

## 2.1 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，仅支持在Device上调用这些接口，本文档中的接口描述包括：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，本文档中的DVPP接口只能在Device上调用：

图 2-1 流程图



DVPP包括如下功能：

- VPC ( vision preprocessing core ) 功能：支持对图片做抠图、缩放、叠加、拼接、格式转换等操作，详细描述请参见[VPC功能](#)。
- JPEGD ( JPEG decoder ) 功能：将.jpg、.jpeg、.JPG、.JPEG图片解码成YUV格式图片，详细描述请参见[JPEGD功能](#)。
- JPEGE ( JPEG encoder ) 功能：将YUV格式图片编码成.jpg图片，详细描述请参见[JPEGE功能](#)。
- PNGD ( PNG decoder ) 功能：实现PNG格式图片的硬件解码，详细描述请参见[PNGD功能](#)。
- VDEC ( video decoder ) 功能：实现视频的解码，详细描述请参见[VDEC功能](#)。
- VENC ( video encoder ) 功能：实现视频的编码，详细描述请参见[VENC功能](#)。

## 2.2 接口列表

您需要参见《[CANN 软件安装指南](#)》部署开发环境，部署完成后，您可以获取以下文件：

- 从“CANN软件安装后文件存储路径/Driver组件的安装目录/driver/include/dvpp”目录下获取调用接口所需的头文件。
- 从“CANN软件安装后文件存储路径/develop/lib64”目录下获取编译DVPP接口所需的库文件。

- 从“CANN软件安装后文件存储路径/include/acl/ops”目录下获取调用内存申请/释放接口所需的头文件acl\_dvpp.h。
- 从“CANN软件安装后文件存储路径/lib64”目录下获取申请/释放内存所需的库文件libacl\_dvpp.so。

接口调用的示例代码，您可以从Sample包中获取，Sample包的获取、安装以及示例代码的编译运行，请参见[样例使用指导](#)。

#### 须知

- 本文档中的DVPP接口只能在Device上调用。
- 编译基于DVPP接口的代码逻辑时，请按照引用的头文件，依赖对应的so文件（申请/释放内存时，依赖acllib组件的libacl\_dvpp.so），引用多余的so文件可能导致后续版本升级时存在兼容性问题。

表 2-1 接口列表

分类	接口	功能说明	定义接口的头文件	依赖的库文件
实现VPC/JPEGE/JPEGD/PNGD功能	<b>CreateDvppApi</b>	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 <b>DvppCtl接口</b> 处理图片，可以跨函数调用，跨线程调用。	Dvpp.h	libDvpp_api.so
	<b>DvppCtl</b>	使用 <b>CreateDvppApi</b> 接口创建的实例来调用 <b>DvppCtl</b> 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。		
	<b>DestroyDvppApi</b>	销毁由 <b>CreateDvppApi</b> 接口创建的dvppapi实例，关闭DVPP执行器。		
	<b>DvppGetOutParameter</b>	获取JPEGD/JPEGE/PNGD模块的输出内存大小。		
实现VDEC功能	<b>CreateVdecApi</b>	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 <b>CreateVdecApi接口</b> 进行视频解码，可以跨函数调用，跨线程调用。	Vdec.h	



分类	接口	功能说明	定义接口的头文件	依赖的库文件
	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。		
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	
	SetVencParam	使用CreateVenc接口创建的实例后调用SetVencParam接口，设置VENC编码参数：如码控等。		
	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。		

## 2.3 VPC 功能

### 功能说明

VPC ( vision preprocessing core ) 功能包括：

- **抠图**，从输入图片中抠出需要用的图片区域，支持一图多框和多图多框。
- **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，用于处理“**widthStride**在32~4096（包括4096）范围内，**heightStride**在6~4096”的输入图片，不同格式的输入图片，widthStride的取值范围不同，详细描述参见表2-2，只支持通过硬件实现非8K缩放。
    - 8K缩放，用于处理“widthStride在4096~8192范围内或heightStride在4096~8192范围内（不包括4096）”的输入图片，支持通过软件实现8K缩放，也支持通过硬件实现8K缩放。

- 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（HiSilicon frame buffer compression），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，目前的输入图片格式、输出图片格式，请参见**VPC参数说明**。
- **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。

表 2-2 关于 VPC 输入/输出的约束

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
widthStride在4096~8192范围内或heightStride在4096~8192范围内（不包括4096）	YUV420 SP（NV12、NV21）	<ul style="list-style-type: none"><li>通过软件实现8K缩放：2*2对齐</li><li>通过硬件实现8K缩放：16*2对齐</li></ul>	8K缩放	<ul style="list-style-type: none"><li>通过软件实现8K缩放：16*16~4096*4096</li><li>通过硬件实现8K缩放：32*6~4096*4096，宽高缩放比例范围是[1/32, 16]</li></ul>	请参见表3-1处的outputFormat参数	<ul style="list-style-type: none"><li>通过软件实现8K缩放：2*2对齐</li><li>通过硬件实现8K缩放：16*2对齐</li></ul>

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
<ul style="list-style-type: none"><li>YUV400SP/ YUV420SP/ YUV422SP/ yuv444sp: width Stride 在 32~4096 (包括 4096) 范围内, heightStride 在 6~4096。</li><li>YUV422packed: width Stride /2在 32~4096 (包括 4096) 范围内, heightStride 在 6~4096。</li><li>YUV444pac</li></ul>	请参见表 3-1 处的 inputFormat 参数	<ul style="list-style-type: none"><li>宽 stride 的对齐: 请参见表 3-1 处的 width Stride 参数</li><li>高 stride 是 2 对齐</li></ul>	非 8K 缩放	32*6~4096*4096	请参见表 3-1 处的 outputFormat 参数	16*2 对齐

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
ked/ RGB888: width Stride/3在32~4096 (包括4096)范围内, heightStride在6~4096。 <ul style="list-style-type: none"><li>• XRGB8888: width Stride/4在32~4096 (包括4096)范围内, heightStride在6~4096。</li></ul>						

- 针对通过硬件实现的缩放功能，贴图/抠图的宽高缩放比例范围：[1/32, 16]。
- 对于软件8K缩放功能，在缩放的同时，可以与格式转换（支持YUV420SP NV12与YUV420SP NV21之间的格式转换）功能组合，但不支持与抠图功能组合；对于硬件8K缩放功能，在缩放的同时，支持与格式转换（支持YUV420SP NV12与YUV420SP NV21之间的格式转换）、抠图功能组合使用。

- VPC的输出作为模型推理的输入时，由于经过VPC处理的输出图片中的贴图区域的宽\*高有16\*2对齐的约束，因此输出图片中的贴图区域的宽、高有一些补边的无效数据，所以在VPC贴图前，应首先使用VPC的缩放功能将贴图区域的分辨率缩放成16\*2对齐，否则无效数据会影响模型推理的精度。
- 实现VPC功能时，各接口的参数约束（例如，输入图片格式、宽高的对齐要求）请参见[VPC参数说明](#)。

## 功能示意图

图 2-2 VPC 功能示意图（抠图+缩放+叠加）

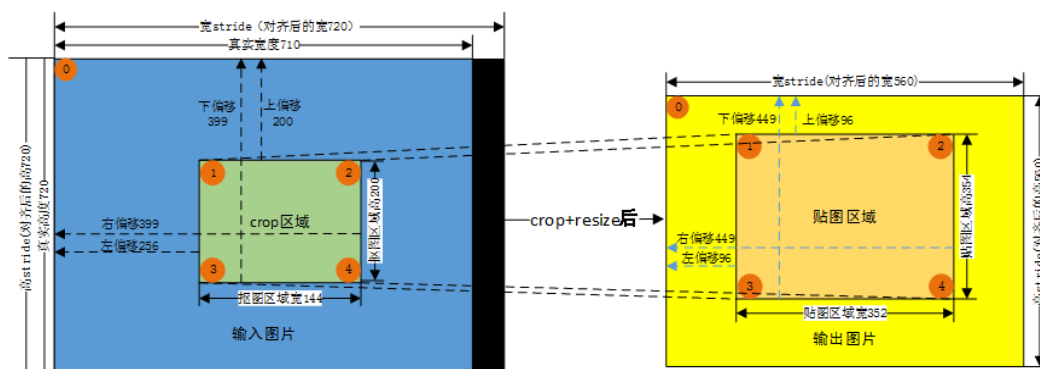


图 2-3 VPC 功能示意图（拼接）

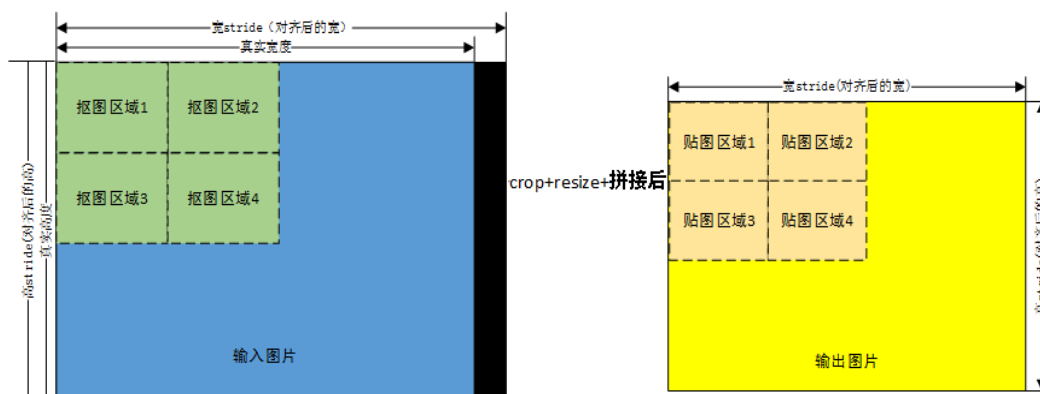


图 2-4 VPC 功能示意图（等比例缩放，即缩放前后图片的宽高比例相同）

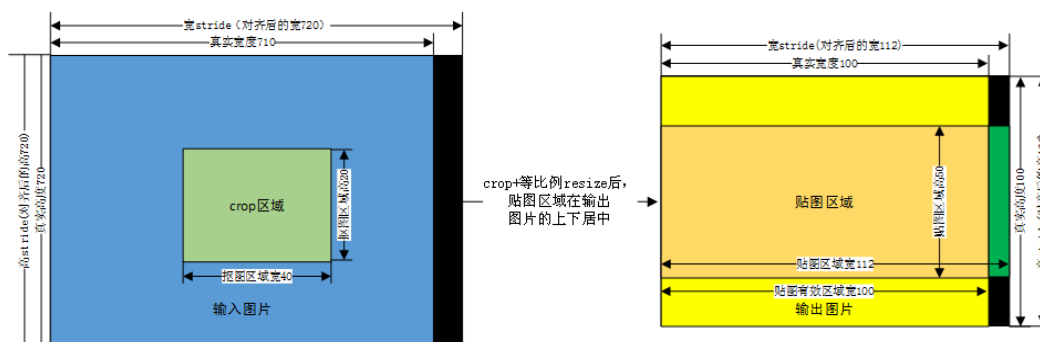


表 2-3 概念解释

概念	描述
宽stride ( widthStride )	<p>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）。</p> <ul style="list-style-type: none"> <li>YUV400SP、YUV420SP、YUV422SP、yuv444sp：输入图像的宽对齐到16。</li> <li>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）。</li> <li>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）。</li> <li>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）。</li> <li>HFBC格式：输入图像的宽。</li> </ul>
高stride ( heightStride )	<p>指图像在内存中的行数，表示输入图片对齐后的高。</p> <p>取值为：输入图像的高对齐到2。高stride最小为6，最大为4096。</p>
上/下/左/右偏移	<p>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。</p> <ul style="list-style-type: none"> <li>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。</li> <li>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。</li> <li>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。</li> <li>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。</li> </ul>
抠图区域	<p>指用户指定的需抠出的图片区域。抠图区奇数、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。</p> <p>抠图区域最小分辨率为10*6，最大分辨率为4096*4096。</p>

概念	描述
贴图区域	<p>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。</p> <p>约束如下：</p> <ul style="list-style-type: none"><li>• 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。</li><li>• 抠图区域不超出输入图片，贴图区域不超出输出图片。</li><li>• 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。</li><li>• 最大贴图个数为256个。</li><li>• 贴图区域相对输出图片的左偏移16对齐。</li><li>• 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。见图2-4，贴图区域旁边的绿色框就表示无效数据。</li><li>• 等比例缩放场景下，当输出图片格式为YUV420SP格式时，贴图左偏移满足16对齐、上偏移2对齐的要求即可。</li></ul>

性能指标说明

- 对于非8K缩放，单个Device的基本场景性能指标参考如下：

场景举例	总帧率
<ul style="list-style-type: none"><li>• 输入图像分辨率：1080p（1920*1080）</li><li>• 输出图像分辨率：1080p（1920*1080）</li><li>• 输入/输出图片格式：YUV420SP</li><li>• n路（n&lt;4，1路对应一个线程）</li></ul>	n*360fps
<ul style="list-style-type: none"><li>• 输入图像分辨率：1080p（1920*1080）</li><li>• 输出图像分辨率：1080p（1920*1080）</li><li>• 输入/输出图片格式：YUV420SP</li><li>• n路（n≥4，1路对应一个线程）</li></ul>	1440fps
<ul style="list-style-type: none"><li>• 输入图像分辨率：4K图像（3840*2160）</li><li>• 输出图像分辨率：4K图像（3840*2160）</li><li>• 输入/输出图片格式：YUV420SP</li><li>• n路（n&lt;4，1路对应一个线程）</li></ul>	n*90fps

场景举例	总帧率
<ul style="list-style-type: none"><li>输入图像分辨率: 4K图像 (3840*2160)</li><li>输出图像分辨率: 4K图像 (3840*2160)</li><li>输入/输出图片格式: YUV420SP</li><li>n路 (n≥4, 1路对应一个线程)</li></ul>	360fps

- 对于软件**8K缩放**, VPC性能与输出图像分辨率强相关, 输出图像分辨率越大, 处理耗时越久, 性能越低。单个Device的典型场景性能指标参考如下:

场景举例	总帧率
<ul style="list-style-type: none"><li>输入图像分辨率: 8K图像 (7680*4320)</li><li>输出图像分辨率: 1080p (1920*1080)</li><li>输入/输出图片格式: YUV420SP</li><li>n路 (n&lt;4, 1路对应一个线程)</li></ul>	n*4fps
<ul style="list-style-type: none"><li>输入图像分辨率: 8K图像 (7680*4320)</li><li>输出图像分辨率: 1080p (1920*1080)</li><li>输入/输出图片格式: YUV420SP</li><li>n路 (n≥4, 1路对应一个线程)</li></ul>	16fps
<ul style="list-style-type: none"><li>输入图像分辨率: 8K图像 (7680*4320)</li><li>输出图像分辨率: 4K图像 (3840*2160)</li><li>输入/输出图片格式: YUV420SP</li><li>n路 (n&lt;4, 1路对应一个线程)</li></ul>	n*1fps
<ul style="list-style-type: none"><li>输入图像分辨率: 8K图像 (7680*4320)</li><li>输出图像分辨率: 4K图像 (3840*2160)</li><li>输入/输出图片格式: YUV420SP</li><li>n路 (n≥4, 1路对应一个线程)</li></ul>	4fps

