Linux VE
开发指南

版本号: 3.0
发布日期: 2024.10.23

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.05.06	AWA0573	初始版本
2.0	2022.10.28	AWA1908	整理及更新一些 api 说明
3.0	2024.10.23	AWA2082	新增部分章节与接口

目　录

1 前言 1
- 1.1 文档简介 1
- 1.2 目标读者 1
- 1.3 适用范围 1
- 1.4 相关术语介绍 1
  - 1.4.1 软件术语 1

2 模块介绍 2
- 2.1 模块功能 2
- 2.2 相关术语介绍 2
- 2.3 源码结构 2
- 2.4 软件框架 3

3 模块接口说明 5
- 3.1 **接口函数** 5
  - 3.1.1 VencCreate 5
  - 3.1.2 VencDestroy 6
  - 3.1.3 VencInit 6
  - 3.1.4 VencStart 6
  - 3.1.5 VencPause 7
  - 3.1.6 VencReset 7
  - 3.1.7 VencFlush 7
  - 3.1.8 VencGetParameter 8
  - 3.1.9 VencSetParameter 8
  - 3.1.10 VencGetValidOutputBufNum 8
  - 3.1.11 VencDequeueOutputBuf 8
  - 3.1.12 VencQueueOutputBuf 9
  - 3.1.13 VencGetValidInputBufNum 9
  - 3.1.14 VencQueueInputBuf 9
  - 3.1.15 VencAllocateInputBuf 10
  - 3.1.16 VencGetVeIommuAddr 10
  - 3.1.17 VencFreeVeIommuAddr 11
  - 3.1.18 VencSetDdrMode 11
  - 3.1.19 VencSetFreq 11
  - 3.1.20 VencJpegEnc 11
  - 3.1.21 VencSetCallbacks 12
- 3.2 内部模块接口设计 12
  - 3.2.1 Frame Buffer Manager 模块接口设计 12
    - 3.2.1.1 VencFbmCreate 13

3.2.1.2 VencFbmDestroy . . . 13
3.2.1.3 VencFbmReset . . . 13
3.2.1.4 VencFbmAllocateBuffer . . . 13
3.2.1.5 VencFbmAddValidBuffer . . . 14
3.2.1.6 VencFbmRequestValidBuffer . . . 14
3.2.1.7 VencFbmReturnValidBuffer . . . 15
3.2.1.8 VencFbmGetValidBufferNum . . . 15
3.2.2 BitStream Manager 模块接口设计 . . . 15
3.2.2.1 BitStreamCreate . . . 16
3.2.2.2 BitStreamDestroy . . . 16
3.2.2.3 BitStreamBaseAddress . . . 16
3.2.2.4 BitStreamBasePhyAddress . . . 16
3.2.2.5 BitStreamEndPhyAddress . . . 17
3.2.2.6 BitStreamBufferSize . . . 17
3.2.2.7 BitStreamFreeBufferSize . . . 17
3.2.2.8 BitStreamFrameNum . . . 17
3.2.2.9 BitStreamWriteOffset . . . 17
3.2.2.10 BitStreamAddOneBitstream . . . 18
3.2.2.11 BitStreamGetOneBitstream . . . 18
3.2.2.12 BitStreamReturnOneBitstream . . . 18
3.2.2.13 BitStreamReset . . . 19
3.2.3 Video Encoder Device 模块接口设计 . . . 19
3.2.3.1 VencoderDeviceCreate . . . 19
3.2.3.2 VencoderDeviceDestroy . . . 19

4 数据结构设计 20

4.1 VencBaseConfig . . . 20
4.2 VencH264ProfileLevel . . . 21
4.3 VencQPRange . . . 21
4.4 MotionParam . . . 21
4.5 VencHeaderData . . . 21
4.6 VencInputBuffer . . . 22
4.7 VencOutputBuffer . . . 22
4.8 VencAllocateBufferParam . . . 23
4.9 VencH264FixQP . . . 23
4.10 VencCyclicIntraRefresh . . . 23
4.11 VencH264Param . . . 23
4.12 VencROIConfig . . . 24
4.13 VENC_DEVICE . . . 24
4.14 VencH264AspectRatio . . . 26
4.15 VencH264VideoSignal . . . 26
4.16 VencH264SVCSkip . . . 27
4.17 VENC_INDEXTYPE . . . 27

5 功能开发 **30**
5.1 功能概述 30
5.2 开发流程 30
5.3 编程示例 31

6 调试方法 **32**
6.1 调试工具 32
6.2 调试节点 32

7 FAQ **33**
7.1 问题描述 33

1 前言

1.1 文档简介

介绍视频编码库对外 API 接口及相关的数据结构，指导基于视频编码库的开发、使用。

1.2 目标读者

基于视频编码库开发和使用的相关人员。

1.3 适用范围

适用于全志公司带有 VE 编码模块的 v 线芯片平台 tina linux sdk。

1.4 相关术语介绍

1.4.1 软件术语

- QP：量化参数；
- Exif：可交换图像文件格式，用于记录照片的属性信息和拍摄数据，通常作为 jpeg 图片的附件信息。

2 模块介绍

2.1 模块功能

视频编码库是一个提供视频编码功能的库，编译输出的库文件为 libvencoder.so。基于视频编码库，应用程序可以在 Allwinner 的各个 IC 平台上实现高效的、多种压缩格式的视频编码功能 ss。

2.2 相关术语介绍

- QP：量化参数；
- Exif：可交换图像文件格式，用于记录照片的属性信息和拍摄数据，通常作为 jpeg 图片的附件信息。

2.3 源码结构

源码结构树状图如下：

```
├── base:用于存放message功能操作函数、配置文件的读取与设置、cdc_log函数等
│
├── config：cedarc.conf，用于动态设置参数，控制与其相关联的功能，如动态log设置
│
├── demo：编解码demo，主要用于测试，可根据自己需要修改其中设置，以达到自测目的
│   ├── vdecoderDemo
│   └── vencoderDemo
│
├── memory：内存管理操作目录
│   ├── ionMemory
│   ├── secureMemory
│   └── sunxiMemory
│
├── vdecoder：解码器
│   ├── sbm：stream buffer manager管理模块
│   ├── videoengine：解码引擎
│
└── vencoder：编码器
  ├── base：Bitstream、FrameBuffer管理操作
  └── libcodec
```

2.4 软件框架

vencoder 调用 Video Encoder device 的编码函数时，vencoder 先从 Frame Buffer Manager 取出输入的图像帧，然后把获取到的图像帧送给 Image Signal Processor，对其进行相关的图像处理，如裁剪，旋转，水印等；然后将处理完的图像帧送给 Video Encoder device，Video Encoder device 从 Bitstream Manager 获取有效的输出 buffer 的起始地址和 offset，并把相应的 Bitstream 起始地址和 offset 配给 VE, 启动 VE 后，将会输出相应的码流到 Bitstream Manager 模块。

