Bitmain FFMPEG 使用指南

注意

您购买的产品、服务或特性等应受比特大陆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，比特大陆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。

技术支持

北京比特大陆科技有限公司

地址：北京市海淀区黑泉路宝盛南路1号院奥北科技园25号楼

邮编：100192

网址：www.sophon.ai

邮箱：support.ai@bitmain.com

发布记录

版本	发布时间	说明
V1.0.0	2020.03.07	1. 增加硬件scale说明, 增加示例等章节。 2. 增加hwupload/hwdownload说明。
V1.1.0	2020.03.17	1. 在scale_bm filter的opt参数处，增加crop和pad说明。 2. 修改例子中CIF写出720P的笔误；修改例子6中文字，以匹配实际命令。 3. 添加例子11和例子12，以说明使用硬件JPEG编解码方法。 4. 添加例子13，以说明软解码与硬编码混用方法。
V1.2.0	2020.03.27	1. 为新接口增加压缩格式说明 2. 增加目录 3. 为各编解码器增加修改参数说明
V1.2.2	2020.03.28	补充scale filter format处解释的不完整。
V 1.3.0	2020.06.16	1. 给视频编码器添加enc-params参数 2. 添加示例14
V 1.3.2	2020.07.02	1. 给scale_bm filter添加flags参数，以及为scale_bm filter添加硬件限制说明。 2.添加示例15
V 1.4.0	2020.08.24	1. 增加章节说明H.264视频编码器bmx264。修改前言。 2. 添加示例16，示例bmx264的使用
V 1.4.2	2020.09.08	1. 为节省内存，推荐硬件视频解码器输出压缩数据。该压缩数据可通过scale_bm filter来解压缩。 2. 优化相关使用示例。 3. 修改章节标题。添加页码。

目　录

前言

BM1684芯片中，有一个8核的A53处理器，同时还内置有视频、图像相关硬件加速模块。在Bitmain提供的FFMPEG SDK开发包中，提供了对这些硬件模块的接口。其中，通过这些硬件接口，提供了如下模块：硬件视频解码器、硬件视频编码器、硬件JPEG解码器、硬件JPEG编码器、硬件scale filter、hwupload filter、hwdownload filter。同时，在PCIE产品中，还支持运行在8核A53处理器上的视频编码模块。

FFMPEG SDK开发包符合FFMPEG hwaccel编写规范，实现了视频转码硬件加速框架，实现了硬件内存管理、各个硬件处理模块流程的组织等功能。

正文中会提到，当前提供的FFMPEG SDK同时提供了新版接口和旧版接口。所谓新版接口，是指的按照FFMPEG hwaccel规范，为BM1684硬件编写的接口。所谓旧版接口，指的是以前没有按照FFMPEG hwaccel规范编写的接口。为支持已有的上层软件，在所提供的SDK开发包中，同时支持这两种接口。

硬件视频解码器

BM1684支持H.264和H.265硬件解码。硬件解码器性能详情如下表所述。

Standard	Profile	Level	Max Resolution	Min Resolution	Bitrate
H.264/AVC	BP/CBP/MP/HP	4.1	8192x4096	16x16	50Mbps
H.265/HEVC	Main/Main10	L5.1	8192x4096	16x16	N/A

在Bitmain的FFMPEG发布包中，H.264硬件视频解码器的名字为*h264_bm*，H.265硬件视频解码器的名字为*hevc_bm*。可通过如下命令，来查询FFMPEG支持的编码器。

```
$ ffmpeg -decoders | grep _bm
```

硬件视频解码器支持的选项

FFMPEG中，BM1684的硬件解码器提供了一些额外选项，可以通过如下命令查询。

```
$ ffmpeg -h decoder=h264_bm
$ ffmpeg -h decoder=hevc_bm
```

这些选项可以使用av_dict_set API来设置。在设置之前，需要对对这些选项有正确的理解。下面详细解释一下这些选项。

output_format:

- 输出数据的格式。
- 设为0，则输出线性排列的未压缩数据；设为101，则输出压缩数据。
- 缺省值为0。
- *推荐设置为101，输出压缩数据。可以节省内存、节省带宽。输出的压缩数据，可以调用后面介绍的scale_bm filter解压缩成正常的YUV数据。具体可参考应用示例中的示例1。*

cbcr_interleave:

- 硬件视频解码器解码输出的帧色度数据是否是交织格式。
- 设为1，则输出为semi-planar yuv图像，譬如nv12；设为0，则输出planar yuv图像，譬如yuv420p。
- 缺省值为1。

extra_frame_buffer_num:

- 硬件视频解码器额外提供硬件帧缓存数量。
- 缺省值为5。最小值为1。

skip_non_idr:

- 跳帧模式。0，关闭；1，跳过Non-RAP帧；2，跳过非参考帧。
- 缺省值为0。

handle_packet_loss

- 出错时，对H.264, H.265解码器使能丢包处理。0, 不做丢包处理；1，进行丢包处理。
- 缺省值为0。

sophon_idx:

- PCIE模式下的sophon设备的编号。与/dev/bm-sophon的设备编号一致。
- 缺省值为0。
- *旧版接口，不推荐使用。*

zero_copy:

- 将设备上的帧数据直接拷贝到AVFrame的data[0]-data[3]所自动申请的内存里。0，关闭拷贝；1，使能拷贝。
- 缺省值为1。
- *仅适用于旧接口的PCIE模式*。新接口提供hwdownload filter，可以显式地把数据从设备内存下载到系统内存。

硬件视频编码器

BM1684中首次添加了硬件视频编码器。 支持H.264/AVC和H.265/HEVC视频编码。

BM1684硬件编码器设计的能力为: 能够实时编码一路1080P30的视频。具体指标如下：

H.265编码器:

- Capable of encoding HEVC Main/Main10/MSP(Main Still Picture) Profile @ L5.1 High-tier

H.264编码器:

- Capable of encoding Baseline/Constrained Baseline/Main/High/High 10 Profiles Level @ L5.2

通用指标

- 最大分辨率 : 8192x8192
- 最小分辨率 : 256x128
- 编码图像宽度须为*8的倍数*
- 编码图像高度宽度须为*8的倍数*

在Bitmain的FFMPEG发布包中，H.264硬件视频编码器的名字为*h264_bm*，H.265硬件视频编码器的名字为*h265_bm*或*hevc_bm*。可通过如下命令, 来查询FFMPEG支持的编码器。

```
$ ffmpeg -encoders
```

硬件视频编码器支持的选项

FFMPEG中， 硬件视频编码器提供了一些额外选项，可以通过如下命令查询。

```
$ ffmpeg -h encoder=h264_bm
$ ffmpeg -h encoder=hevc_bm
```

BM1684硬件视频编码器支持如下选项:

preset: 预设编码模式。推荐通过enc-params设置。

- 0 - fast, 1 - medium, 2 - slow。
- 缺省值为2。

gop_preset: gop预设索引值。推荐通过enc-params设置。

- 1: all I, gopsize 1
- 2: IPP, cyclic gopsize 1
- 3: IBB, cyclic gopsize 1
- 4: IBPBP, cyclic gopsize 2
- 5: IBBBP, cyclic gopsize 4
- 6: IPPPP, cyclic gopsize 4
- 7: IBBBB, cyclic gopsize 4
- 8: random access, IBBBBBBBB, cyclic gopsize 8

qp:

- 恒定量化参数的码率控制方法
- 取值范围为0至51

perf:

- 用于指示是否需要测试编码器性能
- 取值范围为0或1。

enc-params:

- 用于设置视频编码器内部参数。
- 支持的编码参数：preset，gop_preset，qp，bitrate，mb_rc，delta_qp，min_qp，max_qp，bg，nr，deblock，weightp
- 编码参数preset：取值范围为fast, medium, slow或者是0，1，2
- 编码参数gop_preset：gop预设索引值。参考上面已有详细解释。
  1: all I, gopsize 1
  2: IPP, cyclic gopsize 1
  3: IBB, cyclic gopsize 1
  4: IBPBP, cyclic gopsize 2
  5: IBBBP, cyclic gopsize 4
  6: IPPPP, cyclic gopsize 4
  7: IBBBB, cyclic gopsize 4
  8: random access, IBBBBBBBB, cyclic gopsize 8
- 编码参数qp：恒定量化参数，取值范围为[0, 51]。当该值有效时，关闭码率控制算法，用固定的量化参数编码。