- 对于硬件**8K缩放**, 单个Device的基本场景性能指标参考如下:



场景举例	总帧率
<ul style="list-style-type: none"><li>• 输入图像分辨率: 8K图像 ( 7680*4320 )</li><li>• 输出图像分辨率: 1080p ( 1920*1080 )</li><li>• 输入/输出图片格式: YUV420SP</li><li>• n路 ( n&lt;4, 1路对应一个线程 )</li></ul>	n*25fps
<ul style="list-style-type: none"><li>• 输入图像分辨率: 8K图像 ( 7680*4320 )</li><li>• 输出图像分辨率: 1080p ( 1920*1080 )</li><li>• 输入/输出图片格式: YUV420SP</li><li>• n路 ( n≥4, 1路对应一个线程 )</li></ul>	100fps
<ul style="list-style-type: none"><li>• 输入图像分辨率: 8K图像 ( 7680*4320 )</li><li>• 输出图像分辨率: 4K图像 ( 3840*2160 )</li><li>• 输入/输出图片格式: YUV420SP</li><li>• n路 ( n&lt;4, 1路对应一个线程 )</li></ul>	n*25fps
<ul style="list-style-type: none"><li>• 输入图像分辨率: 8K图像 ( 7680*4320 )</li><li>• 输出图像分辨率: 4K图像 ( 3840*2160 )</li><li>• 输入/输出图片格式: YUV420SP</li><li>• n路 ( n≥4, 1路对应一个线程 )</li></ul>	100fps

## 参考说明

RGB、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

## 2.4 JPEGG 功能

### 功能及约束说明

JPEGG ( JPEG encoder ) 将YUV格式图片编码成.jpg图片。

- JPEGG支持以下格式输入：
  - YUV422 Packed ( yuyv,yvyu,uyvy,vyuy )
  - YUV420SP ( NV12, NV21 )
- JPEGG支持的分辨率为：  
最大分辨率：8192 \* 8192，最小分辨率：32 \* 32
- JPEGG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。
- 实现JPEGG功能时，各接口的参数约束（例如，输入图片格式、宽高的对齐要求）请参见[JPEGG参数说明](#)。

### 性能指标说明

1080p指分辨率为1920\*1080的图片；4K指分辨率为3840\*2160的图片。单个Device的基本场景性能指标参考如下：

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

## 2.5 JPEGD 功能

### 功能说明

JPEGD (JPEG decoder) 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。

如果输入图片的码流中包含Orientation信息（代表捕获图像时摄像机相对于场景的方向），则JPEGD在解码时会解析Orientation信息，将图片进行90度、180度、270度或镜像旋转。旋转后输出图片的宽stride、高stride、输出内存仍需满足[JPEGD参数说明](#)的要求。如果输入图片的码流异常，导致JPEGD解码时无法读取Orientation信息，则不能实现图片旋转的功能。

针对不同的源图片格式，解码后，输出如下格式的图片：

jpeg(444) -> YUV444SP V在前U在后、YUV444SP U在前V在后、YUV420 SP V在前U在后、YUV420SP U在前V在后。

jpeg(422) -> YUV422SP V在前U在后、YUV422SP U在前V在后、YUV420SP V在前U在后、YUV420SP U在前V在后。

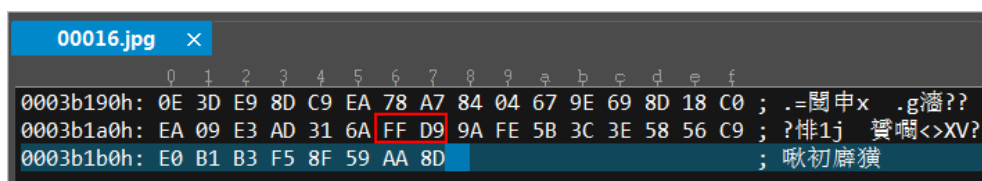
jpeg(420) -> YUV420SP V在前U在后、YUV420SP U在前V在后。

jpeg(400) -> YUV420SP V在前U在后、YUV420SP U在前V在后，UV数据采用0x80填充。

jpeg(440) -> YUV440SP V在前U在后、YUV440SP U在前V在后、YUV420 SP V在前U在后、YUV420SP U在前V在后。

**须知**

- 源图片格式解码是指解码前后的编码格式保持一致，例如解码前输入图片为 jpeg(440)，解码后输出图片为YUV440SP V在前U在后或YUV440SP U在前V在后。JPEGD解码后的输出图片，如果要直接作为模型推理的输入，这时JPEGD使用源图片格式解码（但这里要确保解码后的图片格式模型是支持的），保证模型推理的精度。  
JPEGD解码后的输出图片，如果直接作为VPC的输入，该场景下若使用源图片格式解码时，则需要关注解码后的输出图片格式VPC是否支持（VPC输入图片的格式请参见[VPC功能](#)），如果VPC不支持，则用户需按VPC支持的情况指定JPEGD的输出图片格式。
- 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，源图片格式为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。
  - JPEGD解码后的输出图片，如果要作为模型推理的输入，建议使用源格式解码，保证模型推理的精度。
  - JPEGD+VPC串联使用时，由于JPEGD解码后的输出图片的宽stride\*高stride有128\*16对齐的约束，因此解码后的输出图片的宽、高有一些补边的无效数据，所以VPC在处理图片前，应首先对JPEGD对齐后的输出图片进行原分辨率裁剪，目的是去除无效数据对图像精度的影响。
- 实现JPEGD功能时，各接口的参数约束请参见[JPEGD参数说明](#)。

## 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 15fps。JPEGD解码的输出图片如果涉及旋转，则性能指标低于软件解码的参考值，例如对于1080P的图片，性能指标低于15fps。

1080p指分辨率为1920\*1080的图片；4K指分辨率为3840\*2160的图片。单个Device的基本场景性能指标参考如下：

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

## 2.6 PNGD 功能

### 功能及约束说明

PNGD ( PNG decoder ) 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、GRAY、AGRAY
- PNGD输出格式为：  
RGBA、RGB  
如果输入格式为GRAY，则输出会转码为RGB；如果输入格式为AGRAY，则输出会转码为RGB或RGBA。
- PNGD支持的分辨率  
最大分辨率4096 \* 4096，最小分辨率32 \* 32。
- 实现PNGD功能时，各接口的参数约束请参见[PNGD参数说明](#)。

### 性能指标说明

1080p指分辨率为1920\*1080的图片；4K指分辨率为3840\*2160的图片。单个Device的基本场景性能指标参考如下：

场景举例	总帧率
1080p * n路 ( $1 \leq n \leq 5$ )	$n * 4\text{fps}$
1080p * n路 ( $n \geq 6$ )	24fps
4k * n路 ( $1 \leq n \leq 5$ )	$n * 1\text{fps}$
4k * n路 ( $n \geq 6$ )	6fps

## 2.7 VDEC 功能

### 功能及约束说明

VDEC ( video decoder ) 实现视频的解码。

- VDEC支持两种输入格式：  
H264 bp/mp/hp level5.1 YUV420编码的码流，当前只支持annex-B字节流的裸码流。  
H265 8/10bit level5.1 YUV420编码的码流，当前只支持annex-B字节流的裸码流。
- VDEC支持的输入码流分辨率  
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仅支持解码H264 8bit编码的码流。  
如果用户使用隔行扫描方式编码出来的码流进行VDEC解码，则解码的输出格式仅支持YUV420SP格式。
- 实现VDEC功能时，各接口的参数约束请参见[VDEC功能接口](#)。

### 性能指标说明

720p指分辨率为1280\*720的图片；1080p指分辨率为1920\*1080的图片；4K指分辨率为3840\*2160的图片。

场景举例	总帧率	每路VDEC解码的最大内存消耗 ( 此处以H265格式的输入码流、 参考帧数量9个为例，作为参考 值 )
小于等于720p * n路 ( $8 \leq n \leq 32$ )	1080fps	约86MB
小于等于720 * n路 ( $1 \leq n \leq 7$ )	$n * 120\text{fps}$	约86MB

场景举例	总帧率	每路VDEC解码的最大内存消耗 (此处以H265格式的输入码流、 参考帧数量9个为例, 作为参考 值)
1080p * n路 (4≤n≤32)	480fps	约118MB
1080p * n路 (1≤n≤3)	n*120fps	约118MB
4k * n路 (2≤n≤32)	120fps	约266MB
4k * 1路	60fps	约266MB

下表以1080P分辨率的输入码流为例, 说明每路VDEC解码的最大内存消耗的计算公式, 在计算公式中:

- 输入码流缓存大小: 1080P分辨率以下的输入码流, 输入码流缓存大小默认为6M; 1080P分辨率以上的输入码流, 输入码流缓存大小默认为9M。
- 解码图像帧存大小: 1080P分辨率的输入码流, 该参数值为3MB。其它分辨率时参数值可等量折算。
- 视频解码图像Tmv缓存大小: H264格式、1080P分辨率的输入码流, 该参数值为0.5MB; H265格式、1080P分辨率的输入码流, 该参数值为1MB。其它分辨率时参数值可等量折算。
- 参考帧数量: 以最大参考帧个数为例, 系统内部会根据实际码流的参考帧个数自适应。

场景举例	每路VDEC解码的最大内存消耗 (单位为MB)
输入码流格式H264 输入码流分辨率1080P 输入码流缓存大小为6M 解码图像帧存大小为3MB 视频解码图像Tmv缓存大小为0.5MB 参考帧9个	51MB 计算公式: 4MB + 输入码流缓存大小 * 2 + (解码图像帧存大小 + 视频解码图像Tmv缓存大小) * (参考帧数量 + 1)
输入码流格式H265 输入码流分辨率1080P 输入码流缓存大小为6M 解码图像帧存大小为3MB 视频解码图像Tmv缓存大小为1MB 参考帧9个	56MB 计算公式: 4MB + 输入码流缓存大小 * 2 + (解码图像帧存大小 + 视频解码图像Tmv缓存大小) * (参考帧数量 + 1)

## 关于解码路数与帧率的说明

注意：下表中给出的规格建议供参考，如果单进程内启动的路数超过下表中的建议，则可能出现内存不足或性能不够的问题，进而导致创建解码通道失败或执行解码缓慢。

720p指分辨率为1280\*720的图片；1080p指分辨率为1920\*1080的图片；4K指分辨率为3840\*2160的图片。

典型分辨率	单进程内各启动路数时的规格建议（依据输入帧率得出）				
-	输入帧率 ≥25fps	20fps<输入 帧率<25fps	15fps<输入 帧率≤20fps	10fps<输入 帧率≤15fps	输入帧率 ≤10fps
≤720p	32路	32路	32路	32路	32路
1080p	16路	19路	24路	32路	32路
4K	4路	4路	6路	8路	12路

## 2.8 VENC 功能

VENC（video encoder）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，不支持单进程多线程场景。

- VENC支持以下格式输入：  
YUV420SP 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）
- 实现VENC功能时，各接口的参数约束请参见[VENC功能接口](#)。

### 性能指标说明

1080p指分辨率为1920\*1080的图片。

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。



## 2.9 关于输入输出内存的说明

关于acldvppMalloc/acldvppFree接口的说明，请参见《[应用软件开发指南 \(C&C++, 开放态\)](#)》中的“AscendCL API参考”。

表 2-4 内存要求说明

功能模块	输入内存	输出内存
VPC	<p>请使用AscendCL提供的接口申请/释放内存：</p> <ul style="list-style-type: none"> <li>使用acldvppMalloc接口申请内存，内存满足DVPP的要求（内存起始地址16对齐）；</li> <li>使用acldvppFree接口释放内存。</li> </ul>	<p>请使用AscendCL提供的接口申请/释放内存：</p> <ul style="list-style-type: none"> <li>使用acldvppMalloc接口申请内存，内存满足DVPP的要求（内存起始地址16对齐）；</li> <li>使用acldvppFree接口释放内存。</li> </ul>
JPEGE和JPEG D	<p>请使用AscendCL提供的接口申请/释放内存：</p> <ul style="list-style-type: none"> <li>使用acldvppMalloc接口申请内存，内存满足DVPP的要求（内存起始地址128对齐）；</li> <li>使用acldvppFree接口释放内存。</li> </ul>	<ul style="list-style-type: none"> <li>由用户指定输出内存时，由用户自行释放内存。请使用AscendCL提供的接口申请/释放内存： <ul style="list-style-type: none"> <li>使用acldvppMalloc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<a href="#">DvppGetOutParameter</a>接口获取输出内存大小。</li> <li>使用acldvppFree接口释放内存。</li> </ul> </li> <li>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。</li> </ul>

功能模块	输入内存	输出内存
PNGD	<p>请使用AscendCL提供的接口申请/释放内存：</p> <ul style="list-style-type: none"> <li>使用acldvppMalloc接口申请内存，内存满足DVPP的要求（内存起始地址128对齐）；</li> <li>使用acldvppFree接口释放内存。</li> </ul>	<ul style="list-style-type: none"> <li>由用户指定输出内存时，由用户自行释放内存。请使用AscendCL提供的接口申请/释放内存： <ul style="list-style-type: none"> <li>使用acldvppMalloc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<a href="#">DvppGetOutParameter</a>接口获取输出内存大小。</li> <li>使用acldvppFree接口释放内存。</li> </ul> </li> <li>如果不由用户指定输出内存时，DVPP内部申请内存，需由用户调用PNGD出参结构体中的FreeOutputMemory()函数释放内存。</li> </ul>
VDEC和VENC	<p>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AscendCL提供的acldvppMalloc/acldvppFree接口申请/释放内存。</p>	<p>VDEC输出的HFBC格式数据直接作为VPC的输入。</p> <p>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。</p>

# 3 VPC/JPEGE/JPEGD/PNGD 功能接口

- [3.1 CreateDvppApi](#)
- [3.2 DvppCtl](#)
- [3.3 DestroyDvppApi](#)
- [3.4 DvppGetOutParameter](#)

## 3.1 CreateDvppApi

函数原型	int32_t 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 <a href="#">DvppCtl接口</a> 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	<ul style="list-style-type: none"><li>返回值“0”代表成功。</li><li>其它返回值的说明请参见<a href="#">返回码列表</a>。</li></ul>
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 <a href="#">DestroyDvppApi接口</a> 。

