Multimedia Manual 使用手册

发行版本 (HEAD detached from eb5ee6b)

SOPHGO

2025 年 06 月 19 日

目录

CHAPTER 1

声明

法律声明

注意

您购买的产品、服务或特性等应受算能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，算能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

地址
北京市海淀区丰豪东路 9 号院中关村集成电路设计园（ICPARK）1 号楼

邮编
100094

网址
https://www.sophgo.com/

邮箱
sales@sophgo.com

电话
+86-10-57590723 +86-10-57590724

CHAPTER 2

Release note

版本	发布日期	说明
V0.1.0	2023.08.31	第一次发布，包括 sophon ffmpeg 和 sophon opencv
V0.2.0	2024.05.08	第二次发布，增加 sophon gstreamer

CHAPTER 3

安装 sophon-media

sophon-media 在不同的 Linux 发行版上提供不同类型的安装方式。请根据您的系统选择对应的方式, 不要在一台机器上混用多种安装方式。以下描述中以“1.0.0 ”为示例，当前实际安装版本会有变化。

下文中 $arch $system 根据实际架构进行配置:

- 主机为 x86 cpu 的，$arch 为 amd64, $system 为 x86_64
- 主机为 arm64 或飞腾 cpu 的，$arch 为 arm64, $system 为 aarch64

如果使用 Debian/Ubuntu 系统： sophon-media 安装包由六个文件构成：

- sophon-media-soc-sophon-ffmpeg_1.0.0_arm64.deb
- sophon-media-soc-sophon-ffmpeg-dev_1.0.0_arm64.deb
- sophon-media-soc-sophon-opencv_1.0.0_arm64.deb
- sophon-media-soc-sophon-opencv-dev_1.0.0_arm64.deb
- sophon-media-soc-sophon-gstreamer_1.0.0_arm64.deb
- sophon-media-soc-sophon-gstreamer-dev_1.0.0_arm64.deb

其中：

sophon-media-soc-sophon-ffmpeg/sophon-media-soc-sophon-opencv/sophon-media-soc-sophon-gstreamer 包含了 ffmpeg/opencv/gstreamer 运行时环境 (库文件、工具等); sophon-media-soc-sophon-ffmpeg-dev/sophon-media-soc-sophon-opencv-dev/sophon-media-soc-sophon-gstreamer-dev 包含了开发环境 (头文件、pkgconfig、cmake 等)。如果只是在部署环境上安装，则不需要安装 sophon-media-soc-sophon-ffmpeg-dev/sophon-media-soc-sophon-opencv-dev/sophon-media-soc-sophon-gstreamer-dev。

sophon-media-soc-sophon-ffmpeg 依赖 sophon-libsophon 包，而 sophon-media-soc-sophon-opencv 依赖 sophon-media-soc-sophon-ffmpeg, 因此在安装次序上必须先安装 libsophon, 然后 sophon-media-soc-sophon-ffmpeg, 最后安装 sophon-media-soc-sophon-opencv。sophon-media-soc-sophon-ffmpeg 仅依赖 sophon-libsophon 包，与 sophon-media-soc-sophon-opencv 和 sophon-media-soc-sophon-ffmpeg 不存在依赖关系。

安装步骤如下：安装 libsophon 依赖库—参考《LIBSOPHON 使用手册》[sudo dpkg -i sophon-soc-libsophon_0.4.9_arm64.deb]

安装 plugin 依赖：

```
sudo apt install libgstreamer1.0-dev gstreamer1.0-plugins-base libgstreamer-plugins-base1.0-dev \
gstreamer1.0-plugins-bad libgstreamer-plugins-bad1.0-dev gstreamer1.0-gl gstreamer1.0-tools \
gstreamer1.0-plugins-good
```