编码后，Video Encoder device 将更新 Bitstream Manager 模块中有效码流信息，把相应的码流加入 Bitstream Manager 模块的输出队列。

外部程序通过调用 vencoder 把图像 frame 送给 Frame Buffer Manager 模块，外部应用程序通过 vencoder 从 Bitstream Manager 获取输出的码流。

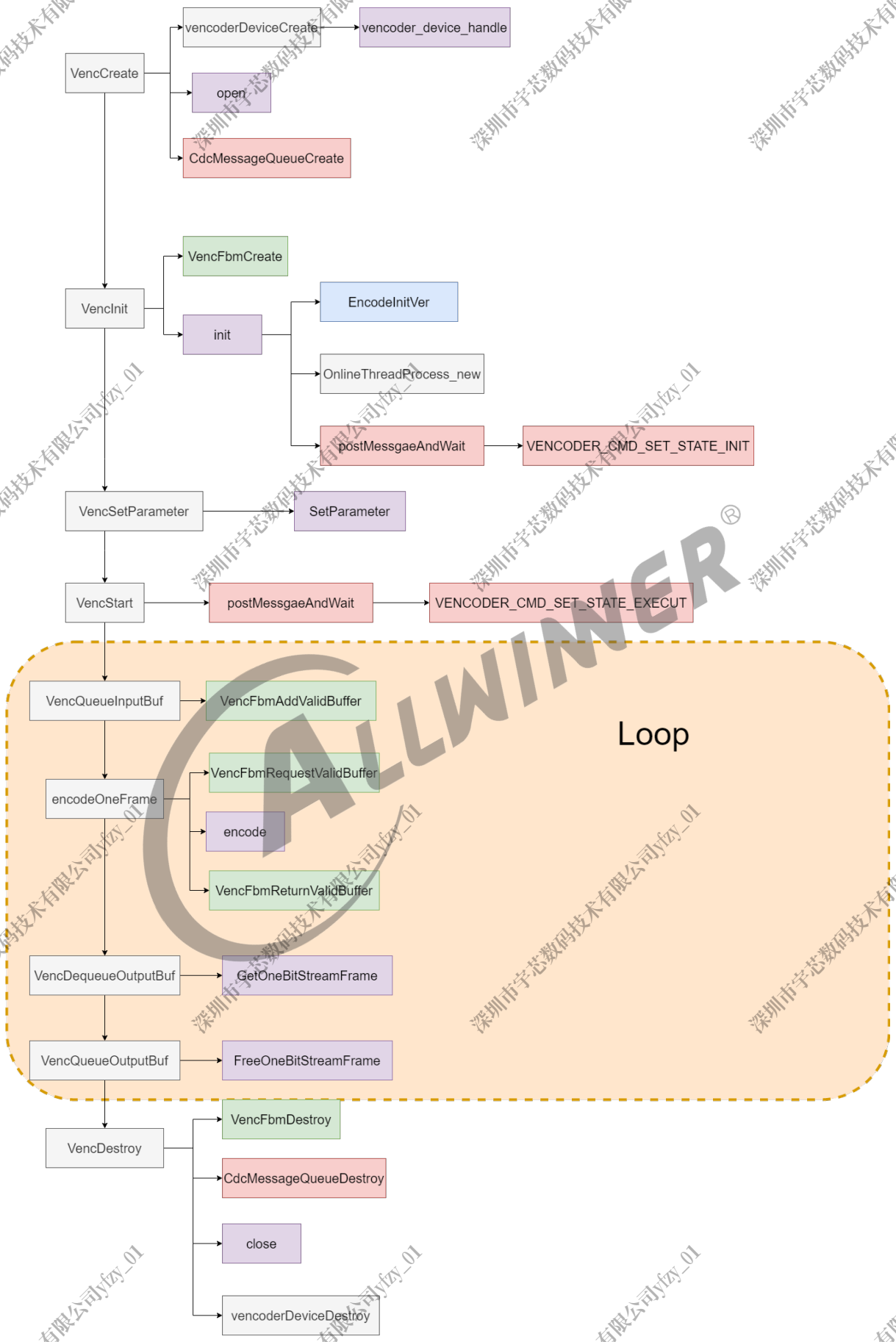

图 2-1: 软件架构

3 模块接口说明

3.1 接口函数

视频编码库 APIs	
VencCreate	创建一个视频编码器
VencDestroy	销毁视频编码器
VencInit	初始化视频编码器
VencStart	启动视频编码器
VencPause	暂停视频编码器
VencReset	重置视频编码器
VencFlush	清理输入或输出数据
VencGetParameter	获取编码器参数
VencSetParameter	设置编码器参数
VencGetValidOutputBufNum	获取有效的输出码流 buffer 的个数
VencDequeueOutputBuf	获取编码器生成的码流数据
VencQueueOutputBuf	还回用于存放码流数据的 buffer
VencGetValidInputBufNum	获取编码器未完成编码的输入 buffer 个数
VencQueueInputBuf	传递用于编码的图像帧输入数据
VencAllocateInputBuf	申请用于存放图像帧数据的 buffer
VencGetVeIommuAddr	获取 iommu 内存的物理地址
VencFreeVeIommuAddr	释放 iommu buffer
VencSetDdrMode	设置 ddr 类型
VencSetFreq	设置 ve 编码器的频率
VencJpegEnc	单独调用此接口实现 jpeg 编码拍照
VencSetCallbacks	设置回调函数

3.1.1 VencCreate

函数原型	VideoEncoder* VencCreate(VENC_CODEC_TYPE eCodecType)
功能	创建一个视频编码器
参数	eCodecType：创建的编码器 codec 类型
返回值	成功：视频编码器指针 失败：返回 NULL
调用说明	视频编码器支持创建多个编码器，支持多路编码。

3.1.2 VencDestroy

函数原型	void VencDestroy(VideoEncoder* pEncoder)
功能	销毁视频编码器
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针。
返回值	无
调用说明	无

3.1.3 VencInit

函数原型	int VencInit(VideoEncoder* pEncoder, VencBaseConfig* pConfig)
功能	初始化视频编码器
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针； pConfig：编码器基本初始化信息，包括是否做 scaler，颜色格式等。
返回值	成功：返回 0 失败：返回-1
调用说明	pConfig：编码器基本初始化信息； 1. nInputWidth：输入图像帧的宽度，以像素为单位； 2. nInputHeight：输入图像帧的高度，以像素为单位； 3. nDstWidth：编码前对输入图像做 scale 后的宽度，以像素为单位； 如果不需要做 scale，nDstWidth 的值保持和 nInputWidth 一致； 4. nDstHeight：编码前对输入图像做 scale 后的高度，以像素为单位， 如果不需要做 scale，nDstHeight 的值保持和 nInputHeight 一致； 5. eInputFormat：输入的颜色格式； 6. nStride: 输入图像帧在内存中的行宽，以像素为单位，编码器要求 nStride 必须 16 对齐。 7. eOutputFormat：输出的颜色格式； 8. bOnlineMode：是否使能在线模式，1：使能，0：关闭； 9. bOnlineChannel：当前编码通道是否为在线编码，1：在线，0：离线； 10. nChannel：通道号； 11. sensor_id：摄像头 id； 12. bk_id：摄像头通道 id； 13. rec_lbc_mode：内存优化压缩倍率； 14. bVcuOn：vcu 功能使能位； 15. bEnableMultiOnlineSensor：双目编码功能使能位； 16. bEnableImageStitching：双目拼接功能使能位；