### 调用示例

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

## 3.2 DvppCtl

### 3.2.1 接口说明

函数原型	<code>int32_t DvppCtl(IDVPPAPI*&amp; pIDVPPAPI, int32_t CMD, dvppapi_ctl_msg* MSG)</code>
功能	使用 <a href="#">CreateDvppApi</a> 接口创建的实例来调用 <a href="#">DvppCtl</a> 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	<ul style="list-style-type: none"><li>“IDVPPAPI”类型指针引用</li><li><a href="#">CMD控制命令字</a></li><li><a href="#">dvppapi_ctl_msg</a>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。</li></ul>
输出说明	根据DVPP各模块的功能，出参也不一样，请参见 <a href="#">dvppapi_ctl_msg</a> 。
返回值说明	<ul style="list-style-type: none"><li>返回值“0”代表成功。</li><li>其它返回值的说明请参见<a href="#">返回码列表</a>。</li></ul>
使用说明	调用方调用“ <a href="#">DvppCtl</a> ”函数，传入“IDVPPAPI”对象指针、传入正确的 <a href="#">CMD控制命令字</a> 、配置好相应功能的 <a href="#">dvppapi_ctl_msg</a> 。
使用约束	无。

### CMD 控制命令字

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
DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM	预留命令字。	8

## dvppapi\_ctl\_msg

成员变量	说明	取值范围
int32_t in_size	入参大小。	in参数结构体的大小。 对于VPC一图多框的功能，in_size为in参数结构体的大小；对于VPC多图多框的功能，in_size为in参数结构体的倍数，倍数与输入源图的个数一致。 对于其它功能，in_size为in参数结构体的大小。
int32_t out_size	出参大小。	out参数结构体的大小。
void *in	入参。	<ul style="list-style-type: none"><li>• VPC功能入参：<a href="#">VpcUserImageConfigure</a></li><li>• JPEGE功能入参：<a href="#">sJpegIn</a></li><li>• JPEGD功能入参：<a href="#">JpegdIn</a>（Jpegd输入结构体的驼峰风格版本，推荐使用）</li><li>• JPEGD功能入参：<a href="#">jpegd_raw_data_info</a>（jpegd输入结构体的内核风格版本，不推荐使用）</li><li>• PNGD功能入参：<a href="#">PngInputInfoAPI</a></li><li>• 查询DVPP引擎入参：<a href="#">device_query_req_stru</a></li></ul>
void *out	出参。	<ul style="list-style-type: none"><li>• VPC功能出参：无</li><li>• JPEGE功能出参：<a href="#">sJpegOut</a></li><li>• JPEGD功能出参：<a href="#">JpegdOut</a>（Jpegd输出结构体的驼峰风格版本，推荐使用）</li><li>• JPEGD功能出参：<a href="#">jpegd_yuv_data_info</a>（jpegd输出结构体的内核风格版本，不推荐使用）</li><li>• PNGD功能出参：<a href="#">PngOutputInfoAPI</a></li><li>• 查询DVPP引擎出参：<a href="#">dvpp_engine_capability_stru</a></li></ul>

### 3.2.2 VPC 参数说明

#### 入参: VpcUserImageConfigure

表 3-1 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AscendCL提供的acldvppMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址16对齐）。  acldvppMalloc接口的说明，请参见《 <a href="#">应用软件开发指南 (C&amp;C++, 开放态)</a> 》中的“AscendCL API参考”。	-
uint32_t bareDataBufferSize	非压缩格式输入图片内存大小，单位为字节，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： <ul style="list-style-type: none"><li>• YUV400SP、YUV420SP：<math>\text{widthStride} \times \text{heightStride} \times 3 / 2</math></li><li>• YUV422SP：<math>\text{widthStride} \times \text{heightStride} \times 2</math></li><li>• YUV444SP：<math>\text{widthStride} \times \text{heightStride} \times 3</math></li><li>• YUV422packed：<math>\text{widthStride} \times \text{heightStride}</math></li><li>• YUV444packed、RGB888：<math>\text{widthStride} \times \text{heightStride}</math></li><li>• XRGB8888：<math>\text{widthStride} \times \text{heightStride}</math></li></ul>

成员变量	说明	取值范围
uint32_t widthStride	图像宽度方向的步长。	<p>宽stride最小为32，最大为4096 * 4（宽是4096的argb格式的图像，1个像素占用4个字节，一行像素就占用4096*4，即widthStride）。</p> <p>对于软件8K缩放，要求宽stride对齐到2；</p> <p>对于硬件8K缩放，要求宽stride对齐到16；</p> <p>对于非8K缩放，对于不同图像格式，widthStride的计算公式不同：</p> <ul style="list-style-type: none"><li>• YUV400SP、YUV420SP、YUV422SP、yuv444sp：输入图像的宽对齐到16。</li><li>• 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）。</li><li>• 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）。</li><li>• 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）。</li><li>• HFBC格式：输入图像的宽。</li></ul>
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。	<p>取值为：输入图像的高对齐到2。</p> <p>对于8K缩放，heightStride取值范围：(4096, 8192]。</p> <p>对于非8K缩放，heightStride取值范围：[6, 4096]。</p>

成员变量	说明	取值范围
enum VpclInputFormat inputFormat	输入图像格式。	<p>对于8K缩放，只支持 INPUT_YUV420_SEMI_PLANNER_UV和 INPUT_YUV420_SEMI_PLANNER_VU。</p> <pre>enum VpclInputFormat {     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</pre>



成员变量	说明	取值范围
		<p>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</p> <p>INPUT_YUV420_SEMI_PLANNER_UV_10BIT, // 18, 此格式仅在VDEC输出HFBC压缩格式数据时使用, 用户无需关注存放该图片的内存大小, 由VDEC内部管理这部分内存</p> <p>INPUT_YUV420_SEMI_PLANNER_VU_10BIT, // 19, 此格式仅在VDEC输出HFBC压缩格式数据时使用, 用户无需关注存放该图片的内存大小, 由VDEC内部管理这部分内存</p> <p>};</p>
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 • <a href="#">VpcUserRoiConfigure</a> 结构体。	-
bool isCompressData	是否是视频解码输出的HFBC压缩图片数据格式。	<p>取值范围:</p> <ul style="list-style-type: none"> <li>• true: 表示HFBC压缩图片数据格式</li> <li>• false: 表示非HFBC压缩图片数据格式</li> </ul> <p>单图裁剪缩放和一图多框裁剪缩放均支持HFBC压缩格式。</p>
VpcCompressDataConfigure compressDataConfigure	视频解码输出的HFBC压缩图片数据配置。详细请参见 • <a href="#">VpcCompressDataConfigure...</a> 。	-

成员变量	说明	取值范围
bool yuvSumEnable	是否需要计算yuvSum值（用于统计y、u、v三个分量的总和），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 • <a href="#">VpcUserYuvSum结构体</a> 。	-
VpcUserPerformanceTuningParameter tuningParameter	预留参数，用于参数调优，详细请参见 • <a href="#">VpcUserPerformanceTuningParameter结构体</a> 。	-
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。 <b>说明</b> <ul style="list-style-type: none"><li>参数集须在Device设备上。</li><li>请确保传入的文件路径是正确路径。</li></ul>	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。	-

成员变量	说明	取值范围
uint8_t reserve1	用于指定通过软件，还是硬件实现8K缩放功能。	取值范围： <ul style="list-style-type: none"><li>● 0：软件8K缩放</li><li>● 1：硬件8K缩放</li></ul>
uint16_t reserve2;	用于指定缩放算法。	取值范围： <ul style="list-style-type: none"><li>● 0：华为自研的高阶滤波算法</li><li>● 1：业界通用的Bilinear算法（与OpenCV的计算精度接近）</li><li>● 2：业界通用的Nearest neighbor 算法（与OpenCV的计算精度接近）</li><li>● 3：业界通用的Bilinear算法（与Tensorflow的计算精度接近）</li><li>● 4：业界通用的Nearest neighbor算法（与Tensorflow的计算精度接近）</li></ul>

## 出参说明

暂无出参。

### 3.2.3 JPEGE 参数说明

#### 入参: sJpegIn

表 3-2 入参 sJpegIn

成员变量	说明
eEncodeFormat format	<p>输入YUV数据的类型，支持YUV422 packed ( yuyv,yvyu,uyvy,vyuy ) 和 YUV420SP ( NV12, NV21 )</p> <pre>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;</pre>
unsigned char* buf	<p>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AscendCL提供的acldvppMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。acldvppMalloc接口的说明，请参见《<a href="#">应用软件开发指南 (C&amp;C++, 开放态)</a>》中的“AscendCL API参考”。</p> <p><b>须知</b> 使用acldvppMalloc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。</p>
uint32_t bufSize	输入buf长度，指宽高对齐后的数据长度，单位为字节。
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	输入图片宽度对齐后的数值，对齐到16，兼容对齐到16的倍数如128（对齐到128时，性能更优）。对于YUV422packed数据，stride应该为width的两倍对齐到16。

成员变量	说明
uint32_t heightAligned	输入图片高度对齐后的数值，取值： <ul style="list-style-type: none"> <li>配置为与输入图片的高度相同的数值；</li> <li>或配置为输入图片的高度向上对齐到16的数值（最小为32）。该取值的使用场景举例：JPEGD的输出图片直接作为JPEGE的输入（JPEGD输出图片高度是向上对齐到16的）</li> </ul>
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 3-3 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。 由用户指定内存时，使用AscendCL提供的acldvppMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。acldvppMalloc接口的说明，请参见《 <a href="#">应用软件开发指南（C&amp;C++，开放态）</a> 》中的“AscendCL API参考”。 <b>须知</b> 使用acldvppMalloc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	存放编码后的jpg图片数据的内存的大小，单位为字节。
JpegCalBackFree	释放输出内存的回调函数。 <ul style="list-style-type: none"> <li>由用户指定输出内存时，由用户自行释放内存。</li> <li>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<a href="#">实现JPEGE功能</a>。</li> </ul>

## 3.2.4 JPEGD 参数说明

### 入参: JpegdIn

表 3-4 入参 JpegdIn

成员变量	说明
unsigned char* jpegData	<p>输入jpg图片数据，起始地址128对齐，2M大页表方式申请。</p> <p>使用AscendCL提供的acldvppMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。acldvppMalloc接口的说明，请参见《<a href="#">应用软件开发指南 (C&amp;C++, 开放态)</a>》中的“AscendCL API参考”。</p> <p><b>须知</b> 使用acldvppMalloc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。</p>
uint32_t jpegDataSize	输入内存的长度，单位为字节。
bool isYUV420Need	<p>是否需要输出YUV420SP格式的数据。</p> <ul style="list-style-type: none"><li>• true：是，默认值为true；</li><li>• false：否，保持源格式输出。</li></ul> <p>JPEGD支持raw格式（包括YUV420SP、YUV422SP、YUV444SP、YUV440SP）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。</p>
bool isVBeforeU	该参数值表示v、u分量的顺序（true表示v在前u在后，false表示u在前v在后，默认值为true）。

## 出参: JpegdOut

表 3-5 出参 JpegdOut

成员变量	说明
unsigned char* yuvData	<p>输出yuv图片数据buf, 该图片的宽高为128*16对齐后的宽高。</p> <p>由用户指定内存时, 使用AscendCL提供的acldvppMalloc接口申请内存, 申请到的内存满足DVPP的要求(首地址128对齐)。acldvppMalloc接口的说明, 请参见《<a href="#">应用软件开发指南 (C&amp;C++, 开放态)</a>》中的“AscendCL API参考”。</p> <p><b>须知</b> 使用acldvppMalloc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。</p>
uint32_t yuvDataSize	输出yuv数据的长度, 单位为字节, 数据长度由对齐后的宽高计算, 也可以通过调用 <a href="#">DvppGetOutParameter接口</a> 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。
JpegCalBackFree cbFree	<p>释放输出内存的回调函数。</p> <ul style="list-style-type: none"><li>由用户指定输出内存时, 由用户自行释放内存。</li><li>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<a href="#">实现JPEGD功能</a>。</li></ul>
jpegd_color_space outFormat	<p>输出yuv数据格式:</p> <pre>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, };</pre>

## 入参: jpegd\_raw\_data\_info

表 3-6 入参 jpegd\_raw\_data\_info

成员变量	说明
unsigned char* jpeg_data	<p>输入jpg图片数据，起始地址128对齐，2M大页表方式申请。</p> <p>使用AscendCL提供的acldvppMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。acldvppMalloc接口的说明，请参见《<a href="#">应用软件开发指南 (C&amp;C++, 开放态)</a>》中的“AscendCL API参考”。</p> <p><b>须知</b> 使用acldvppMalloc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。</p>
uint32_t jpeg_data_size	输入内存的长度，单位为字节。
jpegd_raw_format in_format	<p>输入图片中yuv的采样格式，不需要填充，默认值即可。</p> <pre>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 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, };</pre>
bool IsYUV420Need	<p>是否需要输出YUV420SP格式的数据。</p> <ul style="list-style-type: none"> <li>• true: 是，默认值为true</li> <li>• false: 否，保持源格式输出。</li> </ul> <p>JPEGD支持raw格式（包括YUV420SP、YUV422SP、YUV444SP、YUV440SP）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。</p>



成员变量	说明
bool isVBeforeU	<p>该参数值表示v、u分量的顺序（true表示v在前u在后，false表示u在前v在后，默认值为true），当参数值为false时，仅在如下场景时生效：</p> <ul style="list-style-type: none"> <li>当输入图片格式为jpeg(420)时；</li> <li>当输入图片格式为jpeg(444)、jpeg(422)、jpeg(440)，且isYUV420Need配置为true时。</li> </ul>

## 出参：jpegd\_yuv\_data\_info

表 3-7 出参 jpegd\_yuv\_data\_info

成员变量	说明
unsigned char* yuv_data	<p>输出yuv图片数据buf，该图片的宽高为对齐后的宽高。</p> <p><b>须知</b> 该内存是由DVPP内部申请并管理，不能由用户指定。</p>
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，单位为字节。
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。
uint32_t img_height_aligned	输出图片的对齐后的高度，对齐到16。
JpegCalBackFree cbFree	<p>释放输出内存的回调函数。</p> <p>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<a href="#">实现JPEGD功能</a>。</p>

成员变量	说明
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 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, };

### 3.2.5 PNGD 参数说明

#### 入参：PngInputInfoAPI

表 3-8 入参 PngInputInfoAPI

成员变量	说明
void* inputData	输入图像数据的地址。 使用AscendCL提供的acldvppMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。 acldvppMalloc接口的说明，请参见《 <a href="#">应用软件开发指南（C&amp;C++，开放态）</a> 》中的“AscendCL API参考”。 <b>须知</b> 使用acldvppMalloc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，单位为字节，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。
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或AGRAY转换到RGB，0保留原格式。

