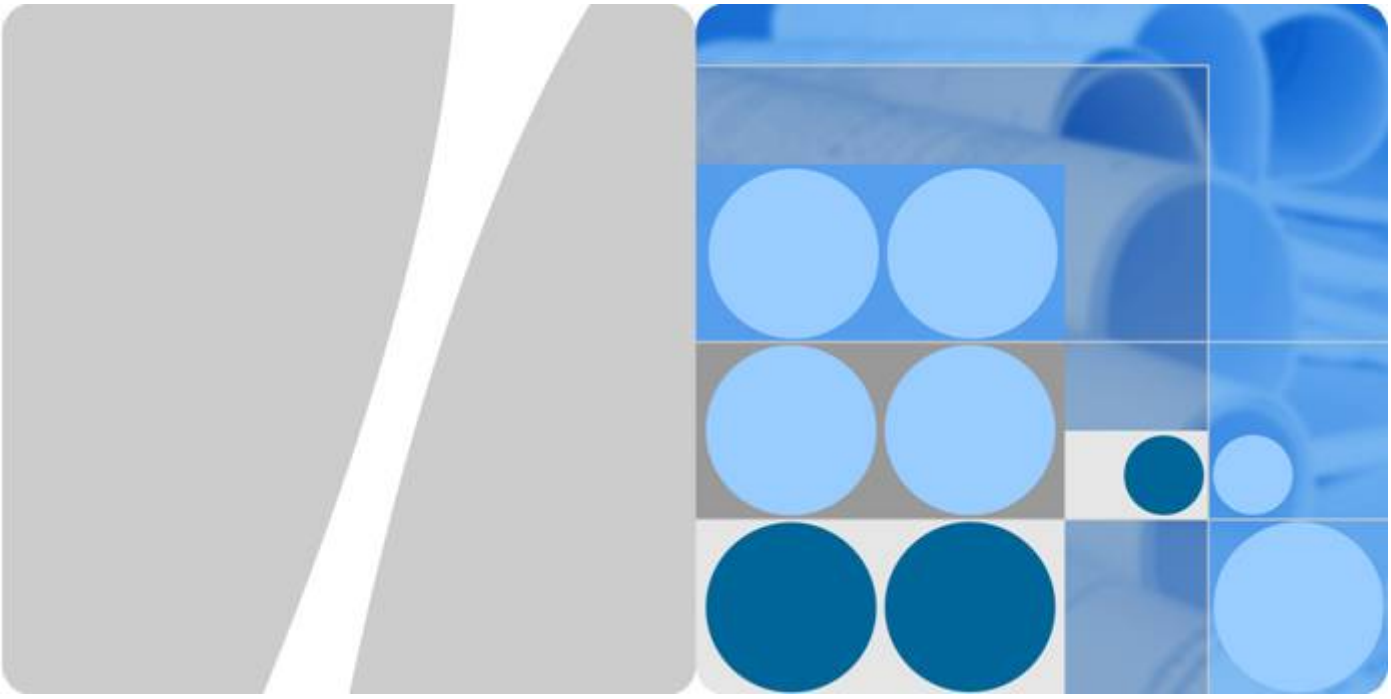


Ascend 310

V100R001

DVPP API 参考

文档版本　01

发布日期　2019-03-12

华为技术有限公司

HUAWEI

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

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

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

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 简介

本文档详细描述了DVPP执行器系统函数接口，以及DVPP执行器与HiAI Engine的接口，接口描述包括：接口函数描述、接口调用说明、整体示例等，适合开发人员、测试人员。

DVPP执行器对外的接口包括两部分：调用方接口和驱动接口。本文提供调用方接口，目的在于指work导调用方正确的设计和开发。

2 DVPP 对外接口

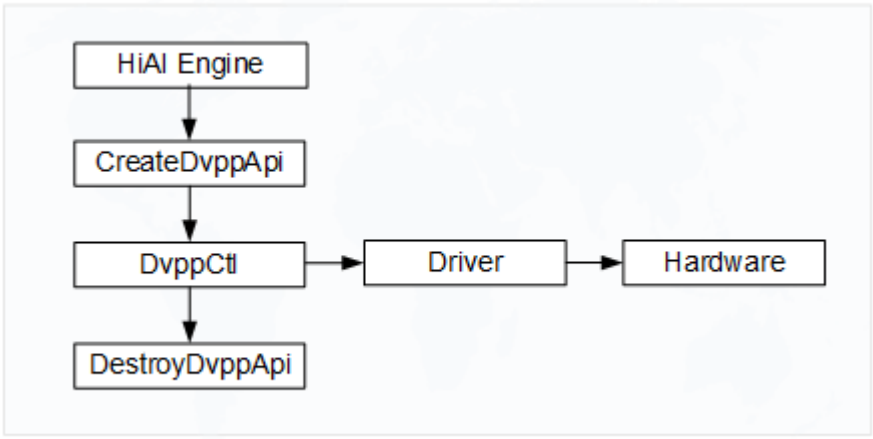
DVPP除VDEC模块对外提供三个接口，如下：

- CreateDvppApi：负责创建DVPP API实例，DVPP API实例类似于打开驱动后返回的文件描述符。
- DvppCtl：为控制DVPP各模块执行（主要调用各个模块暴露的Process接口函数，模块主要包括VPC、VENC、JPEGE、JPEGE、PNGD等，其含义见**A.3 缩略语和术语一览**）。
- DestroyDvppApi：为销毁DVPP API。

VDEC模块由于当前内部不是单例模式，所以其对外提供C接口函数与其他不同。接口使用请参考VDEC章节**2.2.2 VDEC**。

HiAI Engine调用DVPP的实现流程如下：

图 2-1 顺序流程图



2.1 CreateDvppApi

函数原型	int **CreateDvppApi**(**IDVPPAPI** *&pIDVPPAPI)

功能	获取dvppapi实例，相当于DVPP执行器句柄，调用方可以使用申请到的dvppapi实例进行DVPP IP的调用，可以跨函数调用，跨线程调用。调用方负责dvppapi实例的生命周期，即申请与释放，申请使用CreateDvppApi函数，释放使用DestroyDvppApi函数。
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，输出可能为NULL，可能不为NULL，获取失败则为NULL，成功则不为NULL。
返回值说明	返回值“0”代表成功，“-1”代表失败。
使用说明	调用方创建**“IDVPPAPI”**对象指针，初始化为NULL，调用**“CreateDvppApi”**函数将**“IDVPPAPI”**对象指针传入。如果申请成功**“CreateDvppApi”**函数会返回DvppApi实例，否则返回NULL。调用方需要对返回值进行校验。
使用限制	此函数为DVPP执行对外提供的统一调用接口之一，其主要作用相当于获取DVPP执行器句柄。

调用示例

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

2.2 DvppCtl

函数原型	int **DvppCtl(IDVPPAPI** *&pIDVPPAPI, int CMD, **dvppapi_ctl_msg** *MSG)
功能	控制dvpp执行器进行视频编解码，vpc，jpeg编解码，png解码等操作执行。
输入说明	输入为**“IDVPPAPI”**类型指针引用，**CMD控制命令字**，DVPP执行器配置信息MSG，类型为**“dvppapi_ctl_msg”**。DVPP各个组件的配置信息不同，因此每个组件都有相应的配置信息，配置信息会统一在头文件dvpp_config.h供调用方使用。在进行功能调用时一定要配置相应的参数。
输出说明	输出为MSG的配置信息中的输出buffer（此buffer可能为一个，也可能为多个，具体以**2.2 DvppCtl**各个功能的入参为准），以及DVPP的输出状态信息（都存储于MSG中）。
返回值说明	返回值“0”代表成功，“-1”代表失败。
使用说明	调用方调用**“DvppCtl”**函数，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。
使用限制	此函数为DVPP执行对外提供的统一调用接口之一，其主要作用为调用DVPP执行器中的具体功能执行。

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_VDEC_PROC	Vdec功能控制命令字。	4
DVPP_CTL_VENC_PROC	Venc功能控制命令字。	5
DVPP_CTL_DVPP_CAPABILITY	查询DVPP能力控制命令字。	6
DVPP_CTL_CMDLIST_PROC	Cmdlist功能控制命令字。	7
DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM	Vpc获取resize参数配置控制命令字。	8

dvppapi_ctl_msg

成员变量	说明	取值范围
in_size	入参大小。	-
out_size	出参大小。	-
void *in	入参。	resize_param_in_msg vpc_in_msg vdec_in_msg sJpegeIn JpegdIn PngInputInfoAPI venc_in_msg device_query_req_stru
void *out	出参。	resize_param_out_msg vpc_out_msg vdec_out_msg sJpegeOut JpegdOut PngOutputInfoAPI venc_out_msg dvpp_engine_capability_stru

2.2.1 VPC

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

- VPC路数无限制，其性能指标如下。

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

场景	总帧率
1080p * 32路	1440fps
4k * 4路	360fps

当处理过程中图像分辨率大于原图分辨率时，总帧率计算公式参考如下：

总帧率 = (3840*2160 * 360) / (W x H) fps

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

在处理过程中，缩小比例的增大会导致VPC处理性能有略微下降。

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

  注意

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

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

    说明

    对于yuv400sp格式图片，需要调用方申请和yuv420sp格式相同大小的空间，即：width_align_128*high_align_16*1.5。
  - VPC输出内存限制
    - 输出数据地址（os虚拟地址）：128字节对齐。
    - 输出图像宽内存限制：128字节对齐。
    - 输出图像高内存限制：16字节对齐。
  - 同一次任务的缓冲区的输入和输出区的虚拟地址应该在同一 4G 空间。

说明

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

入参：resize_param_in_msg

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

出参：resize_param_out_msg

成员变量	说明	取值范围
int dest_width_stride	输出图像的宽度对齐值。	128
int dest_high_stride	输出图像的高度对齐值。	16
int hmax	与原点在水平方向的最大偏移。	奇数，具体参见**入参：vpc_in_msg**。

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

入参：vpc_in_msg

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

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

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

成员变量	说明	取值范围
char* out_buffer	输出buffer。	此buffer为调用方申请的内存，需保证首地址128对齐，且保证申请的内存在4G空间内，与上一个参数auto_out_buffer_1选填一个即可。
int：out_buffer_1_size	输出buffer1大小。	此值用来进行输出buffer校验。 如果采用shared_ptr<AutoBuffer> auto_out_buffer_1申请输出内存则选填，如果采用 char* out_buffer申请输出内存则必填。
shared_ptr<AutoBuffer> auto_out_buffer_2	此智能指针为vpc双通道的另外一个通道输出，为预留接口	--
int：out_buffer_2_size	此buffersize 为auto_out_buffer_2智能指针输出buffer2大小，为预留接口	--
int：use_flag	使用vpc功能的标志。	0：vpc。 1：vpcrawdata8k。 2：Rawdata叠加。 3：Rawdata拼接。 默认为0。 选填
rawdata_overlay_config：overlay[4]	Rawdata叠加结构体，主要包括文字图片和图像图片的宽度、高度、stride、图像类型、图像格式、位宽、内存地址等。	具体请参见**A.1.1.1 图像叠加结构体rawdata_overlay_config**。
rawdata_collage_config：collage	Rawdata拼接结构体，主要包括叠加后的文字图像图片经过VPC缩放后的宽度、高度、拼接格式、输出地址等。	具体请参见**A.1.1.2 图像拼接结构体rawdata_collage_config**。
RDMACHANNEL rdma;	rdma参数为HFBC数据专用配置结构体。	具体请参见**A.1.1.3 Rdma通道结构体RDMACHANNEL**。
VpcTurningReverse turningReverse1	Vpc优化预留接口。	--
bool isVpcUseTurning1	标志是否使用Vpc优化，预留接口，此标志与turningReverse1配合使用	--

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

出参：vpc_out_msg

暂无，其中输出图片在vpc_in_msg中的auto_out_buffer_1中，用户可读取该内存获取图片。

表 2-1 输入图片格式与输出图片格式对应 rank 配置表

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

输入图片格式	VPC的rank参数配置	输出图片格式
YUV444sp 444spUV	444spUV = 0	YUV420sp NV21
YUV444sp 444spVU	444spVU= 1	YUV420sp NV21
YUV422packet YUYV	YUYV = 2	YUV420sp NV12
YUV422packet YVYU	YVYU = 3	YUV420sp NV21
YUV422packet UYVY	UYVY = 4	YUV420sp NV12
YUV444packet YUV	YUV = 5	YUV420sp NV12
RGB888 RGB	RGB = 6	YUV420sp NV12
RGB888 BGR	BGR = 7	YUV420sp NV21
RGB888 RGB	BGR = 7	YUV420sp NV21
RGB888 BGR	RGB = 6	YUV420sp NV12
XRGB8888 RGBA	RGBA = 8	YUV420sp NV12
XRGB8888 RGBA	BGRA = 9	YUV420sp NV21