3.1.4 VencStart

函数原型	int VencStart(VideoEncoder* pEncoder)
功能	启动视频编码器
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针。
返回值	成功：返回 0 失败：返回-1
调用说明	无

3.1.5 VencPause

函数原型	int VencPause(VideoEncoder* pEncoder)
功能	暂停视频编码器
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针。
返回值	成功：返回 0 失败：返回-1
调用说明	无

3.1.6 VencReset

函数原型	int VencReset(VideoEncoder* pEncoder)
功能	重置视频编码器
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针。
返回值	成功：返回 0 失败：返回-1
调用说明	编码器配置参数不变，仅把输入帧 buffer 队列和输出比特流 buffer 队列清零。

3.1.7 VencFlush

函数原型	int VencFlush(VideoEncoder* pEncoder, VencFlushType eFlushType)
功能	刷新输入或输出数据管理器
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针； eFlushType：参数类型索引 1. VENC_FLUSH_INPUT_BUFFER：刷新输入数据管理器 2. VENC_FLUSH_OUTPUT_BUFFER：刷新输出数据管理器 3. VENC_FLUSH_IN_AND_OUT_BUFFER：刷新输入和输出数据管理器
返回值	成功：返回 0 失败：返回-1
调用说明	编码器配置参数不变，仅把输入帧 buffer 队列和输出比特流 buffer 队列清零。

3.1.8 VencGetParameter

函数原型	int VencGetParameter(VideoEncoder*pEncoder, VENC_INDEXTYPE indexType, void* paramData)
功能	获取编码器参数
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针； indexType：参数类型索引号； paramData（输出）：参数数据指针。
返回值	成功：返回 0 失败：返回-1
调用说明	调用成功后将会返回参数到 paramData 指针所指的地址中。

3.1.9 VencSetParameter

函数原型	int VencSetParameter(VideoEncoder*pEncoder, VENC_INDEXTYPE indexType, void* paramData)
功能	设置编码器参数
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针； indexType：参数类型索引号； paramData（输出）：参数数据指针。
返回值	成功：返回 0 失败：返回-1
调用说明	编码器将从 paramData 指针所指的地址中获取参数信息。

3.1.10 VencGetValidOutputBufNum

函数原型	VencGetValidOutputBufNum(VideoEncoder* pEncoder)
功能	获取有效的的输出码流 buffer 的格式
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针。
返回值	有效的输出码流的个数（value>=0）
调用说明	无

3.1.11 VencDequeueOutputBuf

函数原型	int VencDequeueOutputBuf(VideoEncoder* pEncoder, VencOutputBuffer* pBuffer)
功能	获取有效的的输出码流 buffer 的格式
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针；

	pBuffer（输出）：输出码流 buffer。
返回值	成功：返回 0 失败：返回-1
调用说明	pBuffer 中结构体变量说明： 1. nID：用来识别不同的 buffer； 2. nPts：编码器不对时间戳信息做处理，输出 buffer 中的 pts 对应相应输入 buffer 中的 pts； 3. nSize0：输出码流的第一部分的大小； 4. nSize1：输出码流的第二部分的大小； 5. pData0：输出码流的第一部分的地址； 6. pData1：输出码流的第二部分的地址； 输出的一笔码流可能由两部分组成：nSize0 表示第一部分的大小，nSize1 表示第二部分的大小； nSize0 一定大于 0，当 nSize1 = 0 的时候，输出码流只在地址 pData0 中； 当 nSize1 > 0 时，输出码流由两部分组成，第一部分在 pData0 中，第二部分在 pData1 中，此时需要外部应用程序把这两部分数据组合成一帧。

3.1.12 VencQueueOutputBuf

函数原型	int VencQueueOutputBuf(VideoEncoder* pEncoder, VencOutputBuffer* pBuffer)
功能	还回输出码流 buffer
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针； pBuffer（输入）：由 GetOneBitstreamFrame 获取到的输出码流 buffer。
返回值	成功：返回 0 失败：返回-1
调用说明	pBuffer 表示由 VencDequeueOutputBuf 获取到的输出码流 buffer。

3.1.13 VencGetValidInputBufNum

函数原型	int VencGetValidOutputBufNum(VideoEncoder*pEncoder)
功能	获取编码器未完成编码的输入 buffer 个数。
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针。
返回值	成功：返回 0 失败：返回-1
调用说明	无

3.1.14 VencQueueInputBuf

函数原型	int VencQueueInputBuf(VideoEncoder* pEncoder, VencInputBuffer *inputbuffer)
功能	传递用于编码的图像帧输入数据。
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针。
返回值	成功：返回 0 失败：返回-1
调用说明	编码器用完 input buf 图像帧数据后，通过回调 callback 的方式将 input buf 归还到上层，callback 函数调用 VencSetCallbacks 接口进行设置。

3.1.15 VencAllocateInputBuf

函数原型	int VencAllocateInputBuf(VideoEncoder* pEncoder, > VencAllocateBufferParam*pBufferParam, VencInputBuffer* dst_inputBuf)
功能	通过 vencoder 申请输入图像帧 buffer。
参数	pEncoder: 通过 VideoEncCreate 函数创建的视频编码器指针； pBufferParam：指定申请 buffer 的格式和 size； dst_inputBuf：存放申请出来的 buffer 的相关信息。
返回值	成功：返回 0 失败：返回-1
调用说明	1. 当需要由编码器来提供输入图像帧的 buffer 时，由此接口来申请图像帧 buffer； 2. 当外部模块有自己的 buffer 管理模块，并且所使用的 buffer 为物理连续的 buffer 的时候，从效率上考虑可以不使用此接口来申请输入图像帧 buffer，可以直接把相应的 buffer 的物理地址配给 VE，从而可以少一次数据 copy； 3. 每次调用此接口，只申请 1 个 buffer，如果需要申请多个 buffer，则需要调用多次此接口。

3.1.16 VencGetVeIommuAddr

函数原型	void VencGetVeIommuAddr(VideoEncoder* pEncoder, struct user_iommu_param *pIommuBuf)
功能	获取 iommu 内存的物理地址
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针； pIommuBuf：用于存放获取到的 iommu 内存的物理地址。
返回值	无
调用说明	由于获取 iommu 的物理地址时需要与设备进行绑定，所以上层可调用此接口通过绑定 ve 设备获取得到 iommu 的物理地址。