安装 sophon-media

如果使用 **Debian/Ubuntu** 系统：

sudo dpkg -i sophon-media-soc-sophon-ffmpeg_1.0.0_arm64.deb

sudo dpkg -i sophon-media-soc-sophon-ffmpeg-dev_1.0.0_arm64.deb

sudo dpkg -i sophon-media-soc-sophon-opencv_1.0.0_arm64.deb

sudo dpkg -i sophon-media-soc-sophon-opencv-dev_1.0.0_arm64.deb

sudo dpkg -i sophon-media-soc-sophon-gstreamer_1.0.0_arm64.deb

sudo dpkg -i sophon-media-soc-sophon-gstreamer-dev_1.0.0_arm64.deb

在终端执行如下命令，或者 logout 再 login 当前用户后即可使用安装的工具：

source /etc/profile

注意：位于 SOC 模式时，系统已经预装了：

sophon-media-soc-sophon-ffmpeg

sophon-media-soc-sophon-opencv

只需要按照上述步骤安装：

sophon-media-soc-sophon-ffmpeg-dev_1.0.0_arm64.deb

sophon-media-soc-sophon-opencv-dev_1.0.0_arm64.deb

sophon-media-soc-sophon-gstreamer_1.0.0_arm64.deb

sophon-media-soc-sophon-gstreamer-dev_1.0.0_arm64.deb

安装位置为：

/opt/sophon/

├── libsophon-0.4.9

├── libsophon-current -> /opt/sophon/libsophon-0.4.9

```
├── sophon-ffmpeg_1.0.0
│   ├── bin
│   ├── data
│   ├── include
│   ├── lib
│   │   ├── cmake
│   │   └── pkgconfig
│   └── share
├── sophon-ffmpeg-latest -> /opt/sophon/sophon-ffmpeg_1.0.0
├── sophon-opencv_1.0.0
│   ├── bin
│   ├── data
│   ├── include
│   ├── lib
│   │   ├── cmake
│   │   ├── opencv4
│   │   └── pkgconfig
│   ├── opencv-python
│   ├── share
│   └── test
├── sophon-opencv-latest -> /opt/sophon/sophon-opencv_1.0.0
├── sophon-gstreamer_1.0.0
│   ├── data
│   ├── include
│   ├── lib
└── sophon-gstreamer-latest -> /opt/sophon/sophon-gstreamer_1.0.0
```

deb 包安装方式并不允许您安装同一个包的多个不同版本，但您可能用其它方式在 /opt/sophon 下放置了若干不同版本。在使用 deb 包安装时，/opt/sophon/sophon-ffmpeg-latest，/opt/sophon/sophon-opencv-latest 和 /opt/sophon/sophon-gstreamer-latest 会指向最后安装的那个版本。在卸载后，它会指向余下的最新版本（如果有的话）。

其中 include , lib/cmake lib/pkgconfig 和 include 目录，分别由 sophon-media-soc-sophon-ffmpeg-dev , sophon-media-soc-sophon-opencv-dev 和 sophon-media-soc-sophon-gstreamer-dev 包安装产生

卸载方式：

如果使用 Debian/Ubuntu 系统：

```
sudo apt remove sophon-media-soc-sophon-opencv-dev sophon-media-soc-sophon-opencv
sudo apt remove sophon-media-soc-sophon-ffmpeg-dev sophon-media-soc-sophon-ffmpeg
sudo apt remove sophon-media-soc-sophon-gstreamer-dev sophon-media-soc-sophon-gstreamer
或者:
sudo dpkg -r sophon-media-soc-sophon-opencv-dev
sudo dpkg -r sophon-media-soc-sophon-opencv
sudo dpkg -r sophon-media-soc-sophon-ffmpeg-dev
sudo dpkg -r sophon-media-soc-sophon-ffmpeg
sudo dpkg -r sophon-media-soc-sophon-gstreamer-dev
sudo dpkg -r sophon-media-soc-sophon-gstreamer
```

如果使用其它 Linux 系统：安装包由一个文件构成：

- sophon-media-soc_1.0.0_aarch64.tar.gz

可以通过如下步骤安装:

先按照《LIBSOPHON 使用手册》安装好 libsophon 包，然后

tar -xzvf sophon-media-soc_1.0.0_aarch64.tar.gz

sudo cp -r sophon-media_1.0.0_$system/* /

sudo ln -s /opt/sophon/sophon-ffmpeg_1.0.0 /opt/sophon/sophon-ffmpeg-latest

sudo ln -s /opt/sophon/sophon-opencv_1.0.0 /opt/sophon/sophon-opencv-latest

sudo ln -s /opt/sophon/sophon-sample_1.0.0 /opt/sophon/sophon-sample-latest

sudo sed -i "s/usr/local/opt/sophon/sophon-ffmpeg-latest/g" /opt/sophon/sophon-ffmpeg-latest/lib/pkgconfig/*.pc

sudo sed -i "s/^prefix=.*$/prefix=/opt/sophon/sophon-opencv-latest/g" /opt/sophon/sophon-opencv-latest/lib/pkgconfig/opencv4.pc

最后, 安装 bz2 libc6 libgcc 依赖库 (这部分需要根据操作系统不同，选择对应的安装包，这里不统一介绍) 然后是一些配置工作：

```
添加库和可执行文件路径:
sudo cp /opt/sophon/sophon-ffmpeg-latest/data/01_sophon-ffmpeg.conf /etc/ld.so.conf.d/
sudo cp /opt/sophon/sophon-opencv-latest/data/02_sophon-opencv.conf /etc/ld.so.conf.d/
sudo ldconfig
sudo cp /opt/sophon/sophon-ffmpeg-latest/data/sophon-ffmpeg-autoconf.sh /etc/profile.d/
sudo cp /opt/sophon/sophon-opencv-latest/data/sophon-opencv-autoconf.sh /etc/profile.d/
sudo cp /opt/sophon/sophon-sample-latest/data/sophon-sample-autoconf.sh /etc/profile.d/
source /etc/profile
```