## 出参：PngOutputInfoAPI

表 3-9 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	输出内存长度。当前版本不使用该参数。
void* address	输出内存地址。 由用户指定内存时，使用AscendCL提供的acldvppMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址128对齐）。acldvppMalloc接口的说明，请参见《 <a href="#">应用软件开发指南（C&amp;C++，开放态）</a> 》中的“AscendCL API参考”。 <b>须知</b> 如果用户指定输出内存，则由用户保证内存实际大小与输入参数size一致。
uint64_t size	输出内存大小，单位为字节。
int32_t format	输出图像格式： <ul style="list-style-type: none"><li>• 2表示RGB输出。</li><li>• 6表示RGBA输出。</li></ul>
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对齐。 若输出格式为RGB，则输出图片宽先128对齐后再*3，若输出格式为RGBA，则输出图片宽先128对齐后再*4。
uint32_t highAlign	高度对齐，目前为16对齐。
void FreeOutputMemory()	PngOutputInfoAPI结构体的成员函数，用于释放输出内存的。 <ul style="list-style-type: none"><li>• 如果由用户指定输出内存时，由用户自行释放内存，不需要调用该函数。</li><li>• 如果不由用户指定输出内存时，DVPP内部申请内存，需由用户调用FreeOutputMemory()函数释放内存，调用示例请参见<a href="#">实现PNGD功能</a>。</li></ul>

## 3.2.6 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 3-10 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 3-11 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 <a href="#">dvpp_engine_capability_s tru</a> 中的结构体。
struct dvpp_resolution_stru min_resolution;	最小分辨率。	详细见 <a href="#">dvpp_engine_capability_s tru</a> 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。	<i>enum</i> 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_struct engine_input_format_table[DVPP_VADIO_FORMAT_MAX];	引擎所支持的输入格式表。	详见 <a href="#">dvpp_engine_capability_struct</a> 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_t able[DVPP_VADIO_FOR MAT_MAX];	引擎所支持的输出格式表结构体。	详见 <a href="#">dvpp_engine_capability_s tru</a> 中的结构体。
uint32_t performance_mode_num ;	性能模式的数量。	固定为1。
struct dvpp_performance_unit_s tru performance_mode_tabl e[DVPP_PERFORMANCE_ MODE_MAX];	模块的性能结构体。	详见 <a href="#">dvpp_engine_capability_s tru</a> 中的结构体。
struct dvpp_pre_contraction_str u pre_contraction;	预缩小信息结构体。	详见 <a href="#">dvpp_engine_capability_s tru</a> 中的结构体。
struct dvpp_pos_scale_stru pos_scale;	后缩放信息结构体。	详见 <a href="#">dvpp_engine_capability_s tru</a> 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： <ul style="list-style-type: none"><li>• HFBC</li><li>• YUV或RGB</li></ul>	-
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。	详见 <a href="#">•dvpp_vpc_data_spec_str u...</a> 。