调用示例

- VPC基础功能调用示例

  gXxxxx为配置参数，参数配置与VPC调用分为两个阶段。

  step1：通过DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM命令字调用DvppCtl，获取配置参数，DvppCtl的入参为resize_param_in_msg，出参为resize_param_out_msg。即通过resize_param_in_msg将原始图片的宽高，裁剪区域(hmax,hmin,vamx,vmin)，以及输出宽高等参数传给DVPP，DVPP通过计算将重新生成的裁剪区域(hmax,hmin,vamx,vmin)以及缩放比(hinc,vinc)返回给调用方。

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

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

dvppApiCtlMsg.in = (void *)(&resize_in_param);
```

```
    dvppApiCtlMsg.out = (void *)(&resize_out_param);
    IDVPPAPI *pidvppapi = NULL;
    CreateDvppApi(pidvppapi);
    if (DvppCtl(pidvppapi, DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM, &dvppApiCtlMsg) != 0) {
        printf("call DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM process faild!\n");
        DestroyDvppApi(pidvppapi);
        return;
    }

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

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

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



```
        }

        fread(payloadY, 1, buffersize, rdmaimage);
        vpcInMsg.rdma.luma_head_addr        = (long)headY;
        vpcInMsg.rdma.chroma_head_addr      = (long)headC;
        vpcInMsg.rdma.luma_head_stride      = vpcInMsg.stride;
        vpcInMsg.rdma.chroma_head_stride    = vpcInMsg.stride;
        vpcInMsg.rdma.luma_payload_stride   = gpYStride;
        vpcInMsg.rdma.chroma_payload_stride = gpUVStride;
        fclose(rdmaimage);
        printf("luma payload stride= %d\n", vpcInMsg.rdma.luma_payload_stride );
        printf("chroma payload stride= %d\n", vpcInMsg.rdma.chroma_payload_stride );
    } else {
        alloc_buffer(vpcInMsg.width, vpcInMsg.high, vpcInMsg.stride, bufferSize);
        if (open_image(in_file_name, vpcInMsg.width, vpcInMsg.high, vpcInMsg.stride) != 0) {
            printf("open file failed~\n");
            free(in_buffer);
            return;
        }
        vpcInMsg.in_buffer      = in_buffer;
        vpcInMsg.in_buffer_size = bufferSize;
    }
    //alloc in and out buffer
    shared_ptr<AutoBuffer> auto_out_buffer = make_shared<AutoBuffer>();
    vpcInMsg.auto_out_buffer_1             = auto_out_buffer;
    dvppApiCtlMsg.in                       = (void *)(&vpcInMsg);
    dvppApiCtlMsg.in_size                  = sizeof(vpc_in_msg);

    if (pidvppapi != NULL) {

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

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

    free(in_buffer);

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

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

- RawData8k基础功能调用示例

  参数配置：

```
    dvppapi_ctl_msg dvppApiCtlMsg;
    vpc_in_msg vpcInMsg;
    vpcInMsg.width = gWidth;
    vpcInMsg.high = gHigh;
    vpcInMsg.stride = gStride;
    vpcInMsg.out_width = gOutWidth;
    vpcInMsg.out_high = gOutHigh;
```



```
    printf("width     =%d\n", vpcInMsg.width);
    printf("high      =%d\n", vpcInMsg.high);
    printf("stride    =%d\n", vpcInMsg.stride);
    printf("out_width =%d\n", vpcInMsg.out_width);
    printf("out_high  =%d\n", vpcInMsg.out_high);
    //alloc in and out buffer
    char *in_buffer = (char *)malloc(vpcInMsg.width * vpcInMsg.high * 1.5);
    if (in_buffer = NULL) {
        printf("malloc fail\n");
        return;
    }

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

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

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

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

- Rawdata叠加基础功能调用实例

```
    dvppapi_ctl_msg dvppApiCtlMsg;
    vpc_in_msg vpcInMsg;
    //配置RawData叠加参数
    vpcInMsg.use_flag = 2;
    vpcInMsg.overlay.image_type = 0;
    vpcInMsg.overlay.image_rank_type = 0;
    vpcInMsg.overlay.bit_width = 8;
    vpcInMsg.overlay.in_width_image = 1000;
    vpcInMsg.overlay.in_high_image = 1000;
```



```
    vpcInMsg.overlay.in_width_text = 500;
    vpcInMsg.overlay.in_high_text = 390;
    vpcInMsg.overlay.image_width_stride = 2;
    vpcInMsg.overlay.image_high_stride = 2;
    vpcInMsg.overlay.text_width_stride = 2;
    vpcInMsg.overlay.text_high_stride = 2;
    vpcInMsg.overlay.in_buffer_image = NULL;
    vpcInMsg.overlay.in_buffer_text = NULL;
    vpcInMsg.overlay.auto_overlay_out_buffer = make_shared<AutoBuffer>();
    //初始化图像内存
    if(NULL == (vpcInMsg.overlay.in_buffer_image =
(char*)malloc(vpcInMsg.overlay.in_width_image*vpcInMsg.overlay.in_high_image*3/2)))    {
        printf("in_buffer_image alloc failed!\n");
        return ;
    }
    //打开图片至申请的内存中
    char image_file[50] = "yuv_data_0";
    FILE * yuvimage = fopen(image_file,"rb");
    if(NULL == yuvimage)    {
        printf("VPC_TEST: yuv image open faild!");
        return ;
    }    else    {
        fread(vpcInMsg.overlay.in_buffer_image,
1,vpcInMsg.overlay.in_width_image*vpcInMsg.overlay.in_high_image*3/2,yuvimage);
        fclose(yuvimage);
        yuvimage = NULL;
}

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

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

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



```
        return ;
    }    else    {
        printf("pidvppapi is null!\n");
        return ;
    }
    if(NULL != vpcInMsg.overlay.in_buffer_image)    {
        free(vpcInMsg.overlay.in_buffer_image);
        vpcInMsg.overlay.in_buffer_image = NULL;
    }
    if(NULL != vpcInMsg.overlay.in_buffer_text)    {
        free(vpcInMsg.overlay.in_buffer_text);
        vpcInMsg.overlay.in_buffer_text = NULL;
    }
```

- Rawdata拼接基础功能调用实例

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

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



```
1,vpcInMsg.collage.in_width*vpcInMsg.collage.in_high*6,fp);
        fflush(fp);
        fclose(fp);
        fp=NULL;
        return ;
    }    else    {
        printf("pidvppapi is null!\n");
        return ;
    }
    for(int i=0; i<4; i++) {
        if(NULL != vpcInMsg.collage.in_buffer[i])        {
            free(vpcInMsg.collage.in_buffer[i]);
            vpcInMsg.collage.in_buffer[i] = NULL;
        }
    }
```

- VPCAPI调用

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

- VPC Resize参数配置获取功能调用示例

  gXxxxx为配置参数

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



