Multimedia Technical Reference Manual

SOPHGO

Jan 05, 2025

menu

Release Record

Version	Date of Release	Descriptions
V2.0.2	2019.11.15	The first edition adds OpenCV private API, FFMPEG API, and enhances the introduction of libyuv API interface
V2.1.0	2020.07.06	The second edition optimizes the chapter structure of the FFMPEG part and adds the content introduction of the video coding part
V2.2.0	2020.08.26	The third edition adjusts the font, optimizes the layout, and increases the content of the bmx264 video encoder
V2.2.1	2021.02.09	Update the description of the new interface of Opencv: new interface in bmcv::/av::/Mat:

CHAPTER 1

Multimedia User Guide

1.1 SOPHGO Multimedia Framework Introduction

1.1.1 Introduction

The multimedia framework described in this document is described for the Sophon BM1688 product series of Sophgo. Among them,1) all the content about video hardware encoding in this document is only for BM1688; 2) The functions under the bmcv namespace in Opencv mentioned in this document are only for BM1688.

The coverage of the multimedia framework described in this document includes the video decoding VPU module, video encoding VPU module, image encoding JPU module, image decoding JPU module, and image processing module VPP in BM1688 product family. The functions of these modules are encapsulated in the FFMPEG and OPENCV open-source frameworks. Users can choose FFMPEG API or OPENCV API according to their own development preference. For the image processing module, we also provide the underlying interface of Sophgo' s own BMCV API. This part of the interface is described in a special document. You can refer to the "BMCV Technical Reference Manual" , which will not be described in detail in this document. Only the hierarchical relationship between these three sets of APIs and how to convert each other are introduced.

The three APIs of OPENCV, FFMPEG and BMCV are functionally subsets, but some of them cannot be included, and are specifically marked in the brackets below.

1) BMCV API contains all the image processing acceleration interfaces that can be accelerated by hardware (here image processing hardware acceleration, including hardware image processing VPP module acceleration, and image processing functions implemented by using other hardware modules)

2) FFMPEG API includes all hardware-accelerated video/image codec interfaces, all software-supported video/image codec interfaces (that is, all FFMPEG open-source supported formats), and some hardware-accelerated image processing interfaces supported by the bm_scale filter (these image processing interface, only includes scaling, crop, padding and color conversion functions accelerated by hardware image processing VPP module)

3) OPENCV API includes all hardware and software video codec interfaces supported by FFMPEG (the bottom layer of the video module is supported by FFMPEG, and this part of the function is completely covered), hardware-accelerated JPEG codec interfaces, and all other image codec interfaces supported by software (that is, all image formats supported by open-source opencv), some hardware-accelerated image processing interfaces (referring to the scaling, crop, padding, and color conversion functions accelerated by the image processing VPP module), and all software-supported OPENCV image processing functions.

Among these three frameworks, BMCV focuses on image processing functions and can be accelerated by BM1688 hardware; FFMPEG framework is strong in the encoding and decoding of images and videos and supports almost all formats, the difference is whether it can be accelerated by hardware; OPENCV framework is strong in image processing. Various image processing algorithms are firstly integrated into the OPENCV framework, and the video codec module is implemented by calling FFMPEG at the bottom layer.

Because BMCV only provides image processing interfaces, users generally choose one of the FFMPEG or OPENCV frameworks as the main framework for development. These two frameworks, in terms of functional abstraction, the interface of OPENCV is more concise, and one interface can realize a video encoding and decoding operation; in terms of performance, the performance of these two frameworks is exactly the same, and there is almost no difference. In terms of codec, OPENCV is just a layer of encapsulation for the FFMPEG interface; In terms of flexibility, FFMPEG has more separated interfaces with finer granularity of operations that can be inserted. Most importantly, users still have to make choices based on their familiarity with a certain framework. Only with in-depth understanding can they make good use of the framework.

The hierarchical relationship of these three frameworks is shown in the figure

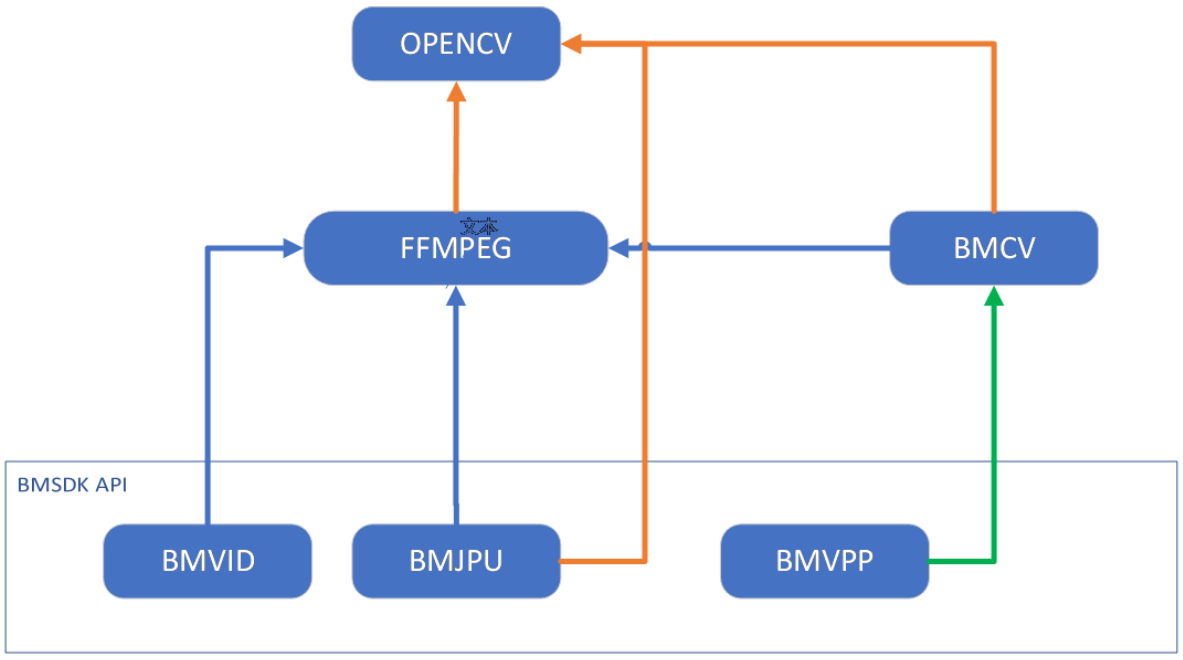


Figure 1 Hierarchical Calling Relationship between OPENCV/FFMPEG/BMCV and BMSDK

In many application scenarios, special functions under a certain framework need to be used, so a flexible conversion scheme between the three frameworks is given is Section 4. This conversion does not require a large number of data copies so there is little performance penalty.

1.1.2 BM1688 Hardware Acceleration Function

This section presents the functions supported by the hardware acceleration module in the multimedia framework. The hardware acceleration module includes video decoding VPU module, video encoding VPU module, image encoding and decoding JPU module, and image processing VPP module.

It is important to note that only the capabilities that can be accelerated with hardware are listed here, along with performance estimates for typical scenarios. For more detailed performance indicators, please refer to the BM1688 product specification.

Video Codec

BM1688 supports hardware decoding acceleration of H264 (AVC), HEVC video format, and supports real-time decoding up to 4K video. Support H264 (AVC), HEVC video format hardware encoding, up to real-time encoding of HD (1080p) video.

The speed of video decoding is highly related to the format of the input video stream, and the decoding speed of streams with different complexity has relatively large fluctuations, such as bit rate, GOP structure, resolution, etc., which will affect the specific test results.

Generally speaking, for video surveillance application scenarios, a single chip of BM1688 can support up to 32 channels of real-time HD decoding.

The speed of video encoding is highly related to the configuration parameters of encoding. Under different encoding configurations, even with the same video content, the encoding speed is not exactly the same. Generally speaking, a single chip of BM1688 can support up to 2 channels of HD real-time encoding.

Image Codec

BM1688 support hardware encoding/decoding acceleration of JPEG baseline format. Note that only the hardware codec acceleration of the JPEG baseline grade is supported. For other image formats, including JPEG2000, BMP, PNG, and JPEG standard grades such as progressive, lossless, etc., the soft decoding is automatically supported. In the OPENCV framework, this compatibility support is transparent to the users and requires no special handling by users during application development.

The processing speed of image hardware codec has a great relationship with the resolution of the image and the image color space (YUV420/422/444), Generally speaking, for a picture with a resolution of 1920x1080 and YUV420 color space, the single-chip image hardware codec can reach about 600fps.

Image Processing

BM1688 has a dedicated video processing VPP unit to perform hardware acceleration processing on images. Supported image operations include color conversion, image scaling, image crop, and image stitch functions. Maximum support up to 4k image input. For some common complex image processing functions not supported by VPP, such as linear transformation ax+b, histogram, etc., we use other hardware units to do special acceleration processing in the BMCV API interface.

1.1.3 Hardware Memory Classification

In the subsequent discussion, the memory synchronization problem is a relatively hidden problem that is often encountered in application debugging. We usually refer to the synchronization between these two types of memory uniformly as device memory and host memory.

SOC mode means that the processor in the BM1688 chip is used as the main control CPU, and the BM1688 product runs the application program independently. Typical products are SE5, SM5-soc modules. In this mode, the ION memory under Linux system is used to manage the device memory. In the SOC mode, the device memory refers to the physical memory allocated by ION, and the system memory is actually the cache. From system memory (cache) to device memory, it is called Upload (essentially cache flush); from device memory to system memory (cache), it is called Download (essentially cache invalidation). In SOC mode, the device memory and the system memory are actually the same physical memory. Most of the time, the operating system will automatically synchronize them, which also makes the phenomenon that the memory is not synchronized in time is more subtle and difficult to reproduce.

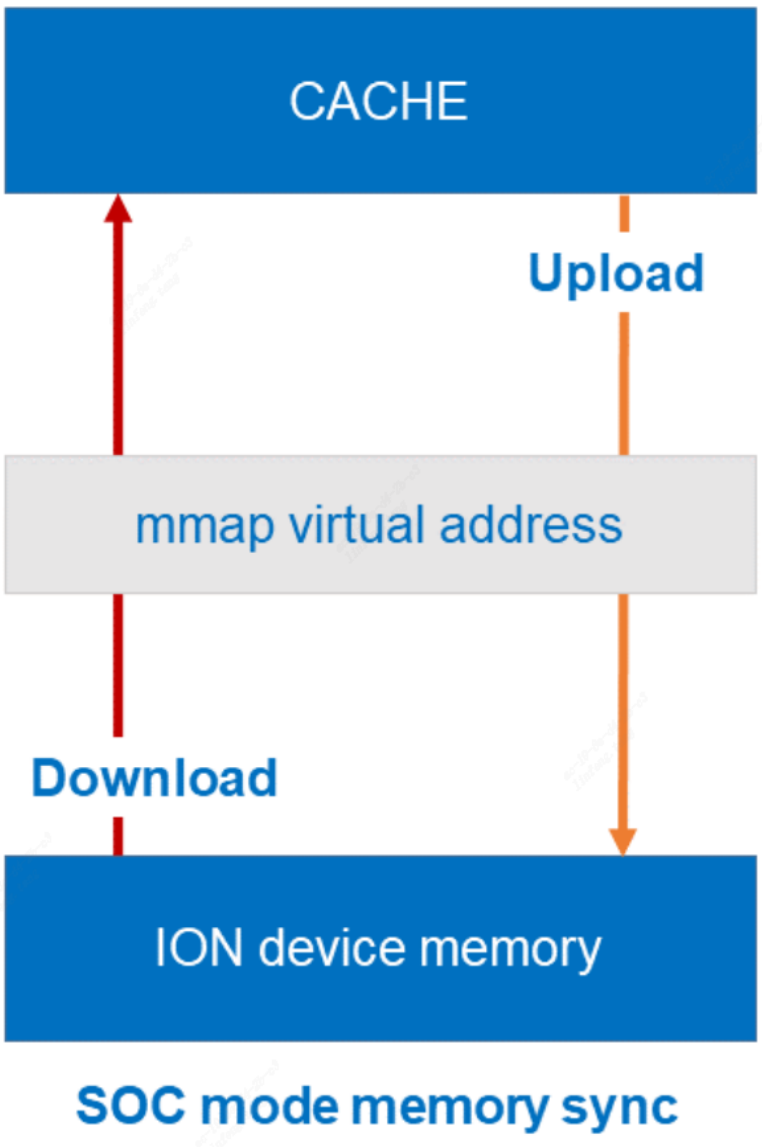


Figure 2 Memory Synchronization Model

Both FFMPEG and OPENCV frameworks provide functions for memory synchronization operations. The BMCV API is only for device memory operations, so there is no memory synchronization problem. When calling the BMCV API, the data needs to be prepared in the device memory.