- 编码参数bitrate：用于编码所指定的码率。单位是Kbps，1Kbps=1000bps。当指定改参数时，请不要设置编码参数qp。
- 编码参数mb_rc：取值范围0或1。当设为1时，开启宏块级码率控制算法；当设为0时，开启帧级码率控制算法。
- 编码参数delta_qp：用于码率控制算法的QP最大差值。该值太大影响视频主观质量。太小影响码率调整的速度。
- 编码参数min_qp和max_qp：码率控制算法中用于控制码率和视频质量的最小量化参数和最大量化参数。取值范围[0, 51]。
- 编码参数bg：是否开启背景检测。取值范围0或1。
- 编码参数nr：是否开启降噪算法。取值范围0或1。
- 编码参数deblock：是否开启环状滤波器。有如下几种用法：
    - 关闭环状滤波器“deblock=0”或“no-deblock”。
    - 简单开启环状滤波器，使用缺省环状滤波器参数”deblock=1”。
    - 开启环状滤波器并设置参数，譬如”deblock=6,6”。
- 编码参数weightp：是否开启P帧、B帧加权预测。取值范围0或1。

sophon_idx: 仅适用于PCIE模式

- 用于指出要使用的Sophon设备的编号。与/dev/bm-sophon的编号一致。
- 最小值为0, 最大值为Sophon设备数量减1
- *仅适用于旧版接口。*

is_dma_buffer:

- 用于提示编码器，输入的帧缓存是否为设备上的连续物理内存地址。
- 在PCIE模式时，值0表示输入的是系统内存虚拟地址；在SoC模式，值0表示输入的是设备内存的虚拟地址。值1表示，输入的是设备上的连续物理地址。
- 缺省值为1。
- *仅适用于旧版接口。*

A53上的H.264视频编码器

在BM1684芯片上有一个8核的A53处理器。通过该处理器，可以提供额外的、更加灵活的视频编码能力。除了上一章的硬件视频编码器外，此处要介绍的bmx264，就是在PCIE卡形式下，通过设备上的A53处理器提供的一个视频编码器。设备上的视频编码器是x264视频编码器的修改版本。在PCIE产品形态下，通过BM1684芯片，同时支持VPU硬编码和A53软编码。在不占用HOST的CPU的前提下，增强了视频编码能力。

值得注意的是，*在SOC模式下，不支持该编码器*。

在Bitmain的FFMPEG发布包中，H.264视频编码器的名字为*bmx264*。可通过如下命令查询所提供的FFMPEG发布包是否支持bmx264编码器。

```
$ ffmpeg -encoders | grep bmx
```

bmx264编码器支持的选项

FFMPEG中， bmx264编码器提供了一些额外选项，可以通过如下命令查询。

```
$ ffmpeg -h encoder=bmx264
```

bmx264视频编码器支持如下选项:

preset: 预设编码模式。

- 按照编码速度从高到低排列：ultrafast, superfast, veryfast, fast, medium, slow, slower, veryslow, placebo。编码速度越快，往往意味着编码效率越低。
- 缺省值为medium。

preset模式	**具体对应的参数**
ultrafast	--no-8x8dct --aq-mode 0 --b-adapt 0 --bframes 0 --no-cabac --no-deblock --no-mbtree --me dia --no-mixed-refs --partitions none --rc-lookahead 0 --ref 1 --scenecut 0 --subme 0 --trellis 0 --no-weightb --weightp 0
superfast	--no-mbtree --me dia --no-mixed-refs --partitions i8x8,i4x4 --rc-lookahead 0 --ref 1 --subme 1 --trellis 0 --weightp 1
veryfast	--no-mixed-refs --rc-lookahead 10 --ref 1 --subme 2 --trellis 0 --weightp 1 - faster: --no-mixed-refs --rc-lookahead 20 --ref 2 --subme 4 --weightp 1
fast	--rc-lookahead 30 --ref 2 --subme 6 --weightp 1
medium	Default settings apply.
slow	--direct auto --rc-lookahead 50 --ref 5 --subme 8 --trellis 2

slower	--b-adapt 2 --direct auto --me umh --partitions all --rc-lookahead 60 --ref 8 --subme 9 --trellis 2
veryslow	--b-adapt 2 --bframes 8 --direct auto --me umh --merange 24 --partitions all --ref 16 --subme 10 --trellis 2 --rc-lookahead 60
placebo	--bframes 16 --b-adapt 2 --direct auto --slow-firstpass --no-fast-pskip --me tesa --merange 24 --partitions all --rc-lookahead 60 --ref 16 --subme 11 --trellis 2