```
vpcInMsg.vinc = resize_out_param.vinc;
vpcInMsg.hinc = resize_out_param.hinc;
```

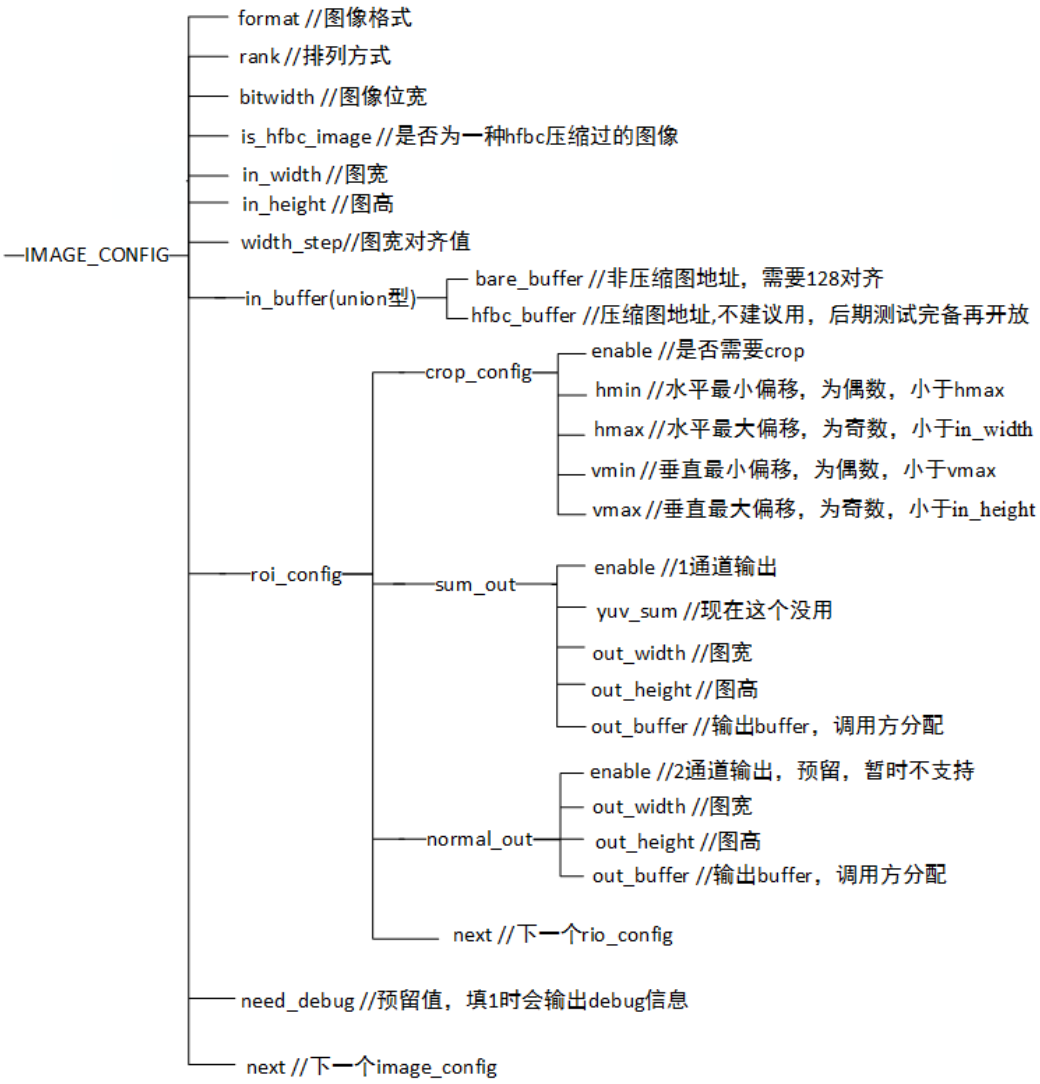
2.2.1.1 CMDLIST

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

入参

cmdlist输入为结构体，详细参数如图2-2、表2-2所示。

图 2-2 cmdlist 结构体



详细信息以及取值范围请参见表2-2。

表 2-2 结构体说明

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

成员变量	说明	取值范围

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

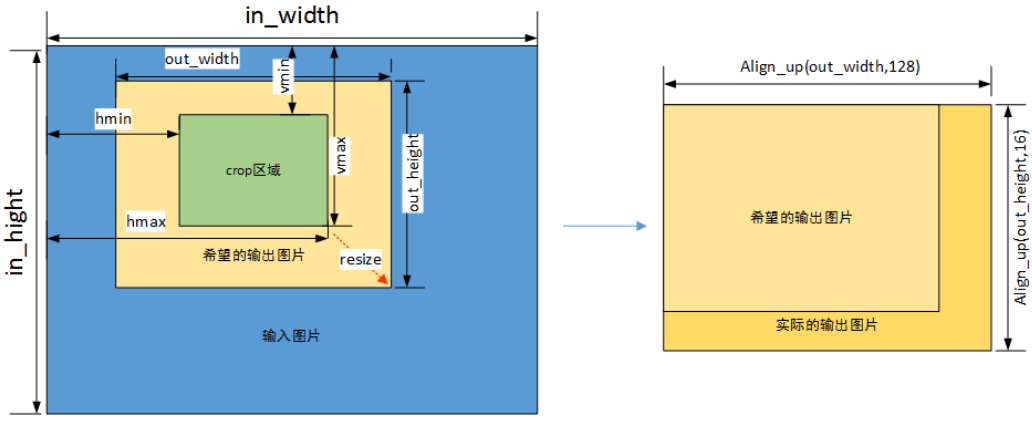
ROI_CONFIG：roi_config	输出参数配置结构体。	详见**ROI_CONFIG结构体**
IMAGE_CONFIG*：next	下一个IMAGE_CONFIG结构体指针。多图时配置，单图时配置为NULL。	--

说明

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

可以通过**图2-3**所示，直观了解输入参数信息。

图 2-3 输入参数示例说明



此图将crop和resize操作归一化到一个图中，理解如下。

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

说明

- 输入图片的宽128对齐，高16对齐，且输入输出图片最大分辨率为4K。
- 内存地址对齐，需要用到如下接口。
  - 输入输出图片内存地址128对齐，并且要保证地址在同一个4G空间，需要用HIAI_DMalloc的大页内存申请
  - 申请接口：HIAI_DMalloc(size, MALLOC_DEFAULT_TIME_OUT, HIAI_MEMORY_HUGE_PAGE)。
- 根据输出图片比上crop后图片计算缩放系数，需要在[0.03125,4]范围内。
- 输出内存大小需要根据输出宽128对齐，高16对齐后的分辨率计算。
- 调用一次cmdlist接口，最多支持32张图，每图最多支持256个框。

出参

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

输入图片格式与输出图片格式对应rank配置表请参见**表2-2**。

调用示例

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



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

2.2.1.2 VPC 和 CMDLIST 统一接口

vpc和cmdlist统一接口，包含vpc所有功能模块，请尽量使用此接口进行开发，将老接口迁移到新接口。

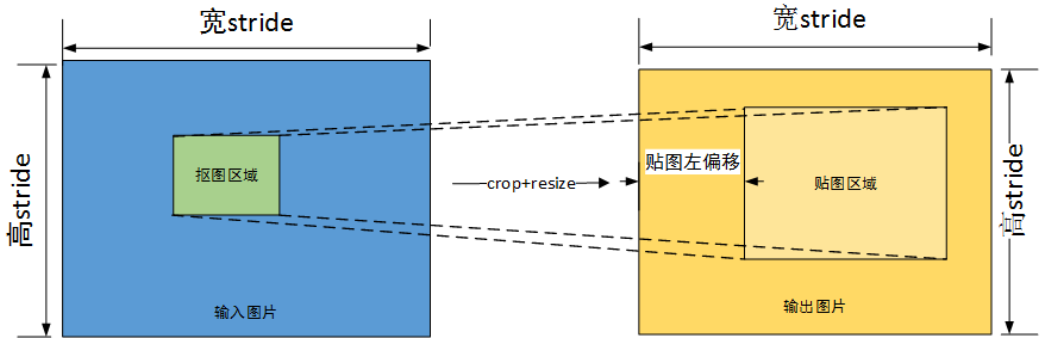
新接口支持的功能有：

- 单图裁剪缩放
- 一图多框裁剪缩放输出
- 等比例缩放

- 指定输出区域

NEW VPC归一化功能示意图如**图1 VPC归一化功能示意图**。

图 2-4 VPC 归一化功能示意图



说明

- 输出图片指用户分配的输出buffer图片。
- 贴图区域是指指定的输出区域。
- 宽stride指一行图像步长，并不完全等同于图像分辨率上宽的含义。
- 高stride指图像在内存中的行数。

例如：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

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。	--
uint32_t bareDataBufferSize;	非压缩格式输入图片内存大小，其中大小是根据图像的宽高stride计算出来的，当图像为压缩格式时，不需要填，默认为0。	--
uint32_t widthStride	图像宽度方向的步长，对于不同图像格式，stride与宽比例不同。	yuv400sp、yuv420sp、yuv422sp、yuv444sp：图像有效区域宽对齐到128。 yuv422packed：图像有效区域宽 * 2后对齐到128。 yuv444packed、rgb888：图像有效区域宽 * 3后对齐到128。 xrgb8888：图像有效区域宽 * 4后对齐到128。 输入为hfbc格式：填图像的宽
uint32_t heightStride	图像高度方向的步长，对于yuv sp图像根据该参数计算uv数据的起始地址。	--

成员变量	说明	取值范围

enum VpcInputFormat inputFormat	输入图像格式。	enum VpcInputFormat {
		INPUT_YUV400, // 0
		INPUT_YUV420_SEMI_PLANNER_UV, // 1
		INPUT_YUV420_SEMI_PLANNER_VU, // 2
		INPUT_YUV422_SEMI_PLANNER_UV, // 3
		INPUT_YUV422_SEMI_PLANNER_VU, // 4
		INPUT_YUV444_SEMI_PLANNER_UV, // 5
		INPUT_YUV444_SEMI_PLANNER_VU, // 6
		INPUT_YUV422_PACKED_YUYV, // 7
		INPUT_YUV422_PACKED_UYVY, // 8
		INPUT_YUV422_PACKED_YVYU, // 9
		INPUT_YUV422_PACKED_VYUY, // 10
		INPUT_YUV444_PACKED_YUV, // 11
		INPUT_RGB, // 12
		INPUT_BGR, // 13
		INPUT_ARGB, // 14
		INPUT_ABGR, // 15
		INPUT_RGBA, // 16
		INPUT_BGRA, // 17
		INPUT_YUV420_SEMI_PLANNER_UV_10BIT, // 18
		INPUT_YUV420_SEMI_PLANNER_VU_10BIT, // 19
		};

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

成员变量	说明	取值范围
VpcUserRoiConfigure* roiConfigure	抠图区域配置，详细请参见**•VpcUserRoiConfigure结构体**。	--
bool isCompressData	是否是视频解码输出的压缩图片数据格式。	当图片来源于VDEC时，需要配置为true，其他格式为false，默认为false。
VpcCompressDataConfigure compressDataConfigure	视频解码输出的压缩图片数据配置。详细请参见**•VpcCompressDataConfigur...**。	--
bool yuvSumEnable	是否需要计算yuvSum值，AIcore有此需求，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。
VpcUserYuvSum yuvSum	yuvSum计算配置。详细请参见**•VpcUserYuvSum结构体**。	--
VpcUserPerformanceTuningParameter tuningParameter	此变量为预留，用于参数调优，详细请参见**•VpcUserPerformanceTunin...**。	--
uint32_t* cmdListBufferAddr	此变量为预留。	--
uint32_t cmdListBufferSize	此变量为预留。	--
uint64_t yuvScalerParaSetAddr	滤波参数集文件路径地址和文件名数组。 **说明** ● 参数集须在Device设备上。 ● 请确保传入的文件路径是正确路径。	参数集文件路径为device侧的绝对文件路径＋文件名，如{"/root/share/vpc/YUVScaler_pra.h"}
uint16_t yuvScalerParaSetSize	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	yuvScalerParaSetIndex变量对应yuvScalerParaSetAddr的索引（默认为0）。	0<=yuvScalerParaSetIndex<10
uint32_t reserve1	预留接口	--

上述接口的定义请见DDK包下include/inc/dvpp/Vpc.h，详细如下：

```
// 支持的输入格式
enum VpcInputFormat {
    INPUT_YUV400, // 0
    INPUT_YUV420_SEMI_PLANNER_UV, // 1
    INPUT_YUV420_SEMI_PLANNER_VU, // 2
    INPUT_YUV422_SEMI_PLANNER_UV, // 3
    INPUT_YUV422_SEMI_PLANNER_VU, // 4
    INPUT_YUV444_SEMI_PLANNER_UV, // 5
    INPUT_YUV444_SEMI_PLANNER_VU, // 6
    INPUT_YUV422_PACKED_YUYV, // 7
    INPUT_YUV422_PACKED_UYVY, // 8
    INPUT_YUV422_PACKED_YVYU, // 9
    INPUT_YUV422_PACKED_VYUY, // 10
    INPUT_YUV444_PACKED_YUV, // 11
    INPUT_RGB, // 12
    INPUT_BGR, // 13
    INPUT_ARGB, // 14
    INPUT_ABGR, // 15
    INPUT_RGBA, // 16
    INPUT_BGRA, // 17
    INPUT_YUV420_SEMI_PLANNER_UV_10BIT, // 18
    INPUT_YUV420_SEMI_PLANNER_VU_10BIT, // 19
};
// 支持的输出格式
enum VpcOutputFormat {
    OUTPUT_YUV420SP_UV,
    OUTPUT_YUV420SP_VU
};
// 压缩格式的数据配置
struct VpcCompressDataConfigure
{
    uint64_t lumaHeadAddr;
    uint64_t chromaHeadAddr;
    uint32_t lumaHeadStride;
    uint32_t chromaHeadStride;
    uint64_t lumaPayloadAddr;
    uint64_t chromaPayloadAddr;
    uint32_t lumaPayloadStride;
    uint32_t chromaPayloadStride;
    VpcCompressDataConfigure()
    {
        lumaHeadAddr = 0;
        chromaHeadAddr = 0;
        lumaHeadStride = 0;
        chromaHeadStride = 0;
        lumaPayloadAddr = 0;
        chromaPayloadAddr = 0;
        lumaPayloadStride = 0;
        chromaPayloadStride = 0;
    }
};
// crop区域的配置
struct VpcUserCropConfigure {
    uint32_t leftOffset; // 左偏移
    uint32_t rightOffset; // 右偏移
    uint32_t upOffset; // 上偏移
    uint32_t downOffset; // 下偏移
    uint64_t reserve1;
    VpcUserCropConfigure()
    {
        leftOffset = 0;
        rightOffset = 0;
        upOffset = 0;
        downOffset = 0;
    }
};
// AI CORE有这个需求，需要统计输出yuv图像每个分量的总量
struct VpcUserYuvSum {
```



```
    uint32_t ySum;
    uint32_t uSum;
    uint32_t vSum;
    uint64_t reserve1;
    VpcUserYuvSum()
    {
        ySum = 0;
        uSum = 0;
        vSum = 0;
    }
};
// 用户抠图部分的输入数据的相关配置
struct VpcUserRoiInputConfigure {
    VpcUserCropConfigure cropArea;
    uint64_t reserve1;
};
// 用户抠图部分的输出数据的相关配置
struct VpcUserRoiOutputConfigure {
    uint8_t * addr; // 输出图片的首地址
    uint32_t bufferSize; // 输出buffer的大小,根据yuv420sp计算
    uint32_t widthStride; // 输出图片的宽stride
    uint32_t heightStride; // 输出为yuv420sp图像，需要根据heightStride计算出uv数据的起始地址
//用户指定输出区域坐标
    VpcUserCropConfigure outputArea;
    uint64_t reserve1;
    VpcUserRoiOutputConfigure()
    {
        addr = 0;
        bufferSize = 0;
        widthStride = 0;
        heightStride = 0;
    }
};
struct VpcUserRoiConfigure {
    VpcUserRoiInputConfigure inputConfigure;
    VpcUserRoiOutputConfigure outputConfigure;
    VpcUserRoiConfigure* next;
    uint64_t reserve1;
    VpcUserRoiConfigure ()
    {
        next = nullptr;
    }
};
// 性能调优预留接口
struct VpcUserPerformanceTuningParameter{
    uint64_t reserve1;
    uint64_t reserve2;
    uint64_t reserve3;
    uint64_t reserve4;
    uint64_t reserve5;
};
// 用户调用vpc的配置
struct VpcUserImageConfigure {
    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
    uint32_t bareDataBufferSize; // 内存大小是根据图像的宽高stride计算出来的，当图像为压缩格式时，
不需要填
    uint32_t widthStride; // 根据widthStride计算出下一行图像的地址,注意不同图像格式，stride与宽比例
不同
    uint32_t heightStride; // 如果是yuv semi planner图像，根据heightStride计算出uv数据的起始地址
enum VpcInputFormat inputFormat;
enum VpcOutputFormat outputFormat;
    VpcUserRoiConfigure* roiConfigure; // 配置抠图区域，支持一图多框
// 进过hfbc压缩过的输入图片数据的配置，压缩过的图片数据要配置多项值，此来源为vdec模块。
bool isCompressData;
    VpcCompressDataConfigure compressDataConfigure;
// AI CORE有统计输出图像每个分量的总量的需求，有这个需求时，只支持一图一框
// 如果想要得到yuvSum值，需要将yuvSumEnable 赋值为ture
bool yuvSumEnable; // 不要yuvSum值时，可以不填，默认值为false
    VpcUserYuvSum yuvSum;
```



```
    VpcUserPerformanceTuningParameter tuningParameter;//// 此变量为预留,用于参数调优
    uint32_t* cmdListBufferAddr; // 此变量为预留
    uint32_t cmdListBufferSize; // 此变量为预留
    uint64_t reserve1;
    uint64_t reserve2;
    VpcUserImageConfigure()
    {
        bareDataAddr = nullptr;
        bareDataBufferSize = 0;
        widthStride = 0;
        heightStride = 0;
        inputFormat = INPUT_YUV420_SEMI_PLANNER_UV;
        outputFormat = OUTPUT_YUV420SP_UV;
        roiConfigure = nullptr;
        isCompressData = false;
        yuvSumEnable = false;
        cmdListBufferAddr = nullptr;
        cmdListBufferSize = 0;
    }
};
```

出参说明

暂无。

约束条件

- 原图有效区域宽高必须为偶数。
- 抠图区域和贴图区域最小分辨分辨率为16*16，最大分辨率为4096*4096。
- 抠图区域奇数、偶数限制仍然与老接口保持一致，即左偏移和上偏移为偶数、右偏移和下偏移为奇数。贴图区域限制也相同，即即左偏移和上偏移为偶数、右偏移和下偏移为奇数。
- 抠图区域不超出输入图片，贴图区域不超出输出图片。
- 贴图区域相对输出图片的左偏移16对齐。

  说明

  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0，参考**图2-4**。
- 输入输出图片宽stride 128对齐，高stride 16对齐。宽stride最大为argb图像最大分辨率 * 4，即4096 * 4。
- 输入输出图片buffer地址在同一个4G空间，首地址128对齐。
- 宽高缩放范围：[1/32, 16]。
- 最大贴图个数为256个。

调用示例

示例1，原图缩放。