In the OPENCV framework, the update flag is provided in the formal parameters of some functions. When the flag is set to true, the function will automatically perform memory synchronization operations. This part can refer to the subsequent API introduction in Chapter 2, Section 3.Users can also actively control memory synchronization through the two functions bmcv::downloadMat() and bmcv::uploadMat(). The basic principles of synchronization are: a) In the OPENCV native function, the data in the device memory in the yuv Mat format is always the latest, and the data in the system memory in the RGB Mat format is always the latest b) When the OPENCV function switches to the BMCV API , according to the previous principle, synchronize the latest data to the device memory; on the contrary, when switching from the BMCV API to the OPENCV function, synchronize the latest data to the system memory under the RGB Mat. c) When frame switching does not occur, minimize memory synchronization operations. Frequent memory synchronization operations can significantly degrade performance.

In the regular FFMPEG framework, there are two classes of codec APIs and filter APIs called soft (regular) and hard (hwaccel). The framework of these two APIs can support the hardware video codec and hardware image filter of BM1688. From this perspective, the underlying performance of soft decoding and hard decoding is exactly the same, but the difference in usage preferences. The usage of the soft codec/filter API is exactly the same as the usual ffmpeg built-in codec. The hard codec/filter API uses -hwaccel to specify and enable the dedicated hardware device for bmcodec. When in the soft codec API and filter API, the flag parameter “is_dma_buffer” or “zero_copy” is passed in through av_dict_set to control whether the internal codec or filter synchronizes the device memory data to the system memory. The specific parameters can be viewed with ffmpeg -h . When the subsequent direct connection to the hardware processing, it usually does not need to synchronize the device memory data to the system memory.

In the hwaccel codec API and filter API, the default memory is only device memory, and no system memory is allocated. If memory synchronization is required, it is done through the hwupload and hwdownload filters.

To sum up, both OPENCV and FFMPEG frameworks provide support for memory synchronization, and applications can choose the corresponding framework according to their own usage preferences to precisely control data synchronization. The BMCV API always works on device memory.

1.1.4 Frame Conversion

In application development, there are always situations where a certain framework cannot fully meet the design requirements. At this time, it is necessary to quickly switch between various frameworks. The BM1688 multimedia framework provides support to meet this demand, and this switching does not perform data copying, which has almost no impact on performance.

Conversion between FFMPEG and OPENCV

The conversion between FFMPEG and OPENCV is mainly the format conversion between the data format AVFrame and cv::Mat.

When the cooperation between FFMPEG and OPENCV is required, it is recommended to use the general non-HWAccel API path. At present, OPENCV internally adopts this method, and the verification is relatively complete.

FFMPEG AVFrame to OPENCV Mat format is as follows.

1. AVFrame * picture;

2. After a series of processing by FFMPEG API, such as avcodec_decode_video2 or avcodec_receive_frame, and then convert the result to Mat

3. card_id is the order number of the device for FFMPEG hardware-accelerated decoding. In the regular codec API, it is specified by sophon_idx of av_dict_set, or in the hwaccel API, it is specified when the hwaccel device is initialized, and the default is 0 in soc mode.

4. cv::Mat ocv_frame(picture, card_id);

5. Or format conversion can be done in a step-by-step manner

6. cv::Mat ocv_frame;

7. ocv_frame.create(picture, card_id);

8. Then you can use ocv_frame for opencv operations. At this time, the ocv_frame format is the yuv mat type extended by BM1688. If you want to convert to the OPENCV standard bgr mat format later, you can do the following.

9. Note: There is a memory synchronization operation here. If not set, FFMPEG is in the device memory by default. If update=false, then the data converted to bgr is always in the device memory, and the system memory is invalid data, If update=true, the device memory is synchronized to the system memory. If the follow-up is still hardware accelerated processing, you can set update=false, which can improve the efficiency. When you need to use the system memory data, you can explicitly call bmcv::downloadMat() to synchronize.

10. cv::Mat bgr_mat;

11. cv::bmcv::toMAT(ocv_frame, bgr_mat, update);

12. Finally, AVFrame *picture will be released by Mat ocv_frame, so there is no need to perform av_frame_free() operation on picture. If you want to call av_frame_free to release the picture externally, you can add card_id = card_id | BM_MAKEFLAG(UMatData::AVFRAME_ATTACHED,0,0), this standard indicates that the creation and release of AVFrame are managed externally

13. ocv_frame.release();

14. picture = nullptr;

It is rare for OPENCV Mat to be converted into FFMPEG AVFrame, because almost all required FFMPEG operations have corresponding encapsulation interfaces in opencv. For example, FFMPEG decoding has a videoCapture class in OPENCV, FFMPEG encoding has a videoWriter class in OPENCV, and FFMPEG' s filter operation corresponding to image processing has an interface under the bmcv namespace and rich native image processing functions in OPENCV.

Generally speaking, converting OPENCV Mat to FFMPEG AVFrame refers to yuv Mat. In this case, the conversion can be done as follows.

1. Create a yuv Mat, if the yuv Mat already exists, you can ignore this step. card_id is the order number of the BM1688 device, and it defaults to 0 in soc mode

2. AVFrame * f = cv::av::create(height, width, AV_PIX_FMT_YUV420P, NULL, 0, -1, NULL, NULL, AVCOL_SPC_BT709, AVCOL_RANGE_MPEG, card_id);

3. cv::Mat image(f, card_id);

4. do something in opencv

5. AVFrame * frame = image.u->frame;

6. call FFMPEG API

7. Note: Before the FFMPEG call is completed, it must be ensured that the Mat image is not released, otherwise the AVFrame will be released together with the Mat image. If you need to separate the two declaration cycles, the image declaration above should be changed to the following format.

8. cv::Mat image(f, card_id | BM_MAKEFLAG(UMatData::AVFRAME_ATTACHED, 0, 0));

9. This way Mat won' t take over the memory release of the AVFrame

Conversion between FFMPEG and BMCV API

FFMPEG often needs to be used in conjunction with the BMCV API, so the conversion between FFMPEG and BMCV is relatively frequent. For this purpose, we have specially given an example ff_bmcv_transcode, which can be found in the bmnnsdk2 release package.

The ff_bmcv_transcode example demonstrates the process of decoding with FFMPEG, converting the decoding result to BMCV for processing, and then converting back to FFMPEG for encoding. The mutual conversion between FFMPEG and BMCV can refer to the avframe_to_bm_image() and bm_image_to_avframe() functions in the ff_avframe_convert.cpp file.

Conversion between OPENCV and BMCV API

For conversion between OPENCV and BMCV API, special conversion functions are provided under the bmcv namespace extended by OPENCV.

Convert OPENCV Mat to BMCV bm_image format:

1. cv::Mat m(height, width, CV_8UC3, card_id);

2. opencv operation

3. bm_image bmcv_image;

4. Here update is used to control memory synchronization. Whether memory synchronization is required depends on the previous OPENCV operation. If the previous operations are completed with hardware acceleration and the latest data is in the device memory, there is no need to perform memory synchronization. If the previous operation is called The OPENCV function does not use hardware acceleration (the subsequent OPENCV chapter 6.2 mentions which functions use hardware acceleration), and memory synchronization is required for the bgr mat format.

5. You can also explicitly call cv::bmcv::uploadMat(m) to achieve memory synchronization before calling the following function

6. cv::bmcv::toBMI(m, &bmcv_image, update);

7. Use bmcv_image to make bmcv api calls. During the call, pay attention to ensure that Mat m cannot be released, because bmcv_image uses the memory space allocated in Mat m. handle can be obtained by bm_image_get_handle()

8. Release: This function must be called because bm_image is created in toBMI, otherwise there will be a memory leak

9. bm_image_destroy(bmcv_image);

10. m.release();

There are two ways to convert from BMCV bm_image format to OPENCV Mat. One is to copy data, so that bm_image and Mat are independent of each other and can be released separately, but there is a performance loss; one is to directly refer to bm_image memory without any performance loss.

1. bm_image bmcv_image;

2. Call bmcv API to allocate memory space to bmcv_image and operate

3. Mat m_copy, m_nocopy;

4. The following interface will copy memory data and convert it into standard bgr mat format.

5. Update controls memory synchronization. You can also use bmcv::downloadMat() to control memory synchronization after calling this function.

6. csc_type is the control color conversion coefficient matrix, which controls the conversion of different yuv color spaces to bgr

7. cv::bmcv::toMAT(&bmcv_image, m_copy, update, csc_type);

8. The following interface will directly refer to bm_image memory (nocopy flag is true), and update is still according to the previous description, choose whether to synchronize memory or not.

9. In subsequent opencv operations, it must be ensured that bmcv_image is not released, because the memory of mat is directly referenced from bm_image

10. cv::bmcv::toMAT(&bmcv_image, &m_nocopy, AVCOL_SPC_BT709, AVCOL_RANGE_MPEG, NULL, -1, update, **true**);

11. For opencv

1.2 SOPHGO OpenCV User Guide

1.2.1 OpenCV Introduction

The multimedia, BMCV and NPU hardware modules in the BM1688 series chips can accelerate the processing of pictures and videos:

1) Multimedia module: hardware-accelerated JPEG codec and Video codec operations.
2) BMCV module: hardware-accelerated image resize, color conversion, crop, split, linear transform, nms, sort and other operations.
3) NPU module: Hardware accelerated split, rgb2gray, mean, scale, int8tofloat32 operations on images.

In order to facilitate customers to use the hardware modules on the chip to accelerate the processing of pictures and videos, and improve the performance of the application OpenCV software, the OpenCV library has been modified by SOPHGO, and the hardware modules are called internally to perform Image and Video-related processing.

The current OpenCV version of SOPHGO is 4.1.0. Except for the following SOPHGO’s own APIs, all other APIs are consistent with the OpenCV API.

In SOC mode, due to hardware limitations, in the Mat object of the OpenCV library, the step value will be automatically set to 64bytes alignment, and the data less than 64bytes will be padded with random numbers.

For example, in a 100*100 picture, the RGB of each pixel is represented by 3 U8 values, the normal step value is 300, but after 64bytes alignment, the step value is finally 320. As shown in the figure below, in the data of the Mat object, the data of each step is a continuous 320 bytes, of which the first 300 are real data, and the last 20 are automatically filled random numbers.

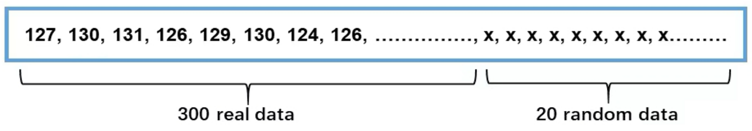

127, 130, 131, 126, 129, 130, 124, 126, ..., x, x, x, x, x, x, x, x, x.........
300 real data
20 random data

Figure 3 Alignment Introduction

In SOC mode, due to the extra random numbers filled, the data variable of the data stored in the Mat object cannot be directly passed to the API of the BMRuntime library for inference, otherwise the accuracy of the model will be reduced. Please set stride to non-aligned mode when the last BMCV does the transformation, and the excess random numbers will be automatically cleared.

1.2.2 Data Structure Extension Description

The color space of OpenCV’ s built-in standard processing is BGR format, but in many cases, for video and image sources, processing directly in the YUV color space can save bandwidth and avoid unnecessary mutual conversion between YUV and RGB. So SOPHGO Opencv extends the Mat class.

1) In Mat.UMatData, the AVFrame member is introduced to extend support for various YUV formats. Where the format definition of AVFrame is compatible with the definition in FFMPEG
2) In Mat.UMatData, the definitions of fd, addr (in soc mode) are added, which represent the corresponding memory management handle and physical memory address respectively
3) Variable fromhardware variable is added to the Mat class to identify whether the current video and picture decoding is done by hardware or software.