3.1.17 VencFreeVeIommuAddr

函数原型	void VencFreeVeIommuAddr(VideoEncoder* pEncoder, struct user_iommu_param *pIommuBuf)
功能	释放通过 VencGetVeIommuAddr 接口获取到的 iommu buffer。
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针； pIommuBuf：通过 VencGetVeIommuAddr 接口获取到的 iommu buffer。
返回值	无
调用说明	无

3.1.18 VencSetDdrMode

函数原型	void VencSetDdrMode(VideoEncoder* pEncoder, int nDdrType)
功能	设置 ddr 类型
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针； nDdrType：需要设置的 ddr 类型参数。
返回值	无
调用说明	1. 一般情况下无需调用此接口； 2. 若芯片方案使用的 ddr 类型比较特殊，则需调用此接口将 ddr 类型设置到编码器，编码器会根据 ddr 类型做相关的操作。

3.1.19 VencSetFreq

函数原型	int VencSetFreq(VideoEncoder* pEncoder, int nVeFreq)
功能	设置 ve 编码器的工作频率
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针； nVeFreq：目标频率值。
返回值	成功：返回 0 失败：返回-1
调用说明	如果上层调用者需要调节编码器的工作频率，可调用此接口进行设置，否则编码器驱动则使用默认的工作频率。

3.1.20 VencJpegEnc

函数原型	int VencJpegEnc(JpegEncInfo* pJpegInfo, EXIFInfo* pExifInfo, void* pOutBuffer, int* pOutBufferSize)
功能	单独调用此接口实现 jpeg 编码拍照。
参数	pJpegInfo: 进行 jpeg 拍照的相关参数；

	pExifInfo：jpeg 图片携带的 exif 信息； pOutBuffer：jpeg 编码器生成的码流数据； pOutBufferSize：jpeg 码流数据的 size 大小。
返回值	成功：返回 0 失败：返回-1
调用说明	1. 由于 jpeg 只需要进行一次编码操作，所以可单独调用此接口实现 jpeg 的拍照功能； 2. 上层也可以调用正常编码器接口实现 jpeg 拍照，如调用 create/start/init/destroy 等正常流程接口。

3.1.21 VencSetCallbacks

函数原型	int VencSetCallbacks(VideoEncoder* pEncoder, VencCbType* pCallbacks, void* pAppData)
功能	设置回调函数
参数	pEncoder: 通过 VencCreate 函数创建的视频编码器指针； pCallbacks：回调函数的操作指针； pAppData：上层私有数据。
返回值	成功：返回 0 失败：返回-1
调用说明	无

3.2 内部模块接口设计

3.2.1 Frame Buffer Manager 模块接口设计

Frame Buffer Manager 模块接口如下所示：

1	VencFbmCreate	创建帧缓冲管理模块
2	VencFbmDestroy	销毁帧缓冲管理模块，释放内存
3	VencFbmReset	帧 buffer 队列清零
4	VencFbmAllocateBuffer	申请输入图像帧物理连续的内存当不使用外部应用 buffer 管理队列时
5	VencFbmAddValidBuffer	添加一帧图像帧到输入帧队列
6	VencFbmRequestValidBuffer	从输入帧队列获取一个图像帧
7	VencFbmReturnValidBuffer	归还由 VencFbmRequestValidBuffer 获取的图像帧 buffer
8	VencFbmGetValidBufferNum	获取未完成编码的帧 buffer 个数

3.2.1.1 VencFbmCreate

函数原型	FrameBufferManager* VencFbmCreate(int num, struct ScMemOpsS *memops, void *veOps, void *pVeopsSelf)
功能	创建图像帧缓冲管理模块
参数	num: 缓冲帧 buffer 节点的个数； memops: memory 管理器接口； VeOps：ve 模块的操作句柄； pVeOpsSelf：ve 模块的私有数据指针。
返回值	成功：返回图像缓冲帧管理模块的指针 失败：返回 NULL
调用说明	Num 的个数一般设置为 2~6 之间

3.2.1.2 VencFbmDestroy

函数原型	void VencFbmDestroy(FrameBufferManager* fbm)
功能	创建图像帧缓冲管理模块
参数	fbm: 通过 VencFbmCreate 创建的图像帧管理模块的指针。
返回值	无
调用说明	无

3.2.1.3 VencFbmReset

函数原型	int VencFbmReset(FrameBufferManager* fbm)
功能	帧 buffer 队列清零
参数	fbm: 通过 VencFbmCreate 创建的图像帧管理模块的指针。
返回值	成功：返回 0 失败：返回-1
调用说明	无

3.2.1.4 VencFbmAllocateBuffer

函数原型	int VencFbmAllocateBuffer(FrameBufferManager* fbm, VencAllocateBufferParam *buffer_param, VencInputBuffer*dst_inputBuf)
功能	申请输入图像帧内存，当不使用外部应用程序的 buffer 管理队列的时候，需要调用此接口来申请物理连续的图像帧内存。
参数	Fbm：通过 VencFbmCreate 创建的图像帧管理模块的指针； dst_inputBuf：存放申请出来的 buffer 的相关信息。

返回值	成功：返回 0 失败：返回-1
调用说明	无

3.2.1.5 VencFbmAddValidBuffer

函数原型	int VencFbmAddValidBuffer(FrameBufferManager* fbm, VencInputBuffer *inputbuffer)
功能	添加一帧图像帧到输入帧队列
参数	fbm: 通过 VencFbmCreate 创建的图像帧管理模块的指针； Inputbuffer（输入）：输入图像帧的信息。
返回值	成功：返回 0 失败：返回-1
调用说明	Inputbuffer 参数中包含输入图像帧的信息，包括 pts，地址等信息。 nID：用来识别不同的图像帧； nPts：输入图像帧的时间戳，以 us 为单位； nFlag: 标记此 buffer 中的数据是否属于关键帧； pAddrPhyY：输入图像帧的 Y 分量的物理地址，当有 C 分量的时候，此地址必须有效，硬件编码需要物理地址，当输入的数据为 RGB 格式的时候，pAddrPhyY 用来存储 RGB 数据的地址； pAddrPhyC：输入图像帧的 C 分量的物理地址，此地址必须有效，硬件编码需要物理地址。当输入的数据为 RGB 格式的时候，此地址不使用，可以为空； pAddrVirY：输入图像帧的 Y 分量的虚拟地址； pAddrVirC：输入图像帧的 C 分量的虚拟地址； BEnableCorp: 标记是否使用 crop； sCropInfo：如果使用 crop，给出 crop 区域信息； ispPicVar: isp 对 YUV 噪声强度的评价，默认不使用； roi_param[4]: 图像处理程序对 roi 区域标定，增减编码时的 QP，仅个别芯片会用到。

3.2.1.6 VencFbmRequestValidBuffer

函数原型	int VencFbmRequestValidBuffer(FrameBufferManager* fbm, VencInputBuffer *inputbuffer)
功能	从输入图像帧队列获取一个图像帧，供编码器使用。
参数	fbm: 通过 VencFbmCreate 创建的图像帧管理模块的指针； inputbuffer：输入图像帧的信息。
返回值	成功：返回 0 失败：返回-1
调用说明	编码器编码之前会先调用此函数来获取一个图像帧