```
void nevVpcTest1()
{
    int32_t ret = 0;
    uint32_t inWidthStride = 1920;
    uint32_t inHeightStride = 1088;
    uint32_t outWidthStride = 1280;
    uint32_t outHeightStride = 720;
    uint32_t inBufferSize = inWidthStride * inHeightStride * 3 / 2; // 1080P yuv420sp图像
    uint32_t outBufferSize = outWidthStride * outHeightStride * 3 / 2; // 720P yuv420sp图像
// 构造伪数据
    uint8_t* inBuffer = (uint8_t*)HIAI_DMalloc(inBufferSize, MALLOC_DEFAULT_TIME_OUT,
```



```
HIAI_MEMORY_HUGE_PAGE); // 构造输入图片
    uint8_t* outBuffer = (uint8_t*)HIAI_DMalloc(outBufferSize, MALLOC_DEFAULT_TIME_OUT,
HIAI_MEMORY_HUGE_PAGE);// 构造输出图片
// 构造输入图片配置
    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;
// 设置抠图区域，抠图区域左上角坐标[0,0],右下角坐标[1919,1079]
    inputConfigure->cropArea.leftOffset  = 0;
    inputConfigure->cropArea.rightOffset = 1919;
    inputConfigure->cropArea.upOffset    = 0;
    inputConfigure->cropArea.downOffset  = 1079;
    VpcUserRoiOutputConfigure* outputConfigure = &roiConfigure->outputConfigure;
    outputConfigure->addr = outBuffer;
    outputConfigure->bufferSize = outBufferSize;
    outputConfigure->widthStride = outWidthStride;
    outputConfigure->heightStride = outHeightStride;
// 设置贴图区域，贴图区域左上角坐标[0,0],右下角坐标[1279,719]
    outputConfigure->outputArea.leftOffset  = 0;
    outputConfigure->outputArea.rightOffset = 1279;
    outputConfigure->outputArea.upOffset    = 0;
    outputConfigure->outputArea.downOffset  = 719;
    imageConfigure->roiConfigure = roiConfigure.get();

    IDVPPAPI *pidvppapi = nullptr;
    ret = CreateDvppApi(pidvppapi);
if (ret != 0) {
        printf("create dvpp api fail.\n");
        UNMAP(inBuffer, inBufferSize);
        UNMAP(outBuffer, outBufferSize);
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 (0 != ret) {
        printf("call vpc dvppctl process faild!\n");
        ret = DestroyDvppApi(pidvppapi);
return;
    } else {
        printf("call vpc dvppctl process success!\n");
    }
    ret = DestroyDvppApi(pidvppapi);
    UNMAP(inBuffer, inBufferSize);
    UNMAP(outBuffer, outBufferSize);
return;
}
```

示例2，抠图缩放，结果输出到输出图片指定位置。

```
void nevVpcTest2()
{
    int32_t ret = 0;
    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图像
```



```
    uint32_t outBufferSize = outWidthStride * outHeightStride * 3 / 2; // 720P yuv420sp图像
// 构造伪数据
    uint8_t* inBuffer = (uint8_t*)HIAI_DMalloc(inBufferSize, MALLOC_DEFAULT_TIME_OUT,
HIAI_MEMORY_HUGE_PAGE); // 构造输入图片
    uint8_t* outBuffer = (uint8_t*)HIAI_DMalloc(outBufferSize, MALLOC_DEFAULT_TIME_OUT,
HIAI_MEMORY_HUGE_PAGE); // 构造输出图片
// 构造输入图片配置
    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;
// 设置抠图区域，抠图区域左上角坐标[100,100],右下角坐标[499,499]
    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;
// 设置贴图区域，贴图区域左上角坐标[256,200],右下角坐标[399,399]
    outputConfigure->outputArea.leftOffset  = 256; // 这个偏移值需要16对齐
    outputConfigure->outputArea.rightOffset = 399;
    outputConfigure->outputArea.upOffset    = 200;
    outputConfigure->outputArea.downOffset  = 399;
    imageConfigure->roiConfigure = roiConfigure.get();

    IDVPPAPI *pidvppapi = nullptr;
    ret = CreateDvppApi(pidvppapi);
if (ret != 0) {
        printf("create dvpp api fail.\n");
        UNMAP(inBuffer, inBufferSize);
        UNMAP(outBuffer, outBufferSize);
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 (0 != ret) {
        printf("call vpc dvppctl process faild!\n");
        ret = DestroyDvppApi(pidvppapi);
return;
    } else {
        printf("call vpc dvppctl process success!\n");
    }
    ret = DestroyDvppApi(pidvppapi);
    UNMAP(inBuffer, inBufferSize);
    UNMAP(outBuffer, outBufferSize);
return;
}
```

示例3，抠多张图片。

```
void nevVpcTest3(int& ret)
{
    int32_t ret = 0;
    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图像
    uint32_t outBufferSize = outWidthStride * outHeightStride * 3 / 2; // 720P yuv420sp图像
// 构造伪数据
    uint8_t* inBuffer = (uint8_t*)HIAI_DMalloc(inBufferSize, MALLOC_DEFAULT_TIME_OUT,
HIAI_MEMORY_HUGE_PAGE);// 构造输入图片
// 构造输入图片配置
    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> lastRoi;
for (int i = 0; i < 5; i++) {
        std::shared_ptr<VpcUserRoiConfigure> roiConfigure(new VpcUserRoiConfigure);
        roiConfigure->next = nullptr;
        VpcUserRoiInputConfigure* inputConfigure = &roiConfigure->inputConfigure;
// 设置抠图区域
        inputConfigure->cropArea.leftOffset  = 100 + i * 16;
        inputConfigure->cropArea.rightOffset = 499 + i * 16;
        inputConfigure->cropArea.upOffset    = 100 + i * 16;
        inputConfigure->cropArea.downOffset  = 499 + i * 16;
        VpcUserRoiOutputConfigure* outputConfigure = &roiConfigure->outputConfigure;
        uint8_t* outBuffer = (uint8_t*)HIAI_DMalloc(outBufferSize, MALLOC_DEFAULT_TIME_OUT,
HIAI_MEMORY_HUGE_PAGE);// 构造输出图片
        outputConfigure->addr = outBuffer;
        outputConfigure->bufferSize = outBufferSize;
        outputConfigure->widthStride = outWidthStride;
        outputConfigure->heightStride = outHeightStride;
// 设置贴图区域
        outputConfigure->outputArea.leftOffset  = 256 + i * 16; // 这个偏移值需要16对齐
        outputConfigure->outputArea.rightOffset = 399 + i * 16;
        outputConfigure->outputArea.upOffset    = 399 + i * 16;
        outputConfigure->outputArea.downOffset  = 399 + i * 16;
if (i == 0) {
            imageConfigure->roiConfigure = roiConfigure.get();
            lastRoi = roiConfigure;
        } else {
            lastRoi->next = roiConfigure.get();
            lastRoi = roiConfigure;
        }
    }

    IDVPPAPI *pidvppapi = nullptr;
    ret = CreateDvppApi(pidvppapi);
if (ret != 0) {
        printf("create dvpp api fail.\n");
        UNMAP(inBuffer, inBufferSize);
while (imageConfigure->roiConfigure != nullptr) {
            UNMAP(imageConfigure->roiConfigure->outputConfigure.addr, outBufferSize);
            imageConfigure->roiConfigure = imageConfigure->roiConfigure->next;
        }
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 (0 != ret) {
        printf("call vpc dvppctl process faild!\n");
        ret = DestroyDvppApi(pidvppapi);
return;
```



```
    } else {
        printf("call vpc dvppctl process success!\n");
    }
    ret = DestroyDvppApi(pidvppapi);
    UNMAP(inBuffer, inBufferSize);
while (imageConfigure->roiConfigure != nullptr) {
        UNMAP(imageConfigure->roiConfigure->outputConfigure.addr, outBufferSize);
        imageConfigure->roiConfigure = imageConfigure->roiConfigure->next;
    }
return;
}
```

2.2.2 VDEC

功能

实现视频的解码。

- VDEC支持两种输入格式：
  H264 bp/mp/hp level5.1 yuv420sp编码的码流。
  H265 8/10bit level5.1 yuv420sp编码的码流。
- VDEC输出格式为：yuv420sp压缩后的HFBC数据。

VDEC 性能指标如下

场景	总帧率
1080p ≥ 2进程（进程数无限制）	480fps
1080p 单进程（2≤线程数≤16）	320fps
1080p单进程 单线程	240fps
4k ≥ 2进程（进程数无限制）	120fps
4k 单进程（2≤线程数≤16）	80fps
4k 单进程 单线程	60fps

VDEC比较特殊，由于当前内部不是单例模式，所以其对外提供C接口函数与其他不同。

VDEC 对外提供 C 接口函数为

- **int CreateVdecApi(IDVPPAPI *&pIDVPPAPI, int singleton)**

函数原型	int CreateVdecApi(IDVPPAPI *&pIDVPPAPI, int singleton)
功能	获取vdecapi实例，相当于vdec执行器句柄。调用方可以使用申请到的vdecapi实例进行vdec的调用，可以跨函数调用，跨线程调用。调用方负责vdecapi实例的生命周期，即申请与释放，申请使用CreateVdecApi函数，释放使用DestroyVdecApi函数。

输入说明	输入为**IDVPPAPI**类型指针引用，输入指针必须为NULL。 输入singleton为内部保留使用，为以后实现pIDVPPAPI单例预留，建议调用方当前设置为0。
输出说明	输出为**IDVPPAPI**类型指针引用，输出可能为NULL，可能不为NULL，获取失败则为NULL，成功则不为NULL。
返回值说明	返回值0代表成功，-1代表失败。
使用说明	调用方创建**IDVPPAPI**对象指针，初始化为NULL，调用CreateVdecApi函数将**IDVPPAPI**对象指针传入。如果申请成功CreateVdecApi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。
使用限制	此函数为VDEC执行对外提供的统一调用接口之一，其主要作用相当于获取VDEC执行器句柄。

- **int VdecCtl(IDVPPAPI *&pIDVPPAPI, int CMD, dvppapi_ctl_msg *MSG, int singleton)**

函数原型	**int VdecCtl(IDVPPAPI *&pIDVPPAPI, int CMD, dvppapi_ctl_msg *MSG, int singleton)**
功能	控制DVPP执行器进行视频解码。
输入说明	输入为**IDVPPAPI**类型指针引用。 控制命令字CMD(VDEC为DVPP_CTL_VDEC_PROC)。 vdec执行器配置信息MSG，类型为**dvppapi_ctl_msg**，其中此结构体中的in和out请见**入参：vdec_in_msg**和**出参：vdec_out_msg**。 输入singleton为内部保留使用，为以后实现pIDVPPAPI单例预留，建议调用方当前设置为0。
输出说明	输出为MSG的配置信息中的输出buffer，以及vdec的输出状态信息(都存储于MSG中)。
返回值说明	返回值0代表成功，-1代表失败。
使用说明	调用方调用**VdecCtl**函数，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。
使用限制	此函数为VDEC执行对外提供的统一调用接口，其主要作用为调用vdec执行器中的具体功能执行。

- **int DestroyVdecApi(IDVPPAPI *&pIDVPPAPI, int singleton)**

函数原型	**int DestroyVdecApi(IDVPPAPI *&pIDVPPAPI, int singleton)**
功能	释放vdecapi，关闭vdec执行器。

输入说明	输入为**IDVPPAPI**类型指针引用。 输入singleton为内部保留使用，为以后实现pIDVPPAPI单例预留，建议调用方当前设置为0。
输出说明	无输出。
返回值说明	返回值0代表成功，-1代表失败。
使用说明	无特殊说明。
使用限制	此函数为VDEC执行对外提供的统一调用接口，其主要作用为释放VDEC执行资源。

入参：vdec_in_msg

成员变量	说明
char video_format[10]	输入视频格式，“h264”或者“h265”，默认是“h264”， 仅支持yuv420sp(NV12、NV21)编码后的h264和h265码流。
char image_format[10]	输出帧格式，“nv12”或者“nv21”，默认是“nv12”。
void (*call_back)(FRAME* frame,void * hiai_data)	调用方回调函数，FRAME为vdec解码后的输出结构体，详见**A.1.3 vdec_in_msg中的结构体和类•FRAME结构体**。
char* in_buffer	输入视频码流内存，此码流为h264或h265裸码流。
int in_buffer_size	输入视频码流内存大小。
void * hiai_data	解码后输出结果帧回调函数的形参指针，指针指向对象由调用方定义具体的结构。 **注意** 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。
std::shared_ptr<HIAI_DATA_SP> hiai_data_sp	调用方回调函数形参指针，其中HIAI_DATA_SP为VDEC内部定义的父类，具体类HIAI_DATA_SP结构请见**A.1.3 vdec_in_msg中的结构体和类•HIAI_DATA_SP类**，用户可以继承衍生子类，使用方式可以参考3.3.2.7示例代码。此指针指向的类中可以设置帧序号等信息，但只支持一帧对应一个帧序号，不支持多帧对应一个帧序号。 **注意** hiai_data_sp智能指针的使用将屏蔽hiai_data，即hiai_data_sp和hiai_data只能选其一。

成员变量	说明
int channelId	输入码流对应的解码通道的Id，不同码流同时解码需设置不同的值，取值范围0~15。
void (*err_report)(VDECERR* vdecErr)	错误上报回调函数，用于将解码过程中出现的异常通知用户，比如码流错误、硬件问题、解码器状态错误等，以便用户决定如何处理。其中形参VDECERR为vdec内部定义结构体，具体请见**A.1.3 vdec_in_msg中的结构体和类•VDECERR结构体**。
bool isEOS	码流结束标志，标识本路解码结束。用户可以不用关心此标志，默认在DestroyVdecApi接口中会将isEOS设置true，并在内部实现结束本路解码同时释放资源。若用户主动配置isEOS为true，则在调用VdecCtl接口中优先使用用户配置，结束本路解码并释放资源。具体使用请见**调用示例**。

出参：vdec_out_msg

暂无，其中输出请参见**A.1.3 vdec_in_msg中的结构体和类•FRAME结构体**，用户可读取该内存得到VDEC输出结果。

调用示例

说明：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。