1.2.3 API Extension Description

bool VideoCapture::get_resampler(int *den, int *num)

Function Prototype	bool VideoCapture::get_resampler(int *den, int *num)
Function	Get the sample rate of the video. For example, den=5, num=3 means 2 frames are discarded every 5 frames.
Input Params	int *den – denominator of sample rate
	int *num – numerator of sample rate
Output Params	None
Return Value	true – successful implementation false - failed implementation
Description	This interface will be deprecated. It is recommended to use double VideoCapture::get(CAP_PROP_OUTPUT_SRC) interface.

bool VideoCapture::set_resampler(int den, int num)

Function Prototype	bool VideoCapture::set_resampler(int den, int num)
Function	Set the sample rate of the video. For example, den=5, num=3 means 2 frames are discarded every 5 frames.
Input Params	int den – denominator of sample rate
	int num – numerator of sample rate
Output Params	None
Return Value	true – successful implementation false - failed implementation
Description	This interface will be deprecated. It is recommended to use bool VideoCapture::set(CAP_PROP_OUTPUT_SRC, double resampler) interface.

double VideoCapture::get(CAP_PROP_TIMESTAMP)

Function Prototype	double VideoCapture::get(CAP_PROP_TIMESTAMP)
Function	Provides the timestamp of the current picture, the time base depends on the time base given in the stream.
Input Params	CAP_PROP_TIMESTAMP – A specific enumeration type indicates getting timestamps, this type is defined by Sophgo
Output Params	None
Return Value	Convert the return value to unsigned int64 data type before use
	0x8000000000000000L-No AV PTS value
	other-AV PTS value

double VideoCapture::get(CAP_PROP_STATUS)

Function Prototype	double VideoCapture::get(CAP_PROP_STATUS)
Function	This function provides an interface for checking the internal running status of video capture.
Input Params	CAP_PROP_STATUS – A specific enumeration type defined by Sophgo
Output Params	None
Return Value	Convert the return value to int data type before use
	0 Video capture is stopped, paused or otherwise inoperable
	1 Video capture is in progress
	2 Video capture is end

bool VideoCapture::set(CAP_PROP_OUTPUT_SRC, double resampler)

Function Prototype	double VideoCapture::get(CAP_PROP_OUTPUT_SRC, double resampler)
Function	Set the sample rate of YUV video. If the resampler is 0.4, means 2 frames are reserved in every 5 frames, and 3 frames are discarded.
Input Params	CAP_PROP_OUTPUT_SRC – A specific enumeration type indicates getting timestamps, this type is defined by Sophgo
	double resampler - sample rate
Output Params	None
Return Value	true - successful implementation
	false - failed implementation

double VideoCapture::get(CAP_PROP_OUTPUT_SRC)

Function Prototype	double VideoCapture::get(CAP_PROP_OUTPUT_SRC)
Function	Get the sample rate of the video.
Input Params	CAP_PROP_OUTPUT_SRC - A specific enumeration type indicates video output, this type is defined by Sophgo
Output Params	None
Return Value	Sample rate value

bool VideoCapture::set(CAP_PROP_OUTPUT_YUV, double enable)

Function Prototype	bool VideoCapture::set(CAP_PROP_OUTPUT_YUV, double enable)
Function	Turns frame output in YUV format on or off. The YUV format in the BM1688 series is I420
Input Params	CAP_PROP_OUTPUT_YUV - A specific enumeration type, referring to the video frame output in YUV format, this type is defined by SOPHGO; double enable - OP code, 1 means open, 0 means close
Output Params	None
Return Value	true – successful implementation false - failed implementation

double VideoCapture::get(CAP_PROP_OUTPUT_YUV)

Function Prototype	double VideoCapture::get(CAP_PROP_OUTPUT_YUV)
Function	Get the state of the YUV video frame output.
Input Params	CAP_PROP_OUTPUT_YUV - A specific enumeration type, referring to the video frame output in YUV format, this type is defined by SOPHGO.
Output Params	None
Return Value	Status of YUV video frame output. 1 means open, 0 means close.

bm_status_t bmcv::toBMI(Mat &m, bm_image *image, bool update = true)

Function Prototype	bm_status_t bmcv::toBMI(Mat &m, bm_image *image, bool date = true)
Function	The OpenCV Mat object is converted into the bm_image data object of the corresponding format in the BMCV interface. This interface directly references the data pointer of the Mat, and no copy operation occurs. This interface only supports 1N mode
Input Params	Mat& m – Mat object, which can be in extended YUV format or standard OpenCV BGR format; bool update – Whether to synchronize cache or memory. If true, the cache will be synchronized after the conversion is completed
Output Params	bm_image *image – BMCV bm_image data object of the corresponding format
Return Value	BM_SUCCESS(0): successful implementation Other: failed implementation
Description	Currently supports format conversion of compressed formats, Gray, NV12, NV16, YUV444P, YUV422P, YUV420P, BGR separate, BGR packed, CV_8UC1

bm_status_t bmcv::toBMI(Mat &m, Mat &m1, Mat &m2, Mat &m3, bm_image *image, bool update = true)

Function Prototype	bm_status_t bmcv::toBMI(Mat &m, Mat &m1, Mat &m2, Mat &m3, bm_image *image, bool update = true)
Function	The OpenCV Mat object is converted into the bm_image data object of the corresponding format in the BMCV interface. This interface directly references the data pointer of the Mat, and no copy operation occurs. This interface is for the 4N mode of BMCV. The input image format of all Mats is required to be the same, only valid for BM1688
Input Params	Mat &m - The first image in 4N, extended YUV format or standard OpenCV BGR format. Mat &m1 - The second image in 4N, extended YUV format or standard OpenCV BGR format. Mat &m2 - The third image in 4N, extended YUV format or standard OpenCV BGR format. Mat &m3 - The fourth image in 4N, extended YUV format or standard OpenCV BGR format. bool update - Whether to synchronize cache or memory. If true, the cache will be synchronized after the conversion is completed
Output Params	bm_image *image - The BMCV bm_image data object of the corresponding format, which contains 4 image data
Return Value	BM_SUCCESS(0): successful implementation Other: failed implementation
Description	Currently supports format conversion of compressed formats, Gray, NV12, NV16, YUV444P, YUV422P, YUV420P, BGR separate, BGR packed, CV_8UC1

bm_status_t bmcv::toMAT(Mat &in, Mat &m0, bool update=true)

Function Prototype	bm_status_t bmcv::toMAT(Mat &in, Mat &m0, bool update = true)
Function	The input MAT object, which can be in various YUV or BGR formats, is converted into a MAT object output in BGR packet format
Input Params	Mat &in - The input MAT object can be in various YUV formats or BGR formats; bool update - Whether to synchronize cache or memory. If true, the cache will be synchronized after the conversion is completed
Output Params	Mat &m0 - The output MAT object is converted into the standard OpenCV BGR format
Return Value	BM_SUCCESS(0): successful implementation Other: failed implementation
Description	Currently supports compressed format, Gray, NV12, NV16, YUV444P, YUV422P, YUV420P, BGR separate, BGR packed, CV_8UC1 to BGR packed format conversion. In the YUV format, the correct color conversion matrix will be automatically selected according to the colorspace and color_range information in the AVFrame structure.

bm_status_t toMAT(bm_image *image, Mat &m, int color_space, int color_range, void* vaddr = NULL, int fd0 = -1, bool update = true, bool nocopy = true)

Function Prototype	bm_status_t bmcv::toMAT(bm_image *image, Mat &m, int color_space, int color_range, void* vaddr=NULL, int fd0=-1, bool update=true, bool nocopy=true)
Function	The input bm_image object, when nocopy is true, directly multiplexes the device memory and converts it to Mat format. When nocopy is false, the behavior is similar to 3.13 toMAT interface, 1N mode.
Input Params	bm_image *image - The input bm_image object can be in various YUV formats or BGR formats; Int color_space – The color space of the input image, which can be AVCOL_SPC_BT709 or AVCOL_SPC_BT470, see the definition in FFMPEG pixfmt.h for details; Int color_range – The color dynamic range of the input image, which can be AVCOL_RANGE_MPEG or AVCOL_RANGE_JPEG, see the definition in FFMPEG pixfmt.h for details; Void* vaddr – Output Mat’ s system virtual memory pointer. If allocated, output Mat directly uses this memory as Mat’ s system memory. If NULL, Mat is internally allocated automatically; Int fd0 – Physical memory handle of the output Mat, if negative, use the device memory handle in bm_image, otherwise use the handle given by fd0 to mmap the device memory; bool update -Whether to synchronize cache or memory. If it is true, the cache will be synchronized to the system memory after the conversion is completed; bool nocopy – If true, it will directly refer to the device memory of bm_image, if false, it will be converted into standard BGR Mat format.
Output Params	Mat &m - The output MAT object, when nocopy is true, outputs Mat in standard BGR format or extended YUV format; when nocopy is false, converts into standard OpenCV BGR format.
Return Value	BM_SUCCESS(0): successful implementation Other: failed implementation
Description	1.The no copy mode only supports the 1N mode, and the 4N mode cannot support reference because of the memory arrangement. 2.When nocopy is false, the correct color conversion matrix will be automatically selected for color conversion according to the parameters colorspace and color_range information. 3.If the system memory vaddr is external, then the external needs to manage the release of this memory, and the memory will not be released when Mat is released

bm_status_t bmcv::toMAT(bm_image *image, Mat &m0, bool update = true, csc_type_t csc = CSC_MAX_ENUM)

Function Prototype	bm_status_t bmcv::toMAT(bm_image *image, Mat &m0, bool update=true, csc_type_t csc=CSC_MAX_ENUM)
Function	The input bm_image object can be in various YUV or BGR formats, converted to MAT object output in BGR format, 1N mode
Input Params	bm_image *image - The input bm_image object can be in various YUV formats or BGR formats; bool update - Whether to synchronize cache or memory. If it is true, the cache will be synchronized after the conversion is completed; csc_type_t csc – Color conversion type, required only when the input bm_image is in YUV format and needs a CSC (Color Space Conversion). The default type is YCbCr2RGB_BT601.
Output Params	Mat &m0 - The output MAT object is converted into the standard OpenCV BGR format
Return Value	BM_SUCCESS(0)：successful implementation Other：failed implementation

bm_status_t bmcv::toMAT(bm_image *image, Mat &m0, Mat &m1, Mat &m2, Mat &m3, bool update=true, csc_type_t csc=CSC_MAX_ENUM)

Function Prototype	bm_status_t bmcv::toMAT(bm_image *image, Mat &m0, Mat &m1, Mat &m2, Mat &m3, bool update=true, csc_type_t csc=CSC_MAX_ENUM)
Function	The input bm_image object can be in various YUV or BGR formats, converted to MAT object output in BGR format, 4N mode, only valid in BM1688
Input Params	bm_image *image - The input bm_image object in 4N mode can be in various YUV formats or BGR formats; bool update - Whether to synchronize cache or memory. If it is true, the cache will be synchronized after the conversion is completed; csc_type_t csc – Color conversion type, required only when the input bm_image is in YUV format and needs a CSC (Color Space Conversion). The default type is YCbCr2RGB_BT601.
Output Params	Mat &m0 - The first MAT object output is converted into the standard OpenCV BGR format; Mat &m1 - The second MAT object output is converted into the standard OpenCV BGR format; Mat &m2 - The third MAT object output is converted into the standard OpenCV BGR format; Mat &m3 - The fourth MAT object output is converted into the standard OpenCV BGR format
Return Value	BM_SUCCESS(0): successful implementation Other: failed implementation

bm_status_t bmcv::resize(Mat &m, Mat &out, bool update = true, int interpolation= BMCV_INTER_NEAREST)

Function Prototype	bm_status_t bmcv::resize(Mat &m, Mat &out, bool update = true, int interpolation = BMCV_INTER_NEAREST)
Function	The input MAT object is scaled to the size given by the output Mat, and the output format is the color space specified by the output Mat. Because MAT supports the extended YUV format, the color space supported by this interface is not limited to BGR packed.
Input Params	Mat &m - The input Mat object can be in standard BGR packed format or extended YUV format; bool update - Whether to synchronize cache or memory. If it is true, the cache will be synchronized after the conversion is completed; int interpolation – The scaling algorithm can be NEAREST or LINEAR algorithm
Output Params	Mat &out - The output scaled Mat object
Return Value	BM_SUCCESS(0): successful implementation Other: failed implementation
Description	Support Gray, YUV444P, YUV420P, BGR/RGB separate, BGR/RGB packed, ARGB packed format scaling

bm_status_t bmcv::convert(Mat &m, Mat &out, bool update=true)

Function Prototype	bm_status_t bmcv::convert(Mat &m, Mat &out, bool update = true)
Function	It realizes color conversion between two mats. The difference between it and the toMat interface is that toMat can only realize color conversion from various color formats to BGR packed, while this interface can support BGR packed or YUV format to BGR packed or YUV convert.
Input Params	Mat &m - The input Mat object can be in extended YUV format or standard BGR packed format; bool update - Whether to synchronize cache or memory. If true, the cache will be synchronized after the conversion is completed
Output Params	Mat &out - The output color-converted Mat object can be in BGR packed or YUV format.
Return Value	BM_SUCCESS(0): successful implementation Other: failed implementation

bm_status_t bmcv::convert(Mat &m, std::vector<Rect> &vrt, std::vector<Size> &vsz, std::vector<Mat> &out, bool update= true, csc_type_t csc=CSC_YCbCr2RGB_BT601, csc_matrix_t *matrix = nullptr, bmcv_resize_algorithm algorithm= BMCV_INTER_LINEAR)