卸载方式：

```
sudo rm -f /etc/ld.so.conf.d/01_sophon-ffmpeg.conf
sudo rm -f /etc/ld.so.conf.d/02_sophon-opencv.conf
sudo ldconfig
sudo rm -f /etc/profile.d/sophon-ffmpeg-autoconf.sh
sudo rm -f /etc/profile.d/sophon-opencv-autoconf.sh
sudo rm -f /etc/profile.d/sophon-sample-autoconf.sh
sudo rm -f /opt/sophon/sophon-ffmpeg-latest
sudo rm -f /opt/sophon/sophon-opencv-latest
sudo rm -f /opt/sophon/sophon-sample-latest
sudo rm -rf /opt/sophon/sophon-ffmpeg_1.0.0
sudo rm -rf /opt/sophon/sophon-opencv_1.0.0
sudo rm -rf /opt/sophon/sophon-sample_1.0.0
sudo rm -rf /opt/sophon/opencv-bmcpu_1.0.0
```

注意事项

- **如果需要用 sophon-opencv 的 python 接口，手动设置环境变量：**
  export PYTHONPATH=$PYTHONPATH:/opt/sophon/sophon-media_1.0.0/opencv-python

CHAPTER 4

使用 sophon-sample

sophon-sample 在不同的 Linux 发行版上提供不同类型的安装方式。请根据您的系统选择对应的方式, 不要在一台机器上混用多种安装方式。以下描述中“1.0.0 ”仅为示例, 视当前实际安装版本会有变化。

下文中 $arch $system 根据实际架构进行配置:

- 主机为 x86 cpu 的,$arch 为 amd64, $system 为 x86_64
- 主机为 arm64 或飞腾 cpu 的,$arch 为 arm64, $system 为 aarch64

如果使用 Debian/Ubuntu 系统:

sophon-sample 安装包由以下文件构成:

- sophon-media-soc-sophon-sample_1.0.0_arm64.deb

其中:

- sophon-media-soc-sophon-sample 包含了数个用于测试 sophon-ffmpeg/sophon-opencv 的应用程序;
- sophon-media-soc-sophon-sample 依赖上一章节的 sophon-ffmpeg/sophon-opencv 包。

安装步骤如下:

```
安装libsophon依赖库(参考《LIBSOPHON使用手册》)
安装sophon-media(参考上一章节)
安装sophon-sample
```

如果使用 Debian/Ubuntu 系统:

- sudo dpkg -i sophon-media-soc-sophon-sample_1.0.0_arm64.deb

安装位置为:

```
/opt/sophon/
├─── libsophon-0.4.9
├─── libsophon-current -> /opt/sophon/libsophon-0.4.9
├─── sophon-ffmpeg_1.0.0
├─── sophon-ffmpeg-latest -> /opt/sophon/sophon-ffmpeg_1.0.0
├─── sophon-opencv_1.0.0
├─── sophon-opencv-latest -> /opt/sophon/sophon-opencv_1.0.0
├─── sophon-gstreamer_1.0.0
├─── sophon-gstreamer-latest -> /opt/sophon/sophon-gstreamer_1.0.0
├─── sophon-sample_1.0.0
|   | ─── bin
|   |   ├─── test_bm_restart
|   |   ├─── test_ff_bmcv_transcode
|   |   ├─── test_ff_scale_transcode
|   |   ├─── test_ff_overlay_transcode
|   |   ├─── test_ff_video_encode
|   |   ├─── test_ff_video_xcode
|   |   ├─── test_ff_hw_bmcv_transcode
|   |   ├─── test_ff_resize_transcode
|   |   ├─── test_ff_bmjpeg
|   |   ├─── test_ff_bmjpeg_dec_recycle
|   |   ├─── test_ff_hw_bmcv_transcode
|   |   ├─── test_ff_sns_xcode
|   |   ├─── test_ocv_jpubasic
|   |   ├─── test_ocv_jpumulti
|   |   ├─── test_ocv_vidbasic
|   |   ├─── test_ocv_video_xcode
|   |   ├─── test_ocv_vidmulti
|   |   ├─── test_gst_transcode
|   |   └─── test_gst_vcmulti
|   ├─── data
|   └─── samples
└─── sophon-sample-latest -> /opt/sophon/sophon-sample_1.0.0
```

deb 包安装方式并不允许您安装同一个包的多个不同版本, 但您可能用其它方式在/opt/sophon 下放置了若干不同版本。在使用 deb 包安装时/opt/sophon/sophon-sample-latest 会指向最后安装的那个版本。在卸载后, 它会指向余下的最新版本（如果有的话)。

注意: soc 模式下, deb 安装包为

sophon-media-soc-sophon-sample_1.0.0_arm64.deb

安装位置同上

卸载方式:

如果使用 Debian/Ubuntu 系统:

- sudo apt remove sophon-media-soc-sophon-sample 或者:
- sudo dpkg -r sophon-media-soc-sophon-sample_1.0.0_arm64.deb