3.2.1.7 VencFbmReturnValidBuffer

函数原型	int VencFbmReturnValidBuffer(FrameBufferManager* fbm, VencInputBuffer *inputbuffer)
功能	归还由 VencFbmRequestValidBuffer 获取到的图像帧。
参数	fbm: 通过 VencFbmCreate 创建的图像帧管理模块的指针； inputbuffer：图像帧的信息。
返回值	成功：返回 0 失败：返回-1
调用说明	无

3.2.1.8 VencFbmGetValidBufferNum

函数原型	int VencFbmGetValidBufferNum(FrameBufferManager* fbm)
功能	获取编码器未完成编码的输入 buffer 个数
参数	fbm: 通过 VencFbmCreate 创建的图像帧管理模块的指针
返回值	编码器未完成编码的输入 buffer 个数
调用说明	无

3.2.2 BitStream Manager 模块接口设计

BitStream Manager 模块接口如下所示：

1	BitStreamCreate	创建码流管理模块，申请一块物理连续的内存
2	BitStreamDestroy	销毁码流管理模块，释放内存
3	BitStreamBaseAddress	获取码流内存的虚拟基地址
4	BitStreamBasePhyAddress	获取码流内存的物理基地址
5	BitStreamEndPhyAddress	获取码流内存的 end 地址
6	BitStreamBufferSize	获取码流 buffer 大小
7	BitStreamFreeBufferSize	获取空余的，未被编码器使用的码流内存大小
8	BitStreamFrameNum	获取码流输出队列中的 buffer 数量
9	BitStreamWriteOffset	获取当前可写的码流内存的 offset
10	BitStreamAddOneBitstream	添加一笔码流到码流输出队列
11	BitStreamGetOneBitstream	从码流输出队列中获取一个码流 buffer
12	BitStreamReturnOneBitstream	还回码流 buffer，更新码流内存有效的数据长度
13	BitStreamReset	码流 buffer 队列清零

3.2.2.1 BitStreamCreate

函数原型	BitStreamManager* BitStreamCreate(int nBufferSize, struct ScMemOpsS *memops)
功能	创建码流管理模块，申请一块物理连续的内存。
参数	nBufferSize：申请的码流内存的大小； memops: memory 管理器接口。
返回值	成功：返回码流管理模块的指针 失败：返回 NULL
调用说明	一般申请码流内存为 4~8MB

3.2.2.2 BitStreamDestroy

函数原型	void BitStreamDestroy(BitStreamManager* handle)
功能	销毁码流管理模块，释放内存
参数	Handle：由 BitStreamCreate 申请的码流管理模块的指针
返回值	无
调用说明	无

3.2.2.3 BitStreamBaseAddress

函数原型	void* BitStreamBaseAddress(BitStreamManager* handle)
功能	获取码流内存的虚拟基地址
参数	Handle：由 BitStreamCreate 申请的码流管理模块的指针
返回值	成功：返回码流的基地址（虚拟地址） 失败：返回 NULL
调用说明	码流输出 buffer 的地址计算需要此函数，通过基地址和 offset 可以计算出对应码率的地址。

3.2.2.4 BitStreamBasePhyAddress

函数原型	void* BitStreamBasePhyAddress(BitStreamManager* handle)
功能	获取码流内存的物理基地址
参数	Handle：由 BitStreamCreate 申请的码流管理模块的指针
返回值	成功：返回码流的基地址（物理地址） 失败：返回 NULL
调用说明	编码器需要调用此函数获取到码流内存的基地址（物理地址），此地址会配给硬件。

3.2.2.5 BitStreamEndPhyAddress

函数原型	void* BitStreamEndPhyAddress(BitStreamManager* handle)
功能	获取码流内存的物理 end 地址
参数	Handle：由 BitStreamCreate 申请的码流管理模块的指针
返回值	成功：返回码流的 end 地址（物理地址） 失败：返回 NULL
调用说明	编码器需要调用此函数获取到码流内存的 end 地址（物理地址）

3.2.2.6 BitStreamBufferSize

函数原型	int BitStreamBufferSize(BitStreamManager* handle)
功能	获取码流 buffer 大小
参数	Handle：由 BitStreamCreate 申请的码流管理模块的指针
返回值	码流内存的总大小
调用说明	无

3.2.2.7 BitStreamFreeBufferSize

函数原型	int BitStreamFreeBufferSize(BitStreamManager* handle)
功能	获取码流内存的物理基地址
参数	Handle：由 BitStreamCreate 申请的码流管理模块的指针
返回值	码流内存空余空余大小
调用说明	无

3.2.2.8 BitStreamFrameNum

函数原型	int BitStreamFrameNum(BitStreamManager* handle)
功能	获取输出码流队列中有效 buffer 的个数
参数	Handle：由 BitStreamCreate 申请的码流管理模块的指针
返回值	输出码流队列中有效 buffer 的个数
调用说明	无

3.2.2.9 BitStreamWriteOffset

函数原型	int BitStreamWriteOffset(BitStreamManager* handle)
功能	获取输出码流队列中有效 buffer 的个数

参数	Handle：由 BitStreamCreate 申请的码流管理模块的指针
返回值	输出码流队列中有效 buffer 的个数
调用说明	无

3.2.2.10 BitStreamAddOneBitstream

函数原型	int BitStreamAddOneBitstream(BitStreamManager* handle, StreamInfo* pStreamInfo)
功能	添加一笔码流到码流输出队列
参数	Handle：由 BitStreamCreate 申请的码流管理模块的指针； pStreamInfo：码流 buffer 信息。
返回值	成功：返回 0 失败：返回-1
调用说明	pStreamInfo 中变量的说明： nStreamOffset：当前码流 buffer，相应码流内存起始地址的 offset； nStreamLength：当前码流 buffer 的长度； nPts：当前码流 buffer 的时间戳； nFlags：当前码流 buffer 的标记，可以用此来标记当前码率是否是关键帧； nID：当前码流 buffer 的 ID, 用来区分不用的码流 buffer； VE 编码结束后，会调用此函数把一笔码流添加到码流 buffer 队列中。

3.2.2.11 BitStreamGetOneBitstream

函数原型	StreamInfo* BitStreamGetOneBitstream(BitStreamManager* handle)
功能	从码流输出队列中获取一个码流 buffer
参数	Handle：由 BitStreamCreate 申请的码流管理模块的指针
返回值	成功：返回码流 buffer 信息的指针 失败：返回 NULL
调用说明	外部模块会调用此函数从码流输出队列中获取一个码流 buffer。

3.2.2.12 BitStreamReturnOneBitstream

函数原型	int BitStreamReturnOneBitstream(BitStreamManager* handle, StreamInfo* pStreamInfo)
功能	还回码流 buffer，更新码流内存有效的数据长度
参数	Handle：由 BitStreamCreate 申请的码流管理模块的指针； pStreamInfo：码流 buffer 信息。
返回值	成功：返回 0 失败：返回-1