### 3.3 DestroyDvppApi

函数原型	int32_t DestroyDvppApi(IDVPPAPI*& pIDVPPAPI)
功能	销毁由 <a href="#">CreateDvppApi</a> 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	<ul style="list-style-type: none"><li>返回值“0”代表成功。</li><li>返回值“-1”代表失败。</li></ul>
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);
```

## 3.4 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	<ul style="list-style-type: none"><li>in指针，各模块对应的指针不同，请参见表3-12。</li><li>out指针，各模块对应的指针不同，请参见表3-12。</li><li>cmd控制命令字</li></ul>
输出说明	各模块输出内存的大小： <ul style="list-style-type: none"><li>JPEGE功能，取sjpegeOut结构体中的jpgSize参数值。</li><li>JPEGD功能，取JpegdOut结构体中的yuvDataSize参数值。</li><li>PNGD功能，取PngOutputInfoAPI结构体中的outputSize参数值。</li></ul>
返回值说明	<ul style="list-style-type: none"><li>返回值“0”代表成功。</li><li>其它返回值的说明请参见返回码列表。</li></ul>
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： <ul style="list-style-type: none"><li>JPEGD模块，请参见实现JPEGD功能。</li><li>JPEGE模块，请参见实现JPEGE功能。</li><li>PNGD模块，请参见实现PNGD功能。</li></ul>
使用限制	无。

表 3-12 入参说明

入参	说明	取值范围
void* in	输入结构体指针	<ul style="list-style-type: none"> <li>JPEGE功能入参: <a href="#">JpegelIn</a></li> <li>JPEGD功能入参: <a href="#">JpegdIn</a></li> <li>PNGD功能入参: <a href="#">PngInputInfoAPI</a></li> </ul> <p><b>须知</b> 该函数不支持JPEGD 的<a href="#">jpegd_raw_data_info</a>结构体。</p>
void* out	输出结构体指针	<ul style="list-style-type: none"> <li>JPEGE功能出参: <a href="#">JpegeOut</a></li> <li>JPEGD功能出参: <a href="#">JpegdOut</a></li> <li>PNGD功能出参: <a href="#">PngOutputInfoAPI</a></li> </ul> <p><b>须知</b> 该函数不支持JPEGD 的<a href="#">jpegd_yuv_data_info</a>结构体。</p>
int32_t cmd	获取输出参数控制命令字	<ul style="list-style-type: none"> <li>JPEGE功能的命令字: GET_JPEGE_OUT_PARAMETER</li> <li>JPEGD功能的命令字: GET_JPEGD_OUT_PARAMETER</li> <li>PNGD功能的命令字: GET_PNGD_OUT_PARAMETER</li> </ul>



# 4 VDEC 功能接口

- 4.1 总体说明
- 4.2 CreateVdecApi
- 4.3 VdecCtl
- 4.4 DestroyVdecApi

## 4.1 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。

## 4.2 CreateVdecApi

函数原型	int32_t CreateVdecApi(IDVPPAPI*& pIDVPPAPI, int32_t 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 <a href="#">CreateVdecApi接口</a> 进行视频解码，可以跨函数调用，跨线程调用。
输入说明	<ul style="list-style-type: none"><li>• IDVPPAPI类型指针引用，输入指针必须为NULL。</li><li>• 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。</li></ul>
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	<ul style="list-style-type: none"><li>• 返回值“0”代表成功。</li><li>• 其它返回值的说明请参见<a href="#">返回码列表</a>。</li></ul>

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。

## 4.3 VdecCtl

函数原型	int32_t VdecCtl(IDVPPAPI*& pIDVPPAPI, int32_t CMD, dvppapi_ctl_msg* MSG, int32_t 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	<ul style="list-style-type: none"><li>• IDVPPAPI类型指针引用。</li><li>• 控制命令字CMD(VDEC为DVPP_CTL_VDEC_PROC)。</li><li>• 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：VdecInMsg（驼峰风格，推荐）、入参：vdec_in_msg（内核风格，兼容旧版本）。</li><li>• 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。</li></ul>
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	<ul style="list-style-type: none"><li>• 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）。</li><li>• 其它返回值的说明请参见<a href="#">返回码列表</a>。</li></ul>
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：VdecInMsg（驼峰风格，推荐）

成员函数	说明
VdecInMsg()	构造函数。

成员函数	说明
~VdecInMsg()	析构函数。
void SetVideoFormat(enum VideoFormat videoFormat)	设置输入码流格式。 enum VideoFormat { H264 = 0, HEVC = 1 //H265 };
enum VideoFormat VideoFormat()	获取配置的输入码流格式，通常用户无需调用。
void SetImageFormat(enum ImageFormat imageFormat)	设置输出图像格式。 enum ImageFormat { YUV420SP_NV12 = 0, YUV420SP_NV21 = 1 };
enum ImageFormat ImageFormat()	获取配置的输出图像格式，通常用户无需调用。
void SetInBuffer(const void* inBuffer)	设置输入码流内存地址。
const void* InBuffer()	获取输入码流的内存地址，通常用户无需调用。
void SetInBufferSize(int32_t inBufferSize)	设置输入码流长度。
int32_t InBufferSize()	获取输入码流长度，通常用户无需调用。
void SetHiaiData(const void* hiaiData)	用于设置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。 <b>须知</b> VDEC内部仅调用第一次传给VDEC的hiaiData，若要想多次使用hiaiData传送不同对象，请用下面智能指针对象。
const void* HiaiData()	获取用户自定义数据hiaiData，通常用户无需调用。

成员函数	说明
void SetHiaiDataSp(const std::shared_ptr< <a href="#">HIAI_DATA_SP</a> > hiaiDataSp)	<p>用于设置调用方回调函数形参指针，如果不涉及帧序号的配置，可不调用该接口，默认为NULL。如果需要设置帧序号，使用方法如下。</p> <p>其中HIAI_DATA_SP为VDEC内部定义的父亲类，具体类HIAI_DATA_SP结构请见 <a href="#">HIAI_DATA_SP类</a>，用户可以继承衍生子类。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。</p> <p><b>使用注意点：</b></p> <ol style="list-style-type: none"> <li>1. 在用户自定义子类hiaiDataSp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hiaiDataSp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；</li> <li>2. 用户自定义子类hiaiDataSp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；</li> <li>3. 使用VDEC进行视频码流解码时，如果使用了hiaiDataSp对象，但视频码流中存在异常帧，VDEC会直接将异常帧的hiaiDataSp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。</li> <li>4. 使用VDEC进行视频码流解码时，如果使用了hiaiDataSp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。</li> <li>5. VDEC会将用户传入的hiaiDataSp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hiaiDataSp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hiaiDataSp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hiaiDataSp对象会在第101帧开始解码时被丢弃。</li> <li>6. hiaiDataSp和hiaiData只能选其一，如果使用hiaiDataSp，则必须与channelId参数配合使用。</li> <li>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。</li> </ol>
std::shared_ptr< <a href="#">HIAI_DATA_SP</a> > HiaiDataSp()	获取用户自定义数据hiaiDataSp，通常用户无需调用。
void SetChannelId(int32_t channelId)	设置解码通道号，取值范围[0,31]。
int32_t ChannelId()	获取解码通道号，通常用户无需调用。

成员函数	说明
void SetFrameReturn(FrameReturnFp frameReturnFp)	设置解码回调函数，正常解码或具体某一帧解码异常时均通过此接口返回帧的相关信息，具体包含哪些信息，请参见表6-1。 回调函数的原型： void FrameReturn(FrameData& frameData)
const FrameReturnFp FrameReturn()	获取解码回调函数指针，通常用户无需调用。
void SetErrReport(ErrReportFp FperrReportFp)	设置异常错误上报函数，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。 异常错误上报函数的原型如下： void ErrReport(VDECERR* vdecErr)
const ErrReportFp FpErrReport()	获取异常错误上报函数指针，通常用户无需调用。
void SetEos(bool isEOS)	设置码流结束符EOS（End Of Stream），标识本路解码是否结束，true表示结束，false表示未结束。 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。
bool IsEos()	获取码流结束符的值，通常用户无需调用。
void SetOneInOneOutMode(bool isOneInOneOutMode)	设置是否实时出帧（即发送一帧解码一帧，无需依赖后续帧的传入），取值范围： <ul style="list-style-type: none"> <li>false：默认出帧模式，由于解码过程中存在缓存帧，无法实时输出，因此DVPP需要在收到码流中的多帧数据后，才开始输出解码结果。</li> <li>true：快速出帧模式，DVPP获取码流中的一帧数据后，就开始实时输出解码结果。只支持简单参考关系的H264/H265标准码流（无长期参考帧，无B帧）。</li> </ul>
bool IsOneInOneOutMode()	获取实时出帧模式的值，通常用户无需调用。

成员函数	说明
<code>void SetChnlWidth(uint32_t chnlWidth);</code>	设置通道的宽，即输入码流的宽。当前版本VDEC内部会根据输入码流解析得到码流的宽。 <ul style="list-style-type: none"><li>若用户不调用本接口设置通道的宽，或调用SetChnlWidth接口和调用SetChnlHeight接口设置通道的宽、高后，宽*高&gt;1920*1088，则VDEC内部会按较大默认值来申请码流缓存占用的内存。</li><li>若用户调用SetChnlWidth接口、调用SetChnlHeight接口设置的通道宽、高后，宽*高&lt;=1920*1088，则VDEC内部会按较小默认值来申请码流缓存占用的内存，每路内存大约节省6M。</li></ul>
<code>uint32_t ChnlWidth();</code>	获取通道的宽。当前版本用户无需调用，VDEC内部会解析得到输入码流的宽。
<code>void SetChnlHeight(uint32_t chnlHeight);</code>	设置通道的高，即输入码流的高。当前版本VDEC内部会根据输入码流解析得到码流的高。 <ul style="list-style-type: none"><li>若用户不调用本接口设置通道的高，或调用SetChnlWidth接口和调用SetChnlHeight接口设置通道的宽、高后，宽*高&gt;1920*1088，则VDEC内部会按较大默认值来申请码流缓存占用的内存。</li><li>若用户调用SetChnlWidth接口、调用SetChnlHeight接口设置的通道宽、高后，宽*高&lt;=1920*1088，则VDEC内部会按较小默认值来申请码流缓存占用的内存，每路内存大约节省6M。</li></ul>
<code>uint32_t ChnlHeight();</code>	获取通道的高。当前版本用户无需调用，VDEC内部会解析得到输入码流的高。

## 入参：vdec\_in\_msg（内核风格，兼容旧版本）

表 4-1 入参 vdec\_in\_msg

成员变量	说明
<code>char video_format[10]</code>	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。
<code>char image_format[10]</code>	输出帧格式，“nv12”或者“nv21”，默认是“nv12”。

成员变量	说明
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• <a href="#">FRAME结构体</a> 。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，单位为字节。
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。  <b>须知</b> 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
<code>std::shared_ptr&lt;HIAI_DATA_SP&gt; hiai_data_sp</code>	<p>表示调用方回调函数形参指针，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。</p> <p>其中HIAI_DATA_SP为VDEC内部定义的父亲类，具体类HIAI_DATA_SP结构请见<a href="#">•HIAI_DATA_SP类</a>，用户可以继承衍生子类，使用方式可以参考<a href="#">实现VDEC功能</a>。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。</p> <p><b>使用注意点：</b></p> <ol style="list-style-type: none"><li>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；</li><li>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；</li><li>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。</li><li>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。</li><li>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。</li><li>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。</li><li>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。</li></ol>
<code>int32_t channelId</code>	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[0,31]。



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，以使用户决定如何处理，具体的使用方法请参见 <a href="#">实现VDEC功能</a> 。其中形参VDECERR为VDEC内部定义结构体，具体请见 <a href="#">VDECERR结构体</a> 。
bool isEOS	码流结束符EOS（End Of Stream），标识本路解码结束，true表示结束，false表示未结束。用户不用关心此标志，默认在 <a href="#">DestroyVdecApi</a> 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 <a href="#">实现VDEC功能</a> 。
int32_t isOneInOneOutMode	是否实时出帧（即发送一帧解码一帧，无需依赖后续帧的传入），取值范围： <ul style="list-style-type: none"><li>0：默认出帧模式，由于解码过程中存在缓存帧，无法实时输出，因此DVPP需要在收到码流中的多帧数据后，才开始输出解码结果。</li><li>1：快速出帧模式，DVPP获取码流中的一帧数据后，就开始实时输出解码结果。只支持简单参考关系的H264/H265标准码流（无长期参考帧，无B帧）。</li></ul>

## 4.4 DestroyVdecApi

函数原型	int32_t DestroyVdecApi(IDVPPAPI*& pIDVPPAPI, int32_t singleton)
功能	释放由 <a href="#">CreateVdecApi</a> 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	<ul style="list-style-type: none"><li>返回值“0”代表成功。</li><li>其它返回值的说明请参见<a href="#">返回码列表</a>。</li></ul>
使用说明	无特殊说明。
使用约束	一旦调用 <a href="#">DestroyVdecApi</a> 接口，如果还想在继续调用VDEC，需要重新创建vdecapi实例。

# 5 VENC 功能接口

- 5.1 总体说明
- 5.2 CreateVenc
- 5.3 SetVencParam
- 5.4 RunVenc
- 5.5 DestroyVenc

## 5.1 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。

## 5.2 CreateVenc

函数原型	int32_t CreateVenc(struct <b>VencConfig</b> * 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 <b>RunVenc</b> 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： <a href="#">入参：VencConfig</a> 。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	<ul style="list-style-type: none"><li>返回值非负数代表创建VENC编码实例成功</li><li>其它返回值的说明请参见<a href="#">返回码列表</a>。</li></ul>
使用说明	调用方创建VencConfig对象指针，调用 <b>CreateVenc</b> 接口将VencConfig对象指针传入。如果申请成功， <b>CreateVenc</b> 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 <a href="#">DestroyVenc接口</a> 。
------	---

## 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 5-1 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 <ul style="list-style-type: none"><li>0: H265 main level ( 仅支持Slice码流 )。</li><li>1: H264 baseline level。</li><li>2: H264 main level。</li><li>3: H264 high level。</li></ul>
uint32_t yuvStoreType	YUV图像存储格式。	0或者1 <ul style="list-style-type: none"><li>0: YUV420 semi-planner ( nv12 )</li><li>1: YVU420 semi-planner ( nv21 )</li></ul>
uint32_t keyFrameInterval	I帧间隔	取值范围：(0,65536)
VencOutMsgCallBack vencOutMsgCallBack	回调函数，用于处理编码结果。回调函数的格式为： <pre>typedef void (*VencOutMsgCallBack) (struct VencOutMsg* vencOutMsg, void* userData);</pre> 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	不可为空

成员变量	说明	取值范围
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

## 5.3 SetVencParam

函数原型	<b>int32_t SetVencParam(int32_t vencHandle, uint32_t paramType, uint32_t length, const void* param)</b>
功能	使用 <a href="#">CreateVenc</a> 接口创建的实例调用SetVencParam接口，设置VENC编码参数：如码率控制等。
输入说明	<ul style="list-style-type: none"><li>• vencHandle：句柄号，为CreateVenc接口的返回值。</li><li>• paramType：可设置的参数类型，取值范围如下：<ul style="list-style-type: none"><li>- 0：表示RC_MODE，码率控制模式</li><li>- 1：表示MAX_BITRATE，输出码率</li><li>- 2：表示SRC_FRAME_RATE，输入码流帧率</li><li>- 3：表示MAX_IP_PROP，一个GOP内单个I帧bit数和单个P帧bit数的比例</li></ul></li><li>• length：参数长度，void* param指针数据的长度</li><li>• param：参数指针，数据长度为length<ul style="list-style-type: none"><li>- 当paramType设置为0时，需在param参数处设置具体的码流控制模式，1表示变码率VBR模式，2表示定码率CBR，数据类型为uint32_t。</li><li>- 当paramType设置为1时，需在param参数处设置具体的输出码率，单位kbps，取值范围[10,30000]，数据类型为uint32_t。</li><li>- 当paramType设置为2时，需在param参数处设置具体的输入码流帧率，单位fps，取值范围[1, 120]，数据类型为uint32_t。注：如果该值和实际输入码流帧率相差太大，会影响输出码率。</li><li>- 当paramType设置为3时，需在param参数处设置具体的IP比例系数，取值范围[1,100]，数据类型为uint32_t。注：VBR模式下此值默认为80，CBR模式下此值默认为70。</li></ul></li></ul>
输出说明	无输出。
返回值说明	<ul style="list-style-type: none"><li>• 返回值“0”代表成功。</li><li>• 其它返回值的说明请参见<a href="#">返回码列表</a>。</li></ul>
使用说明	调用方先用CreateVenc创建编码通道，然后多次调用该接口设置RC_MODE、MAX_BITRATE、SRC_FRAME_RATE、MAX_IP_PROP码控参数。
使用约束	用户在调用RunVenc前调用该接口进行设置码控参数。

## 5.4 RunVenc

函数原型	<code>int32_t RunVenc(int32_t vencHandle, struct <a href="#">VencInMsg</a>* vencInMsg)</code>
功能	使用 <a href="#">CreateVenc</a> 接口创建的实例调用 <a href="#">RunVenc</a> 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： <a href="#">入参：VencInMsg</a> 。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	<ul style="list-style-type: none"><li>返回值“0”代表成功。</li><li>其它返回值的说明请参见<a href="#">返回码列表</a>。</li></ul>
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 <a href="#">CreateVenc接口</a> 创建实例后，必须使用同一个实例调用 <a href="#">RunVenc</a> 接口进行视频编码，最后再调用一次 <a href="#">DestroyVenc</a> 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 5-2 入参 VencInMsg

成员变量	说明	取值范围
<code>void* inputData</code>	输入数据地址。	非空
<code>uint32_t inputDataSize</code>	输入数据大小，单位为字节。	不能大于输入数据buffer大小，推荐值和输入数据buffer大小一样
<code>uint32_t keyFrameInterval</code>	I帧间隔。	大于等于0小于65535，0表示这个参数不起作用
<code>uint32_t forcIframe</code>	强制重新开始I帧间隔： <ul style="list-style-type: none"><li>0：不强制；</li><li>1：强制重新开始I帧</li></ul>	0或者1
<code>uint32_t eos</code>	是否为结束帧： <ul style="list-style-type: none"><li>0：不是；</li><li>1：是结束帧</li></ul>	0或者1

## 出参: VencOutMsg

表 5-3 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
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

## 5.5 DestroyVenc

函数原型	int32_t DestroyVenc(int32_t vencHandle)
功能	释放由 <a href="#">CreateVenc</a> 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	<ul style="list-style-type: none"><li>返回值“0”代表成功。</li><li>其它返回值的说明请参见<a href="#">返回码列表</a>。</li></ul>
使用说明	无特殊说明。
使用约束	一旦调用 <a href="#">DestroyVenc</a> 接口，如果还想在继续调用VENC，需要重新创建VENC编码实例。

# 6数据类型

- 6.1 VpcUserImageConfigure中的结构体
- 6.2 VdeclnMsg中的类
- 6.3 vdec\_in\_msg中的结构体和类
- 6.4 dvpp\_engine\_capability\_stru中的结构体

## 6.1 VpcUserImageConfigure 中的结构体

- VpcUserRoiConfigure结构体

成员变量	说明
VpcUserRoiInputConfigure inputConfigure	用户ROI输入配置，详细见 •VpcUserRoiInputConfigur...。 若实现软件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。
uint32_t lumaHeadStride	y分量头stride，与•FRAME结构体中的stride_head参数值保持一致。

成员变量	说明
uint32_t chromaHeadStride	uv分量头stride, 与 <a href="#">FRAME结构体</a> 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 <a href="#">FRAME结构体</a> 中的stride_payload参数值保持一致。
uint32_t chromaPayloadStride	uv分量数据的stride, 与 <a href="#">FRAME结构体</a> 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 <a href="#">VpcUserCropConfigure 结构...</a>
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AscendCL提供的acldvppMalloc接口申请内存，申请到的内存满足DVPP的要求（首地址16对齐）。 acldvppMalloc接口的说明，请参见《应用软件开发指南 (C&C++, 开放态)》中的“AscendCL API参考”。
uint32_t bufferSize	输出buffer的大小，根据yuv420sp计算，单位为字节。
uint32_t widthStride	输出图片的宽步长，需要16对齐。若实现非8K缩放或硬件8K缩放功能时，宽stride最小为32，最大为4096；若实现软件8K缩放功能时，宽stride最小为16，最大为4096。
uint32_t heightStride	输出图片的高步长，需要2对齐。若实现非8K缩放或硬件8K缩放功能时，高stride最小为6，最大为4096；若实现软件8K缩放功能时，高stride最小为16，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 <a href="#">•VpcUserCropConfigure 结构...</a> 。 若实现软件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2-3](#)。

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。

## 6.2 VdecInMsg 中的类

表 6-1 FrameData 类

成员函数	说明
FrameData()	构造函数。
~FrameData()	析构函数。
void SetChanId(int32_t channelId)	设置通道号，通常用户无需调用，VDEC内部调用该接口，设置的通道号对应 <a href="#">入参：VdecInMsg（驼峰风格，推荐）</a> 中用户通过SetChannelId接口设置的通道号。
int32_t ChanId()	获取通道号，对应 <a href="#">入参：VdecInMsg（驼峰风格，推荐）</a> 中用户通过SetChannelId接口设置的通道号。
void SetFrameId(uint64_t frameId)	设置解码返回的帧序号，通常用户无需调用。
uint64_t FrameId()	获取解码返回的帧序号，若用户使用了hiai_data_sp设置了帧序号，则此处值即为hiai_data_sp中的帧序号，否则为VDEC内部分配的帧序号。
void SetStatus(enum Status status)	设置解码返回帧的状态，通常用户无需调用。 <pre>enum Status {     FAIL = -1, // decode one frame unsuccessfully     SUCC = 0, // decode one frame successfully     NOPIC = 1 //decode no pic out(Interlaced scanning decode) };</pre>
enum Status Status()	获取解码返回帧的状态，取值SUCC或FAIL，用户自定义的回调函数中，需首先检查返回帧的状态是否为FAIL，若是则应该不继续回调函数中的后续流程，否则可能出现未定义的行为。FAIL表示本帧解码失败，因此 <a href="#">表6-1</a> 中设置的其它参数无效。
void SetAlignedWidth(int32_t alignedWidth)	设置解码返回帧对齐后的宽，通常用户无需调用。
int32_t AlignedWidth()	获取解码返回帧对齐后的宽，H264码流的解码返回帧按照16对齐，H265码流的解码返回帧按照64对齐。 例如，码流实际宽高为800*800，H264码流的解码返回帧对齐后的宽为800，则H265码流的解码返回帧对齐后的宽为832。 此参数一般场景使用不到，用户通常使用下面的RealWidth接口获取码流的真实宽。

成员函数	说明
void SetAlignedHeight(int32_t alignedHeight)	设置解码返回帧对齐后的高，通常用户无需调用。
int32_t AlignedHeight()	<p>获取解码返回帧对齐后的高，H264码流的解码返回帧按照16对齐，H265码流的解码返回帧按照64对齐。</p> <p>例如，码流实际宽高为800*800，H264码流的解码返回帧对齐后的高为800，则H265码流的解码返回帧对齐后的高为832。</p> <p>此参数一般场景使用不到，用户通常使用下面的RealHeight接口获取真实宽。</p>
void SetRealWidth(int32_t realWidth)	设置解码返回帧的真实宽，通常用户无需调用。
int32_t RealWidth()	获取解码返回帧的真实宽，如码流实际宽高为800*800，则此值为800，建议用户在回调函数中后续流程使用此值。
void SetRealHeight(int32_t realHeight)	设置解码返回帧的真实高，通常用户无需调用。