tune: 针对应用场景进行特定编码优化。

- 预设的编码场景有film，animation，grain，stillimage，psnr，ssim，fastdecode，zerolatency。
- 缺省情况不进行特定优化。

tune值	**说明**	**对应的具体参数**
film	电影片源，对视频的质量非常严格时使用	--deblock -1:-1 --psy-rd <unset>:0.15
animation	动画片源	--bframes {+2} --deblock 1:1 --psy-rd 0.4:<unset> --aq-strength 0.6 --ref {Double if >1 else 1}
grain	颗粒状片源。来自低照度环境拍摄视频、低质相机或者旧时影片等各种因素。	--aq-strength 0.5 --no-dct-decimate --deadzone-inter 6 --deadzone-intra 6 --deblock -2:-2 --ipratio 1.1 --pbratio 1.1 --psy-rd <unset>:0.25 --qcomp 0.8
stillimage	静态图像较多视频。 譬如PPT类讲课视频。	--aq-strength 1.2 --deblock -3:-3 --psy-rd 2.0:0.7
psnr	最大化编码视频的PSNR值	--aq-mode 0 --no-psy
ssim	最大化编码视频的SSIM值	--aq-mode 2 --no-psy
fastdecode	生成有利于快速解码的码流	--no-cabac --no-deblock --no-weightb --weightp 0
zerolatency	主要用于视频直播等零延迟应用场景	--bframes 0 --force-cfr --no-mbtree --sync-lookahead 0 --sliced-threads --rc-lookahead 0

forced-idr:

- 将关键帧编码为IDR帧
- 取值范围为0至1
- 缺省值为0

perf:

- 用于指示是否需要测试编码器性能
- 取值范围为0或1。

enc-params:

- 用于设置视频编码器内部参数。bmx264的配置选项与x264的基本一致。但不支持interlaced编码、场编码等选项，只支持帧编码。
- 字符串形式：”参数1=参数1值:参数2=参数2值:”。多个参数之间用冒号分隔。
- 鉴于x264选项繁多，可以参考x264 --fullhelp的输出或其他一些文献资料 X264 Settings - MeWiki等。
- 实际应用中，首先设置preset的值，在特定需求时可设置tune值。接着，根据需要通过enc-params来细调部分参数。下面结合着某个应用场景，对参数的设定过程进行说明。

典型应用场景下参数配置示例

在上云网关项目应用场景，可对bmx264编码器进行如下设置。首先，preset的缺省设置为medium。在设置为medium的同时，为满足上云网关应用特点，可做如下额外设置：

```
av_dict_set(&opts, "preset", "medium", 0);
av_dict_set(&opts, "enc-params",
“force-cfr=1:repeat-headers=1:qpmin=10:qpmax=51:bframes=0:k
eyint=50:keyint-min=50:rc-lookahead=10:scenecut=0", 0);
```

参数设置	**说明**
force-cfr=1	设置恒定帧率
repeat-headers=1	输出的每个关键帧都有头信息
bframes=0	没有B帧。 当解码器不支持B帧时，设置编码器不生成B帧数据。
keyint=50:keyint-min=50	50帧一个关键帧。 设置编码器上下文的gop_size也可以实现同样目的。
scenecut=0	不开启场景切换检测及设置关键帧功能
rc-lookahead=10	码率控制的lookahead为10帧。 该值越大则增加延时；该为0则影响码率控制效果。
qpmin=10	最小QP为10
qpmax=51	最大QP为51

上面的示例只是bmx264编码器在上云网关应用的一个示例。在不同的应用中，需要根据应用特点修改上面的选项。譬如，在对编码路数要求不高，而编码质量要求很高的情况下，可以将preset设为slow；在解码器支持B帧时，可以设bframes=2，进一步提高压缩效率。

硬件JPEG解码器

在BM1684芯片中，硬件JPEG编码器提供硬件JPEG图像解码输入能力。这里介绍一下，如何通过FFMPEG来实现硬件JPEG解码。

在FFMPEG中, 硬件JPEG解码器的名称为 *jpeg_bm*。可以通过如下命令, 来查看FFMPEG中是否有*jpeg_bm*解码器。

```
$ ffmpeg -decoders | grep jpeg_bm
```