调用说明	无

3.2.2.13 BitStreamReset

函数原型	int BitStreamReset(BitStreamManager* handle, struct ScMemOpsS *memops)
功能	码流 buffer 队列清零
参数	Handle：由 BitStreamCreate 申请的码流管理模块的指针； memops：memory 管理器接口。
返回值	成功：返回 0 失败：返回-1
调用说明	无

3.2.3 Video Encoder Device 模块接口设计

Video Encoder Device 模块接口如下所示：

1	VencoderDeviceCreate	创建编码设备
2	VencoderDeviceDestroy	销毁编码设备

3.2.3.1 VencoderDeviceCreate

函数原型	VENC_DEVICE *VencoderDeviceCreate(VENC_CODEC_TYPE type)
功能	创建编码设备
参数	type：Codec 类型
返回值	成功：返回编码设备的指针 失败：返回 NULL
调用说明	编码设备的返回值是类型为 VENC_DEVICE 的指针，此结构体中封装了对编码器操作的相关函数。

3.2.3.2 VencoderDeviceDestroy

函数原型	void VencoderDeviceDestroy(void *handle)
功能	销毁编码设备
参数	handle：由 VencoderDeviceCreate 创建的编码设备的指针
返回值	无
调用说明	无

4 数据结构设计

4.1 VencBaseConfig

名称	VencBaseConfig	
功能描述	初始化编码器时的基本配置信息	
属性	类型	描述
nInputWidth	unsigned int	输入图像帧的宽度
nInputHeight	unsigned int	输入图像帧的高度
nDstWidth	unsigned int	编码输出的图像宽度
nDstHeight	unsigned int	编码输出的图像高度
nStride	unsigned int	输入图像在内存中的宽度
eInputFormat	VENC_PIXEL_FMT	输入图像的颜色格式： typedef enum VENC_PIXEL_FMT { VENC_PIXEL_YUV420SP, VENC_PIXEL_YVU420SP, VENC_PIXEL_YUV420P, VENC_PIXEL_YVU420P, VENC_PIXEL_YUV422SP, VENC_PIXEL_YVU422SP, VENC_PIXEL_YUV422P, VENC_PIXEL_YVU422P, VENC_PIXEL_YUYV422, VENC_PIXEL_UYVY422, VENC_PIXEL_YVYU422, VENC_PIXEL_VYUY422, VENC_PIXEL_ARGB, VENC_PIXEL_RGBA, VENC_PIXEL_ABGR, VENC_PIXEL_BGRA, VENC_PIXEL_TILE_32X32, VENC_PIXEL_TILE_128X32, }VENC_PIXEL_FMT
memops	struct ScMemOpsS*	memory 管理器接口

4.2 VencH264ProfileLevel

名称	VencH264ProfileLevel	
功能描述	H264 编码的 profile 和 level	
属性	类型	描述
nProfile	VENC_H264PROFILETYPE	无
nLevel	VENC_H264LEVELTYPE	无

4.3 VencQPRange

名称	VencQPRange	
功能描述	H264 编码的 QP 区间	
属性	类型	描述
nMaxqp	int	取值范围（0~51）
nMinqp	int	取值范围（0~51）

4.4 MotionParam

名称	MotionParam	
功能描述	移动侦测的参数	
属性	类型	描述
nMotionDetectEnable	int	0：关闭移动侦测 1：打开移动侦测
nMotionDetectRatio	int	取值范围（0~12） 0 为最高灵敏度，值越小灵敏度越高，值越大灵敏度越低。

4.5 VencHeaderData

名称	VencHeaderData	
功能描述	存储头信息的结构体	
属性	类型	描述
pBuffer	unsigned char*	H264 编码，会用此来存储 SPS、PPS 信息；JPEG 编码无此结构体。
nLength	unsigned int	头信息的长度

4.6 VencInputBuffer

名称	VencInputBuffer	
功能描述	输入图像帧的信息	
属性	类型	描述
nID	int	用来区分不同的 buffer
nPts	long long	当前图像帧的时间戳
nFlag	unsigned int	标记此 buffer 的数据是否属于关键帧
pAddrPhyY	unsigned char*	当前图像帧 Y 分量的物理地址，配给硬件使用
pAddrPhyC	unsigned char*	当前图像帧的 C 分量的物理地址，配给硬件使用
pAddrVirY	unsigned char*	当前图像帧 Y 分量的虚拟地址
pAddrVirC	unsigned char*	当前图像帧 C 分量的虚拟地址
bEnableCorp	int	0: 关闭 corp 1：打开 corp
sCropInfo	VencRect	当 corp 打开的时候的 corp 矩形区域
ispPicVar	int	isp 对 YUV 数据的噪声评价，默认不使用
roi_param[4]	VencROIConfig	图像处理识别的 ROI 区域，编码器会对这些区域进行 QP 特殊调整, 仅个别芯片会用到。

4.7 VencOutputBuffer

名称	VencOutputBuffer	
功能描述	输出图像帧的信息	
属性	类型	描述
nID	int	用来区分不同的 buffer
nPts	long long	当前输出 buffer 的时间戳
nFlag	int	用来标记是否为关键帧
nSize0	unsigned int	输出码流的第一部分长度，存储的数据在地址 pData0 中。
nSize1	unsigned int	nSize1 != 0，表示输出码流的第二部分长度，数据在地址 pData1 中。
pData0	unsigned char*	输出码流的第一部分地址
pData1	unsigned char*	输出码流的第二部分地址
frame_info	FrameInfo	buffer 中的数据所属帧的 QP 和 GOP 信息，用于码率控制。

4.8 VencAllocateBufferParam

名称	VencAllocateBufferParam	
功能描述	申请图像帧内存的参数	
属性	类型	描述
nBufferNum	unsigned int	申请的图像帧个数
nSizeY	unsigned int	申请图像帧 Y 分量的大小
nSizeC	unsigned int	申请图像帧的 C 分量的大小

4.9 VencH264FixQP

名称	VencH264FixQP	
功能描述	固定 QP 参数	
属性	类型	描述
bEnable	int	0: 码率控制固定 QP 模式关闭 1：码率控制估计 QP 模式打开
nIQp	int	I 帧的 QP(0~51)
nPQp	int	P 帧的 QP(0~51)

4.10 VencCyclicIntraRefresh

名称	VencCyclicIntraRefresh	
功能描述	Cyclic Intra Refresh 信息	
属性	类型	描述
bEnable	int	0: 关闭 1：打开
nBlockNumber	int	一个图像帧划分的区域个数