int32_t RealHeight()	获取解码返回帧的真实高，如码流实际宽高为800*800，则此值为800，建议用户在回调函数中后续流程使用此值。
void SetOutBuffer(const uint8_t* outBuffer)	设置解码返回帧的输出内存地址，通常用户无需调用。
const uint8_t* OutBuffer()	<p>获取解码返回帧的输出内存地址。</p> <p>隔行码流场景下，隔行码流每帧发送两场，解码时其中一场无图像输出，属于正常现象，会返回ERR_DECODE_NOPIC = 0x20000错误码；隔行码流的解码输出数据都在奇数场对应的输出buffer中。</p>
void SetOutBufferSize(int32_t outBufferSize)	设置解码返回帧的输出数据长度，通常用户无需调用。
int32_t OutBufferSize()	获取解码返回帧的输出数据长度。
void SetOffsetPayloadY(uint32_t offsetPayloadY)	设置HFBC payload数据Y方向数据起始地址偏移量，通常用户无需调用。
uint32_t OffsetPayloadY()	获取HFBC payload数据Y方向数据起始地址偏移量，相对于输出内存地址而言。
void SetOffsetPayloadC(uint32_t offsetPayloadC)	设置HFBC payload数据UV方向数据起始地址偏移量，通常用户无需调用。

成员函数	说明
uint32_t OffsetPayloadC()	获取HFBC payload数据UV方向数据起始地址偏移量，相对于输出内存地址而言。
void SetOffsetHeadY(uint32_t offsetHeadY)	设置HFBC head数据Y方向数据起始地址偏移量，通常用户无需调用。
uint32_t OffsetHeadY()	获取HFBC head数据Y方向数据起始地址偏移量，相对于输出内存地址而言。
void SetOffsetHeadC(uint32_t offsetHeadC)	设置HFBC head数据UV方向数据起始地址偏移量，通常用户无需调用。
uint32_t OffsetHeadC()	获取HFBC head数据UV方向数据起始地址偏移量，相对于输出内存地址而言。
void SetStridePayload(uint32_t stridePayload)	设置HFBC payload数据的对齐后的数值，通常用户无需调用。
uint32_t StridePayload()	获取HFBC payload数据的对齐后的数值。
void SetStrideHead(uint32_t strideHead)	设置HFBC head数据的对齐后的数值，通常用户无需调用。
uint32_t StrideHead()	获取HFBC head数据的对齐后的数值。
void SetBitdepth(uint32_t bitdepth)	设置解码返回帧的位深，通常用户无需调用。
uint32_t Bitdepth()	获取解码返回帧的位深，取值为8或10。
void SetVideoFormat(enum VideoFormat videoFormat)	设置本路解码的输入码流格式，通常用户无需调用，VDEC内部调用该接口，设置的输入码流格式对应 <a href="#">入参：VdecInMsg（驼峰风格，推荐）</a> 中用户通过SetVideoFormat接口设置的输入码流格式。
uint32_t VideoFormat()	获取本路解码的输入码流格式，对应 <a href="#">入参：VdecInMsg（驼峰风格，推荐）</a> 中用户通过SetVideoFormat接口设置的输入码流格式。 <pre>enum VideoFormat {     H264 = 0,     HEVC = 1 //H265 };</pre>
void SetImageFormat(enum ImageFormat imageFormat)	设置本路解码的输出图像格式，通常用户无需调用，VDEC内部调用该接口，设置的输出图像格式对应 <a href="#">入参：VdecInMsg（驼峰风格，推荐）</a> 中用户通过SetImageFormat接口设置的输出图像格式。。

成员函数	说明
uint32_t ImageFormat()	获取本路解码的输出图像格式，对应 <b>入参：VdecInMsg（驼峰风格，推荐）</b> 中用户通过SetImageFormat接口设置的输出图像格式。 <pre>enum ImageFormat {     YUV420SP_NV12 = 0,     YUV420SP_NV21 = 1 };</pre>
void SetHiaiData(const void* hiaiData)	设置用户自定义数据，通常用户无需调用，VDEC内部调用该接口，设置的自定义数据对应 <b>入参：VdecInMsg（驼峰风格，推荐）</b> 中用户通过SetHiaiDataSp或SetHiaiData接口设置的自定义数据。
const void* HiaiData()	获取用户自定义数据，对应 <b>入参：VdecInMsg（驼峰风格，推荐）</b> 中用户通过SetHiaiDataSp或SetHiaiData接口设置的自定义数据。
void SetCompressStatus(bool isCompressData)	用于设置输出图片是否为HFBC压缩格式，用户无需调用，VDEC内部调用该接口： <ul style="list-style-type: none"><li>• true：表示HFBC压缩格式</li><li>• false：表示非HFBC压缩格式，仅是YUV420SP格式</li></ul> <b>说明</b> 如果用户使用隔行扫描方式编码出来的码流进行VDEC解码，则解码的输出格式仅支持YUV420SP格式。
bool IsCompressData()	用户通过该接口获取输出图片是否为HFBC压缩格式： <ul style="list-style-type: none"><li>• true：表示HFBC压缩格式</li><li>• false：表示非HFBC压缩格式，仅是YUV420SP格式</li></ul> <b>说明</b> 在调用VPC功能前，用户需调用该接口判断VDEC解码的输出格式，便于用户在编写代码逻辑时，针对不同的输出格式进行处理。
void SetErrType(enum ERRTYPE errType);	VDEC内部调用该接口设置错误码，用户无需调用。
enum ERRTYPE ErrType();	用户通过该接口获取解码返回的错误码。

## 6.3 vdec\_in\_msg 中的结构体和类

- 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）。

成员变量	说明
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的对齐后的数值。
unsigned int stride_head;	输出图像head的对齐后的数值。
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。
void setFrameBuffer(void * frameBuff)	设置frameBuffer地址。
void * getFrameBuffer()	获取frameBuffer地址。

成员变量或函数	说明
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	<p>错误类型。</p> <pre>enum ERRTYPE{ //成功 ERR_NONE = 0x0 //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, //DVPP内部申请内存失败，用户需要检查系统是否有可用 内存，若忽略错误，则可能导致用户持续送入码流却无任 何解码结果输出 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（用户需要检查系统 是否有可用内存）等异常，若忽略错误，则可能导致用户 持续送入码流却无任何解码结果输出 ERR_ALLOC_DYNAMIC_MEM_FAIL, //系统内部申请VDEC的输入、输出buffer异常，用户需要 检查系统是否有可用内存，若忽略错误，则可能导致用户 持续送入码流却无任何解码结果输出 ERR_ALLOC_IN_OR_OUT_PORT_MEM_FAIL, //码流错误（如语法解析失败、重发eos或首帧即发送 eos），用户需检查输入的视频流数据是否正确。 ERR_BITSTREAM, //输入视频格式错误，用户需检查video_format参数（表 示输入视频的格式）是否为h264或h265 ERR_VIDEO_FORMAT, //输出格式配置错误，用户需检查image_format参数（表 示输出图像的格式）是否为nv12或nv21 ERR_IMAGE_FORMAT, //回调函数为空错误，用户需检查配置的回调函数是否为 空 ERR_CALLBACK,</pre>



成员变量	说明
	<p>//输入buffer为空错误, 用户需检查in_buffer参数 (表示输入buffer) 是否为空 ERR_INPUT_BUFFER,</p> <p>//输入buffer大小&lt;=0错误, 用户需检查in_buffer_size参数 (表示输入buffer大小) 是否小于等于0 ERR_INBUF_SIZE,</p> <p>//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, 用户需要检查系统中资源 (例如: 线程、内存等) 是否可用 ERR_THREAD_CREATE_FBD_FAIL,</p> <p>//创建解码实例失败, 用户需要重新创建解码实例 ERR_CREATE_INSTANCE_FAIL,</p> <p>//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,</p> <p>//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,</p> <p>//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,</p> <p>//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,</p> <p>//其它错误 ERR_OTHER</p> <p>//隔行码流场景下使用, 隔行码流每帧发送两场, 解码时其中一场无图像输出, 属于正常现象, 会返回该错误码; 隔行码流的解码输出数据都在奇数场对应的输出buffer中。 ERR_DECODE_NOPIC = 0x20000 };</p>
unsigned short channelId	解码错误的通道。

## 6.4 dvpp\_engine\_capability\_stru 中的结构体

- 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
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	<pre>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_8BI T_LIN, DVPP_COLOR_YVU420_SP_8BI T_LIN, DVPP_COLOR_YUV420_SP_8BI T_HFBC, DVPP_COLOR_YVU420_SP_8BI T_HFBC, DVPP_COLOR_YUV420_SP_10 BIT_HFBC, DVPP_COLOR_YVU420_SP_10 BIT_HFBC, DVPP_COLOR_YUV420_P_8BIT _LIN, </pre>

[illegible]

成员变量	说明	取值范围
		G_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BAG R_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GRA G_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GRB A_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GAB R_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GAR B_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GBR A_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GBA R_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\_performance\_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_P RE_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： <ul style="list-style-type: none"><li>• 0: 表示HFBC类型</li><li>• 1: 表示YUV或RGB类型</li></ul>
struct dvpp_resolution_stru min_resolution;	最小分辨率。	请参见 <a href="#">struct dvpp_resolution_...</a>
struct dvpp_resolution_stru max_resolution;	最大分辨率	请参见 <a href="#">struct dvpp_resolution_...</a>
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 返回码列表

- 7.1 公共返回码
- 7.2 VPC返回码
- 7.3 JPEGD返回码
- 7.4 JPEGG返回码
- 7.5 VDEC返回码
- 7.6 VENC返回码
- 7.7 PNGD返回码

## 7.1 公共返回码

错误代码	描述
0xA0028001	无效的Device ID。
0xA0028002	无效的channel ID。
0xA0028003	参数不合法，例如不合法的枚举值。
0xA0028004	资源已存在。
0xA0028005	资源不存在。
0xA0028006	函数参数中有空指针。
0xA0028007	使能系统、Device或通道前未配置对应的参数。
0xA0028008	不支持的参数或者功能。
0xA0028009	该操作不允许，如试图修改静态配置参数。
0xA002800C	分配内存失败，如系统内存不足。
0xA002800D	分配缓存失败，如申请的数据缓冲区太大。
0xA002800E	缓冲区中无数据。

错误代码	描述
0xA002800F	缓冲区中数据满。
0xA0028010	系统没有初始化或者相关依赖的模块没有加载。
0xA0028011	地址错误。
0xA0028012	系统忙。
0xA0028013	缓存小于实际需要的大小。
0xA0028014	硬件或软件处理超时。
0xA0028015	内部系统错误。
0xA002803F	最大的返回码，该模块的错误码必须小于该值。

## 7.2 VPC 返回码

错误代码	描述
0xA0078001	无效的Device ID。
0xA0078002	无效的channel ID。
0xA0078003	参数不合法，例如不合法的枚举值。
0xA0078004	资源已存在。
0xA0078005	资源不存在。
0xA0078006	函数参数中有空指针。
0xA0078007	使能系统、Device或通道前未配置对应的参数。
0xA0078008	不支持的参数或者功能。
0xA0078009	该操作不允许，如试图修改静态配置参数。
0xA007800C	分配内存失败，如系统内存不足。
0xA007800D	分配缓存失败，如申请的数据缓冲区太大。
0xA007800E	缓冲区中无数据。
0xA007800F	缓冲区中数据满。
0xA0078010	系统没有初始化或者相关依赖的模块没有加载。
0xA0078011	地址错误。
0xA0078012	系统忙。
0xA0078013	缓存小于实际需要的大小。
0xA0078014	硬件或软件处理超时。

错误代码	描述
0xA0078015	内部系统错误。
0xA007803F	最大的返回码，该模块的错误码必须小于该值。

## 7.3 JPEGD 返回码

错误代码	描述
0xA00E8001	无效的Device ID。
0xA00E8002	无效的channel ID。
0xA00E8003	参数不合法，例如不合法的枚举值。
0xA00E8004	资源已存在。
0xA00E8005	资源不存在。
0xA00E8006	函数参数中有空指针。
0xA00E8007	使能系统、Device或通道前未配置对应的参数。
0xA00E8008	不支持的参数或者功能。
0xA00E8009	该操作不允许，如试图修改静态配置参数。
0xA00E800C	分配内存失败，如系统内存不足。
0xA00E800D	分配缓存失败，如申请的数据缓冲区太大。
0xA00E800E	缓冲区中无数据。
0xA00E800F	缓冲区中数据满。
0xA00E8010	系统没有初始化或者相关依赖的模块没有加载。
0xA00E8011	地址错误。
0xA00E8012	系统忙。
0xA00E8013	缓存小于实际需要的大小。
0xA00E8014	硬件或软件处理超时。
0xA00E8015	内部系统错误。
0xA00E803F	最大的返回码，该模块的错误码必须小于该值。

## 7.4 JPEG 返回码

错误代码	描述
0xA00B8001	无效的Device ID。
0xA00B8002	无效的channel ID。
0xA00B8003	参数不合法，例如不合法的枚举值。
0xA00B8004	资源已存在。
0xA00B8005	资源不存在。
0xA00B8006	函数参数中有空指针。
0xA00B8007	使能系统、Device或通道前未配置对应的参数。
0xA00B8008	不支持的参数或者功能。
0xA00B8009	该操作不允许，如试图修改静态配置参数。
0xA00B800C	分配内存失败，如系统内存不足。
0xA00B800D	分配缓存失败，如申请的数据缓冲区太大。
0xA00B800E	缓冲区中无数据。
0xA00B800F	缓冲区中数据满。
0xA00B8010	系统没有初始化或者相关依赖的模块没有加载。
0xA00B8011	地址错误。
0xA00B8012	系统忙。
0xA00B8013	缓存小于实际需要的大小。
0xA00B8014	硬件或软件处理超时。
0xA00B8015	内部系统错误。
0xA00B803F	最大的返回码，该模块的错误码必须小于该值。

## 7.5 VDEC 返回码

错误代码	描述
0xA0058001	无效的Device ID。
0xA0058002	无效的channel ID。
0xA0058003	参数不合法，例如不合法的枚举值。
0xA0058004	资源已存在。

错误代码	描述
0xA0058005	资源不存在。
0xA0058006	函数参数中有空指针。
0xA0058007	使能系统、Device或通道前未配置对应的参数。
0xA0058008	不支持的参数或者功能。
0xA0058009	该操作不允许，如试图修改静态配置参数。
0xA005800C	分配内存失败，如系统内存不足。
0xA005800D	分配缓存失败，如申请的数据缓冲区太大。
0xA005800E	缓冲区中无数据。
0xA005800F	缓冲区中数据满。
0xA0058010	系统没有初始化或者相关依赖的模块没有加载。
0xA0058011	地址错误。
0xA0058012	系统忙。
0xA0058013	缓存小于实际需要的大小。
0xA0058014	硬件或软件处理超时。
0xA0058015	内部系统错误。
0xA005803F	最大的返回码，该模块的错误码必须小于该值。

## 7.6 VENC 返回码

错误代码	描述
0xA0088001	无效的Device ID。
0xA0088002	无效的channel ID。
0xA0088003	参数不合法，例如不合法的枚举值。
0xA0088004	资源已存在。
0xA0088005	资源不存在。
0xA0088006	函数参数中有空指针。
0xA0088007	使能系统、Device或通道前未配置对应的参数。
0xA0088008	不支持的参数或者功能。
0xA0088009	该操作不允许，如试图修改静态配置参数。
0xA008800C	分配内存失败，如系统内存不足。

错误代码	描述
0xA008800D	分配缓存失败，如申请的数据缓冲区太大。
0xA008800E	缓冲区中无数据。
0xA008800F	缓冲区中数据满。
0xA0088010	系统没有初始化或者相关依赖的模块没有加载。
0xA0088011	地址错误。
0xA0088012	系统忙。
0xA0088013	缓存小于实际需要的大小。
0xA0088014	硬件或软件处理超时。
0xA0088015	内部系统错误。
0xA008803F	最大的返回码，该模块的错误码必须小于该值。

## 7.7 PNGD 返回码

错误代码	描述
0xA0408001	无效的Device ID。
0xA0408002	无效的channel ID。
0xA0408003	参数不合法，例如不合法的枚举值。
0xA0408004	资源已存在。
0xA0408005	资源不存在。
0xA0408006	函数参数中有空指针。
0xA0408007	使能系统、Device或通道前未配置对应的参数。
0xA0408008	不支持的参数或者功能。
0xA0408009	该操作不允许，如试图修改静态配置参数。
0xA040800C	分配内存失败，如系统内存不足。
0xA040800D	分配缓存失败，如申请的数据缓冲区太大。
0xA040800E	缓冲区中无数据。
0xA040800F	缓冲区中数据满。
0xA0408010	系统没有初始化或者相关依赖的模块没有加载。
0xA0408011	地址错误。
0xA0408012	系统忙。

错误代码	描述
0xA0408013	缓存小于实际需要的大小。
0xA0408014	硬件或软件处理超时。
0xA0408015	内部系统错误。
0xA040803F	最大的返回码，该模块的错误码必须小于该值。



# 8 调用示例

- [8.1 获取示例代码](#)
- [8.2 实现VPC功能](#)
- [8.3 实现JPEGE功能](#)
- [8.4 实现JPEGD功能](#)
- [8.5 实现PNGD功能](#)
- [8.6 实现VDEC功能](#)
- [8.7 实现VENC功能](#)

## 8.1 获取示例代码

单击[示例代码](#)，查看样例代码以及样例的编译运行指导。

## 8.2 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[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3-1](#)。

### 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 blmage
    // set context
    aclrtSetCurrentContext(context);
    uint8_t* inBuffer = nullptr;
    acldvppMalloc((void**)&(inBuffer), inBufferSize); // Construct an input picture.
    if (inBuffer == nullptr) {
        printf("can not alloc input buffer\n");
        return;
    }
    uint8_t* outBuffer = nullptr;
    acldvppMalloc((void**)&(outBuffer), outBufferSize); // Construct an output picture.
    if (outBuffer == nullptr) {
        printf("can not alloc output buffer\n");
        acldvppFree(inBuffer);
        inBuffer = nullptr;
        return;
    }

    FILE* fp = fopen("dvpp_vpc_1920x1080_nv12.yuv", "rb+");
    if (fp == nullptr) {
        printf("fopen file failed\n");
        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) {
    printf("create dvpp api fail.\n");
    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) {
    printf("call vpc dvppctl process failed!\n");
    ret = DestroyDvppApi(pidvppapi);
    DFreeInOutBuffer(inBuffer, outBuffer);
    return;
}

FILE* outImageFp = fopen("NewVpcTest1Out.yuv", "wb");
if (outImageFp == nullptr) {
    printf("open NewVpcTest1Out.yuv failed!\n");
    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
    // set context
    aclrtSetCurrentContext(context);
    uint8_t* inBuffer = nullptr;
    aclvppMalloc((void**)&(inBuffer), inBufferSize); // Construct an input picture.
    if (inBuffer == nullptr) {
        printf("can not alloc input buffer\n");
        return;
    }
    uint8_t* outBuffer = nullptr;
    aclvppMalloc((void**)&(outBuffer), outBufferSize); // Construct an output picture.
    if (outBuffer == nullptr) {
        printf("can not alloc output buffer\n");
        aclvppFree(inBuffer);
        inBuffer = nullptr;
        return;
    }
    int32_t safeFuncRet = memset_s(outBuffer, outBufferSize, 0x80, outBufferSize);
    if (safeFuncRet != 0) {
        printf("memset_s fail");
        return;
    }

    FILE* fp = fopen("dvpp_vpc_1920x1080_nv12.yuv", "rb+");
    if (fp == nullptr) {
        printf("fopen file failed\n");
        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) {
    printf("create dvpp api fail.\n");
    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) {
    printf("call vpc dvppctl process failed!\n");
    ret = DestroyDvppApi(pidvppapi);
    DFreeInOutBuffer(inBuffer, outBuffer);
    return;
}

FILE* outImageFp = fopen("NewVpcTest2Out.yuv", "wb");
if (outImageFp == nullptr) {
    printf("open NewVpcTest2Out.yuv failed!\n");
    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。关于拼接的功能介绍，请参见[VPC功能](#)。

**示例代码：**本示例是从yuv420sp格式、1080p的图片中抠出4张图，经过缩放后，将缩放后的4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
    // set context
    aclrtSetCurrentContext(context);
    uint8_t* inBuffer = nullptr;
    aclvppMalloc((void**)&(inBuffer)), inBufferSize); // Construct an input picture.
    if (inBuffer == nullptr) {
        printf("can not alloc input buffer\n");
        return;
    }

    FILE* fp = fopen("dvpp_vpc_1920x1080_nv12.yuv", "rb+");
    if (fp == nullptr) {
        printf("fopen file failed\n");
        aclvppFree(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;
    uint8_t* outBuffer = nullptr;
    aclvppMalloc((void**)&(outBuffer)), outBufferSize); // Construct an output picture.
    if (outBuffer == nullptr) {
        printf("can not alloc output buffer\n");
        aclvppFree(inBuffer);
        inBuffer = nullptr;
        return;
    }
    int32_t safeFuncRet = memset_s(outBuffer, outBufferSize, 0x80, outBufferSize);
    if (safeFuncRet != 0) {
        printf("memset_s fail\n");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    for (uint32_t i = 0; i < 4; 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 = i * 384;
```

```
inputConfigure->cropArea.rightOffset = i * 384 + 383;
inputConfigure->cropArea.upOffset   = i * 256;
inputConfigure->cropArea.downOffset = i * 256 + 255;
VpcUserRoiOutputConfigure* outputConfigure = &roiConfigure->outputConfigure;
outputConfigure->addr = outBuffer;
outputConfigure->bufferSize = outBufferSize;
outputConfigure->widthStride = outWidthStride;
outputConfigure->heightStride = outHeightStride;
// Set the map area.
outputConfigure->outputArea.leftOffset = i * 224; // The offset value must be 16-pixel-aligned.
outputConfigure->outputArea.rightOffset = i * 224 + 223;
outputConfigure->outputArea.upOffset   = i * 128;
outputConfigure->outputArea.downOffset = i * 128 + 127;
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) {
    printf("create dvpp api fail.\n");
    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) {
    printf("call vpc dvppctl process failed!\n");
    ret = DestroyDvppApi(pidvppapi);
    DFreeInOutBuffer(inBuffer, outBuffer);
    return;
}

FILE* outImageFp = fopen("NewVpcTest3Out.yuv", "wb");
if (outImageFp == nullptr) {
    printf("open NewVpcTest3Out.yuv failed!\n");
    DestroyDvppApi(pidvppapi);
    DFreeInOutBuffer(inBuffer, outBuffer);
    return;
}
fwrite(outBuffer, 1, outBufferSize, outImageFp);
fclose(outImageFp);
outImageFp = nullptr;

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

## 示例 4：软件 8K 缩放功能

对于软件**8K缩放**功能，在缩放的同时，可以与格式转换（支持YUV420SP NV12与YUV420SP NV21之间的格式转换）功能组合，但不支持与抠图功能组合。

**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
// set context
aclrtSetCurrentContext(context);
uint8_t* inBuffer = nullptr;
aclvppMalloc((void**)&(inBuffer)), inBufferSize); // Construct an input picture.
if (inBuffer == nullptr) {
    printf( "can not alloc input buffer\n");
    return;
}
uint8_t* outBuffer = nullptr;
aclvppMalloc((void**)&(outBuffer)), outBufferSize); // Construct an output picture.
if (outBuffer == nullptr) {
    printf( "can not alloc output buffer\n");
    aclvppFree(inBuffer);
    inBuffer = nullptr;
    return;
}

FILE* fp = fopen("dvpp_vpc_8192x8192_nv12.yuv", "rb+");
if (fp == nullptr) {
    printf( "fopen file failed\n");
    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
VpcUserRoiOutputConfigure* outputConfigure = &roiConfigure->outputConfigure;
outputConfigure->addr = outBuffer;
outputConfigure->bufferSize = outBufferSize;
outputConfigure->widthStride = outWidthStride;
outputConfigure->heightStride = outHeightStride; // Set the map area
imageConfigure->roiConfigure = roiConfigure.get();
IDVPPAPI *pidvppapi = nullptr;
int32_t ret = CreateDvppApi(pidvppapi);
if (ret != 0) {
    printf( "create dvpp api fail\n");
    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) {
    printf( "call vpc dvppctl process failed!\n");
    ret = DestroyDvppApi(pidvppapi);
    DFreeInOutBuffer(inBuffer, outBuffer);
    return;
}

FILE* outImageFp = fopen("NewVpcTest4Out.yuv", "wb");
if (outImageFp == nullptr) {