用例介绍: test_bm_restart

此用例主要用于测试 ffmpeg 模块下的视频解码功能和性能, 支持多路解码和断线重连功能。用户可以通过用例监测视频、码流的解码情况。

test_bm_restart [api_version] [yuv_format] [pre_allocation_frame] [codec_name] [sophon_idx] [zero_copy] [input_file/url] [input_file/url]

参数:

-api_version
指定解码过程使用的 ffmpegAPI 版本

0: 使用老版本的解码 avcodec_decode_video2 接口

1: 使用新版解码 avcodec_send_packet 接口

2: 使用 av_parser_parse2 的 API 用于抓包

-yuv_format
是否压缩数据,

0 表示不压缩

1 表示压缩

-pre_allocation_frame
允许的缓存帧数, 最多为 64

-codec_name
指定解码器, 可选择 h264_bm/hevc_bm,no 为不指定

-sophon_idx
若处于 SOC 模式, 该选项可以随意设置（不可为空）, 其值将会被忽略

-zero_copy
SOC 模式,0 表示启用 Host memory, 1 表示不启用

-input_file_or_url
输入的文件路径或码流地址

e.g
test_bm_restart 1 0 1 no 0 0 ./example0.mp4 ./example1.mp4 ./example2.mp4

test_ff_bmcv_transcode

此用例主要用于测试 ffmpeg 模块下的视频转码功能和性能, 通过调用 ff_video_decode,ff_video_encode 用例中的数据类型和函数, 来实现先解码后编码的转码过程, 以此保证解码和编码功能的正确性。同时此用例也可测试 ffmpeg 下的转码性能, 运行时程序会输出即时转码帧率供参考。

test_ff_bmcv_transcode [platform] [src_filename] [output_filename] [encode_pixel_format] [codecer_name] [width] [height] [frame_rate] [bitrate] [thread_num]

参数:

-platform
平台: soc

-src_filename
输入文件名如 x.mp4 x.ts 等

-output_filename
转码输出文件名如 x.mp4,x.ts 等

-encode_pixel_format
编码格式如 I420.

-encoder_name
编码 h264_bm,h265_bm.

-width
编码宽度 (32,4096]

-height
编码高度 (32,4096]

-frame_rate
编码帧率

-bitrate
编码比特率 encode bitrate 500 < bitrate < 10000

-thread_num
线程数量

e.g

soc mode example:
test_ff_bmcv_transcode soc example.mp4 test.ts I420 h264_bm 800 400 25 3000 3

test_ff_scale_transcode

此用例主要用于测试 ffmpeg 下视频转码的功能和性能。此功能通过先解码再编码的过程实现，主要调用了 ff_video_decode, ff_video_encode 中的数据类型和函数。

test_ff_scale_transcode [src_filename] [output_filename] [encode_pixel_format] [codecer_name] [height] [width] [frame_rate] [bitrate] [thread_num] [zero_copy] [sophon_idx]

参数:

-src_filename
输入文件名如 x.mp4 x.ts···

-output_filename
输出文件名如 x.mp4 x.ts···

-encode_pixel_format
编码格式如 I420

-codecer_name
编码名如 h264_bm,hevc_bm,h265_bm

-height
编码高度

-width
编码宽度

-frame_rate
编码帧率

-bitrate
编码比特率

-thread_num
使用线程数

-zero_copy
0: copy host mem,1: nocopy.

-sophon_idx
设备索引

e.g

test_ff_scale_transcode example.mp4 test.ts I420 h264_bm 800 400 25 3000 3 0 0

test_ff_overlay_transcode

此用例主要用于测试 ffmpeg 下视频添加滤镜的功能和性能。此功能通过先解码再编码的过程实现，主要调用了 ff_video_decode, ff_video_encode 中的数据类型和函数。

test_ff_overlay_transcode [src_filename] [output_filename] [encoder_name] [zero_copy] [sophon_idx] [overlay_num] [overlay_filepath_1] [x] [y] [overlay_filepath_2] [x] [y]

参数:

-src_filename
输入文件名如 x.mp4 x.ts···

-output_filename
输出文件名如 x.mp4 x.ts···

-encoder_name
编码名如 h264_bm,hevc_bm,h265_bm

-zero_copy
0: copy host mem,1: nocopy.