Function Prototype	bm_status_t bmcv::convert(Mat &m, std::vector<Rect> &vrt, std::vector<Size> &vsz, std::vector<Mat> &out, bool update = true, csc_type_t csc=CSC_YCbCr2RGB_BT601, csc_matrix_t *matrix=nullptr, bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR)
Function	The interface adopts the built-in VPP hardware acceleration unit, which integrates crop, resize and csc. According to the given multiple rect boxes and given multiple scaling sizes, the input Mat object is output to multiple Mat objects, and the output is OpenCV standard BGR pack format or extended YUV format
Input Params	Mat &m - The input Mat object can be in extended YUV format or standard BGR packed format; std::vector<Rect> &vrt - Multiple rect boxes, the ROI area in the input Mat. The number of rectangular boxes and the number of resize should be the same; std::vector<Size> &vsz - Multiple resize sizes, one-to-one correspondence with the rectangular box of vrt; bool update - Whether to synchronize cache or memory. If it is true, the cache will be synchronized after the conversion is completed; csc_type_t csc – Color conversion matrix, can specify the appropriate color conversion matrix according to the color space; csc_matrix_t *matrix – When the color conversion matrix is not in the list, an external user-defined conversion matrix can be given; bmcv_resize_algorithm algorithm – The scaling algorithm can be NEAREST or LINEAR algorithm
Output Params	std::vector<Mat> &out - Output scaled, cropped, and color-converted Mat objects in standard BGR pack format or YUV format.
Return Value	BM_SUCCESS(0): successful implementation Other: failed implementation
Description	The interface can complete the three operations of resize, crop, and csc in one step, with the highest efficiency. Use this interface as much as possible to improve efficiency

bm_status_t bmcv::convert(Mat &m, std::vector<Rect> &vrt, bm_image *out, bool update= true)

Function Prototype	bm_status_t bmcv::convert(Mat &m, std::vector<Rect> &vrt, bm_image *out, bool update= true)
Function	The interface adopts the built-in VPP hardware acceleration unit, which integrates crop, resize and csc. According to the given multiple rect boxes, according to the size specified in multiple bm_images, the input Mat objects are output to multiple bm_image objects, and the output format is determined by the bm_image initialization value. Note that bm_image must be initialized by the caller, and the number corresponds to vrt one-to-one.
Input Params	Mat &m - The input Mat object can be in extended YUV format or standard BGR packed format; std::vector<Rect> &vrt - Multiple rect boxes, the ROI area in the input Mat. The number of rectangular boxes and the number of resize should be the same; bool update - Whether to synchronize cache or memory. If true, the cache will be synchronized after the conversion is completed
Output Params	bm_image *out - Output scaling, crop, and color-converted bm_image objects. The output color format is determined by the bm_image initialization value. At the same time, the initialized size and color information contained in the bmimage parameter are also used as input information for processing.
Return Value	BM_SUCCESS(0): successful implementation Other: failed implementation

void bmcv::uploadMat(Mat &mat)

Function Prototype	void bmcv::uploadMat(Mat &mat)
Function	Cache synchronization or device memory synchronization interface. When this function is executed, the content in the cache will be flushed to the actual memory (SOC mode)
Input Params	Mat &mat - The input mat object that needs memory synchronization
Output Params	None
Return Value	None
Description	Reasonably calling this interface can effectively control the number of memory synchronizations, and only call it when needed.

void bmcv::downloadMat(Mat &mat)

Function Prototype	void bmcv::downloadMat(Mat &mat)
Function	Cache synchronization or device memory synchronization interface. When this function is executed, the content in the cache will be invalidated (SOC mode). The memory synchronization direction of this interface is exactly opposite to the 3.21 interface.
Input Params	Mat &mat - The input mat object that needs memory synchronization
Output Params	None
Return Value	None
Description	Reasonably calling this interface can effectively control the number of memory synchronizations, and only call it when needed.

bm_status_t bmcv::stitch(std::vector<Mat> &in, std::vector<Rect>& srt, std::vector<Rect>& drt, Mat &out, bool update = true, bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR)

Function Prototype	bm_status_t bmcv::stitch(std::vector<Mat> &in, std::vector<Rect> &src, std::vector<Rect> &drt, Mat &out, bool update=true, bmcv_resize_alogrithm algorithm=BMCV_INTER_LINEAR)
Function	Image stitching, scaling and stitching multiple input Mats into one Mat according to the given position
Input Params	std::vector<Mat> &in – Multiple input Mat objects, which can be in extended YUV format or standard BGR pack format; std::vector<Rect> &src – The display content box corresponding to each Mat object; std::vector<Rect> &drt – Corresponding to the display position of each display content in the target Mat; bool update - Whether to synchronize cache or memory. If it is true, the cache will be synchronized after the conversion is completed; bmcv_resize_algorithm algorithm – The scaling algorithm can be NEAREST or LINEAR algorithm
Output Params	Mat &out – Output the spliced Mat object, which can be BGR packed or YUV format
Return Value	BM_SUCCESS(0)： successful implementation Other： failed implementation
Description	For bm1688, input and output Mats only support 64-aligned stride

void bmcv::print(Mat &m, bool dump = false)

Function Prototype	void bmcv::print(Mat &m, bool dump = false)
Function	Debug interface, print the color space, width, height and data of the input Mat object.
Input Params	Mat &m - The input Mat object can be in extended YUV format or standard BGRpacked format; bool dump - When true, the data value in the Mat is printed, and it is not printed by default. If true, the mat_dump.bin file will be generated in the current directory
Output Params	None
Return Value	None
Description	Currently supports dump OpenCV standard BGRpacked or CV_8UC1 data, as well as extended NV12, NV16, YUV420P, YUV422P, GRAY, YUV444P and BGRSeparate format data

void bmcv::print(bm_image *image, bool dump)

Function Prototype	void bmcv::print(bm_image *image, bool dump)
Function	Debug interface, print the color space, width, height and data of the input bm_image object.
Input Params	bm_image *image - The input bm_image object; bool dump - When true, the data value in Mat is printed. By default, it is not printed. If true, a BMI- "width" x" height" .bin file will be generated in the current directory.
Output Params	None
Return Value	None
Description	Currently supports dump BGR packed, NV12, NV16, YUV420P, YUV422P, GRAY, YUV444P and BGR Separate format bm_image data

void bmcv::dumpMat(Mat &image, const String &fname)

Function Prototype	void bmcv::dumpMat(Mat &image, const String &fname)
Function	Debug interface, specifically dumpMat data to the specified named file. The function is the same as the function when dump is true in 3.23.
Input Params	Mat &image - The input Mat object can be in extended YUV format or standard BGR packed format; const String &fname – output dump filename
Output Params	None
Return Value	None
Description	Currently supports dump OpenCV standard BGR packed or CV_8UC1 data, as well as extended NV12, NV16, YUV420P, YUV422P, GRAY, YUV444P and BGR Separate format data

void bmcv::dumpBMImage(bm_image *image, const String &fname)

Function Prototype	void bmcv::dumpBMImage(bm_image *image, const String &fname)
Function	Debug interface, specifically dump bm_image data to the specified named file. The function is the same as the function when dump is true in 3.25.
Input Params	bm_image *image - input bm_image object; const String &fname – output dump filename
Output Params	None
Return Value	None
Description	Currently supports dump BGR packed, NV12, NV16, YUV420P, YUV422P, GRAY, YUV444P and BGR Separate format bm_image data

bool Mat::avOK()

Function Prototype	bool Mat::avOK()
Function	Determine whether the current Mat is in extended YUV format
Input Params	None
Output Params	None
Return Value	true – Indicates that the current Mat is in extended YUV format false – Indicates that the current Mat is in standard OpenCV format
Description	Combined with interface 3.21 3.22 downloadMat and uploadMat, it can effectively manage memory synchronization. Generally, a Mat whose avOK is true has the latest physical memory, and a Mat whose avOK is false has the latest data in its cache or host memory. You can decide whether to call uploadMat or downloadMat based on this information. If it is always working through the hardware acceleration unit in device memory, memory synchronization can be omitted and downloadMat is called only when it needs to be swapped into system memory.

int Mat::avCols()

Function Prototype	int Mat::avCols()
Function	Get the width of Y in YUV extended format
Input Params	None
Output Params	None
Return Value	Returns the Y width in extended YUV format, or 0 if it is in standard OpenCV Mat format

int Mat::avRows()

Function Prototype	int Mat::avRows()
Function	Get the height of Y in YUV extended format
Input Params	None
Output Params	None
Return Value	Returns the Y height in extended YUV format, or 0 if it is in standard OpenCV Mat format

int Mat::avFormat()

Function Prototype	int Mat::avFormat()
Function	Get YUV format information
Input Params	None
Output Params	None
Return Value	Returns extended YUV format information, if it is standard OpenCV Mat format, returns 0

int Mat::avAddr(int idx)

Function Prototype	int Mat::avAddr(int idx)
Function	Get the physical address of each component of YUV
Input Params	int idx – Specifies the order number of the YUV plane
Output Params	None
Return Value	Returns the physical head address of the specified plane, if it is in standard OpenCV Mat format, returns 0

int Mat::avStep(int idx)

Function Prototype	int Mat::avStep(int idx)
Function	Get the line size of the specified plane in YUV format
Input Params	int idx - Order number of the specified YUV plane
Output Params	None
Return Value	The line size of the specified plane, if it is in standard OpenCV Mat format, return 0

AVFrame* av::create(int height, int width, int color_format, void *data, long addr, int fd, int* plane_stride, int* plane_size, int color_space = AVCOL_SPC_BT709, int color_range = AVCOL_RANGE_MPEG, int id = 0)

Function Prototype	AVFrame* av::create(int height, int width, int clor_format, void *data, long addr, int fd, int* plane_stride, int* plane_size, int color_space = AVCOL_SPC_BT709, int color_range = AVCOL_RANGE_MPEG, int id = 0)
Function	AVFrame creation interface, allowing external creation of system memory and physical memory, the created format is compatible with the AVFrame definition in FFMPEG
Input Params	int height – The height of the created image data; int width – The width of the created image data; int color_format – The format of the created image data，see the FFMPEG pixfmt.h definition for details; void *data – System memory address. When it is null, it means that the interface creates its own management; long addr – Device memory address; int fd – Handle to the device memory address. If it is -1, it means that the device memory is allocated internally, otherwise it is given by the addr parameter. int* plane_stride – Array of stride per row for each layer of image data; int* plane_size – The size of each layer of the image data; int color_space – The color space of the input image can be AVCOL_SPC_BT709 or AVCOL_SPC_BT470, see the definition in FFMPEG pixfmt.h for details, the default is AVCOL_SPC_BT709; int color_range – The color dynamic range of the input image, which can be AVCOL_RANGE_MPEG or AVCOL_RANGE_JPEG, see the definition in FFMPEG pixfmt.h for details, the default is AVCOL_RANGE_MPEG; int id – Specified device card number and the flag of the HEAP location, see 5.1 for details, the default value of this parameter is 0
Output Params	None
Return Value	AVFrame structure pointer
Description	1.This interface supports the creation of AVFrame data structures in the following image formats: AV_PIX_FMT_GRAY8, AV_PIX_FMT_GBRP, AV_PIX_FMT_YUV420P, AV_PIX_FMT_NV12, AV_PIX_FMT_YUV422P horizontal, AV_PIX_FMT_YUV444P, AV_PIX_FMT_NV16 2.When both the device memory and the system memory are given externally, in soc mode, the external address must match, that is, the system memory is the virtual address mapped from the device memory; when the device memory is given externally, the system memory is null; when the device memory is not provided and the system memory is also null, the interface will automatically create the internal memory; when the device memory is not provided and the system memory is provided externally, the interface will create failed

AVFrame* av::create(int height, int width, int id = 0)

Function Prototype	AVFrame* av::create(int height, int width, int id = 0)
Function	Simple creation interface of AVFrame, all memory is created and managed internally, only supports YUV420P format
Input Params	int height – The height of the created image data; int width – The width of the created image data; int id – Specified device card number and the flag of the HEAP location, see 5.1 for details, this parameter defaults to 0
Output Params	None
Return Value	AVFrame structure pointer
Description	This interface only supports the creation of AVFrame data structures in YUV420P format

int av::copy(AVFrame *src, AVFrame *dst, int id)

Function Prototype	int av::copy(AVFrame *src, AVFrame *dst, int id)
Function	The deep copy function of AVFrame, copies the valid image data of src to dst
Input Params	AVFrame *src – input AVFrame raw data pointer; int id – Specified device card number, see 5.1 for details
Output Params	AVFrame *dst – Output AVFrame target data pointer
Return Value	Returns the number of valid image data for copy, if it is 0, no copy occurs
Description	1.This interface only supports the copy of image data in the same device card number, that is, the id is the same 2.The id in the function only needs to specify the device card number, no other flags are required

int av::get_scale_and_plane(int color_format, int wscale[], int hscale[])

Function Prototype	int av::get_scale_and_plane(int color_format, int wcale[], int hscale[])
Function	Get the aspect ratio of the specified image format relative to YUV444P
Input Params	int color_format – Specify the image format, see the definition in FFMPEG pixfmt.h for details
Output Params	int wscale[] – Corresponding format relative to the width ratio of each layer of YUV444P; int hscale[] - Corresponding format relative to the height ratio of each layer of YUV444P
Return Value	Returns the number of plane layers for the given image format
Description	

cv::Mat(int height, int width, int total, int _type, const size_t* _steps, void* _data, unsigned long addr, int fd, SophonDevice device=SophonDevice())

Function Prototype	cv::Mat(int height, int width, int total, int _type, const size_t* _steps, void* _data, unsigned long addr, int fd, SophonDevice device=SophonDevice())
Function	Added Mat constructor interface. Opencv standard format or extended YUV Mat format can be created, and both system memory and device memory are allowed to be given by external allocation
Input Params	int height – the height of the input image data; int width – the width of the input image data; int total – The size of the memory, which can be the internal memory to be allocated, or the size of the external allocated memory; int _type – Mat type, this interface only supports CV_8UC1 or CV_8UC3, the format _type of the extended YUV Mat is always CV_8UC1; const size_t *steps – The step information of the created image data, if the pointer is null, it is AUTO_STEP; void *_data – System memory pointer, if null, the memory is allocated internally; unsigned long addr – Device physical memory address, any value is considered a valid physical address; int fd – The handle corresponding to the physical memory of the device. If negative, device physical memory is allocated internally; SophonDevice device – The specified device card number and the sign of the HEAP location, see 5.1 for details, this parameter defaults to 0
Output Params	Constructed standard BGR or extended YUV Mat data type
Return Value	None
Description	1.SophonDevice is a type introduced to avoid function matching errors caused by C++ implicit type matching. SophonDevice(int id) can be used to convert directly from the ID in section 5.1 2.When both the device memory and the system memory are given externally, in the soc mode, the external address of the two must be matched, that is, the system memory is the virtual address mapped from the device memory; when the device memory is given externally, the system memory is null When the device memory is not provided and the system memory is also null, the interface will automatically create the internal memory; when the device memory is not provided and the system memory is provided externally, the interface will create.