```



```
printf( "open NewVpcTest4Out.yuv failed!\n");
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

**acldvppMalloc((void\*\*>(&(outBuffer)), 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);
```

## 8.3 实现 JPEG 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_JPEG_CUSTOM_MEMORY()
{
    sJpegIn inData;
    sJpegOut 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;
    }
    // set context
    aclrtSetCurrentContext(context);
    void* addrOrig = nullptr;
    acldvppMalloc((void**>(&(addrOrig)), inData.bufSize); // Construct an input picture.
    if (addrOrig == nullptr) {
        printf( "can not alloc input buffer\n");
        return;
    }
    inData.buf = reinterpret_cast<unsigned char*>(addrOrig);

    unsigned char* tmpAddr = nullptr;

    do {
        // load img file
        FILE* fpln = fopen(g_inFileName, "rb");
        if (nullptr == fpln) {
            printf( "can not open input file\n");
            return;
        }
    } while (0);
```

```
        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
        // 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
        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) {
        printf("call DvppGetOutParameter process failed\n");
        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
    printf("outdata size is %d", outData.jpgSize);
    aclDvppMalloc((void**)&(tmpAddr), outData.jpgSize); // Construct an input picture.
    if (tmpAddr == nullptr) {
        printf("can not alloc output buffer\n");
        break;
    }
    uint32_t size = outData.jpgSize;

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

    if (!HandleJpegCtlSucc(dvppApiCtlMsg, tmpAddr, size, &outData)) {
        break;
    }
} while (0); // for resource inData.buf
if (addrOrig != nullptr) {
    aclDvppFree(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
    aclDvppFree(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 JpegProcess()
{
    sJpegIn inData;
    sJpegOut 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;
}
// set context
aclrtSetCurrentContext(context);
void* addrOrig = nullptr;
aclDvppMalloc((void**)&addrOrig), inData.bufSize); // Construct an input picture.
if (addrOrig == nullptr) {
    printf("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) {
        printf("can not open input file\n");
        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
        // 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
        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 jpeg 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) {
        printf("can not open dvppapi engine\n");
        break;
    }

    JpegProcessBranch(pidvppapi, dvppApiCtlMsg, outData);

    DestroyDvppApi(pidvppapi);
} while (0); // for resource inData.buf
if (addrOrig != nullptr) {
    aclDvppFree(addrOrig);
    addrOrig = nullptr;
}
```

```
    }  
}  
  
/*  
 * only handle jpeg process.  
 */  
void JpegeProcessBranch(IDVPPAPI*& pidvppapi, dvppapi_ctl_msg& dvppApiCtlMsg, sjpegeOut&  
outData)  
{  
    do {  
        if (DvppCtl(pidvppapi, DVPP_CTL_JPEGE_PROC, &dvppApiCtlMsg)) {  
            printf( "call jpeg encoder fail\n");  
            break;  
        }  
  
        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 {  
            printf( "call not save result file %s \n", outFile.str().c_str());  
        }  
  
        outData.cbFree();  
        outData.jpgData = nullptr;  
        printf( "jpeg encode process completed");  
    } while (0); // for resource pddvppapi  
}
```

## 8.4 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;  
    }  
    jpegdInData.isVBeforeU = g_isVBeforeU;  
  
    FILE *fpIn = fopen(g_inFileName, "rb");  
    if (fpIn == nullptr) {  
        printf( "can not open file %s.\n", g_inFileName);  
        return;  
    }  
  
    do { // for resource fpIn  
        fseek(fpIn, 0, SEEK_END);  
        uint32_t fileLen = ftell(fpIn);  
        jpegdInData.jpegDataSize = fileLen;  
        fseek(fpIn, 0, SEEK_SET);  
        // set context  
        aclrtSetCurrentContext(context);  
        void* addrOrig = nullptr;  
        aclDvppMalloc((void**>(&addrOrig)), jpegdInData.jpegDataSize); // Construct an input picture.  
        if (addrOrig == nullptr) {  
            printf( "can not alloc input buffer\n");  
        }  
    } while (0);  
}
```

```
        fclose(fpln);
        fpln = nullptr;
        break;
    }

    jpegdInData.jpegData = reinterpret_cast<unsigned char*>(addrOrig);
    do { // for resource inBuf
        fread(jpegdInData.jpegData, 1, fileLen, fpln);
        fclose(fpln);
        fpln = nullptr;
        int32_t ret = DvppGetOutParameter((void*)&jpegdInData, (void*)&jpegdOutData,
GET_JPEGD_OUT_PARAMETER);
        if (ret != 0) {
            printf( "call DvppGetOutParameter process failed\n");
            break;
        }
        aclvppMalloc((void**>(&jpegdOutData.yuvData)), jpegdOutData.yuvDataSize); // Construct
an input picture.
        if (jpegdOutData.yuvData == nullptr) {
            printf( "can not alloc output buffer\n");
            break;
        }
        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) {
            if (0 != DvppCtl(pidvppapi, DVPP_CTL_JPEGD_PROC, &dvppApiCtlMsg)) {
                printf( "call dvppctl process failed\n");
                DestroyDvppApi(pidvppapi);
                break;
            }
            DestroyDvppApi(pidvppapi);
        } else {
            printf( "can not create dvpp api\n");
            break;
        }
        WriteTestJpegdResultToFile(&jpegdOutData);
    } while (0); // for resource inBuf

    if (addrOrig != nullptr) {
        aclvppFree(addrOrig);
        addrOrig = nullptr;
    }
    if (jpegdOutData.yuvData != nullptr) {
        aclvppFree(jpegdOutData.yuvData);
        jpegdOutData.yuvData = nullptr;
    }
} while (0); // for resource fpln
}
```

- 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;
    }
    jpegdInData.isVBeforeU = g_isVBeforeU;

    FILE *fpln = fopen(g_inFileName, "rb");
    if (fpln == nullptr) {
```

```
printf( "can not open file %s.\n", g_inFileName);
return;
}

do { // for resource fpln
    fseek(fpln, 0, SEEK_END);
    // the buf len should 8 byte larger, the driver asked
    uint32_t fileLen = ftell(fpln);
    jpegInData.jpeg_data_size = fileLen;
    fseek(fpln, 0, SEEK_SET);
    // set context
    aclrtSetCurrentContext(context);
    void* addrOrig = nullptr;
    aclvppMalloc(&(addrOrig), jpegInData.jpeg_data_size); // Construct an input picture.

    if (addrOrig == nullptr) {
        printf( "can not alloc input buffer\n");
        break;
    }

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

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

        IDVPPAPI *pidvppapi = nullptr;
        CreateDvppApi(pidvppapi);

        if (pidvppapi != nullptr) {
            if (DvppCtl(pidvppapi, DVPP_CTL_JPEGD_PROC, &dvppApiCtlMsg) != 0) {
                printf( "call dvppctl process failed\n");
                DestroyDvppApi(pidvppapi);
                break;
            }
            DestroyDvppApi(pidvppapi);
        } else {
            printf( "can not create dvpp api\n");
            break;
        }

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

    } while (0); // for resource inBuf

    if (addrOrig != nullptr) {
        aclvppFree(addrOrig);
        addrOrig = nullptr;
    }