-sophon_idx
设备索引

-overlay_num
滤镜个数

-overlay_filepath_1
滤镜 1 输入路径

-x
在源视频上 overlay1 的水平位置

-y
在源视频上 overlay1 的垂直位置

-overlay_filepath_2
滤镜 2 输入路径

-x
在源视频上 overlay2 的水平位置

-y
在源视频上 overlay2 的垂直位置

e.g

test_ff_overlay_transcode src.mp4 out.ts h264_bm 0 0 2 overlay_1.264 10 10 overlay_2.264 500 500

test_ff_video_encode

此用例主要用于测试 ffmpeg 模块下视频的编码功能。输入的视频限制为 I420 和 NV12 格式。通过调用此用例用户可以得到封装好的视频文件,ffmpeg 支持的视频格式均可。

test_ff_video_encode <input file> <output file> <encoder> <width> <height> <roi_enable> <input pixel format> <bitrate(kbps)> <frame rate>

参数:

-input_file
输入视频路径

-output_file
输出视频文件名

-encoder
H264 或者 H265, 默认为 H264

-width
视频宽度, 输出与输入需一致,256 <= width <= 8192

-height
视频高度, 输出与输入需一致,128 <= height <= 8192

-roi_enable
是否开启 roi,0 表示不开启,1 表示开启

-input_pixel_format
I420(YUV, 默认), NV12.

-bitrate
输出比特率,10 < bitrate <= 100000, 默认为帧率 x 宽 x 高/8

-framerate
输出帧率,10 < framerate <= 60, 默认为 30

e.g

- test_ff_video_encode <input file> <output file> H264 width height 0 I420 3000 30
- test_ff_video_encode <input file> <output file> H265 width height 0 I420
- test_ff_video_encode <input file> <output file> H265 width height 0 NV12
- test_ff_video_encode <input file> <output file> H265 width height 0

test_ff_video_xcode

此用例主要用于测试 ffmpeg 下视频转码的功能和性能。此功能通过先解码再编码的过程实现, 主要调用了 ff_video_decode,ff_video_encode 中的数据类型和函数。转码后的视频分辨率与原视频一致, 比特率不能超过 10000kbps 或小于 500kbps, 否则会被置为默认值 3000kbps。转码后的视频如比特率与原视频一致, 那么时长也应一致。有一些丢帧属于正常现象。

test_ff_video_xcode <input file> <output file> encoder framerate bitrate(kbps) isdmabuffer

参数:

-input_file
输入文件

-output_file
输出文件

-encoder
编码器 H264 或者 H265.

-isdmabuffer
是否开启内存一致,1 表示不开启,0 表示开启

e.g

- test_ff_video_xcode ./file_example_MP4_1920_18MG.mp4 tran5.ts H264 30 3000 1
- test_ff_video_xcode ./file_example_MP4_1920_18MG.mp4 tran5.ts H264 30 3000 0

test_ff_hw_bmcv_transcode

此用例主要用于测试 ffmpeg 下视频转码的功能和性能。此功能通过先解码再编码的过程实现, 主要调用了 ff_video_decode,ff_video_encode 中的数据类型和函数。

test_ff_hw_bmcv_transcode [platform] [src_filename] [output_filename] [codecer_name] [width] [height] [frame_rate] [bitrate] [thread_num] [en_bmx264] [zero_copy] [sophon_idx]

参数:

-platform
soc

-src_filename
输入文件

-output_filename
输出文件

-codecer_name
编码器 H264 或者 H265.

-width
编码视频宽度

-height
编码视频高度

-frame_rate
编码帧率

-bitrate
编码比特率

-thread_num
线程数

-en_bmx264
soc：0

-zero_copy
soc：0

-sophon_idx
soc：0

e.g

- test_ff_hw_bmcv_transcode soc INPUT.264 out.264 h264_bm 1920 1080 25 3000 0 0 0

test_ff_sns_xcode

此用例主要用于测试基于 v4l2 与 ffmpeg 的视频转码功能和性能, 该功能通过先解码再编码的过程实现, 它支持从视频采集设备 (如摄像头) 直接采集视频流, 经过解码、宽动态 WDR、ISP 处理、编码等流程, 最终输出为指定格式的视频文件。它可以验证转码、编码、基于 v4l2 的 vi 与 isp 驱动的采集、处理等全流程的功能和性能。

test_ff_sns_xcode [devnum] [input_dev] [output_file] [wdr_on] ···[encoder] [framerate] [bitrate(kbps)] [use_isp] [v4l2_buf_num] [framenum] [loopflag]

参数:

-devnum
输入设备 (通常为摄像头) 的数量, 范围为 [1,12]

-input_dev
输入文件或设备路径, 如 /dev/video0

-output_file
输出文件

-wdr_on
是否开启宽动态 (WDR) 功能

-encoder
编码名如 H264 或 H265, 默认为 H264

-framerate
编码帧率

-bitrate(kbps)
编码比特率

-use_isp
是否使用 ISP 硬件处理, 0 表示不使用, 1 表示使用

-v4l2_buf_num
v4l2 缓冲区数量

-framenum
采集帧数

-loopflag
当 loopflag=0 时, 程序会根据设定的帧数 (framenum) 进行采集、编码和解码, 处理完成后自动结束, 生成有效的视频文件; 当 loopflag=1 时, 程序会持续不断地进行采集、编码和解码, 不会根据帧数停止, 也不会生成有效的视频文件, 主要用于持续性能测试或压力测试

e.g

test_ff_sns_xcode 6 /dev/video0 /mnt/video0.264 0 /dev/video1 /mnt/video1.264 0 /dev/video2 /mnt/video2.264 0 /dev/video3 /mnt/video3.264 0 /dev/video4 /mnt/video4.264 0 /dev/video5 /mnt/video5.264 0 H264 30 3000 1 10 901 0