Mat::Mat(SophonDevice device)

Function Prototype	Mat::Mat(SophonDevice device)
Function	The newly added Mat construction interface, specifying that the subsequent operations of the Mat are on the given device device
Input Params	SophonDevice device - The specified device card number and the sign of the HEAP location, see 5.1 for details
Output Params	Declared Mat data type
Return Value	None
Description	1.This constructor only initializes the device index inside the Mat, and does not actually create memory 2.The biggest function of this constructor is that for some internal create memory functions, the device number and HEAP location for creating memory can be specified in advance through this constructor, so as to avoid allocating a large amount of memory on the default device number 0

void Mat::create(int height, int width, int total, int _type, const size_t* _steps, void* _data, unsigned long addr, int fd, int id = 0)

Function Prototype	void Mat::create(int height, int width, int total, int type, const size_t* _steps, void* _data, unsigned long addr, int fd, int id = 0)
Function	Mat' s allocation memory interface, which allows both system memory and device memory to be given by external allocation, and can also be allocated internally.
Input Params	int height – The height of the input image data; int width – The width of the input image data; int total – Size of the memory, which can be the internal memory to be allocated, or the size of the external allocated memory; int _type – Mat type, this interface only supports CV_8UC1 or CV_8UC3, the format _type of the extended YUV Mat is always CV_8UC1; const size_t *steps – The step information of the created image data, if the pointer is null, it is AUTO_STEP; void *_data – System memory pointer, if null, the memory is allocated internally; unsigned long addr – Device physical memory address, any value is considered a valid physical address; int fd – The handle corresponding to the physical memory of the device. If negative, device physical memory is allocated internally; int id – The specified device card number and HEAP location flag, see 5.1 for details, this parameter defaults to 0
Output Params	None
Return Value	None
Description	1.The extended memory allocation interface, the main improvement purpose is to allow the external specified device physical memory, when the device or system memory is created by the external, the external must be responsible for the release of the memory, otherwise it will cause memory leaks 2.When both the device memory and the system memory are given externally, in the soc mode, the external address of the two must be matched, that is, the system memory is the virtual address mapped from the device memory; when the device memory is given externally, the system memory is null; when the device memory is not provided and the system memory is also null, the interface will automatically create the internal memory; when the device memory is not provided and the system memory is provided externally, the interface will create The Mat only has system memory in soc mode

void VideoWriter::write(InputArray image, char *data, int *len)

Function Prototype	void VideoWriter::write(InputArray image, char *data, int len)
Function	Added video encoding interface. Different from the OpenCV standard VideoWriter::write interface, it provides the function of outputting the encoded video data to the buffer for subsequent processing
Input Params	InputArray image – Input image data Mat structure
Output Params	char *data – output encoded data cache; int *len – output encoded data length
Return Value	None

virtual bool VideoCapture::grab(char *buf, unsigned int len_in, unsigned int *len_out);

Function Prototype	bool VideoCapture::grab(char *buf, unsigned int len_in, usigned int *len_out);
Function	Added stream decoding interface. Different from the OpenCV standard VideoWriter::grab interface, it provides the function of outputting the video data before decoding to buf.
Input Params	char *buf – Memory allocated and freed externally. unsigned int len_in – Size of the buf.
Output Params	char *buf – Output the video data before decoding. int *len_out – The actual size of the output buf.
Return Value	true - the stream decoding is successful; false - the stream decoding is failed.

virtual bool VideoCapture::read_record(OutputArray image, char *buf, unsigned int len_in, unsigned int *len_out);

Function Prototype	bool VideoCapture::read_record(OutputArray image, char *buf, unsigned int len_in, unsigned int *len_out);
Function	Added read stream video interface. It provides the function of outputting the video data before decoding to buf, and outputting the decoded data to image.
Input Params	char *buf – Memory allocated and freed externally. unsigned int len_in – Size of the buf.
Output Params	OutputArray image - Output decoded video data. char *buf – Output the video data before decoding. int *len_out – The actual size of the output buf.
Return Value	true - the stream decoding is successful; false - the stream decoding is failed.

1.2.4 OpenCV Extension for Hardware JPEG Decoder

In BM1688 series chips, JPEG hardware codec module is provided. To use these hardware modules, the SDK software package extends the API functions related to JPEG image processing in OpenCV, such as: cv::imread(), cv::imwrite(), cv::imdecode(), cv:: iencode() etc. When you use these functions for JPEG encoding and decoding, the functions will automatically call the underlying hardware acceleration resources, thus greatly improving the efficiency of encoding and decoding. If you want to keep the original OpenCV API usage of these functions, you can skip this section; but if you still want to learn about the simple and easy-to-use extension functions we provide, this section may be very helpful to you.

Output Image Data in YUV Format

The native cv::imread() and cv::imdecode() API functions of OpenCV perform the decoding operation of JPEG images and return a Mat structure. The Mat structure stores the image data in BGR packed format, which can be regarded as an extensible API. The function function can return the original YUV format data after the JPEG image is decoded. The usage is as follows: When the second parameter flags of these two functions is set to cv::IMREAD_AVFRAME, it means that data in YUV format is stored in the Mat structure out returned after decoding. The specific format of YUV data depends on the image format of the JPEG file. When flags is set to other values or omitted, it means decoding and outputting Mat data in OpenCV’ s native BGR packed format. The description of the extended data format of the input and output of the decoder is shown in the following table:

Input Image format	Input YUV format	FFMPEG corresponding format
I400	I400	AV_PIX_FMT_GRAY8
I420	NV12	AV_PIX_FMT_NV12
I422	NV16	AV_PIX_FMT_NV16
I444	I444 planar	AV_PIX_FMT_YUV444P

The specific FFmpeg format corresponding to the current data can be obtained through the Mat::avFormat() extension function. You can use the Mat::avOK() extension function to know whether the out returned by cv::imdecode(buf, cv::IMREAD_AVFRAME, &out) decoding is the Mat data format extended by SOPHGO.

In addition, when the cv::IMREAD_RETRY_SOFTDEC flag is added to the flags in these two interfaces, it will try to switch the software decoding when the hardware decoding fails. This function can also be achieved by setting the environment variable OPENCV_RETRY_SOFTDEC=1.

List of Functions Supporting YUV Format

At present, SOPHGO Opencv has supported the function interface list of YUV Mat extended format as follows:

- Video decoding class interface
  - Member functions of the VideoCapture class

  Such member functions, such as read and grab, use the hardware acceleration of the BM1688 series for the commonly used HEVC and H264 video formats, and support the YUV Mat extension format.
- Video encoding class interface
  - Member functions of the VideoWriter class

  Such member functions, such as write, have used the hardware acceleration of the BM1688 series for the commonly used HEVC and H264 video formats, and support the YUV Mat extension format.
- JPEG encoding class interface
- JPEG decoding class interface
  - Imread
  - Imwrite
  - Imdecode
  - Imencode

The above interfaces have used the hardware acceleration function of the BM1688 series when processing the JPEG format, and support the YUV Mat extension format.

- Image processing class interface
  - cvtColor
  - resize

    These two interfaces support YUV Mat extended format in BM1688 series SOC mode and are optimized with hardware acceleration.

    In particular, it should be noted that the cvtColor interface only supports hardware acceleration and YUV Mat format when YUV is converted to BGR or GRAY output, that is, only the input is in YUV Mat format, and hardware acceleration is performed, and the output does not support YUV Mat format.
- line
- rectangle
- circle
- putText

  The above four interfaces all support the YUV extension format. Note that these four interfaces do not use hardware acceleration, but use the CPU’s support for the YUV Mat extension format.
- Basic operation class interface
  - Part of the Mat class interface
    - Create release interface: create, release, Mat declaration interface
    - Memory assignment interface: clone, copyTo, cloneAll, copyAllTo, assignTo, operator =,
    - Extended AV interface: avOK, avComp, avRows, avCols, avFormat, avStep, avAddr

  The above interfaces all support the YUV extension format, especially the copyTo and clone interfaces are accelerated by hardware.
- Extended class interface
  - Bmcv interface: see opencv2/core/bmcv.hpp for details
  - AvFrame interface: see opencv2/core/av.hpp for details

  The above SOPHGO extension class interfaces all support the YUV Mat extension format and are optimized for hardware accelerated processing.

Note: The interface supporting the YUV Mat extension format is not equivalent to using hardware acceleration, and some interfaces are implemented through CPU processing. Pay special attention to this.

1.2.5 The Calling Principles of OpenCV and BMCV API

The BMCV API makes full use of the acceleration capability of the hardware unit in the BM1688 series chips, which can improve the efficiency of data processing. The OpenCV software provides very rich image and graphics processing capabilities. The organic combination of the two enables users to develop not only the rich function library of OpenCV, but also the acceleration of hardware-supported functions. This is the main purpose of this section.

In the process of switching between BMCV API and OpenCV functions and data types, the most important thing is to avoid data copying as much as possible to minimize the switching cost. Therefore, the following principles should be followed in the calling process.

1) To switch from OpenCV Mat to BMCV API, you can use the toBMI() function, which converts the data in the Mat into the bm_image type required by the BMCV API call in a zero-copy manner.
2) When the BMCV API needs to switch to OpenCV Mat, the last step should be implemented through the bmcv function in OpenCV. This not only completes the required image processing operations, but also completes the data type preparation for subsequent OpenCV operations. Because OpenCV generally requires the color space of BGR Pack, the toMat() function is generally used as the last step before switching.
3) Generally, the data processed by the neural network is RGB planar data without padding, and there are specific requirements for the input size. Therefore, it is recommended to use the resize() function as the last step before calling the neural network NPU interface.
4) When the three operations of crop, resize, and color conversion are continuous, it is strongly recommended that customers use the convert() function, which can obtain ideal benefits in both bandwidth optimization and speed optimization. Even if a subsequent copy may be required, the cost is worth it because the copy occurs on the scaled image.

1.2.6 Introduction to National Standard GB28181 Interface in OpenCV

SOPHGO reuses the native Cap interface of OpenCV, and provides the playback support of GB28181 by extending the url definition. Therefore, users do not need to be familiar with the interface again, as long as they understand the extended url definition, they can play GB28181 video consistently like rtsp video.

Note: The SIP proxy registration steps in the national standard need to be managed by the user. When the front-end device list is obtained, it can be played directly by url.

General Steps Supported by National Standard GB28181

- Start SIP proxy (usually deployed by the customer or provided by the platform)
- Customer' s subordinate application platform registered to SIP proxy
- The client application gets the list of front-end devices as shown below. Among them, 34010000001310000009 etc. are the 20-bit codes of the device.

  { "devidelist" :

  [{ "id" : "34010000001310000009" }

  { "id" : "34010000001310000010" }

  { "id" : "34020000001310101202" }]}
- Organize the GB28181 url to directly call the OpenCV Cap interface for playback

GB28181 Url Format Definition

UDP Real-time Stream Address Definition

```
gb28181://34020000002019000001:123456@35.26.240.99:5666?deviceid
=34010000001310000009#localid=12478792871163624979#localip=172.
10.18.201#localmediaport=20108:
```

```
gb28181://34020000002019000001:123456@35.26.240.99:5666?deviceid
=34010000001310000009#localid=12478792871163624979#localip=172.
10.18.201#localmediaport=20108:
```

Hint

34020000002019000001:123456@35.26.240.99:5666:

sip server GB code: sip server password@sip server ip address: sip server port deviceid:

Front-end device 20-bit code

localid:

Local 20-bit code, optional

localip:

Local ip, optional

localmediaport:

The video stream port of the media receiving end needs to be mapped to two ports (rtp:11801, rtcp:11802), and the in and out of the two port mappings must be the same. The same core board port cannot be repeated.