硬件JPEG解码器支持的选项

FFMPEG中， 可以通过如下命令, 来查看*jpeg_bm*解码器支持的选项

```
$ ffmpeg -h decoder=jpeg_bm
```

解码选项的说明如下。硬件JPEG解码器中这些选项, 可以使用 `av_dict_set()`API 函数对其进行重置。

bs_buffer_size: 用于设置硬件JPEG解码器中输入比特流的缓存大小(KBytes)。

- 取值范围(0到INT_MAX)
- 缺省值5120

cbcr_interleave: 用于指示JPEG解码器输出的帧数据中色度数据是否为交织的格式。

- 0: 输出的帧数据中色度数据为plannar的格式
- 1: 输出的帧数据中色度数据为interleave的格式
- 缺省值为0

num_extra_framebuffers: JPEG解码器需要的额外帧缓存数量

- 对于Still JPEG的输入, 建议该值设为0
- 对于Motion JPEG的输入, 建议该值至少为2
- 取值范围(0到INT_MAX)
- 缺省值为2

sophon_idx: 仅适用于PCIE模式。

- 用于指出要使用的Sophon设备的编号。 与/dev/bm-sophon的编号一致
- 最小值为0, 最大值为Sophon设备数量减1

- *仅适用于旧版接口。*

zero_copy:

- 将设备上的帧数据直接拷贝到AVFrame的data[0]-data[3]所自动申请的内存里。0，关闭拷贝；1，使能拷贝。
- 缺省值为1。
- *仅适用于旧接口的PCIE模式*。新接口提供hwdownload filter，可以显式地把数据从设备内存下载到系统内存。

硬件JPEG编码器

在BM1684芯片中，硬件JPEG编码器提供硬件JPEG图像编码输出能力。这里介绍一下，*如何通过FFMPEG来实现硬件JPEG编码*。

在FFMPEG中，硬件JPEG编码器的名称为 *jpeg_bm*。可以通过如下命令，来查看FFMPEG中是否有jpeg_bm编码器。

```
$ ffmpeg -encoders | grep jpeg_bm
```

硬件JPEG编码器支持的选项

FFMPEG中， 可以通过如下命令, 来查看jpeg_bm编码器支持的选项

```
$ ffmpeg -h encoder=jpeg_bm
```

编码选项的说明如下:

sophon_idx: 仅适用于PCIE模式

- 用于指出要使用的Sophon设备的编号。 与/dev/bm-sophon的编号一致。
- 最小值为0, 最大值为Sophon设备数量减1
- *仅适用于旧版接口。*

is_dma_buffer:

- 用于提示编码器，输入的帧缓存是否为设备上的连续物理内存地址。
- 在PCIE模式时，值0表示输入的是系统内存虚拟地址；在SoC模式，值0表示输入的是设备内存的虚拟地址。值1表示，输入的是设备上的连续物理地址。
- 缺省值为1。
- *仅适用于旧版接口。*

硬件JPEG编码器中这些选项, 可以使用 `av_dict_set()` API 函数对其进行重置。

硬件scale filter

BM1684硬件scale filter用于将输入的图像进行"缩放/裁剪/补边"操作。譬如，转码应用。在将1080p的视频解码后，使用硬件scale缩放成720p的，再进行压缩输出。

内容	最大分辨率	最小分辨率	最大缩放倍数
硬件限制	4096x4096	8x8	32

在FFMPEG中，硬件scale filter的名称为 *scale_bm*。

```
$ ffmpeg -filters | grep bm
```

硬件scale filter支持的选项

FFMPEG中，可以通过如下命令, 来查看scaler_bm编码器支持的选项

```
$ ffmpeg -h filter=scale_bm
```

scale_bm选项的说明如下:

w:

- 缩放输出视频的宽度。 请参考ffmpeg scale filter的用法。

h:

- 缩放输出视频的高度。 请参考ffmpeg scale filter的用法。

format:

- 缩放输出视频的像素格式。 请参考ffmpeg scale filter的用法。
- 输入支持的格式为nv12, yuv420p, yuvj420p, yuvj422p, yuvj444p。
- 输出支持yuv420p和yuvj420p两种格式。
- 缺省值"same"。 即输出像素格式与输入像素格式相同。输入为yuv420p，输出为yuv420p; 输入为yuvj420p，输出为yuvj420p。输入为nv12时，缺省输出为yuv420p。
- 支持nv12到yuv420p、nv12到yuvj420p、yuv420p到yuvj420p、yuvj422p到yuvj420p、yuvj422p到yuv420p的格式转换。

opt:

- 缩放操作 (from 0 to 2) (default 0)

- 值0 - 仅支持缩放操作。缺省值。
- 值1 - 支持缩放+自动裁剪操作。命令行参数中可用crop来表示。
- 值2 - 支持缩放+自动补黑边操作。命令行参数中可用pad来表示。

flags:

- 缩放方法 (from 0 to 2) (default 2)
- 值0 - bilinear滤波器。命令行参数中，可用bilinear来表示。
- 值1 - nearest滤波器。命令行参数中，可用nearest来表示。
- 值2 - bicubic滤波器。命令行参数中，可用bicubic来表示。

AVFrame特殊定义说明

遵从FFMPEG的规范, 硬件解码器是通过AVFrame来提供输出的，硬件编码器是通过AVFrame来提供输入的。因此，在通过API方式，调用FFMPEG SDK、进行硬件编解码处理时，需要注意到AVFrame的如下特殊规定。AVFrame是线性YUV输出。在AVFrame中，data为数据指针, 用于保存物理地址，linesize为每个平面的线跨度。

硬件解码器输出的AVFrame接口定义

旧版接口

data数组的定义

下标	说明
0	Y的虚拟地址
1	cbcr_interleave=1 时，CbCr的虚拟地址； cbcr_interleave=0 时，Cb的虚拟地址
2	cbcr_interleave=0 时，Cr的虚拟地址
3	未使用
4	Y的物理地址
5	cbcr_interleave=1 时，CbCr的物理地址； cbcr_interleave=0 时，Cb的物理地址
6	cbcr_interleave=0 时，Cr的物理地址
7	未使用

linesize数组的定义

下标	说明
0	Y的虚拟地址的跨度
1	cbcr_interleave=1时，CbCr的虚拟地址的跨度； cbcr_interleave=0时，Cb的虚拟地址的跨度
2	cbcr_interleave=0时，Cr的虚拟地址的跨度
3	未使用
4	Y物理地址的跨度
5	cbcr_interleave=1时，CbCr的物理地址的跨度； cbcr_interleave=0时，Cb的物理地址的跨度

6	cbcr_interleave=0时，Cr的物理地址的跨度
7	未使用

新版接口

data数组的定义

下标	未压缩格式说明	压缩格式说明
0	Y的物理地址	压缩的亮度数据的物理地址
1	cbcr_interleave=1 时，CbCr的物理地址； cbcr_interleave=0 时，Cb的物理地址	压缩的色度数据的物理地址
2	cbcr_interleave=0 时，Cr的物理地址	亮度数据的偏移量表的物理地址
3	保留	色度数据的偏移量表的物理地址
4	保留	保留

linesize数组的定义

下标	未压缩格式说明	压缩格式说明
0	Y的物理地址的跨度	亮度数据的跨度
1	cbcr_interleave=1时，CbCr的物理地址的跨度； cbcr_interleave=0时，Cb的物理地址的跨度	色度数据的跨度
2	cbcr_interleave=0时，Cr的物理地址的跨度	亮度偏移量表的大小
3	未使用	色度偏移量表的大小

硬件编码码器输入的AVFrame接口定义

旧版接口

data数组的定义

下标	说明
0	Y的虚拟地址
1	Cb的虚拟地址
2	Cr的虚拟地址
3	保留

4	Y的物理地址
5	Cb的物理地址
6	Cr的物理地址
7	未使用

linesize数组的定义

下标	说明
0	Y的虚拟地址的跨度
1	Cb的虚拟地址的跨度
2	Cr的虚拟地址的跨度
3	未使用
4	Y的物理地址的跨度
5	Cb的物理地址的跨度
6	Cr的物理地址的跨度
7	未使用