[注意]: [input_dev] [output_file] [wdr_on] 这三个参数的数量都要跟 [devnum] 保持一致, 多输入设备时这三个参数需要连续输入。

test_ff_resize_transcode

此用例主要用于测试 ffmpeg 下视频转码的功能和性能。此功能通过先解码再编码的过程实现, 主要调用了 ff_video_decode,ff_video_encode 中的数据类型和函数。

test_ff_resize_transcode [src_filename] [output_filename] [encode_pixel_format] [codecer_name] [height] [width] [frame_rate] [bitrate] [thread_num] [zero_copy] [sophon_idx]

参数:

-platform
soc

-src_filename
输入文件

-output_filename
输出文件

-encode_pixel_format
编码格式如 I420

-codecer_name
编码名如 h264_bm , hevc_bm , h265_bm

-width
编码视频宽度

-height
编码视频高度

-frame_rate
编码帧率

-bitrate
编码比特率

-thread_num
线程数

-zero_copy
0: copy host mem,1: nocopy.

-sophon_idx
设备索引

e.g

test_ff_resize_transcode example.mp4 test.ts I420 h264_bm 800 400 25 3000 3 0 0

test_gst_transcode

此用例主要用于测试 gstreamer 下视频转码的功能。此功能通过先解码再编码的过程实现, 支持 H.264/H.265/JPEG 等格式输入。

test_gst_transcode [video_path] [output_path] [dec_type] [enc_type] [bps] [gop] [gop_preset] [cqp] [qp_min] [qp_max] [q_factor]

参数:

-video_path
输入文件

-output_path
输出文件

-dec_type
输入编解码格式, 1 表示 h264, 2 表示 h265, 3 表示 JPEG, 4 表示 decode bin

-enc_type
输出编解码格式, 1 表示 h264, 2 表示 h265, 3 表示 JPEG

-bps
编码中的目标比特率

-gop
h26x 编码两个 I 帧之间的间隔

-gop_preset
h26x 编码 GOP 结构预设选项 [1-7]

-cqp
h26x 编码器中使用恒定质量模式, 需满足 bsp >= 0

-qp_min
h26x 编码过程中允许使用的最小量化参数值

-qp_max
h26x 编码过程中允许使用的最大量化参数值

-q_factor
JPEG 压缩中的质量参数

e.g

test_gst_transcode -v 1920x1080.mp4 -d 4 -e 1 -g 50 -b 2000000 -o tc_case24.h264

test_gst_vcmulti

此用例主要用来测试帧率，内部使用 videotestsrc 生成 1080p 的 yuv 测试数据，输出每个通道的帧率，不会输出码流或 yuv。

test_gst_vcmulti [video_path] [codec_type] [is_enc] [num_chl] [disp] [trans_type]

参数:

-video_path
视频文件路径

-codec_type
编解码器类型, 1 表示 h264, 2 表示 h265, 3 表示 JPEG

-is_enc
0 表示解码, 1 表示编码, 2 表示转码

-num_chl
编解码通道数

-disp
1 表示使能帧率实时显示

-trans_type
当 is_enc 设置为 2 时, 进行转码测试，编码类型: 1 表示 h264, 2 表示 h265, 3 表示 JPEG

e.g

test_gst_vcmulti -c 1 -e 1 -n 4 -d 1

gst-launch-1.0

此用例主要用于测试 gstreamer 下 jpeg 编码的功能和性能。

e.g

功能测试:

```
gst-launch-1.0 filesrc location=640x480_420p.yuv \
! rawvideoparse format=i420 width=640 height=480 ! bmjpegenc \
! filesink location=case0.jpg
```

性能测试:

```
gst-launch-1.0 filesrc location=640x480_420p.yuv \
! rawvideoparse format=i420 width=640 height=480 ! bmjpegenc \
! fakesink
```

管道详细步骤:

filesrc
从文件中读取数据

rawvideoparse
解析原始视频数据, 指定格式为 I420(YUV420p), 宽度为 640, 高度为 480

bmjpegenc
使用 BMJPEG 编码器将视频数据编码为 JPEG 格式

filesink
将编码后的 JPEG 数据写入文件中

fakesink
直接将编码后的数据丢弃, 不写入文件

gst-launch-1.0

此用例主要用于测试 gstreamer 下 jpeg 解码的功能和性能, 调用了 bmdec 插件实现图像解码的硬件加速。

e.g

功能测试:

```
gst-launch-1.0 filesrc location=JPEG_1920x1088_yuv420_planar.jpg ! jpegparse \
! bmdec ! filesink location=case12.yuv
```