4.11 VencH264Param

名称	VencH264Param	
功能描述	H264 参数	
属性	类型	描述
sProfileLevel	VencH264ProfileLevel	Profile 和 level 信息
bEntropyCodingCABAC	int	0：熵编码使用 CAVLC 1：熵编码使用 CABAC
sQPRange	VencQPRange	设置 QP 区间

nFramerate	int	帧率，单位为：fps
nSrcFramerate	int	实际帧率，单位为：fps
nBitrate	int	码率，单位为：bps
nMaxKeyInterval	int	关键帧间隔
nCodingMode	VENC_CODING_MODE	可以选择帧编码，还是场编码： VENC_FRAME_CODING VENC_FIELD_CODING
sGopParam	VencGopParam	设置参考帧相关信息，如参考模式等
sRcParam	VencRcParam	设置码率模式相关信息

4.12 VencROIConfig

名称	VencROIConfig	
功能描述	ROI 感兴趣区域设置	
属性	类型	描述
bEnable	int	0：关闭 1：打开
index	int	可使用 4 个 ROI，区域，index 的值可设为（0~3），来选择这四个 ROI 区域。
nQPoffset	int	通过 nQPoffset 可以设置 QP：ROI 区域的 QP 为码率控制产生的 QP 与用户设定的 nQPoffset 的差。 例如： 一帧图像使用固定 QP=30， nQPoffset = 10， 那么 ROI 区域的 QP 为： 30 - 10 = 20
sRect	VencRect	感兴趣区域所表示的矩形区域

4.13 VENC_DEVICE

名称	VENC_DEVICE	
功能描述	编码器内部功能调用接口，用于调用 JPEG 编码、H264 编码等。	
方法	描述	说明
open	原型	void* (*open)()
	参数	无
	返回值	成功：返回值为编码设备上下文信息指针 失败：返回值为 NULL

	功能	打开编码设备
init	原型	Int (*init)(void *handle, VencBaseConfig* pBaseConfig)
	参数	handle: 通过 void* (*open)() 创建的编码设备上下文； pBaseConfig：编码基本配置信息。
	返回值	成功：返回值为 0 失败：返回值为 -1
	功能	初始化编码设备上下文信息
uninit	原型	int (*uninit)(void *handle)
	参数	handle: 通过 void* (*open)() 创建的编码设备上下文。
	返回值	成功：返回值为 0 失败：返回值为 -1
	功能	去初始化编码设备
close	原型	void (*close)(void *handle)
	参数	handle: 通过 void* (*open)() 创建的编码设备上下文。
	返回值	无
	功能	关闭编码设备
encode	原型	void (*encode)(void *handle, VencInputBuffer* pInBuffer)
	参数	handle: 通过 void* (*open)() 创建的编码设备上下文； VencInputBuffer：输入的图像帧。
	返回值	VENC_RESULT_ERROR(-1): 编码出错； VENC_RESULT_OK（0）：编码成功； VENC_RESULT_NO_FRAME_BUFFER（1）：无法获取到输入帧； VENC_RESULT_BITSTREAM_IS_FULL（2）：输出码流 buffer 已经溢出。
	功能	编码一个图像帧
GetParameter	原型	int (*GetParameter)(void *handle, int indexType, void* param)
	参数	handle: 通过 void* (*open)() 创建的编码设备上下文。 indexType：参数索引 Param：参数信息
	返回值	成功：返回值为 0 失败：返回值为 -1
	功能	获取编码器参数
SetParameter	原型	int (*SetParameter)(void *handle, int indexType, void* param)

	参数	handle: 通过 void* (*open)() 创建的编码设备上下文； indexType：参数索引； Param：参数信息。
	返回值	成功：返回值为 0 失败：返回值为 -1
	功能	设置编码器参数

4.14 VencH264AspectRatio

名称	VencH264AspectRatio	
功能描述	VUI 扩展选项，对播放视频时的显示比例限制	
属性	类型	描述
aspect_ratio_idc	unsigned char	一般取值 255，表示启用自定义显示比例
sar_width	unsigned short	显示比例宽度
sar_height	unsigned short	显示比例高度

4.15 VencH264VideoSignal

名称	VencH264VideoSignal	
功能描述	VUI 扩展选项，对颜色空间控制	
属性	类型	描述
video_format	VENC_VIDEO_FORMAT	视频制式，一般取值 5
full_range_flag	unsigned char	全范围色彩空间标识
src_colour_primaries	VENC_COLOR_SPACE	输入源色彩空间： typedef enum { RESERVED0 = 0, VENC_BT709 = 1, RESERVED1 = 2, RESERVED2 = 3, RESERVED3 = 4, VENC_BT601 = 5, BT601_525 = 6, RESERVED4 = 7, VENC_YCC = 8, }VENC_COLOR_SPACE
dst_colour_primaries	VENC_COLOR_SPACE	输出图色彩空间

4.16 VencH264SVCSkip

名称	VencH264SVCSkip	
功能描述	时域可伸缩编码及跳帧，不能与插帧混用	
属性	类型	描述
nTemporalSVC	T_LAYER	时域分层数： typedef enum { NO_T_SVC = 0, T_LAYER_2 = 2, T_LAYER_3 = 3, T_LAYER_4 = 4 }T_LAYER
nSkipFrame	SKIP_FRAME	跳帧倍数，若 nTemporalSVC 为 0，则可独立使用；否则没意义，实际跳帧受 nTemporalSVC 控制： typedef enum { NO_SKIP = 0, SKIP_2 = 2, SKIP_4 = 4, SKIP_8 = 8 }SKIP_FRAME。

4.17 VENC_INDEXTYPE

对函数 VideoEncGetParameter 与 VideoEncSetParameter 中用到的枚举变量 VENC_INDEXTYPE 进行说明：

VENC_INDEXTYPE	引用的数据类型	描述
VENC_IndexParamBitrate	int	单位为：bps
VENC_IndexParamFramerate	int	单位为：fps
VENC_IndexParamMaxKeyInterval	int	设置关键帧最大间隔
VENC_IndexParamIfilter	int	I 帧滤波开关
VENC_IndexParamRotation	int	设置旋转方向（支持 4 个方向）： 0：不旋转； 90：旋转 90 度； 180：旋转 180 度； 270：旋转 270 度。
VENC_IndexParamSliceHeight	int	设置一个 slice 的高度，一帧图像可以支持多个 slice，单位为像素，16 对齐。