UDP Playback Stream Address Definition

gb28181_playback://34020000002019000001:123456@35.26.240.99:5666?deviceid =35018284001310090010#devicetype=3#localid=12478792871163624979#localip= 172.10.18.201#localmediaport=20108#begtime=20191018160000#endtime =20191026163713:

gb28181_playback://34020000002019000001:123456@35.26.240.99:5666?deviceid =35018284001310090010#devicetype=3#localid=12478792871163624979#localip= 172.10.18.201#localmediaport=20108#begtime=20191018160000#endtime =20191026163713:

Hint

34020000002019000001:123456@35.26.240.99:5666:

sip server GB code: sip server password@sip server ip address: sip server port

deviceid:

Front-end device 20-bit code

devicetype:

Video storage type

localid:

Local 20-bit code, optional

localip:

Local ip, optional

localmediaport:

The video stream port of the media receiving end needs to be mapped to two ports (rtp:11801, rtcp:11802), and the in and out of the two port mappings must be the same. The same core board port cannot be repeated.

begtime:

Recording start time

endtime:

Recording end time

TCP Real-time Stream Address Definition

gb28181://34020000002019000001:123456@35.26.240.99:5666?deviceid
=35018284001310090010#localid=12478792871163624979#localip=172.10.18.201:

gb28181://34020000002019000001:123456@35.26.240.99:5666?deviceid
=35018284001310090010#localid=12478792871163624979#localip=172.10.18.201:

Hint

34020000002019000001:123456@35.26.240.99:5666:

sip server GB code: sip server password@sip server ip address: sip server port

deviceid:

Front-end device 20-bit code

localid:

Local 20-bit code, optional

localip:

Local ip, optional

TCP Playback Stream Address Definition

gb28181_playback://34020000002019000001:123456@35.26.240.99:5666?deviceid
=35018284001310090010#devicetype=3#localid=12478792871163624979#localip=
172.10.18.201#begtime=20191018160000#endtime=20191026163713:

gb28181_playback://34020000002019000001:123456@35.26.240.99:5666?deviceid

=35018284001310090010#devicetype=3#localid=12478792871163624979#localip=
172.10.18.201#begtime=20191018160000#endtime=20191026163713:

Hint

34020000002019000001:123456@35.26.240.99:5666:

sip server GB code: sip server password@sip server ip address: sip server port

deviceid:

Front-end device 20-bit code

devicetype:

Video storage type

localid:

Local 20-bit code, optional

localip:

Local ip, optional

begtime:

Recording start time

endtime:

Recording end time

1.2.7 Code Example

For code examples, see examples/multimedia in the bmnnsdk2 package.

1.3 SOPHGO FFMPEG User Guide

1.3.1 Preface

In the BM1688 series chip, there is an 8-core A53 processor, and it also has built-in video and image related hardware acceleration modules. Interfaces to these hardware modules are provided in the FFMPEG SDK development kit provided by SOPHGO. Among them, through these hardware interfaces, the following modules are provided: hardware video decoder, hardware video encoder, hardware JPEG decoder, hardware JPEG encoder, hardware scale filter, hwupload filter, hwdownload filter.

The FFMPEG SDK development kit complies with the FFMPEG hwaccel writing specification, implements the video transcoding hardware acceleration framework, and implements functions such as hardware memory management and the organization of each hardware processing module process. At the same time, the FFMPEG SDK also provides an interface compatible with common CPU decoders to match the usage habits of some users. We call these two sets of interfaces the HWAccel interface and the general interface. They share the BM1688 hardware acceleration module at the bottom and are the same in performance. The only difference is that 1) HWAccel needs to initialize the hardware device 2) The HWAccel interface is only for device memory, while the general interface allocates both device memory and system memory 3) There are slight differences in their parameter configuration and interface calls.

In the following description, unless otherwise specified, it applies to both the general interface and the HWAccel interface.

1.3.2 Hardware Video Decoder

The BM1688 series supports H.264 and H.265 hardware decoding. The hardware decoder performance details are described in the table below.

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

In SophGo’ s FFMPEG release package, the name of the H.264 hardware video decoder is h264_bm, and the name of the H.265 hardware video decoder is hevc_bm. The following commands can be used to query the encoders supported by FFMPEG.

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

Options Supported by Hardware Video Decoder

In FFMPEG, the hardware decoder of BM1688 series provides some additional options, which can be queried by the following commands.

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

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

These options can be set using the av_dict_set API. A proper understanding of these options is required before setting. These options are explained in detail below.

output_format:

- The format of the output data.

- Set to 0 to output linearly arranged uncompressed data; set to 101 to output compressed data.
- Default value is 0.
- The recommended setting is 101 to output compressed data. It can save memory and save bandwidth. The output compressed data can be decompressed into normal YUV data by calling the scale_bm filter described later. For details, please refer to Example 1 in Application Examples.

cbcr_interleave:

- Whether the frame chroma data output by hardware video decoder decoding is in interleaved format.
- Set to 1, the output is a semi-planar yuv image, such as nv12; set to 0, the output is a planar yuv image, such as yuv420p.
- Default value is 1.

extra_frame_buffer_num:

- The number of additional hardware frame buffers provided by the hardware video decoder.
- Default value is 2. Minimum value is 1.

skip_non_idr:

- Frame skip mode. 0, off; 1, skip Non-RAP frames; 2, skip non-reference frames.
- Default value is 0.

handle_packet_loss

- When an error occurs, enable packet loss processing for H.264 and H.265 decoders. 0, no packet loss processing; 1, packet loss processing.
- Default value is 0.

zero_copy:

- Copy the frame data on the device directly to the system memory automatically applied by data[0]-data[3] of AVFrame. 1, disable copy; 0, enable copy.
- Default value is 1.

1.3.3 Hardware Video Encoder

Hardware video encoders were added for the first time since BM1688. It supports H.264/AVC and H.265/HEVC video encoding.

The capability of BM1688 hardware encoder design is: It can encode one channel 1080P30 video in real time. The specific indicators are as follows:

H.265 encoder:

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

H.264 encoder:

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

General indicator

- Maximum resolution : 8192x8192
- Minimum resolution : 256x128
- Encoded image width must be a multiple of 8
- Encoded image height must be a multiple of 8

In SophGo’ s FFMPEG release package, the name of the H.264 hardware video encoder is h264_bm, and the name of the H.265 hardware video encoder is h265_bm or hevc_bm. The following commands can be used to query the encoders supported by FFMPEG.

```
$ ffmpeg -encoders
```

Options Supported by Hardware Video Encoders

In FFMPEG, the hardware video encoder provides some additional options, which can be queried by the following commands.

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

The BM1688 hardware video encoder supports the following options:

preset: preset encoding mode. It is recommended to set it through enc-params.

- 0 - fast, 1 - medium, 2 - slow。
- Default value is 2。

gop_preset: gop preset index value. It is recommended to set it through enc-params.

- 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:

- Rate control method with constant quantization parameter
- The value range is 0 to 51

perf:

- Used to indicate if encoder performance needs to be tested
- The value is 0 or 1

enc-params:

- Used to set the internal parameters of the video encoder.
- Supported encoding parameters: preset, gop_preset, qp, bitrate, mb_rc, delta_qp, min_qp, max_qp, bg, nr, deblock, weightp
- Encoding parameter preset: the value range is fast, medium, slow or 0, 1, 2
- Encoding parameter gop_preset: gop preset index value. Refer to the above for a detailed explanation.

  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
- Encoding parameter qp: constant quantization parameter, the value range is [0, 51]. When this value is valid, the rate control algorithm is turned off, and the fixed quantization parameter is used for coding.
- Encoding parameter bitrate: used to encode the specified bitrate. The unit is Kbps, 1Kbps=1000bps. When specifying the parameter, please do not set the encoding parameter qp.

- Encoding parameter mb_rc: the value is 0 or 1. When set to 1, the macroblock-level rate control algorithm is enabled; when it is set to 0, the frame-level rate control algorithm is enabled.
- Encoding parameter delta_qp: the maximum difference of QP for the rate control algorithm. Too large a value will affect the subjective quality of the video. Too small will affect the speed of bit rate adjustment.
- Encoding parameters min_qp and max_qp: the minimum and maximum quantization parameters used to control the code rate and video quality in the rate control algorithm. The value range is [0, 51].
- Encoding parameter bg: whether to enable background detection. The value is 0 or 1.
- Encoding parameter nr: Whether to enable the noise reduction algorithm. The value is 0 or 1.
- Encoding parameter deblock: whether to enable the ring filter. There are the following usages:
  - Turn off the ring filter "deblock=0" or "no-deblock".
  - Simply turn on the ring filter, using the default ring filter parameter "deblock=1".
  - Turn on the ring filter and set the parameters, such as "deblock=6,6".
- Encoding parameter weightp: Whether to enable P frame, B frame weighted prediction. The value is 0 or 1.

is_dma_buffer:

- Used to inform the encoder whether the input frame buffer is a contiguous physical memory address on the device.
- In SoC mode, 0 indicates that the virtual address of the device memory is input. 1 means that the input is a contiguous physical address on the device.
- Default value is 1.
- Applies only to regular interfaces.

1.3.4 Hardware JPEG Decoder

In BM1688 series chips, hardware JPEG decoder provides hardware JPEG image decoding input capability. Here is how to implement hardware JPEG decoding through FFMPEG.

In FFMPEG, the name of the hardware JPEG decoder is jpeg_bm. You can use the following command to check whether there is a jpeg_bm decoder in FFMPEG.

$ ffmpeg -decoders | grep jpeg_bm

Options Supported by the Hardware JPEG Decoder

In FFMPEG, you can use the following commands to view the options supported by the jpeg_bm decoder

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

The decoding options are described below. These options in the hardware JPEG decoder can be reset using the av_dict_set() API function.

bs_buffer_size: Used to set the buffer size (KBytes) of the input bitstream in the hardware JPEG decoder.

- Value range (0 to INT_MAX)
- Default value is 5120

cbcr_interleave: Used to indicate whether the chroma data in the frame data output by the JPEG decoder is in interleaved format.

- 0: The chroma data in the output frame data is in plannar format
- 1: The chroma data in the output frame data is in interleaved format
- Default value is 0

num_extra_framebuffers: number of extra framebuffer required by JPEG decoder

- For Still JPEG input, it is recommended to set this value to 0
- For Motion JPEG input, it is recommended that the value be at least 2
- Value range (0 to INT_MAX)
- Default value is 2

zero_copy:

- Copy the frame data on the device directly to the memory automatically applied by data[0]-data[3] of AVFrame. 0, disable copy; 1, enable copy.
- Default value is 1.

The new interface provides the hwdownload filter, which can explicitly download data from device memory to system memory.

1.3.5 Hardware JPEG Encoder

In BM1688 series chips, hardware JPEG encoder provides hardware JPEG image encoding output capability. Introduce here,How to implement hardware JPEG encoding through FFMPEG。

In FFMPEG, the name of the hardware JPEG encoder is jpeg_bm。 You can use the following command to check whether there is a jpeg_bm encoder in FFMPEG.

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

Options Supported by the Hardware JPEG Encoder

In FFMPEG, you can use the following command to view the options supported by the jpeg_bm encoder

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

The encoding options are described below. These options in the hardware JPEG encoder can be reset using the av_dict_set() API function.

is_dma_buffer:

- Used to inform the encoder whether the input frame buffer is a contiguous physical memory address on the device.
- In SoC mode, 0 indicates that the virtual address of the device memory is input. 1 means that the input is a contiguous physical address on the device.
- Default value is 1.
- Applies only to regular interfaces.

1.3.6 Hardware Scale Filter

The BM1688 series hardware scale filter is used to "scale/crop/fill" the input image. For example, transcoding applications. After decoding the 1080p video, use hardware scale to scale it to 720p, and then compress and output.

Content	Maximum Resolution	Minimum Resolution	Magnification
Hardware Limitations	4096 * 4096	8*8	32

In FFMPEG, the name of the hardware scale filter is scale_bm。

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

Options Supported by the Hardware Scale Filter

In FFMPEG, you can use the following command to view the options supported by the scaler_bm encoder

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

The description of the scale_bm options is as follows:

w:

- The width of the scaled output video. Please refer to the usage of ffmpeg scale filter.

h:

- The height of the scaled output video. Please refer to the usage of ffmpeg scale filter.

format:

- The output format of the scaled output video. Please refer to the usage of ffmpeg scale filter.
- For the supported formats of input and output, see Appendix 7.1.
- The default value is "none". That is, the output pixel format is system automatic. Input is yuv420p, output is yuv420p; input is yuvj420p, output is yuvj420p. When the input is nv12, the default output is yuv420p.
- Under the HWAccel framework: support the format conversion from nv12 to yuv420p, nv12 to yuvj420p, yuv420p to yuvj420p, yuvj422p to yuvj420p, and yuvj422p to yuv420p. See Appendix 7.1 for support in normal mode without the HWAccel framework enabled.