性能测试:

```
gst-launch-1.0 filesrc location=JPEG_1920x1088_yuv420_planar.jpg ! jpegparse \
! bmdec ! fakesink
```

管道详细步骤:

filesrc
从文件中读取数据

jpegparse
解析 JPEG 数据

bmdec
使用 BM 解码器解码 JPEG 数据

filesink
将解码后的 YUV 数据写入文件中

fakesink
直接将解码后的数据丢弃, 不写入文件

gst-launch-1.0

此用例主要用于测试 gstreamer 下 H.265 视频编码的功能和性能。

e.g

功能测试:

```
gst-launch-1.0 filesrc location=1080p_nv12.yuv \
! rawvideoparse format=nv12 width=1920 height=1080 \
! bmh265enc gop=50 ! filesink location=case0.h265
```

性能测试:

```
gst-launch-1.0 filesrc location=1080p_nv12.yuv \
! rawvideoparse format=nv12 width=1920 height=1080 \
! bmh265enc gop=50 ! fakesink
```

管道详细步骤:

filesrc
从文件中读取数据

rawvideoparse
解析原始视频数据, 指定格式为 NV12, 宽度为 1920, 高度为 1080

bmh265enc
使用 BM H.265 编码器对视频进行 H.265 编码, 设置 GOP 为 50

filesink
将编码后的 H.265 视频数据写入文件中

fakesink
直接将编码后的数据丢弃, 不写入文件

gst-launch-1.0

此用例主要用于测试 gstreamer 下 H.264 视频解码的功能和性能。

e.g

功能测试:

```
gst-launch-1.0 filesrc location=1920x1080.mp4 ! qtdemux ! h264parse ! bmdec \
! video/x-raw,format=NV12 ! filesink location=case1.nv12
```

性能测试:

```
gst-launch-1.0 filesrc location=1920x1080.mp4 ! qtdemux ! h264parse ! bmdec \
! video/x-raw,format=NV12 ! fakesink
```

管道详细步骤：

filesrc
从文件中读取数据

qtdemux
解封装 QuickTime 格式文件的元素，用于提取其中的音视频流

h264parse
解析 H.264 数据

bmdec
使用 BM 解码器解码视频数据

video/x-raw,format=NV12
将解码后的视频数据转换为 NV12 格式

filesink
将转换后的 NV12 视频数据写入文件中

fakesink
直接将解码后的数据丢弃, 不写入文件

gst-launch-1.0

此用例主要用于测试 gstreamer 下视频转码的功能和性能，此功能通过先解码再编码的过程实现。

e.g

功能测试：

```
gst-launch-1.0 -e filesrc location=1920x1080.mp4 ! qtdemux ! h264parse ! bmdec \
! bmh265enc gop=50 bps=1000000 qp-min=24 qp-max=45 gop-preset=5 \
! filesink location=case4.h265
```

性能测试：

```
gst-launch-1.0 -e filesrc location=1920x1080.mp4 ! qtdemux ! h264parse ! bmdec \
! bmh265enc gop=50 bps=1000000 qp-min=24 qp-max=45 gop-preset=5 \
! fakesink
```

管道详细步骤：

filesrc
从文件中读取数据

qtdemux
解封装 QuickTime 格式文件的元素，用于提取其中的音视频流

h264parse
解析 H.264 数据

bmdec
使用 BM 解码器解码视频数据

bmh265enc
使用 BM H.265 编码器对视频进行 H.265 编码, 设置 GOP 为 50, 比特率为 1000000, 最小量化参数为 24, 最大量化参数为 45, GOP 预设值为 5

filesink
将编码后的 H.265 视频数据写入文件中

fakesink
直接将编码后的数据丢弃, 不写入文件

gst-launch-1.0

此用例主要用于测试 gstreamer 下 VPSS 色彩转换的功能和性能。

e.g

功能测试:

```
gst-launch-1.0 filesrc location=1920x1080_yuv420p.bin blocksize=3110400 num-buffers=1 \
! video/x-raw, format=I420, width=1920, height=1080, framerate=1/1 ! bmvpss \
! video/x-raw, format=RGB, width=1920, height=1080, framerate=1/1 \
! filesink location=vpss_case01.bin
```

性能测试:

```
gst-launch-1.0 filesrc location=1920x1080_yuv420p.bin blocksize=3110400 num-buffers=1 \
! video/x-raw, format=I420, width=1920, height=1080, framerate=1/1 ! bmvpss \
! video/x-raw, format=RGB, width=1920, height=1080, framerate=1/1 \
! fakesink
```