```
//自定义的子类，因是示例代码，所以此子类除了继承父类实现了析构函数外，未作其他任何改动
class HIAI_DATA_SP_SON: public HIAI_DATA_SP
{
public:
           ~ HIAI_DATA_SP_SON ()
           {
               //destruct here;
           }
};

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

//回调函数举例，调用方需根据自己的需求来重新定义回调函数
void frame_return(FRAME * frame,void * hiai_data)
{
    static int image_cnt = 0;
    image_cnt++;
    printf("call save frame number:[%d], width:[%d], height:[%d]\n", image_cnt, frame->width,
frame->height);

    if(hiai_data != NULL){
            //如果采用了智能指针，此处需要转成用户自定义的类指针；如果是普通指针，则转换为用户自定义的结构类型
            HIAI_DATA_SP_SON * hiai_data_sp = (HIAI_DATA_SP_SON*) hiai_data;
            printf("frame index : %d\n", hiai_data_sp->getFrameIndex());
    }

    //call vpc interface to decompress hfbc
```

```
    printf("call vpc interface\n");
    if (pidvppapi_vpc != NULL) {
        dvppapi_ctl_msg dvppApiCtlMsg;
        vpc_in_msg vpcInMsg;
        vpcInMsg.format = 0;  //YUV420_SEMI_PLANNAR=0
        vpcInMsg.rank = 1;    //nv12 =0 ,nv21 =1
        vpcInMsg.bitwidth = frame->bitdepth;
        vpcInMsg.width = frame->width;
        vpcInMsg.high = frame->height;
        shared_ptr<AutoBuffer> auto_out_buffer = make_shared<AutoBuffer>();
        vpcInMsg.in_buffer = (char *)frame->buffer;
        vpcInMsg.in_buffer_size = frame->buffer_size;
        vpcInMsg.rdma.luma_head_addr = (long)(frame->buffer + frame->offset_head_y);
        vpcInMsg.rdma.chroma_head_addr = (long)(frame->buffer + frame->offset_head_c);
        vpcInMsg.rdma.luma_payload_addr = (long)(frame->buffer + frame->offset_payload_y);
        vpcInMsg.rdma.chroma_payload_addr = (long)(frame->buffer + frame-
>offset_payload_c);
        vpcInMsg.rdma.luma_head_stride = frame->stride_head;
        vpcInMsg.rdma.chroma_head_stride = frame->stride_head;
        vpcInMsg.rdma.luma_payload_stride = frame->stride_payload;
        vpcInMsg.rdma.chroma_payload_stride = frame->stride_payload;
        vpcInMsg.cvdr_or_rdma = 0;
        vpcInMsg.hmax = frame->realWidth-1;
        vpcInMsg.hmin = 0;
        vpcInMsg.vmax = frame->realHeight-1;
        vpcInMsg.vmin = 0;
        vpcInMsg.vinc = 1;
        vpcInMsg.hinc = 1;
        vpcInMsg.auto_out_buffer_1 = auto_out_buffer;
        dvppApiCtlMsg.in = (void*)(&vpcInMsg);
        dvppApiCtlMsg.in_size = sizeof(vpc_in_msg);

        if (DvppCtl(pidvppapi_vpc,DVPP_CTL_VPC_PROC,&dvppApiCtlMsg) != 0) {
           printf("call dvppctl fail\n");
            return;
        }

        char filename_vpc[30];
        int ret = memset_s(filename_vpc, 30, 0, 30);
        if (ret != 0) {
            printf("memset_s fail\n");
        }

        ret = strncpy_s(filename_vpc, sizeof(filename_vpc), "output_dir/image_vpc",
strlen("output_dir/image_vpc"));
        if (ret != 0) {
            printf("strncpy_s fail\n");
            return;
        }

        ret = sprintf_s(filename_vpc + strlen("output_dir/image_vpc"), sizeof(filename_vpc) -
strlen("output_dir/image_vpc"), "%d",image_cnt);
        if (ret == -1) {
            printf("sprintf_s fail\n");
            return;
        }

        FILE *fp_vpc = NULL;
        fp_vpc = fopen(filename_vpc, "wb+");
        if (fp_vpc == NULL) {
            printf("open file %s failed~\n", filename_vpc);
            return;
        }
        fwrite(vpcInMsg.auto_out_buffer_1->getBuffer(), 1, vpcInMsg.auto_out_buffer_1-
>getBufferSize(), fp_vpc);
        fflush(fp_vpc);
        fclose(fp_vpc);
        printf("save vpc result:%s, write size: %d\n",filename_vpc, vpcInMsg.auto_out_buffer_1-
>getBufferSize());
```



```
        }
        else {
         printf("create vpc fail\n");
        }
    DestroyDvppApi(pidvppapi_vpc);
    return;
}
//测试函数主入口
int main()
{
    char output_dir[20];
    int ret = 0;
    ret = memset_s(output_dir, 20, 0, 20);
    if (ret != 0) {
        printf("memset_s fail\n");
        return;
    }

    ret = strncpy_s(output_dir, sizeof(output_dir), "output_dir", strlen("output_dir"));
    if (ret !=0) {
        printf("strncpy_s fail\n");
        return;
    }

    if (access(output_dir, 0) == -1) {
        if (mkdir(output_dir, S_IREAD | S_IWRITE) < 0) {
            printf("create dir failed.\n");
            return;
        }
    }

        //初始化vpc实例对象
        ret = CreateDvppApi(pidvppapi_vpc);
        //初始化vdec实例对象
        ret = CreateVdecApi(pidvppapi_vdec, 0);

        if(ret == 0 && pidvppapi_vpc != NULL && pidvppapi_vdec != NULL){
         FILE* fp = fopen(in_file_name,"rb");
         if(fp != NULL){
               fseek(fp,0L,SEEK_END);
               int file_size = ftell(fp);
               fseek(fp,0L,SEEK_SET);
               int rest_len=file_size;
               int len = file_size;//2*1024*1024;

               vdec_in_msg vdec_msg;
               //用户自定义frame_return函数，可参考上面的frame_return函数示例
                vdec_msg.call_back = frame_return;
               //用户自定义hiai_data结构，本处不使用设为NULL
                vdec_msg.hiai_data = NULL;
               //用户设置解码通道Id
                vdec_msg.channelId = 0;
               //用户设置码流协议类型
                if(gFormat == 0){
                      memcpy(vdec_msg.video_format,"h264",4);
                }else{
                      memcpy(vdec_msg.video_format,"h265",4);
               }
               //用户设置输出格式类型，本处为nv12
                memcpy(vdec_msg.image_format,"nv12",4);
               //用户自己申请存放码流的内存
               vdec_msg.in_buffer = (char*)malloc(len);
               if(vdec_msg.in_buffer != NULL){
                      dvppapi_ctl_msg dvppApiCtlMsg;
                      dvppApiCtlMsg.in = (void*)(&vdec_msg);
                      dvppApiCtlMsg.in_size = sizeof(vdec_in_msg);

                      int cnt = 0;
                      while(rest_len>0){
```

```
                                int read_len = fread(vdec_msg.in_buffer,1,len,fp);
                                printf("rest_len is %d,read len is %d.\n",rest_len,read_len);
                                vdec_msg.in_buffer_size = read_len;
                                //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
                                std::shared_ptr< HIAI_DATA_SP_SON> dc_sp = make_shared<
HIAI_DATA_SP_SON>();
                                dc_sp->setFrameIndex(++cnt);
                                vdec_msg.hiai_data_sp = 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
                                vdec_msg.channelId = 0;
                                //调用VdecCtl接口解码
                                if (0!= VdecCtl(pidvppapi_vdec, DVPP_CTL_VDEC_PROC,
&dvppApiCtlMsg, 0))
                                {
                                      printf("send stream data, call dvppctl process failed!\n");
                                }
                                rest_len=rest_len-len;
                          }
                          //下面几行主动设置eos标志并调用VdecCtl接口发送，可以不设置，若不设置，
DestroyVdecApi接口中默认设置
                          vdec_msg.isEOS = true;
                          if (0!= VdecCtl(pidvppapi_vdec, DVPP_CTL_VDEC_PROC, &dvppApiCtlMsg, 0)
                                {
                                      printf("send eos flag, call dvppctl process failed!\n");
                                }
                          //释放buffer
                          free(vdec_msg.in_buffer);
                    }
             fclose(fp);
             }
           }
           DestroyVdecApi(pidvppapi_vdec, 0);
           DestroyDvppApi(pidvppapi_vpc);
           return ;
}
```

2.2.3 JPEGE

功能：实现编码生成.jpg图片。

- JPEGE支持以下格式输入：
  - YUV422 packed（yuyv,yvyu,uyvy,vyuy）
  - YUV420 Semi-planar（NV12，NV21）
- JPEGE输出格式为：

  jpeg
- 内存限制
  - 缓冲区起始地址应 128 对齐；
  - 同一次任务的缓冲区的输入和输出区的虚拟地址应该在同一 4G 空间；

JPEGE性能指标如下：

场景	总帧率
1080p ≥ 1路（路数无限制）	64fps
4k ≥ 1路（路数无限制）	16fps

入参：sJpegeIn

成员变量	说明
eEncodeFormat format	输入YUV数据的类型，支持YUV422 packed（yuyv,yvyu,uyvy,vyuy）和YUV420 Semi-planar（NV12，NV21） typedef enum { JPGENC_FORMAT_UYVY = 0x0, JPGENC_FORMAT_VYUY = 0x1, JPGENC_FORMAT_YVYU = 0x2, JPGENC_FORMAT_YUYV = 0x3, JPGENC_FORMAT_NV12 = 0x10, JPGENC_FORMAT_NV21 = 0x11, } eEncodeFormat;
unsigned char* buf	yuv输入数据,需要调用方申请，需要合理对齐，见参数stride，heightAligned。
unsigned long bufSize	输入buf长度，对齐后的数据长度并对齐到 PAGE_SIZE(4k)。
unsigned int width	输入图片的宽度，范围[32,8192]。
unsigned int height	输入图片的高度，范围[32,8192]。
unsigned int stride	输入图片对齐后宽度，对齐到16，兼容对齐到16的倍数如128，对于YUV422packed数据，stride应该为width的两倍对齐到16。
unsigned int heightAligned	输入图片对齐后高度，可与高度相同不对齐，兼容对齐如16对齐。
unsigned int level	编码质量范围(0 100]，数值越小输出图片质量越差。

出参：sJpegeOut

成员变量	说明
unsigned char* jpg_data	输出缓冲区中jpg数据的起始地址。
unsigned int jpgSize	编码后的jpg图片数据长度。
JpegCalBackFree	释放输出buf的回调函数。

调用示例

```
void TEST_5() // jpeg encoder base function
{
```

```
    sJpegeIn  inData;
    sJpegeOut outData;

    inData.width         = gWidth;
    inData.height        = gHigh;
    inData.heightAligned = gHigh;
    inData.format        = (eEncodeFormat)gFormat;
    inData.level         = 100;