Input	Output	Whether to Support Scaling	Whether to Support Color Conversion
GRAY8	GRAY8	Yes	Yes
NV12(compressed)	YUV420P	Yes	Yes
	YUV422P	No	Yes
	YUV444P	Yes	Yes
	BGR	Yes	Yes
	RGB	Yes	Yes
	RGBP	Yes	Yes
	BGRP	Yes	Yes
NV12(uncompressed	YUV420P	Yes	Yes
	YUV422P	No	Yes
	YUV444P	Yes	Yes
	BGR	Yes	Yes
	RGB	Yes	Yes
	RGBP	Yes	Yes
	BGRP	Yes	Yes
YUV420P	YUV420P	Yes	Yes
	YUV422P	No	Yes
	YUV444P	Yes	Yes
	BGR	Yes	Yes
	RGB	Yes	Yes
	RGBP	Yes	Yes
	BGRP	Yes	Yes
YUV422P	YUV420P	Yes	Yes
	YUV422P	No	No
	YUV444P	No	No
	BGR	Yes	Yes

continues on next page

Table 1.1 – continued from previous page

Input	Output	Whether to Support Scaling	Whether to Support Color Conversion
	RGB	Yes	Yes
	RGBP	Yes	Yes
	BGRP	Yes	Yes
YUV444P	YUV420P	Yes	Yes
	YUV422P	No	Yes
	YUV444P	Yes	Yes
	BGR	Yes	Yes
	RGB	Yes	Yes
	RGBP	Yes	Yes
	BGRP	Yes	Yes
BGR、RGB	YUV420P	Yes	
	YUV422P	No	Yes
	YUV444P	Yes	Yes
	BGR	Yes	Yes
	RGB	Yes	Yes
	RGBP	Yes	Yes
	BGRP	Yes	Yes
RGBP、BGRP	YUV420P	Yes	
	YUV422P	No	Yes
	YUV444P	Yes	Yes
	BGR	Yes	Yes
	RGB	Yes	Yes
	RGBP	Yes	Yes
	BGRP	Yes	Yes

Table 7.1 scale_bm Pixel Format Support List

opt:

- Scale operation (from 0 to 2) (default 0)
- Value 0 - Only scale operations are supported. Default value.
- Value 1 - Support scale + auto crop operation. The command line parameters can be represented by crop.
- Value 2 - Support scale + padding operation. The command line parameters can be represented by pad.

flags:

- Scale method (from 0 to 2) (default 1)
- Value 0 - nearest filter. In the command line parameters, it can be represented by nearest.

- Value 1 - bilinear filter. In the command line parameters, it can be represented by bilinear.
- Value 2 - bicubic filter. In the command line parameters, it can be represented by bicubic.

sophon_idx:

- Device ID , start from 0.

zero_copy:

- Value 0 - Indicates that the output AVFrame of scale_bm will contain both device memory and host memory pointers, with the best compatibility and slightly lower performance.
- Value 1 - Indicates that the AVFrame output from scale_bm to the next level will only contain valid device address, and will not synchronize data from device memory to system memory. It is recommended that for the next level to use SOPHGO' s encoding/filter, you can choose to set it to 1, and others are recommended to be set to 0.
- Default value is 0

1.3.7 AVFrame Special Definition Description

Following the FFMPEG specification, the hardware decoder provides output through AVFrame, and the hardware encoder provides input through AVFrame. Therefore, when calling the FFMPEG SDK and performing hardware codec processing through the API method, please pay attention to the following special provisions of AVFrame. AVFrame is linear YUV output. In AVFrame, data is the data pointer, which is used to save the physical address, and linesize is the line span of each plane.

Definition of Avframe Interface Output by Hardware Decoder

General Interfaces

Definition of data array

Index	Description
0	Virtual address of Y
1	The virtual address of CbCr when cbcr_interleave=1; The virtual address of Cb when cbcr_interleave=0
2	The virtual address of Cr when cbcr_interleave=0
3	Unused
4	Physical address of Y
5	The physical address of CbCr when cbcr_interleave=1; The physical address of Cb when cbcr_interleave=0
6	The physical address of Cr when cbcr_interleave=0
7	Unused

Definition of linesize array

Index	Description
0	The span of virtual address of Y
1	The span of the virtual address of CbCr when cbcr_interleave=1; The span of the virtual address of Cb when cbcr_interleave=0
2	The span of the virtual address of Cr when cbcr_interleave=0
3	Unused
4	The span of physical address of Y
5	The span of the physical address of CbCr when cbcr_interleave=1; The span of the physical address of Cb when cbcr_interleave=0
6	The span of the physical address of Cr when cbcr_interleave=0
7	Unused

HWAccel Interface

Definition of data array

Index	Uncompressed Format Description	Compressed Format Description
0	Physical address of Y	Physical address of compressed luminance data
1	The physical address of CbCr when cbcr_interleave=1; The physical address of Cb when cbcr_interleave=0	The physical address of compressed chroma data
2	The physical address of Cr when cbcr_interleave=0	The physical address of the offset table for luminance data
3	Reserved	The physical address of the offset table for chroma data
4	Reserved	Reserved

Definition of linesize array

Index	Uncompressed Format Description	Compressed Format Description
0	The span of physical address of Y	The span of luminance data
1	The span of the physical address of CbCr when cbcr_interleave=1; The span of the physical address of Cb when cbcr_interleave=0	The span of chroma data
2	The span of the physical address of Cr when cbcr_interleave=0	The size of the luminance offset table
3	Unused	The size of the chroma offset table

Definition of Avframe Interface Input by Hardware Encoder

General Interfaces

Definition of data array

Index	Description
0	Virtual address of Y
1	Virtual address of Cb
2	Virtual address of Cr
3	Reserved
4	Physical address of Y
5	Physical address of Cb
6	Physical address of Cr
7	Unused

Definition of linesize array

Index	Description
0	The span of virtual address of Y
1	The span of virtual address of Cb
2	The span of virtual address of Cr
3	Unused
4	The span of physical address of Y
5	The span of physical address of Cb
6	The span of physical address of Cr
7	Unused

HWAccel Interface

Definition of data array

Index	Description
0	Physical address of Y
1	Physical address of Cb
2	Physical address of Cr
3	Reserved
4	Reserved

Definition of linesize array

Index	Description
0	The span of physical address of Y
1	The span of physical address of Cb
2	The span of physical address of Cr
3	Unused

AVFrame Interface Definition of Hardware Filter Input and Output

1.When the HWAccel acceleration function is not enabled, the definition of the AVFrame interface adopts the memory layout of the regular interface.

Definition of data array

Index	Description
0	virtual address of Y
1	virtual address of Cb
2	virtual address of Cr
3	Reserved
4	Physical address of Y
5	Physical address of Cb
6	Physical address of Cr
7	Unused

Definition of linesize array

Index	Description
0	The span of virtual address of Y
1	The span of virtual address of Cb
2	The span of virtual address of Cr
3	Unused
4	The span of physical address of Y
5	The span of physical address of Cb
6	The span of physical address of Cr
7	Unused

2.AVFrame interface definition in HWAccel interface

Definition of data array

Index	Description	Compressed Format Input Interface
0	Physical address of Y	Physical address of compressed luminance data
1	Physical address of Cb	Physical address of compressed chroma data
2	Physical address of Cr	Physical address of the offset table for luminance data
3	Reserved	Physical address of the offset table for chroma data
4	Reserved	Reserved

Definition of linesize array

Index	Description	Compressed Format Input Interface
0	The span of the physical address of Y	The span of luminance data
1	The span of the physical address of Cb	The span of chroma data
2	The span of the physical address of Cr	The size of the luminance offset table
3	Unused	The size of the chroma offset table

1.3.8 Application Examples of Hardware Acceleration in FFMPEG Command

The FFMPEG command line parameters corresponding to the normal mode and the HWAccel mode are given below at the same time.

To facilitate understanding, here is a summary of the description:

- In normal mode, whether the output memory of the bm decoder is synchronized to the system memory is controlled by zero_copy, the default is 1.
- In normal mode, whether the input memory of the bm encoder is in system memory or device memory is controlled by is_dma_buffer, the default value is 1.
- In normal mode, the bm filter will automatically determine the synchronization of the input memory, and whether the output memory is synchronized to the system memory is controlled by zero_copy, the default value is 0.
- In HWAccel mode, the synchronization between device memory and system memory is controlled by hwupload and hwdownload.

- In normal mode, use sophon_idx to specify the device, the default is 0; in HWAccel mode, use hwaccel_device to specify.

Example 1

Use device 0. Decode H.265 video, output compressed frame buffer, scale_bm decompress compressed frame buffer and scale into CIF, and then encode into H.264 code stream.

Normal mode:

```
ffmpeg -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
-vf "scale_bm=352:288:zero_copy=1" \
-c:v h264_bm -g 256 -b:v 256K \
-y wkc_100_cif_scale.264
```

HWAccel mode:

```
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
```

Example 2

Use device 0. Decode H.265 video, scale and auto crop to CIF, then encode to H.264 stream.

Normal mode:

```
ffmpeg -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
-vf "scale_bm=352:288:opt=crop:zero_copy=1" \
-c:v h264_bm -g 256 -b:v 256K \
-y wkc_100_cif_scale_crop.264
```

HWAccel mode:

```
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
```

Example 3

Use device 0. Decode H.265 video, scale and automatically padding into CIF, and then encode into H.264 stream.

Normal mode:

```
ffmpeg -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
-vf "scale_bm=352:288:opt=pad:zero_copy=1" \
-c:v h264_bm -g 256 -b:v 256K \
-y wkc_100_cif_scale_pad.264
```

HWAccel mode:

```
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
```

Example 4

Demonstration video screenshot function. Use device 0. Decode H.265 video, scale and automatically padding to CIF, then encode to jpeg image.

Normal mode:

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

HWAccel mode:

```
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
```

Example 5

Demonstrate video transcoding + video screenshot function. Use device 0. Hardware decode H.265 video, scale it into CIF, and then encode it into H.264 code stream; at the same time, it scales to 720p, and then encode it into JPEG image.

Normal mode:

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

HWAccel mode:

```
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
```

Example 6

Demonstrates the hwdownload function. Hardware decode H.265 video, and then download and store it as a YUV file.

Filter hwdownload is dedicated to the HWAccel interface and is used to synchronize device memory and system memory. In normal mode, this step can be achieved by specifying the zero_copy option in the codec, so the hwdownload filter is not required.

Normal mode:

```
ffmpeg -c:v hevc_bm -cbcr_interleave 0 -zero_copy 0 \
-i src/wkc_100.265 -y test_transfer.yuv
```

HWAccel mode:

```
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
```

Example 7

Demonstrates the hwdownload function. Hardware decode H.265 video, scales it into CIF format, and then download and store it as a YUV file.

In normal mode, scale_bm will automatically determine whether to synchronize memory according to the filter chain, so hwdownload is not required.

Normal mode:

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

HWAccel mode:

```
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
```

Example 8

Demonstrates the hwupload function. Use device 0. Upload YUV video, then encode H.264 video.

Filter hwupload is dedicated to the HWAccel interface and is used to synchronize device memory and system memory. In normal mode, this step can be achieved by specifying the is_dma_buffer option in the encoder, so the hwupload filter is not required.

Normal mode:

```
ffmpeg -s 1920x1080 -pix_fmt yuv420p -i test_transfer.yuv \
-c:v h264_bm -b:v 3M -is_dma_buffer 0 -y test_transfer.264
```

HWAccel mode:

```
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
```

Here foo is an alias for device 0.

Example 9

Demonstrates the hwupload function. Use device 1. Upload YUV video, scale to CIF, then encode H.264 video.

Normal mode:

```
ffmpeg -s 1920x1080 -i test_transfer.yuv \
-vf "scale_bm=352:288:sophon_idx=0:zero_copy=1" \
-c:v h264_bm -b:v 256K -sophon_idx 0 \
-y test_transfer_cif.264
```

Description: 1) -pix_fmt yuv420p is not specified here because the default input is yuv420p format

2) In normal mode, bm_scale filter, decoder, encoder specifies which device to use through the parameter sophon_idx

HWAccel mode:

```
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
```

Description: Here foo is the alias of device 1. In HWAccel mode, the specific hardware device is specified by init_hw_device.

Example 10

Demonstrates the hwdownload function. Hardware decode the JPEG video of YUVJ444P, and then download and store it as a YUV file.