管道详细步骤:

filesrc
从文件中读取数据, 设置了 blocksize(读取数据块的大小) 和 num-buffers(读取数据包数量)

video/x-raw, format=I420, width=1920, height=1080, framerate=1/1
输入数据为 I420 格式, 指定了宽度、高度和帧率

bmvpss
使用 BMVPSS 插件处理视频流

video/x-raw, format=RGB, width=1920, height=1080, framerate=1/1
处理后的数据为 RGB 格式, 指定了宽度、高度和帧率

filesink
将处理后的 RGB 格式数据写入文件中

fakesink
直接将处理后的数据丢弃, 不写入文件

gst-launch-1.0

此用例主要用于测试 gstreamer 下 VPSS 缩放的功能和性能。

e.g

功能测试:

```
gst-launch-1.0 filesrc location=2048x2048_rgb.bin blocksize=12582912 num-buffers=1 \
! video/x-raw, format=RGB, width=2048, height=2048,framerate=1/1 ! bmvpss \
! video/x-raw, format=RGB, width=16, height=16, framerate=1/1 \
! filesink location=vpss_case15.bin
```

性能测试:

```
gst-launch-1.0 filesrc location=2048x2048_rgb.bin blocksize=12582912 num-buffers=1 \
! video/x-raw, format=RGB, width=2048, height=2048,framerate=1/1 ! bmvpss \
! video/x-raw, format=RGB, width=16, height=16, framerate=1/1 \
! fakesink
```

管道详细步骤:

filesrc
从文件中读取数据, 设置了 blocksize(读取数据块的大小) 和 num-buffers(读取数据包数量)

video/x-raw, format=RGB, width=2048, height=2048, framerate=1/1
数据数据为 RGB 格式, 指定了宽度、高度和帧率

bmvpss
使用 BMVPSS 插件处理视频流

video/x-raw, format=RGB, width=16, height=16, framerate=1/1
处理后的数据为 RGB 格式, 指定了宽度、高度和帧率

filesink
将处理后的 RGB 格式数据写入文件中

fakesink
直接将处理后的数据丢弃, 不写入文件

CHAPTER 5

使用 sophon-media 开发

在安装完 sophon-media 后，用户可以用两种方式将 sophon-media 的库链接到自己的编译程序中。

** 如果使用 Make 编译系统 **

推荐用户使用 pkgconfig 来寻找 sophon-media 库。

在之前的安装中，我们已经将 sophon-media 的 pkgconfig 路径加入到环境变量 PKG_CONFIG_PATH。因此, 用户可以在 Makefile 中添加如下语句:

```
CFLAGS = -std=c++11

# add libsophon dependency because sophon-ffmpeg rely on it
CFLAGS += -I/opt/sophon/libsophon-current/include/
LDFLAGS += -L/opt/sophon/libsophon-current/lib -lbmcv -lbmlib -lbmvideo -lbmvpuapi -
↪lbmvpulite -lbmjpuapi -lbmjpulite -lbmion

# add sophon-ffmpeg
CFLAGS += $(shell pkg-config --cflags libavcodec libavformat libavfilter libavutil libswscale)
LDFLAGS += $(shell pkg-config --libs libavcodec libavformat libavfilter libavutil libswscale)

# add sophon-opencv
CFLAGS += $(shell pkg-config --cflags opencv4)
LDFLAGS += $(shell pkg-config --libs opencv4)
```

然后就可以在 Makefile 中使用 sophon-ffmpeg 和 sophon-opencv 的库。

注意：当系统在/usr/lib 或者/usr/local/lib 下还安装了另一份 ffmpeg 或者 opencv 的时候，要注意检查下，是否搜索到了正确的 sophon-ffmpeg/sophon-opencv 路径。如果搜索不正确，则需要显式地指定头文件位置和库文件位置

** 如果使用 CMake 编译系统 **

用户可以在 CMakeLists.txt 中添加如下语句:

```
# add libsophon
find_package(libsophon REQUIRED)
include_directories(${LIBSOPHON_INCLUDE_DIRS})
link_directories(${LIBSOPHON_LIB_DIRS})

# add sophon-ffmpeg
set(FFMPEG_DIR /opt/sophon/sophon-ffmpeg-latest/lib/cmake)
find_package(FFMPEG REQUIRED NO_DEFAULT_PATH)
include_directories(${FFMPEG_INCLUDE_DIRS})
link_directories(${FFMPEG_LIB_DIRS})

# add sophon-opencv
set(OpenCV_DIR /opt/sophon/sophon-opencv-latest/lib/cmake/opencv4)
find_package(OpenCV REQUIRED NO_DEFAULT_PATH)
include_directories(${OpenCV_INCLUDE_DIRS})

add_executable(${YOUR_TARGET_NAME} ${YOUR_SOURCE_FILES})

target_link_libraries(${YOUR_TARGET_NAME} ${FFMPEG_LIBS} ${OpenCV_LIBS})
```

在用户的代码中即可以调用 sophon-ffmpeg 和 sophon-opencv 中的函数:

```
#include <opencv2/opencv.hpp>

int main(int argc, char const *argv[])
{
    cv::Mat img = cv::imread(argv[1]);
    return 0;
}
```