新版接口

data数组的定义

下标	说明
0	Y的物理地址
1	Cb的物理地址
2	Cr的物理地址
3	保留
4	保留

linesize数组的定义

下标	说明
0	Y的物理地址的跨度
1	Cb的物理地址的跨度

2	Cr的物理地址的跨度
3	未使用

硬件filter输入输出的AVFrame接口定义

该接口仅适用于FFMPEG SDK新版接口。SDK老版接口不支持bmcodec硬件filter。

data数组的定义

下标	**说明**	**压缩格式的输入接口**
0	Y的物理地址	压缩的亮度数据的物理地址
1	Cb的物理地址	压缩的色度数据的物理地址
2	Cr的物理地址	亮度数据的偏移量表的物理地址
3	保留	色度数据的偏移量表的物理地址
4	保留	保留

linesize数组的定义

下标	**说明**	**压缩格式的输入接口**
0	Y的物理地址的跨度	亮度数据的跨度
1	Cb的物理地址的跨度	色度数据的跨度
2	Cr的物理地址的跨度	亮度偏移量表的大小
3	未使用	色度偏移量表的大小

硬件加速在FFMPEG命令中的应用示例

- 示例1

使用设备0。解码H.265视频，输出compressed frame buffer，scale_bm解压缩compressed frame buffer并缩放成CIF，然后编码成H.264码流。

```
$ ffmpeg -hwaccel bmcodec -hwaccel_device 0 \
         -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
         -vf "scale_bm=352:288" \
         -c:v h264_bm -g 256 -b:v 256K \
         -y wkc_100_cif_scale.264
```

- 示例2

使用设备0。解码H.265视频，按比例缩放并自动裁剪成CIF，然后编码成H.264码流。

```
$ ffmpeg -hwaccel bmcodec -hwaccel_device 0 \
         -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
         -vf "scale_bm=352:288:opt=crop" \
         -c:v h264_bm -g 256 -b:v 256K \
         -y wkc_100_cif_scale_crop.264
```

- 示例3

使用设备0。解码H.265视频，按比例缩放并自动补黑边成CIF，然后编码成H.264码流。

```
$ ffmpeg -hwaccel bmcodec -hwaccel_device 0 \
         -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
         -vf "scale_bm=352:288:opt=pad" \
         -c:v h264_bm -g 256 -b:v 256K \
         -y wkc_100_cif_scale_pad.264
```

- 示例4

使用设备0。解码H.265视频，缩放成CIF，并降帧率成25fps，然后编码成H.264码流。

```
$ ffmpeg -hwaccel bmcodec -hwaccel_device 0 \
         -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
         -vf "scale_bm=352:288,fps=fps=25" \
         -c:v h264_bm -g 256 -b:v 256K \
         -y wkc_100_cif_scale_25fps.264
```

- 示例5

演示视频截图功能。

使用设备0。解码H.265视频，按比例缩放并自动补黑边成CIF，然后编码成jpeg图片。

```
$ ffmpeg -hwaccel bmcodec -hwaccel_device 0 \
       -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
       -vf "scale_bm=352:288:opt=pad:format=yuvj420p" \
       -c:v jpeg_bm -vframes 1 \
       -y wkc_100_cif_scale.jpeg
```

- 示例6

演示视频转码+视频截图功能。

使用设备0。硬件解码H.265视频，缩放成CIF，然后编码成H.264码流；同时缩放成720p，然后编码成JPEG图片。

```
$ ffmpeg -hwaccel bmcodec -hwaccel_device 0 \
       -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
       -filter_complex
"[0:v]scale_bm=352:288[img1];[0:v]scale_bm=1280:720:format=yuvj420p[
img2]" \
       -map '[img1]' -c:v h264_bm -b:v 256K -y img1.264 \
       -map '[img2]' -c:v jpeg_bm -vframes 1 -y img2.jpeg
```

- 示例7

演示hwdownload功能。硬件解码H.265视频，然后下载存储成YUV文件。

```
$ ffmpeg -hwaccel bmcodec -hwaccel_device 0 \
  -c:v hevc_bm -cbcr_interleave 0 -i src/wkc_100.265 \
  -vf "hwdownload,format=yuv420p|bmcodec" \
  -y test_transfer.yuv
```

- 示例8

演示hwdownload功能。硬件解码H.265视频，缩放成CIF格式，然后下载存储成YUV文件。

```
$ ffmpeg -hwaccel bmcodec -hwaccel_device 0 \
  -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
  -vf "scale_bm=352:288,hwdownload,format=yuv420p|bmcodec" \
  -y test_transfer_cif.yuv
```

- 示例9

演示hwupload功能。使用设备0。上传YUV视频，然后编码H.264视频。

```
$ ffmpeg -init_hw_device bmcodec=foo:0 \
  -s 1920x1080 -i test_transfer.yuv \
  -filter_hw_device foo -vf "format=yuv420p|bmcodec,hwupload" \
  -c:v h264_bm -b:v 3M -y test_transfer.264
```