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

## 8.5 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* fpln = fopen(g_inFileName, "rb");
if (nullptr == fpln) {
    printf( "can not open file %s.\n", g_inFileName);
    return;
}

fseek(fpln, 0, SEEK_END);
uint32_t fileLen = ftell(fpln);
fseek(fpln, 0, SEEK_SET);
// set context
aclrtSetCurrentContext(context);
void* inbuf = nullptr;
aclvppMalloc((void**)&(inbuf), fileLen); // Construct an input picture.
if (inbuf == nullptr) {
    printf( "can not alloc input buffer\n");
    fclose(fpln);
    fpln = nullptr;
    return;
}

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

fread(inputPngData.inputData, 1, fileLen, fpln);
fclose(fpln);
fpln = 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) {
    printf( "call DvppGetOutParameter process failed\n");
    aclvppFree(inbuf);
    inbuf = nullptr;
    return;
}
printf("pngd out size is %d", outputPngData.size);
aclvppMalloc((void**)&(outputPngData.address), outputPngData.size); // Construct an input
picture.
if (outputPngData.address == nullptr) {
    printf( "can not alloc output buffer\n");
    aclvppFree(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
    if (DvppCtl(pidvppapi, DVPP_CTL_PNGD_PROC, &dvppApiCtlMsg) != 0) {
        printf( "call dvppctl process failed\n");
        aclvppFree(inbuf);
        inbuf = nullptr;
    }
}
```

```
        acldvppFree(outputPngData.address);
        outputPngData.address = nullptr;
        DestroyDvppApi(pidvppapi); // destroy dvppapi
        return;
    }
} else {
    printf( "can not get dvpp api\n");
}

DestroyDvppApi(pidvppapi);

char* pAddr = (char*)outputPngData.address;
FILE* fpOut = fopen(g_outFileName, "wb");
if (fpOut == nullptr) {
    printf( "can not open file %s.\n", g_outFileName);
    acldvppFree(inbuf);
    inbuf = nullptr;
    acldvppFree(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;
acldvppFree(inbuf);
inbuf = nullptr;
acldvppFree(outputPngData.address);
outputPngData.address = nullptr;
return;
}
```

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

```
void TEST_PNGD()
{
    // load test file
    FILE* fpln = fopen(g_inFileName, "rb");
    if (fpln == nullptr) {
        printf( "can not open file %s.\n", g_inFileName);
        return;
    }

    fseek(fpln, 0, SEEK_END);
    uint32_t fileLen = ftell(fpln);
    fseek(fpln, 0, SEEK_SET);
    // set context
    aclrtSetCurrentContext(context);
    void* inbuf = nullptr;
    acldvppMalloc((void**)&(inbuf), fileLen); // Construct an input picture.
    if (inbuf == nullptr) {
        printf( "can not alloc input buffer\n");
        fclose(fpln);
        fpln = nullptr;
        return;
    } else {
        fread(inbuf, 1, fileLen, fpln);
    }

    fclose(fpln);
}
```



```
fpIn = nullptr;

// prepare msg
struct PngInputInfoAPI inputPngData;
inputPngData.inputData = inbuf;
inputPngData.inputSize = fileLen;
if (g_transform == 1) {
    inputPngData.transformFlag = 1; // RGBA -> RGB
} else {
    inputPngData.transformFlag = 0;
}

struct PngOutputInfoAPI outputPngData;

dvppapi_ctl_msg dvppApiCtlMsg;
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);

if (pidvppapi != nullptr) {
    if (DvppCtl(pidvppapi, DVPP_CTL_PNGD_PROC, &dvppApiCtlMsg) != 0) {
        printf("call dvppctl process failed\n");
        DestroyDvppApi(pidvppapi);
        aclvppFree(inbuf);
        inbuf = nullptr;
        outputPngData.FreeOutputMemory();
        return;
    }
} else {
    printf("can not get dvpp api\n");
    aclvppFree(inbuf);
    inbuf = nullptr;
    return;
}

DestroyDvppApi(pidvppapi);

char* pAddr = (char*)outputPngData.outputData;
FILE* fpOut = fopen(g_outFileName, "wb");
if (fpOut == nullptr) {
    printf("can not open file %s.\n", g_outFileName);
    aclvppFree(inbuf);
    inbuf = nullptr;
    outputPngData.FreeOutputMemory();
    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;
aclvppFree(inbuf);
inbuf = nullptr;
outputPngData.FreeOutputMemory();
}
```

## 8.6 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_;
};

//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(FrameData& frameData)
{
    static int32_t imageCount = 0;
    imageCount++;
    // 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.
    if (frameData.HiaiData() != NULL) {
        // 强转为用户自定义子类指针，用户自行决定如何使用强转后的指针hiai_data_sp_son
        HIAI_DATA_SP_SON* haii_data_sp_son = (HIAI_DATA_SP_SON*)frameData.HiaiData();
    }

    // 如果是隔行扫描码流，则vdec输出的是yuv数据，无需解压缩
    if (frameData.IsCompressData() == false) {
        int32_t ret = FrameReturnSaveYuvResult(frameData, imageCount);
        if (ret != 0) {
            printf("save yuv result failed\n");
        }
        return;
    }
    IDVPPAPI* dvppHandle = nullptr;
    int32_t ret = CreateDvppApi(dvppHandle);
    if (ret != 0) {
        printf("create dvpp api fail\n");
        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 = frameData.RealWidth();
userImage->heightStride = frameData.RealHeight();
// Configuration input format
if (frameData.Bitdepth() == 8) {
    if (frameData.ImageFormat() == 0) {
        // nv12
        userImage->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV;
    } else {
        // nv21
        userImage->inputFormat = INPUT_YUV420_SEMI_PLANNER_VU;
    }
} else {
    if (frameData.ImageFormat() == 0) {
        // nv12
        userImage->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV_10BIT;
    } else {
        // nv21
        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)frameData.OutBuffer();
compressDataConfigure->lumaHeadAddr = baseAddr + frameData.OffsetHeadY();
compressDataConfigure->chromaHeadAddr = baseAddr + frameData.OffsetHeadC();
compressDataConfigure->lumaPayloadAddr = baseAddr + frameData.OffsetPayloadY();
compressDataConfigure->chromaPayloadAddr = baseAddr + frameData.OffsetPayloadC();
compressDataConfigure->lumaHeadStride = frameData.StrideHead();
compressDataConfigure->chromaHeadStride = frameData.StrideHead();
compressDataConfigure->lumaPayloadStride = frameData.StridePayload();
compressDataConfigure->chromaPayloadStride = frameData.StridePayload();

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 = frameData.RealWidth() - 1;
roiInput->cropArea.upOffset = 0;
roiInput->cropArea.downOffset = frameData.RealHeight() - 1;
VpcUserRoiOutputConfigure* roiOutput = &roiConfigure->outputConfigure;
roiOutput->outputArea.leftOffset = 0;
roiOutput->outputArea.rightOffset = frameData.RealWidth() - 1;
roiOutput->outputArea.upOffset = 0;
roiOutput->outputArea.downOffset = frameData.RealHeight() - 1;

// set context
aclrtSetCurrentContext(context);
// vpc宽、高分别为16, 2 对齐, buffer大小为: 宽*高*3/2
roiOutput->bufferSize = ALIGN_UP(frameData.RealWidth(), 16) * ALIGN_UP(frameData.RealHeight(), 2)
* 3 / 2;
aclDvppMalloc((void**)&(roiOutput->addr), roiOutput->bufferSize);
if (roiOutput->addr == nullptr) {
    printf("can not alloc roiOutput buffer\n");
    DestroyDvppApi(dvppHandle);
    return;
}

roiOutput->widthStride = ALIGN_UP(frameData.RealWidth(), 16); // vpc宽为16对齐
roiOutput->heightStride = ALIGN_UP(frameData.RealHeight(), 2); // vpc高为2对齐

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) {
    printf( "call dvppctl fail\n");
    aclvppFree(roiOutput->addr);
    roiOutput->addr = nullptr;
    DestroyDvppApi(dvppHandle);
    return;
}
ret = FrameReturnSaveVpcResult(frameData, roiOutput->addr, imageCount);
if (ret != 0) {
    printf( "save vpc result fail\n");
}
aclvppFree(roiOutput->addr);
roiOutput->addr = nullptr;

DestroyDvppApi(dvppHandle);
return;
}

int32_t CreateOutputDir()
{
    char outputDir[20] = {0};
    int32_t safeFuncRet = memset_s(outputDir, sizeof(outputDir), 0, sizeof(outputDir));
    if (safeFuncRet != EOK) {
        printf( "memset_s fail");
        return -1;
    }
    safeFuncRet = strncpy_s(outputDir, sizeof(outputDir), "output_dir", strlen("output_dir"));
    if (safeFuncRet != EOK) {
        printf( "strncpy_s fail");
        return -1;
    }
    if (access(outputDir, F_OK) == -1) {
        if (mkdir(outputDir, 0700) < 0) { // directory authority: 0700
            printf( "create dir failed.\n");
            return -1;
        }
    } else if (access(outputDir, R_OK | W_OK | X_OK) == -1) {
        printf( "output directory authority is not correct, use 700 instead.\n");
        return -1;
    }
    return 0;
}

/*
*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) {
        printf( "vdec error code is %#x, channelId = %u\n",
            vdecErr->errType, vdecErr->channelId);
    }
}

//测试函数主入口
void TEST_VDEC()
{
    int32_t ret = CreateOutputDir();
    if (ret != 0) {
        printf( "check output directory fail.\n");
        return;
    }

    IDVPPAPI *pidvppapi = nullptr;
    ret = CreateVdecApi(pidvppapi, 0);
    if (ret != 0 || pidvppapi == nullptr) {
```

```
printf( "create dvpp api fail.\n");
return;
}

VdecInMsg vdecInMsg;
vdecInMsg.SetFrameReturn(FrameReturn);
vdecInMsg.SetErrReport(ErrReport);
(g_format == 0) ? vdecInMsg.SetVideoFormat(H264) : vdecInMsg.SetVideoFormat(HEVC);

FILE* fp = fopen(g_inFileName, "rb");
if (fp != nullptr) {
    fseek(fp, 0L, SEEK_END);
    int fileSize = ftell(fp);
    fseek(fp, 0L, SEEK_SET);

    if (fileSize > 0) {
        char *inBuffer = (char*)malloc(fileSize);
        if(inBuffer == nullptr) {
            printf( "malloc inBuffer failed\n");
            fclose(fp);
            DestroyVdecApi(pidvppapi, 0);
            return;
        } else {
            vdecInMsg.SetInBuffer(inBuffer);
        }

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

        uint32_t readSize = fread(inBuffer, 1, fileSize, fp);
        vdecInMsg.SetInBufferSize(readSize);

        // 下面三行是用户定义hiai_data_sp对象和设置帧序号等，不需要可跳过，
        // 若需要则要求fileSize必须为1帧大小
        static uint64_t cnt = 0;
        std::shared_ptr< HIAI_DATA_SP_SON> dc_sp = make_shared< HIAI_DATA_SP_SON>();
        dc_sp->setFrameIndex(++cnt);
        vdecInMsg.SetHiaiDataSp(dc_sp);
        // 设置解码通道Id，不同解码通道设置不同值，取值范围0~15，
        // 此处示例代码设置为0
        vdecInMsg.SetChannelId(0);

        //调用VdecCtl接口解码
        if (VdecCtl(pidvppapi, DVPP_CTL_VDEC_PROC, &dvppApiCtlMsg, 0) != 0) {
            printf( "call dvppctl process fail!");
        }
        fseek(fp, 0L, SEEK_SET);

        //主动设置eos标志并调用VdecCtl接口发送，若不设置，DestroyVdecApi接口中默认设置，否则优先使用
        //此处设置
        vdecInMsg.SetEos(true);
        if (VdecCtl(pidvppapi, DVPP_CTL_VDEC_PROC, &dvppApiCtlMsg, 0) != 0) {
            printf( "send eos flag, call dvppctl process fail\n");
        }

        free(inBuffer);
        inBuffer = nullptr;
    } else {
        printf( "fileSize is %d, but it should be greater than 0.\n", fileSize);
    }
    fclose(fp);
} else {
    printf( "open file: %s failed.\n", g_inFileName);
}
DestroyVdecApi(pidvppapi, 0);
return;
}
```

## 8.7 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) {
        printf( "get venc out file fail!\n");
        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> fpln(fopen(g_inFileName, "rb"), fclose);
    vencOutFile.reset(fopen(vencOutFileName.c_str(), "wb"), fclose);

    if (fpln.get() == nullptr || vencOutFile.get() == nullptr) {
        printf( "open open venc in/out file failed.\n");
        return;
    }

    fseek(fpln.get(), 0, SEEK_END);
    uint32_t fileLen = ftell(fpln.get());
    fseek(fpln.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 != 0) {
        printf( "CreateVenc fail!\n");
        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) {
        printf( "alloc input buffer failed\n");
        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) {
        printf("error in read input file\n");
        DestroyVenc(vencHandle);
        return;
    }

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

    int32_t ret = RunVenc(vencHandle, &vencInMsg);
    if (ret != 0) {
        printf("call video encode fail\n");
        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());

DestroyVenc(vencHandle);
return;
}
```

# 9 样例使用指导

---

单击[示例代码](#)，查看样例代码以及样例的编译运行指导。



# 10 附录

10.1 辅助功能接口

## 10.1 辅助功能接口

头文件	Class	接口
Jpeg.h	JpegCalBackFree	函数原型 JpegCalBackFree()
		函数原型 ~JpegCalBackFree()
		函数原型 JpegCalBackFree(JpegCalBackFree& others)
		函数原型 void setBuf(void* addr4Free)
		函数原型 void setBuf(void* addr4Free, size_t siz4Free)
		函数原型 void operator() ()
	-	函数原型 int32_t DvppJpegDecodeToRgb(JpegDecodeToRgbln & jpegDecodeToRgbln, JpegDecodeToRgbOut& jpegDecodeToRgbOut); 说明 内部使用，用户无需调用。
DvppCommon.h	AutoBuffer	函数原型 AutoBuffer();

头文件	Class	接口
		函数原型 ~AutoBuffer()
		函数原型 void Reset()
		函数原型 char* allocBuffer(int32_t size)
		函数原型 char* getBuffer() const
		函数原型 int32_t getBufferSize()
	IDVPPAPI	函数原型 virtual ~IDVPPAPI(void)
Vdec.h	HIAI_DATA_SP	函数原型 HIAI_DATA_SP()
		函数原型 virtual ~HIAI_DATA_SP()
		函数原型 void setFrameIndex(unsigned long long index)
		函数原型 unsigned long long getFrameIndex()
		函数原型 void setFrameBuffer(void * frameBuff)
		函数原型 void * getFrameBuffer()
		函数原型 void setFrameSize(uint32_t size)
		函数原型 uint32_t getFrameSize()