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

    void* addrOrig = HIAI_DMalloc(inData.bufSize + 128, MALLOC_DEFAULT_TIME_OUT,
HIAI_MEMORY_HUGE_PAGE);

    if (addrOrig == MAP_FAILED) {
        printf("TEST_5, can not alloc input buffer\n");
        return;
    }
    inData.buf = (unsigned char*)ALIGN_UP( (uint64_t)addrOrig, 128 );

    do{
        int err = 0;
        // load img file
        FILE* fpIn = fopen( in_file_name, "rb" );
        if (fpIn == NULL) {
            printf("TEST_5, can not open input file\n");
            break;
        }

        if (JPGENC_FORMAT_YUV420 == (inData.format&JPGENC_FORMAT_BIT) ) {
            // for yuv422packed format, like uyvy / vyuy / yuyv / yvyu
            printf("TEST_5, input yuv 420 data\n");
            // 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 yuv420semi-planar format, like nv12 / nv21
            printf("TEST_5, input yuv 422 data\n");
            for(uint32_t j = 0; j < inData.height; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width * 2, fpIn );
            }

        }
        fclose( fpIn );
        if (err == -1) {
            break;
        }

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



```
        IDVPPAPI *pidvppapi = NULL;
        CreateDvppApi(pidvppapi);
        if (pidvppapi == NULL) {
            printf("TEST_5, can not open dvppapi engine\n");
            break;
        }

        JpegeProcessBranch(pidvppapi, dvppApiCtlMsg, outData);

        DestroyDvppApi(pidvppapi);

    } while(0); // for resource inData.buf
    if(addrOrig != NULL) {
        munmap( addrOrig, ALIGN_UP( inData.bufSize+128 , MAP_2M ) );
    }

}
```

2.2.4 JPEGD

功能是：实现.jpg图片的解码。

- JPEGD支持如下图片格式输入：

  jpeg (colorspace: yuv, subsample: 444/422/420/400)只支持huffman编码，不支持算术编码，不支持渐进编码，不支持jpeg2000格式。
- JPEGD支持的分辨率为：

  最大分辨率：8192 x 8192

  最小分辨率：32 x 32
- JPEGD输出格式为：

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

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

  jpeg(420) -> yuv420 半平面V在前U在后。

  jpeg(400) -> yuv420，uv数据采用0 x 80填充。
- 内存限制
  - 缓冲区起始地址应 128 对齐；
  - 同一次任务的缓冲区的输入和输出区的虚拟地址应该在同一 4G 空间；

JPEGD性能指标如下：

场景	总帧率
1080p * 1路	128fps
1080p ≥ 2路（路数无限制）	256fps
4k * 1路	32fps
4k ≥ 2路（路数无限制）	64fps

注意

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接口供调用方使用，所以JPEGD暂时支持两套接口供用户使用。

入参：JpegdIn

B590版本更新接口。

成员变量	说明
unsigned char* jpegData	输入jpg图片数据，起始地址128对齐，2M大页表方式申请, 同一次任务缓冲区虚拟地址应在同一4G 空间。
uint32_t jpegDataSize	输入buf的长度，硬件约束比实际码流长度长8 byte。
bool IsYUV420Need	是否需要yuv420 semi-planar输出数据。true（是），false（否）。Jpegd支持raw格式和yuv420的半平面格式输出。其中灰度图片输出的yuv420为fake420形式。
bool isVBeforeU	Jpged输出v在u前，字段保留。

出参：JpegdOut

B590版本更新接口。

成员变量	说明
unsigned char* yuvData	输出yuv图片数据buf，该图片的宽高为对齐后的宽高。
uint32_t yuvDataSize	输出yuv数据的长度，数据长度由对齐后的宽高计算。
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	释放输出buf的回调函数，详见调用示例中的回调函数示例。

成员变量	说明

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

入参：jpegd_raw_data_info

成员变量	说明
unsigned char* jpeg_data	输入jpg图片数据，起始地址128对齐，2M大页表方式申请。
uint32_t jpeg_data_size	输入buf的长度，硬件约束比实际码流长度长8 byte。

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

成员变量	说明
bool IsYUV420Need	是否需要yuv420 semi-planar输出数据。Jpegd支持raw格式和yuv420的半平面格式输出。其中灰度图片输出的yuv420为fake420形式。
bool isVBeforeU	Jpged输出只支持true，v在u前，字段保留。

出参：jpegd_yuv_data_info

成员变量	说明
unsigned char* yuv_data	输出yuv图片数据buf，该图片的宽高为对齐后的宽高。
uint32_t yuv_data_size	输出yuv数据的长度，数据长度由对齐后的宽高计算。
uint32_t img_width	输出yuv图片的宽度。
uint32_t img_height	输出yuv图片的高度。
uint32_t img_width_aligned	输出图片的对齐后的宽度，对齐到128。
uint32_t img_height_aligned	输出图片的对齐后的高度，对齐到16。
JpegCalBackFree cbFree	释放输出buf的回调函数，详见调用示例。

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

调用示例

- 回调函数示例：

```
/**
* 硬解和软解(硬编和软编)输出内存的申请和释放不一致，封装了function对象，
* 来屏蔽输出内存的释放细节，且需要具备bind的作用，将二元函
* 数和一元函数处理成无参函数，方便作为回调函数提供给调用方调用;
* usage: outBuf.cbFree = JpegCalBackFree( .. )
*        outBuf.cbFree()
```

```
*/
class JpegCalBackFree
{
 public:
 JpegCalBackFree( )
 : isMunmapUse_( false ), addr4Free_( NULL ), siz4Free_( 0 ) {}
 void setBuf( void* addr4Free )
 {
  isMunmapUse_ = false;
  addr4Free_   = addr4Free;
 }
 void setBuf( void* addr4Free, size_t siz4Free )
 {
  isMunmapUse_ = true;
  addr4Free_   = addr4Free;
  siz4Free_    = siz4Free;
 }
 void operator() ()
 {
  // if isMunmapUse_ is true, use munmap() to free the buf
  if( isMunmapUse_  && NULL != addr4Free_ )
  {
   if( -1 == munmap( addr4Free_, ALIGN_UP( siz4Free_ , MAP_2M ) ) )
   std::cerr << "free buf with munmap error" << std::endl;
  }
  // if isMunmapUse_ is false, use free() to free the buf
  if( !isMunmapUse_ && NULL != addr4Free_ )  free( addr4Free_ );
  addr4Free_ = NULL;  // safe free
 }
 private:
 // no copyable to avoid double free, private copy constructor and =
 JpegCalBackFree( const JpegCalBackFree& );
 const JpegCalBackFree& operator= ( const JpegCalBackFree& );
 private:
 bool   isMunmapUse_;
 void* addr4Free_;
 size_t siz4Free_;
};
```

- 功能调用示例：

```
void TEST_4() // jpegd base test
{
 struct jpegd_raw_data_info jpegdInData;     // 输入结构体
 struct jpegd_yuv_data_info jpegdOutData;    // 输出结构体
 if(gRank) jpegdInData.IsYUV420Need = false; // rRank 用户输入
 FILE *fpIn = fopen(in_file_name, "rb");
 if (NULL == fpIn)
 {
  printf("can not open file %s.\n", in_file_name);
  return;
 }
 do{// for resource fpIn
  fseek(fpIn, 0, SEEK_END);
  uint32_t fileLen   = ftell(fpIn);
  // the buf len should 8 byte larger, the driver asked
  jpegdInData.jpeg_data_size = fileLen + 8;
  fseek(fpIn, 0, SEEK_SET);
  // 大页表方式申请输入 buf, jpeg_data_size+128为了起始地址 128 字节对齐
        unsigned char* addrOrig = (unsigned char*)HIAI_DMalloc(jpegdInData.jpeg_data_size
+128, MALLOC_DEFAULT_TIME_OUT, HIAI_MEMORY_HUGE_PAGE);
  if( NULL == addrOrig )
  {
   API_LOGE("TEST_4, can not alloc input buffer\n");
   break;
  }
  // 起始地址 128 字节对齐
  jpegdInData.jpeg_data = (unsigned char*)ALIGN_UP( (uint64_t)addrOrig, 128 );
  do{ // for resource inBuf
   // 读入输入文件
   fread( jpegdInData.jpeg_data, 1, fileLen, fpIn );
```



```
  // dvppCtl 输入数据
  dvppapi_ctl_msg dvppApiCtlMsg;
  dvppApiCtlMsg.in = (void *)&jpegdInData;
  dvppApiCtlMsg.in_size = sizeof(jpegdInData );
  dvppApiCtlMsg.out = (void *)&jpegdOutData;
  dvppApiCtlMsg.out_size = sizeof(jpegdOutData);
  IDVPPAPI *pidvppapi = NULL;
  CreateDvppApi(pidvppapi);  // 创建 dvppApi 句柄
  if (pidvppapi)
  {
   // 调用 DVPP - JPEGD 处理
   if (0 != DvppCtl(pidvppapi, DVPP_CTL_JPEGD_PROC, &dvppApiCtlMsg))
   {
   API_LOGE("call dvppctl process failed\n");
   DestroyDvppApi(pidvppapi);
   break;
   }
   // 释放 dvppapi 句柄
   DestroyDvppApi(pidvppapi);
  }
  else
  {
   API_LOGE("can not create dvpp api\n");
   break;
  }
  char outFl[56] = "";
  snprintf(outFl, sizeof(outFl), "dvppapi_jpegd_w%d_h%d_%d.yuv",
  jpegdOutData.img_width_aligned,
  jpegdOutData.img_height_aligned,
  jpegdOutData.out_format);
  FILE* fpOut = fopen( outFl, "wb" );
  if( fpOut )
  {
   fwrite( jpegdOutData.yuv_data, 1, jpegdOutData.yuv_data_size, fpOut );
   fflush( fpOut );
   fclose( fpOut );
  }
  else API_LOGE("can not open output file %s \n", outFl );
  jpegdOutData.cbFree(); // 释放输出 buf
 }while(0);// for resource inBuf
 if( NULL != addrOrig ) munmap( addrOrig, ALIGN_UP( jpegdInData.jpeg_data_size+128 ,
MAP_2M ) ); // 释放输入 buf
}while(0); // for resource fpIn
fclose( fpIn );
return;
}
```

2.2.5 PNGD

功能：实现PNG格式图片的硬件解码。

- PNGD支持以下两种输入格式：
  RGBA、RGB
- PNGD输出格式为：
  RGBA、RGB
- PNGD支持的分辨率
  最大分辨率4096 x 4096，最小分辨率 32 x 32。
- 内存限制
  - 缓冲区起始地址应 128 对齐；
  - 同一次任务的缓冲区的输入和输出区的虚拟地址应该在同一 4G 空间；

PNGD性能指标如下：

场景	总帧率
1080p * 1路	4fps
1080p * 2路	8fps
1080p * 3路	12fps
1080p * 4路	16fps
1080p * 5路	20fps
1080p ≥ 6路（路数无限制）	24fps
4k * 1路	1fps
4k * 2路	2fps
4k * 3路	3fps
4k * 4路	4fps
4k * 5路	5fps
4k ≥ 6路（路数无限制）	6fps

入参：struct PngInputInfoAPI

成员变量	说明
void* inputData	输入图像数据。
unsigned long inputSize	输入buf长度，用于校验输入数据。
void* address	内部申请的内存地址，无需用户配置。
unsigned long size	申请内存大小，无需用户配置。
int transformFlag	转换标志，1表示RGBA转换到RGB，0保留原格式。

出参：struct PngOutputInfoAPI

成员变量	说明
void* outputData	输出图像数据。
unsigned long outputSize	输出buf长度。
void* address	输出内存地址，需用户手动释放。
unsigned long size	内存大小。

成员变量	说明
int format	输出图像格式： ● 2表示RGB输出。 ● 6表示RGBA输出。
unsigned int width	输出图像宽度。
unsigned int high	输出图像高度。
unsigned int widthAlign	宽度内存对齐，图片一行数据占用的内存大小进行128位对齐。若输出格式为RGB，则width*3后128位对齐，若输出格式为RGBA，则width*4后128位对齐。
unsigned int highAlign	高度对齐，目前为16位对齐。