这里foo为设备0的别名。

- 示例10

演示hwupload功能。使用设备1。上传YUV视频，并缩放成CIF，然后编码H.264视频。

```
$ ffmpeg -init_hw_device bmcodec=foo:1 \
```

```
  -s 1920x1080 -i test_transfer.yuv \
  -filter_hw_device foo \
  -vf "format=yuv420p|bmcodec,hwupload,scale_bm=352:288" \
  -c:v h264_bm -b:v 256K -y test_transfer_cif.264
```

这里foo为设备1的别名。

- 示例11

演示hwdownload功能。硬件解码YUVJ444P的JPEG视频，然后下载存储成YUV文件。

```
$ ffmpeg -hwaccel bmcodec -hwaccel_device 0 \
  -c:v jpeg_bm -i src/car/1920x1080_yuvj444.jpg \
  -vf "hwdownload,format=yuvj444p|bmcodec" \
  -y car_1080p_yuvj444_dec.yuv
```

- 示例12

演示hwupload功能。使用设备1。上传YUVJ444P图像数据，然后编码JPEG图片。

```
 $ ffmpeg -init_hw_device bmcodec=foo:1 \
   -s 1920x1080 -pix_fmt yuvj444p -i car_1080p_yuvj444.yuv \
   -filter_hw_device foo -vf 'format=yuvj444p|bmcodec,hwupload' \
   -c:v jpeg_bm -y car_1080p_yuvj444_enc.jpg
```

这里foo为设备1的别名。

- 示例13

演示软解码和硬编码混合使用的方法。使用设备2。使用ffmpeg自带的h264软解码器，解码H.264视频，上传解码后数据到芯片2，然后编码H.265视频。

```
 $ ffmpeg -init_hw_device bmcodec=foo:2 \
   -c:v h264 -i src/1920_18MG.mp4 \
   -filter_hw_device foo -vf 'format=yuv420p|bmcodec,hwupload' \
   -c:v h265_bm -g 256 -b:v 5M \
   -y test265.mp4
```

这里foo为设备2的别名。

- 示例14

演示使用enc-params设置视频编码器的方法。使用设备0。解码H.265视频，缩放成CIF，然后编码成H.264码流。

```
 $ ffmpeg -hwaccel bmcodec -hwaccel_device 0 \
          -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
          -vf "scale_bm=352:288" \
          -c:v h264_bm -g 50 -b:v 32K \
          -enc-params
"gop_preset=2:mb_rc=1:delta_qp=3:min_qp=20:max_qp=40" \
          -y wkc_100_cif_scale_ipp_32Kbps.264
```

- 示例15

使用设备0。解码H.265视频，使用bilinear滤波器，按比例缩放成CIF，并自动补黑边，然后编码成H.264码流。

```
 $ ffmpeg -hwaccel bmcodec -hwaccel_device 0 \
          -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
          -vf "scale_bm=352:288:opt=pad:flags=bilinear" \
          -c:v h264_bm -g 256 -b:v 256K \
          -y wkc_100_cif_scale_pad.264
```

- 示例16

演示使用bmx264编码器，设置enc-params视频编码器的方法。使用设备0。解码H.265视频，缩放成CIF，然后编码成H.264码流。此处根据实际情况，修改固件路径。

```
 $ export BMCPU_FIPFILE="/data/firmware/fip.bin"
 $ export BMCPU_RAMBOOTFILE="/data/firmware/ramboot_rootfs.itb"
 $ export BMX264_LIBFILE="/data/firmware/libx264.so"
 $ ffmpeg -hwaccel bmcodec -hwaccel_device 0 \
          -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
          -vf "scale_bm=352:288" \
```

```
          -c:v bmx264 -b:v 128K -maxrate 256K -bufsize 256K \
          -enc-params
"force-cfr=1:repeat-headers=1:qpmin=10:qpmax=51:bframes=0:keyint=50:
keyint-min=50:rc-lookahead=10:scenecut=0" \
          -y wkc_100_cif_scale_ipp_128Kbps.264
```

通过调用API方式来使用硬件加速功能

旧版接口用法，具体请参考samples/ff_video_xcode/这个例子。

新版接口用法，具体请参考samples/ff_video_scale/bmcodec_scale.c这个例子。