VENC_INDEXTYPE	引用的数据类型	描述
VENC_IndexParam ForceKeyFrame	int	在编码过程中，可以强制设置下一帧为关键帧。
VENC_IndexParam MotionDetectEnable	MotionParam	移动侦测开关
VENC_IndexParam MotionDetectStatus	int	编码一帧结束后，可以使用此接口获取当前图像帧是否有物体运动： 0：静止； 1：移动。
VENC_IndexParam Rgb2Yuv	VENC_COLOR_SPACE	颜色空间转换
VENC_IndexParamYuv2Yuv	VENC_YUV2YUV	颜色空间标准转换
VENC_IndexParam ROIConfig	VencROIConfig	人眼感兴趣区域增强
VENC_IndexParamStride	int	图片在内存中的行宽值
VENC_IndexParam ColorFormat	int(VENC_PIXEL_FMT)	输入编码器数据的颜色格式
VENC_IndexParamSize	VencSize	只读。获取图片输入的宽高
VENC_IndexParam SetVbvSize	unsigned int	预设申请 VBV（编码输出）buffer 大小。
VENC_IndexParamVbvInfo	VbvInfo	只读。获取 VBV（编码输出）buffer 信息。
VENC_IndexParam SuperFrameConfig	VencSuperFrameConfig	超大帧重编码处理设置
VENC_IndexParamSetPSkip	unsigned int	插帧开关
VENC_IndexParamResetEnc	int	复位编码器输入输出 buffer，I 帧 QP 更改。
VENC_IndexParam H264QPRange	VencQPRange	设置 QP 波动范围
VENC_IndexParam H264ProfileLevel	VencProfileLevel	nProfile 和 nLevel 取值参考 vencoder.h
VENC_IndexParam H264EntropyCodingCABAC	int	设置熵编码格式。0：CAVLC；1:CABAC
VENC_IndexParam H264CyclicIntraRefresh	VencCyclicIntraRefresh	循环帧内刷新，网络码流使用。
VENC_IndexParam H264FixQP	VencH264FixQP	不使用码率控制，固定 QP

VENC_INDEXTYPE	引用的数据类型	描述
VENC_IndexParamH264SVCSkip	VencH264SVCSkip	此选项不能与插帧选项混用。时域SVC 和跳帧，时域分层取值0/2/3/4。跳帧倍数取值 0/2/4/8。若时域分层不为 0，则跳帧倍数受时域分层控制，对其取值无意义；否则跳帧倍数可独立使用。
VENC_IndexParamH264AspectRatio	VencH264AspectRatio	VUI 扩展选项。限制视频播放时的显示比例。一般把 aspect_ratio_idc 设为 255，显示比例取 sar_width 和 sar_height 的比值。
VENC_IndexParamH264FastEnc	unsigned int	快速编码开关，简化编码操作，编码速度加快，但图像质量和压缩率下降。
VENC_IndexParamH264VideoSignal	VencH264VideoSignal	VUI 扩展选项。选择编码颜色空间。video_format 一般为 5。src_colour_primaries 取 5，dst_colour_primaries 取 8，颜色最明亮；两者取值相反，效果次之；其它取值颜色最灰暗。
VENC_IndexParamH264IQpOffset	int	I 帧 QP 偏移值
VENC_IndexParamJpegEncMode	int	JPEG 编码方式，若编单幅图片，设为 0；若编 MJPEG 序列，设为 1。
VENC_IndexParamJpegVideoSignal	VencJpegVideoSignal	颜色空间选择，同 H264 类似选项。
VENC_IndexParamJpegQuality	int	（0~100）值越大，编码质量越高。
VENC_IndexParamJpegExifInfo	EXIFInfo	JPEG 图片的描述信息，包括快门速度，曝光时间，GPS 信息，缩略图信息等。

5 功能开发

5.1 功能概述

VE 编码模块功能具体使用场景为需要进行视频硬件编码的场景，如需要将一张 YUV/ARGB 格式的图像进行编码压缩为 jpeg/264/265 码流，或进行一些如旋转缩放等图像预处理的工作时，可以使用 VE 编码模块。

5.2 开发流程

1. 首先，调用 VencCreate 接口对编码器进行创建，该接口主要进行三种操作，一是调用 vencoderDeviceCreate 接口创建对应编码格式的句柄，并会将创建后获得的句柄保存在 vencoder_device_handle 中，这样就可以调用该句柄中的接口与 ve 硬件层进行交互；二是调用编码子模块句柄中的 open 接口；三是创建与编码线程进行交互的消息队列。
2. 第二步，会调用 VencInit 接口对创建的编码器进行初始化，该接口主要进行以下的操作：一是对 FBM 模块进行创建，这是为了方便对输入图像帧数据进行管理；二是调用编码子模块句柄中的 init 接口，这一步主要是将一些编码参数和功能参数根据上层设置的进行配置和初始化；三是在消息队列中写入初始化成功的状态发送给编码线程。
3. 第三步，则是调用 VencSetParameter 接口对编码的参数和特性功能进行设置，如启用 roi 编码功能，overlay 功能，或是设置强制 I 帧，调整 QP 值等参数设置，都是通过该接口调用对应 cmd 命令进行设置。
4. 第四步，是调用 VencStart 接口，这个接口的作用就是向编码线程发送可以编程的状态，让线程切换到编码状态。
5. 在开始编码之前，需要先将输入图像帧的数据放入内存中，这样编码器才能有数据进行编码，所以这一步，会去调用 VencQueueInputBuf 接口，而这个接口实际上调用的就是 FBM 模块中的 VencFbmAddValidBuffer 接口，即将输入图像帧的数据通过 FBM 模块管理到内存中，方便编码器获取。
6. 在有可用的编码数据之后，则会调用 encodeOneFrame 接口开始真正的编码，那么该接口主要的实现流程也有三步：首先是调用 VencFbmRequestValidBuffer 接口获取输入图像帧的数据，然后调用编码子模块句柄中的 encode 接口，在调用 BSM 模块保存编码后产生的码流，即完成了一次编码；最后，在调用 FBM 模块中的 VencFbmReturnValidBuffer 接口将用过的 buffer 进行 return。
7. 在结束编码后，上层可以调用 VencDequeueOutputBuf 接口，实际上也就是调用 BSM 模块中的 BitStreamGetOneBitstream 接口去获得当前帧所编出的码流，将其存放到文件中。

8. 在使用完输出的码流之后，就会去调用 VencQueueOutputBuf 接口，实际上就是调用 BSM 模块中的 BitStreamReturnOneBitstream 接口，将使用过的码流进行返还，保证输出 buffer 的剩余空间。

5.3 编程示例

SDK 的 libcedarc/demo 中存放着开发用例，可参考进行开发。

6 调试方法

6.1 调试工具

1. 观察 yuv 工具可使用：7yuv
2. 观察码流工具可使用：VQAnalyzer Elecard StreamEye Tools

6.2 调试节点

1. ve 的调试节点路径：sys/kernel/debug/mpp/ve_*
2. 开启该调试节点除了正常打开 debug 节点的操作外，还需要使用 VencSetParameter 接口配置相关参数使能调试节点。

7 FAQ

7.1 问题描述

问题现象：编码出来的视频有马赛克

问题分析：大概率是和码流及 QP 值的配置相关

问题解决方法：如：1920x1080 + 30fps 的场景，当码率小于 5Mbps，就可以认为是码率较低的场景，就有可能出现马赛克现象，但如果马赛克现象较为严重，也有可能是其他原因导致。

几个场景的经验值：

- 低码率场景：1080p + 30fps + 2Mbps:qp_min = 30, qp_max = 51
- 中码率场景：1080p + 30fps + 10Mbps:qp_min = 20, qp_max = 45
- 高码率场景：1080p + 30fps + 20Mbps:qp_min = 10, qp_max = 45

著作权声明

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明





（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的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所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。