调用示例

```
int test() // pngd 基本用例
{
 // load test file
 FILE *fpIn = fopen("dvpp_png_decode_001", "rb");
 if(NULL == fpIn){
  printf("can not open file dvpp_png_decode_001\n");
  return -1;
 }
 fseek(fpIn, 0, SEEK_END);
 uint32_t fileLen = ftell(fpIn);
 fseek(fpIn, 0, SEEK_SET);
 void* inbuf = malloc(fileLen);
 if(NULL == inbuf){
  printf("can not alloc memmory\n");
  fclose(fpIn);
  return -1;
 }else{
     fread(inbuf, 1, fileLen, fpIn);
 }
 fclose(fpIn);
 // prepare msg
 struct PngInputInfoAPI inputInfo; // 输入数据
    inputInfo.inputData = inbuf; // 输入png数据
    inputInfo.inoutSize = fileLen; // 输入png数据长度
    inputInfo.transformFlag = 0; // 是否格式转换
    struct PngOutputInfoAPI outputInfo; // 输出数据
    dvppapi_ctl_msg dvppApiCtlMsg; // 调用接口的 msg
    dvppApiCtlMsg.in       = (void*)(&inputInfo);
    dvppApiCtlMsg.in_size  = sizeof(struct PngInputInfoAPI);
    dvppApiCtlMsg.out      = (void*)(&outputInfo);
    dvppApiCtlMsg.out_size = sizeof(struct PngOutputInfoAPI);
    // use interface
    IDVPPAPI *pidvppapi = NULL;
    CreateDvppApi(pidvppapi); // 创建 dvppapi 调用对象
    if(pidvppapi){ // 调用 DvppCtl 接口进行 DVPP_CTL_PNGD_PROC 处理
    if(0 != DvppCtl(pidvppapi, DVPP_CTL_PNGD_PROC, &dvppApiCtlMsg)){
        printf("call dvppctl process failed\n");
        DestroyDvppApi(pidvppapi); // 销毁 dvppapi 对象
        return -1;
    }
    }else{
        printf("can not get dvpp api\n");
    }
    free(inbuf);
```



```
    munmap(outputInfo.address, ALIGN_UP(outputInfo.size, MAP_2M));
    DestroyDvppApi(pidvppapi);
}
```

2.2.6 VENC

功能

实现YUV420/YVU420图片数据的编码。

- VENC支持以下格式输入：
  YUV420 semi-planner NV12/NV21-8bit
- VENC输出格式为：
  H264 BP/MP/HP
  H265 MP

VENC 性能指标

场景	总帧率
1080p * 1路（不支持多路）	30fps

入参：venc_in_msg

成员变量	说明	取值范围
Int width	图像宽度。	128~1920，且为偶数。
Int height	图像高度。	128~1920，且为偶数。
Int coding_type	视频编码协议H265-main level（0）、H264-baseline level（1）、H264-main level(2)、H264-high level（3）	0~3 ● 0：H265 main level。 ● 1：H264 baseline level。 ● 2：H264 main level。 ● 3：H264high level。
Int YUV_store_type	YUV图像存储格式。	0或者1 ● 0：YUV420 semi-planner ● 1：YVU420 semi-planner
char* input_data	输入图像数据地址。	不能为NULL。
Int input_data_size	输入图像数据大小。	正数。

成员变量	说明	取值范围
shared_ptr<AutoBuffer> output_data_queue	输出编码码流数据地址。	需要配置此buffer，此buffer为调用方申请智能指针当做参数传入DVPP。 必填

出参：venc_out_msg

暂无，其中输出根据venc_in_msg中的output_data_queue来读取。

调用示例

```
void TEST_3() //venc demo
{
    int read_file_size;
    int unit_file_size;
    FILE *fp = fopen(in_file_name, "rb");
    if (fp == NULL) {
        printf("open file: %s failed.\n", in_file_name);
        return;
    }
    printf("open yuv success \n");
    fseek(fp, 0L, SEEK_END);
    int file_size = ftell(fp);
    fseek(fp, 0L, SEEK_SET);

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

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

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



```
                fflush(outputBufferFile);
            } else {
                printf("venc output data is too small : %d \n", venc_msg.output_data_queue-
>getBufferSize());
            }

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

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

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

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

2.2.7 查询 DVPP 引擎能力

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：struct device_query_req_stru

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

出参：struct dvpp_engine_capability_stru

成员变量	说明	取值范围
int 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.1.5 dvpp_engine_capability_stru中的结构体**。
struct dvpp_resolution_stru min_resolution;	最小分辨率。	详细见**A.1.5 dvpp_engine_capability_stru中的结构体**。
unsigned int 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

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

成员变量	说明	取值范围
unsigned int input_format_num;	支持的输入格式数量。	VDEC：2 JPEGD：1 PNGD：1 JPEGE：6 VPC：51 VENC：2
struct dvpp_format_unit_stru engine_input_format_table[DVPP_VADIO_FORMAT_MAX];	引擎所支持的输入格式表。	详细见**A.1.5 dvpp_engine_capability_stru中的结构体**。
unsigned int output_format_num;	支持的输出格式数量。	VDEC：4 JPEGD：4 PNGD：2 JPEGE：1 VPC：2 VENC：2
struct dvpp_format_unit_stru engine_output_format_table[DVPP_VADIO_FORMAT_MAX];	引擎所支持的输出格式表结构体。	详细见**A.1.5 dvpp_engine_capability_stru中的结构体**。
unsigned int performance_mode_num;	性能模式的数量。	固定为1。
struct dvpp_perfomance_unit_stru performance_mode_table[DVPP_PERFOMANCE_MODE_MAX];	模块的性能结构体。	详细见**A.1.5 dvpp_engine_capability_stru中的结构体**。
struct dvpp_pre_contraction_stru pre_contraction;	预缩小信息结构体。	详细见**A.1.5 dvpp_engine_capability_stru中的结构体**。
struct dvpp_pos_scale_stru pos_scale;	后缩放信息结构体。	详细见**A.1.5 dvpp_engine_capability_stru中的结构体**。

2.3 DestroyDvppApi

函数原型	int **DestroyDvppApi(IDVPPAPI** *&pIDVPPAPI)
功能	释放dvppapi，关闭DVPP执行器。 **说明** 一旦调用DestroyDvppApi函数，如果还想在继续调用DVPP，需要重新获取DvppApi实例。
输入说明	输入为**“IDVPPAPI”**类型指针引用。
输出说明	无输出。
返回值说明	返回值“0”代表成功，“-1”代表失败。
使用说明	无特殊说明。
使用限制	此函数为DVPP执行对外提供的统一调用接口之一，其主要作用为释放DVPP执行资源。

调用示例

```
DestroyDvppApi(pidvppapi);
```

3 异常处理

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

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

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

说明

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

A 附录

A.1 入参结构体说明

A.1.1 vpc_in_msg 中的结构体

A.1.1.1 图像叠加结构体 rawdata_overlay_config

成员变量	说明	取值范围
int：image_type	图像类型、图像图片和文字图片的图像类型必须为“yuv420_semi_plannar”	0 (yuv420_semi_plannar)
int：image_rank_type	图像格式(NV12或NV21)。	0或1。 0对应NV12；1对应NV21。
int：bit_width	位深。	必须为8。
int：in_width_image	图像图片宽度。	必须在[16，4096]区间范围内，必须为偶数。
int：in_high_image	图像图片高度。	必须在[16，4096]区间范围内，必须为偶数。
int：in_width_text	文字图片宽度，文字图片的宽度必须小于图像图片的宽度。	必须在[16，4096]区间范围内，必须为偶数。
int：in_high_text	文字图片高度，文字图片的高度必须小于图像图片的高度。	必须在[16，4096]区间范围内，必须为偶数。

成员变量	说明	取值范围
int：image_width_stride	图像图片宽度步长。	图像数据在内存中的stride，如果图像在内存没有进行对齐，图像图片宽度步长和图像图片的宽度一致。否则按对齐数字填写。
int：image_high_stride	图像图片高度步长。	图像数据在内存中的stride，如果图像在内存没有进行对齐，图像图片高度步长和图像图片的高度一致。否则按对齐数字填写。
int：text_width_stride	文字图片宽度步长。	图像数据在内存中的stride，如果图像在内存没有进行对齐，文字图片宽度步长和文字图片的宽度一致。否则按对齐数字填写。
int：text_high_stride	文字图片高度步长。	图像数据在内存中的stride，如果图像在内存没有进行对齐，文字图片高度步长和文字图片的高度一致。否则按对齐数字填写。
char *：in_buffer_image	图像图片输入buffer。	调用方需保证图像图片输入buffer的大小与图像图片分别按照image_width_stride和image_high_stride对齐后的宽度和高度一致。 对于yuv420 semi planner nv12格式的图片，图像图片的大小为：(对齐后的宽)*(对齐后的高)*1.5，而输入的buffer大小与如上计算结果不符，则会导致VPC调用失败。

成员变量	说明	取值范围
char *：in_buffer_text	文字图片输入buffer。	调用方需保证文字图片输入buffer的大小与文字图片分别按照text_width_stride和text_high_stride对齐后的宽度和高度一致。 对于yuv420 semi planner nv12格式的图片，文字图片的大小为：(对齐后的宽)*(对齐后的高)*1.5，而输入的buffer大小与如上计算结果不符，则会导致VPC调用失败。
shared_ptr<AutoBuffer> auto_overlay_out_buffer	叠加后的输出buffer。	此buffer为调用方申请智能指针当做参数传入VPC。 必填

A.1.1.2 图像拼接结构体 rawdata_collage_config

成员变量	说明	取值范围
Int：image_type	图像类型，拼接图像类型必须为yuv420_semi_plannar。	0 (yuv420_semi_plannar)
Int：image_rank_type	图像格式(NV12或NV21)，用于拼接的4张图片的图像格式必须一致。	0或1。 0对应NV12，1对应NV21。
Int：bit_width	位深。	默认为8，可不配置。
Int：in_width	图片宽度。	必须在[32，4096]区间范围内，必须为偶数。
Int：in_high	图片高度。	必须在[32，4096]区间范围内，必须为偶数。
Int：width_stride	图片宽度步长。	图像数据在内存中的stride,如果图像在内存没有进行对齐，图像图片宽度步长和图像图片的宽度一致。否则按对齐数字填写。
Int：high_stride	图片高度步长。	图像数据在内存中的stride,如果图像在内存没有进行对齐，图像图片高度步长和图像图片的高度一致。否则按对齐数字填写。

成员变量	说明	取值范围
int collage_type	拼接图片格式，1x4(一行四列)，4x1(四行一列)，2x2(默认，二行二列)。	0或1或2。 0对应二行二列(默认)，1对应一行四列，2对应四行一列。
char *：in_buffer[collage_num]	用于拼接的4张图片输入buffer。	调用方需保证图像图片输入buffer的大小与图像图片分别按照width_stride和high_stride对齐后的宽度和高度一致。 对于yuv420 semi planner nv12格式的图片，图像图片的大小为：(对齐后的宽)*(对齐后的高)*1.5，而输入的buffer大小与如上计算结果不符，则会导致调用失败。
shared_ptr<AutoBuffer> auto_out_buffer	拼接图片的输出地址。	此buffer为调用方申请智能指针当做参数传入DVPP。

A.1.1.3 Rdma 通道结构体 RDMACHANNEL

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

A.1.1.4 Vpc 内部优化结构体 VpcTurningReverse

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

A.1.2 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...**。
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	预留接口。

- 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。
uint32_t chromaHeadStride	uv分量头stride。
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。
uint32_t chromaPayloadStride	uv分量数据的stride。

- 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	预留接口。

- VpcUserPerformanceTuningParameter结构体

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

- VpcUserRoiInputConfigure 结构体

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

- VpcUserRoiOutputConfigure结构体

成员变量	说明
uint8_t* addr	输出图片的首地址。
uint32_t bufferSize	输出buffer的大小，根据yuv420sp计算。
uint32_t widthStride	输出图片的宽步长。
uint32_t heightStride	输出图片的高步长，输出为yuv420sp图像，需要根据heightStride计算出uv数据的起始地址。
VpcUserCropConfigure outputArea	用户指定输出区域坐标，详细见**•VpcUserCropConfigure 结构...**。
uint64_t reserve1	预留接口。

- VpcUserCropConfigure 结构体

成员变量	说明
uint32_t leftOffset	左偏移，必须为偶数。
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	预留接口。

A.1.3 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）。
int width	输出图像的宽（对齐后的值）。
int realHeight	真实图像的高。
int realWidth	真实图像的宽。
unsigned char* buffer	输出图像的内存地址。
int buffer_size	输出图像的内存大小。
unsigned int offset_payload_y	输出图像payload的Y分量偏移量，payload的Y分量地址=buffer+ offset_payload_y。
unsigned int offset_payload_c;	输出图像payload的C分量偏移量，payload的C分量地址=buffer+ offset_payload_c。
unsigned int offset_head_y;	输出图像head的Y分量偏移量，head的Y分量地址=buffer+ offset_head_y。
unsigned int offset_head_c;	输出图像head的C分量偏移量，head的C分量地址=buffer+ offset_head_c。
unsigned int stride_payload;	输出图像payload的stride。
unsigned int stride_head;	输出图像head的stride。
unsigned short bitdepth;	输出图像的位深。
char video_format[10];	输入视频的格式，为“h264”或“h265”。
char image_format[10];	输出图像的格式，为“nv12”或“nv21”。

- HIAI_DATA_SP类

成员变量或函数	说明
unsigned long long frameIndex	帧序号，若不需要，可不配置。
void * frameBuffer	用户申请用于存放输出帧的内存，若不需要，可不配置。
unsigned int frameSize	用户申请用于存放输出帧的内存大小，若不需要，可不配置。
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。
void setFrameSize(unsigned int size)	设置frameSize大小。
unsigned int getFrameSize()	获取frameSize大小。

- VDECERR结构体

成员变量	说明

ERRTYPE errType	错误类型。
	enum ERRTYPE{
	//decoder state error
	ERR_INVALID_STATE = 0x10001,
	//hardware error
	ERR_HARDWARE,
	//vdm decode fail
	ERR_SCD_CUT_FAIL,
	//vdm decode fail
	ERR_VDM_DECODE_FAIL,
	//vdm decode fail
	ERR_ALLOC_MEM_FAIL,
	//bit stream error
	ERR_BITSTREAM,
	//other error
	ERR_OTHER
	};

成员变量	说明
unsigned short channelId	解码错误的通道。

A.1.4 IMAGE_CONFIG 中的结构体

- ROI_CONFIG结构体

成员变量	说明	取值范围
CROP_CONFIG：crop_config	裁剪参数配置结构体。	详细见**•CROP_CONFIG结构体**。
YUV_SUM_OUT_CONFIG：sum_out	输出参数配置结构体1。	详见**•YUV_SUM_OUT_CONFIG结构体**。
NORMAL_OUT_CONFIG：normal_out	输出参数配置结构体2(预留，暂时不支持)。	详见**•NORMAL_OUT_CONFIG结构体**。
ROI_CONFIG*：next	下一个ROI_CONFIG结构体指针。	多框时配置，单框时配置为NULL。

- CROP_CONFIG结构体

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

- YUV_SUM_OUT_CONFIG结构体

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

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

- NORMAL_OUT_CONFIG结构体

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

A.1.5 dvpp_engine_capability_stru 中的结构体

- struct dvpp_resolution_stru结构体

成员变量	说明	取值范围
unsigned int 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
unsigned int 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

- struct dvpp_format_unit_stru 结构体

成员变量	说明	取值范围

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

成员变量	说明	取值范围
		/*400*/ DVPP_COLOR_YVU400_SP_8BIT, /*rgb888*/ DVPP_COLOR_RGB888_RGB_P_8BIT_LIN, DVPP_COLOR_RGB888_RBG_P_8BIT_LIN, DVPP_COLOR_RGB888_GBR_P_8BIT_LIN, DVPP_COLOR_RGB888_GRB_P_8BIT_LIN, DVPP_COLOR_RGB888_BRG_P_8BIT_LIN, DVPP_COLOR_RGB888_BGR_P_8BIT_LIN, /*argb888*/ DVPP_COLOR_ARGB8888_ARGB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_ARBG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_AGBR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_AGRB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_ABRG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_ABGR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RAGB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RABG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RGBA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RGAB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RBAG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RBGA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BRGA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BRAG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BGAR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BGRA_P_8BIT_LIN, DVPP_COLOR_ARGB888

成员变量	说明	取值范围
		8_BARG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BAGR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GRAG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GRBA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GABR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GARB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GBRA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GBAR_P_8BIT_LIN, PIC_JPEG, PIC_PNG, VIO_H265, VIO_H264 };

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

unsigned int 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
		};

unsigned int out_mem_alignment;	输出内存对齐参数	--

- struct dvpp_perfomance_unit_stru结构体

成员变量	说明	取值范围
unsigned int resolution_high;	高度分辨率	VDEC：1920 JPEGD：1920 PNGD：1920 JPEGE：1920 VPC：3840 VENC：1920
unsigned int resolution_width;	宽度分辨率	VDEC：1080 JPEGD：1080 PNGD：1080 JPEGE：1080 VPC：2160 VENC：1080
unsigned int stream_num;	流大小	VDEC：16 JPEGD：0 PNGD：0 JPEGE：0 VPC：0 VENC：1

成员变量	说明	取值范围
unsigned long fps;	帧率	VDEC：30 JPEGD：256 PNGD：24 JPEGE：64 VPC：90 VENC：30

- struct 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
		};

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

- struct 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
unsigned int min_scale;	最小缩放系数	vpc：1 others：1

成员变量	说明	取值范围
unsigned int 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

A.2 DVPP 执行器工具使用说明

DVPP执行器工具(sample_dvpp)集成了DVPP六个IP的基础功能，包括VPC、JPEGE、JPEGD、VENC、VDEC以及PNGD。

A.2.1 环境准备

步骤1 在DDK安装路径$HOME/tools/che/ddk/ddk/sample/dvpp中执行make -j命令，编译出来的可执行文件在out目录下。

步骤2 将编译生成的可执行文件“sample_dvpp_hlt_ddk”复制到device侧任意路径。

说明

依赖SO为： libc_sec.so libDvpp_api.so libDvpp_jpeg_decoder.so libDvpp_jpeg_encoder.so libDvpp_png_decoder.so libDvpp_vpc.so libOMX_common.so libOMX_hisi_vdec_core.so libOMX_hisi_video_decoder.so libOMX_hisi_video_encoder.so libOMX_Core.so libOMX_Core_VENC.so libslog.so

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

步骤3 输入数据文件放入到可执行文件所在路径下，运行**./sample_dvpp_hlt_ddk paramList**。

----结束

A.2.2 DVPP 执行器工具入参说明

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

参数	描述
in_format	输入图片的格式如： yuv420_semi_plannar =0, //0 yuv422_semi_plannar, //1 yuv444_semi_plannar, //2 yuv422_packed, //3 yuv444_packed, //4 rgb888_packed, //5 xrgb8888_packed, //6 yuv400_semi_plannar , //7 jpege的参数如下： JPGENC_FORMAT_UYVY = 0, JPGENC_FORMAT_VYUY = 1, JPGENC_FORMAT_YVYU = 2, JPGENC_FORMAT_YUYV = 3, JPGENC_FORMAT_NV12 = 16, JPGENC_FORMAT_NV21 = 17, 例子：如果输入图片为yuv420sp格式的，此参数设置为0，依次类推。
in_bitwidth	输入图像的位宽，8或者10。
cvdr_or_rdma	输入图像走cvdr通道（1）还是rdma通道（0），默认走cvdr通道。
hinc	水平缩放比率，此值的计算公式： (hmax-hmin+1)/img_width。其中“hmax-hmin+1”为水平裁剪区域，如果没有裁剪，这个值用out_width替代。
vinc	竖直缩放比率，此值的计算公式： (vmax-vmin+1)/img_height。其中“vmax-vmin+1”为竖直裁剪区域，如果没有裁剪，这个值用out_high替代。
hmax	与原点在水平方向的最大偏移，具体约束详见**A.1.1 vpc_in_msg中的结构体**。
hmin	与原点在水平方向的最小偏移，具体约束详见**A.1.1 vpc_in_msg中的结构体**。
vmax	与原点在垂直方向的最大偏移，具体约束详见**A.1.1 vpc_in_msg中的结构体**。
vmin	与原点在垂直方向的最小偏移，具体约束详见**A.1.1 vpc_in_msg中的结构体**。
stride	图像步长，具体约束详见**A.1.1 vpc_in_msg中的结构体**。
out_width	输出图像宽度。

参数	描述
out_high	输出图像高度。
in_image_file	输入图像名称。
out_image_file	输出图像名称。
rank	图像排列方式： 具体约束详见**A.1.1 vpc_in_msg中的结构体**。
test_type	输入数据：1、2、3、4、5、6、7、9。 1：VPC基础功能(裁剪,缩放)。 2：VPC Rawdata8K。 3：VENC。 4：JPEGD。 5：JPEGE。 6：VDEC。 7：JPEGD+VPC+JPEGE。 8：CMDLIST。 9：PNGD。

A.2.3 VPC 使用说明

A.2.3.1 VPC 基础功能

```
./sample_dvpp_hlt_ddk --in_image_file 1920_1080_420sp --out_image_file youyou --img_width 1920  --img_height 1080  --in_format 0 --in_bitwidth 8  --cvdr_or_rdma 1  --hmax 1919 --hmin 0 --vmax 1079 --vmin 0 --hinc 0.3125 --vinc 0.3703703703703704 --stride 1920  --test_type 1
```

示例描述：

将一张分辨率1920x1080，格式为yuv420sp的图片，缩放成600x400的yuv420sp图像。默认输出在当前目录下。

A.2.3.2 VPC RawData 8K

```
./sample_dvpp_hlt_ddk --in_image_file jpeg_7680x4320_nv12 --out_image_file youyou --img_width 7680  --img_height 4320  --out_width 1000  --out_high 800 --test_type  2 --stride 7680 --rank 1
```

示例描述：

将一张分辨率7680x4320，格式为yuv420sp的图片，缩放成1000x800的yuv420sp图像。默认输出在当前目录下。

A.2.4 CMDLIST 使用说明

```
./sample_dvpp_hlt_ddk  --test_type 8
```

示例代码请参见：**2.2.1.1 CMDLIST**。

A.2.5 VDEC 使用说明

```
./sample_dvpp_hlt_ddk --in_image_file h264_three_frame  --test_type 6   --in_format 0
./sample_dvpp_hlt_ddk --in_image_file h265_three_frame  --test_type 6   --in_format 1
```

示例描述：

这两个用例分别是将h264_three_frame和h265_three_frame这两种码流送到VDEC去解码，解码后的文件在./output_dir下。（注：此目录需要用户提前建好。）

A.2.6 VENC 使用说明

```
./sample_dvpp_hlt_ddk --in_image_file fileName  --test_type 3  --img_width 1920  --img_height 1080
--in_format 0 --in_bitwidth 0
```

示例描述：

此用例将1920*1080的yuv图片编码成h265码流，需要注意的是当测试venc时将in_format对应到**入参：venc_in_msg**中coding_type, in_bitwidth对应到**入参：venc_in_msg**中YUV_store_type。编码文件及结果在当前目录下。

A.2.7 JPEGE 使用说明

```
./sample_dvpp_hlt_ddk --in_image_file dvppapi_jpegd_w1280_h720_YUV420 --test_type 5  --img_width
1280  --img_height 720 --in_format 17
```

示例描述：

- 此用例将1280*720的yuv图片编码成jpg图片。
- Jpege无需指定输出文件。
- 编码文件及结果在当前目录下。

A.2.8 JPEGD 使用说明

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_jpeg_decode_025 --test_type 4 --rank 1
```

示例描述：

此用例将dvpp_jpeg_decode_025的jpg图片解码成yuv图片。

- Jpegd无需指定输出文件，输出文件名为dvppapi_jpegd_wxx_hxx_fmt.yuv。
- -- rank为0或为空，isYUV420Need为真，输出NV21格式数据，-- rank不为0输出原格式的半平面数据。
- 解码文件及结果在当前目录下。

A.2.9 PNGD 使用说明

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_pngd_02.png --test_type 9  --transform 1
```

示例描述：

- 此用例将dvpp_pngd_02.png图片解码成rgb或者rgba图片，其中输出文件为/var/result.pngd。
- 参数transform是指rgba转rgb。

A.3 缩略语和术语一览

缩略语	英文全名	中文解释
VDEC	Video Decoder	视频解码
VENC	Video Encoder	视频编码
JPEGD	JPEG Decode	JPEG解码
JPEGE	JPEG Encoder	JPEG编码
PNGD	Portable Network Graphics Decoder	便携式网络图形解码器
VPC	Vision Preprocess Core	视觉预处理核
DVPP	Digital Vision Pre-Process	数字视觉预处理

A.4 辅助功能接口

- JpegCalBackFree类辅助接口

class JpegCalBackFree：

JpegCalBackFree()

~JpegCalBackFree()

JpegCalBackFree(JpegCalBackFree& others)

void setBuf(void* addr4Free)

void setBuf(void* addr4Free, size_t size4Free)

void operator() ()

const JpegCalBackFree& operator= (JpegCalBackFree& others)

- AutoBuffer类辅助接口

class AutoBuffer：

AutoBuffer()

~AutoBuffer()

void Reset()

char* allocBuffer(int size)

char* getBuffer()

int getBufferSize()

AutoBuffer(const AutoBuffer&)

const AutoBuffer& operator= (const AutoBuffer&)

- HIAI_DATA_SP类辅助接口

class HIAI_DATA_SP：

HIAI_DATA_SP()

virtual ~HIAI_DATA_SP()

void setFrameIndex(unsigned long long index)

unsigned long long getFrameIndex()

void setFrameBuffer(void * frameBuff)

void * getFrameBuffer()

void setFrameSize(unsigned int size)

unsigned int getFrameSize()

A.5 不推荐使用接口列表

class IDVPPCAPABILITY：

virtual ~IDVPPCAPABILITY(void)

virtual int process(dvppapi_ctl_msg *MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

class IJPEGDAPI：

virtual ~IJPEGDAPI(void)

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

class IJPEGEAPI：

virtual ~IJPEGEAPI(){}

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

class IVDECAPI：

virtual ~IVDECAPI(void)

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

class IVENCAPI：

virtual ~IVENCAPI(void)

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

class IVPCAPI：

virtual ~IVPCAPI(void)

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