Normal mode:

```
ffmpeg -c:v jpeg_bm -zero_copy 0 -i src/car/1920x1080_yuvj444.jpg \
-y car_1080p_yuvj444_dec.yuv
```

HWAccel mode:

```
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
```

Example 11

Demonstrates the hwupload function. Use device 1. Upload the YUVJ444P image data, and then encode the JPEG image.

Normal mode:

```
ffmpeg -s 1920x1080 -pix_fmt yuvj444p -i car_1080p_yuvj444.yuv \
-frames:v 1 -c:v jpeg_bm -sophon_idx 1 -is_dma_buffer 0 \
-y car_1080p_yuvj444_enc.jpg
```

HWAccel mode:

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

Here foo is the alias of device 1.

Example 12

Demonstrate the method of mixing soft decoding and hard encoding. Use device 2. Use the h264 soft decoder that comes with ffmpeg to decode the H.264 video, upload the decoded data to chip 2, and then encode the H.265 video.

Normal mode:

```
ffmpeg -c:v h264 -i src/1920_18MG.mp4 \
-c:v h265_bm -is_dma_buffer 0 -sophon_idx 2 -g 256 -b:v 5M \
-y test265.mp4
```

HWAccel mode:

```
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
```

Here foo is the alias of device 2.

Example 13

Demonstrates how to set the video encoder using enc-params. Use device 0. Decode H.265 video, scale to CIF, and encode to H.264 stream.

Normal mode:

```
ffmpeg -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
-vf "scale_bm=352:288:zero_copy=1" -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
```

HWAccel mode:

```
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
```

Example 14

Use device 0. Decode H.265 video, use bilinear filter, scale to CIF, and automatically padding, and then encode into H.264 code stream.

Normal mode:

```
ffmpeg -c:v hevc_bm -output_format 101 -i src/wkc_100.265 \
-vf "scale_bm=352:288:opt=pad:flags=bilinear:zero_copy=1" \
-c:v h264_bm -g 256 -b:v 256K \
-y wkc_100_cif_scale_pad.264
```

HWAccel mode:

```
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
```

1.3.9 Use Hardware Acceleration Function by Calling the API

The example in examples/multimedia/ff_bmcv_transcode demonstrates the entire process of using ffmpeg for codec and bmcv for image processing.

The example in examples/multimedia/ff_video_decode demonstrates the process of decoding using ffmpeg.

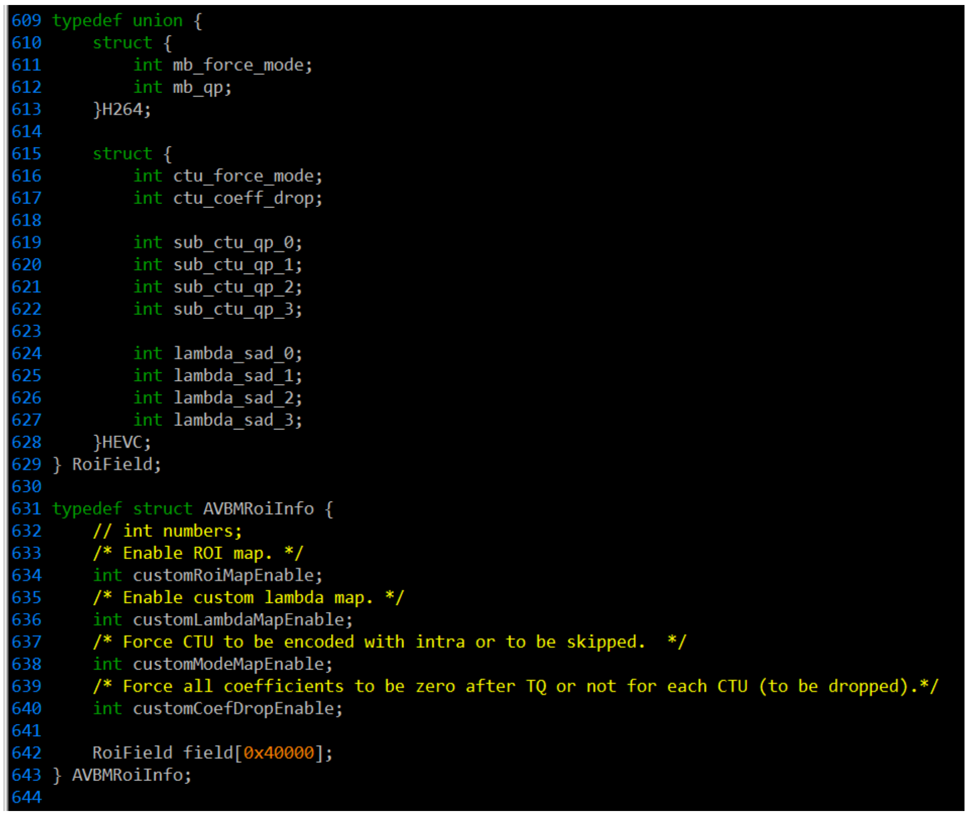
The example in examples/multimedia/ff_video_encode demonstrates the process of encoding using ffmpeg.

1.3.10 Hardware encoding Supporting roi encoding

According to the example in examples/multimedia/ff_video_encode/, you can enable roi encoding by setting roi_enable.

Roi encode data is passed through av_frame side data.

Roi data structure is defined as:



```
typedef union {
    struct {
        int mb_force_mode;
        int mb_qp;
    }H264;

    struct {
        int ctu_force_mode;
        int ctu_coeff_drop;

        int sub_ctu_qp_0;
        int sub_ctu_qp_1;
        int sub_ctu_qp_2;
        int sub_ctu_qp_3;

        int lambda_sad_0;
        int lambda_sad_1;
        int lambda_sad_2;
        int lambda_sad_3;
    }HEVC;
} RoiField;

typedef struct AVBMRoiInfo {
    // int numbers;
    /* Enable ROI map. */
    int customRoiMapEnable;
    /* Enable custom lambda map. */
    int customLambdaMapEnable;
    /* Force CTU to be encoded with intra or to be skipped.  */
    int customModeMapEnable;
    /* Force all coefficients to be zero after TQ or not for each CTU (to be dropped).*/
    int customCoefDropEnable;

    RoiField field[0x40000];
} AVBMRoiInfo;
```

Field Description:

- QP Map

  The QP in H264 is given in units of macroblock 16x16. QP in HEVC is given in units of sub-ctu (32x32). QP corresponds to Qstep in video encoding, and the value range is 0-51.

- Lamda Map

  lamda is used to control and adjust the RC calculation formula inside the IP

  cost = distortion + lamda * rate

  This tuning parameter is only valid in HEVC and allows control in units of 32x32 sub-CTU modules.

- Mode Map

  This parameter is used to specify the mode selection. 0 - not applicable 1 - skip mode 2- intra mode. It is controlled in units of 16x16 macroblocks in H264, and in units of CTU 64x64 in HEVC.

- Zero-cut Flag

  Only valid in HEVC. All the current CTU 64x64 residual coefficients are set to 0, thereby saving more bits for other more important parts.

1.4 SOPHGO LIBYUV User Guide

1.4.1 Introduction

Various hardware modules in the BM1688 series chips can accelerate the processing of pictures and videos. In terms of color conversion, using dedicated hardware to accelerate, the speed will be very fast.

However, in some occasions, there will also be some special cases that cannot be covered by dedicated hardware. At this time, the software implementation optimized by SIMD acceleration is adopted, which becomes a powerful supplement to the dedicated hardware.

SOPHGO Enhanced **libyuv** , is a component released with the SDK.The purpose is to make full use of the 8-core A53 processor provided by the BM1688 series chips, supplementing the limitations of the hardware through software.

In addition to the standard functions provided by libyuv, for the needs of AI, 27 extension functions are added in the SOPHGO enhanced libyuv.

Note: This refers to the A53 processor running on the BM1688 series, not the host processor. This makes sense from the point of view of device acceleration , which avoids taking up the host' s CPU.

1.4.2 Libyuv Extension Description

The following APIs have been added to enhance AI applications.

fast_memcpy

void* fast_memcpy(void *dst, const void *src, size_t n)

Function	The CPU SIMD instruction implements the memcpy function. Copy n bytes from memory area src to memory area dst.	
Params	src	source memory area
	n	number of bytes to copy
	dst	destination memory area
Return Value	returns a pointer to dst	

RGB24ToI400

int RGB24ToI400(const uint8_t* src_rgb24, int src_stride_rgb24, uint8_t* dst_y, int dst_stride_y, int width, int height);

Function	This API can convert a frame of BGR data to BT.601 grayscale data.	
Params	src_rgb24	The virtual address of the memory where the packed BGR image data is located
	src_stride_rgb24	The actual span of each line of BGR image in memory
	dst_y	Virtual address of grayscale image
	dst_stride_y	The actual span of each line of grayscale image in memory
	width	The number of packed BGRs in each row of BGR image data
	height	Number of valid lines of BGR image data
Return Value	0, normal termination; others, abnormal parameter.	

RAWToI400

int RAWToI400(const uint8_t* src_raw, int src_stride_raw, uint8_t* dst_y, int dst_stride_y, int width, int height);

Function	This API can convert a frame of RGB data to BT.601 grayscale data.	
Params	src_row	The virtual address of the memory where the packed BGR image data is located
	src_stride_row	The actual span of each line of BGR image in memory
	dst_y	Virtual address of grayscale image
	dst_stride_y	The actual span of each line of grayscale image in memory
	width	The number of packed BGRs in each row of BGR image data
	height	Number of valid lines of BGR image data
Return Value	0, normal termination; others, abnormal parameter.	

I400ToRGB24

int I400ToRGB24(const uint8_t* src_y, int src_stride_y, uint8_t* dst_rgb24, int dst_stride_rgb24, int width, int height);

Function	This API can convert a frame of BT.601 grayscale data to BGR data.	
Params	src_y	Virtual address of grayscale image
	src_stride_y	The actual span of each line of grayscale image in memory
	dst_rgb24	The virtual address of the memory where the packed BGR image data is located
	dst_stride_rgb24	The actual span of each line of BGR image in memory
	width	The number of packed BGRs in each row of BGR image data
	height	Number of valid lines of BGR image data
Return Value	0, normal termination; others, abnormal parameter.	

I400ToRAW

int I400ToRAW(const uint8_t* src_y, int src_stride_y, uint8_t* dst_raw, int dst_stride_raw, int width, int height);

Function	This API can convert a frame of BT.601 grayscale data to RGB data.	
Params	src_y	Virtual address of grayscale image
	src_stride_y	The actual span of each line of grayscale image in memory
	dst_rgb24	The virtual address of the memory where the packed RGB image data is located
	dst_stride_rgb24	The actual span of each line of RGB image in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

J400ToRGB24

int J400ToRGB24(const uint8_t* src_y, int src_stride_y, uint8_t* dst_rgb24, int dst_stride_rgb24, int width, int height);

Function	This API can convert a frame of BT.601 full range grayscale data to BGR data.	
Params	src_y	Virtual address of grayscale image
	src_stride_y	The actual span of each line of grayscale image in memory
	dst_rgb24	The virtual address of the memory where the packed BGR image data is located
	dst_stride_rgb24	The actual span of each line of BGR image in memory
	width	The number of packed BGRs in each row of BGR image data
	height	Number of valid lines of BGR image data
Return Value	0, normal termination; others, abnormal parameter.	

RAWToJ400

int RAWToJ400(const uint8_t* src_raw, int src_stride_raw, uint8_t* dst_y, int dst_stride_y, int width, int height);

Function	This API can convert a frame of RGB data to BT.601 full range grayscale data.	
Params	src_raw	The virtual address of the memory where the packed RGB image data is located
	src_stride_raw	The actual span of each line of RGB image in memory
	dst_y	Virtual address of grayscale image
	dst_stride_y	The actual span of each line of grayscale image in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

J400ToRAW

int J400ToRAW(const uint8_t* src_y, int src_stride_y, uint8_t* dst_raw, int dst_stride_raw, int width, int height);

Function	This API can convert a frame of BT.601 full range grayscale data to RGB data.	
Params	src_y	Virtual address of grayscale image
	src_stride_y	The actual span of each line of grayscale image in memory
	dst_rgb24	The virtual address of the memory where the packed RGB image data is located
	dst_stride_rgb24	The actual span of each line of RGB image in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

RAWToNV12

int RAWToNV12(const uint8_t* src_raw, int src_stride_raw, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_uv, int dst_stride_uv, int width, int height);

Function	This API can convert a frame of RGB data to semi-planar YCbCr 420 data of BT.601 limited range.	
Params	src_raw	The virtual address of the memory where the packed RGB image data is located
	src_stride_raw	The actual span of each line of RGB image in memory
	dst_y	Virtual address of Y
	dst_stride_y	The actual span of each row of Y data in memory
	dst_uv	Virtual address of CbCr
	dst_stride_uv	The actual span of each row of CbCr data in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

RGB24ToNV12

int RGB24ToNV12(const uint8_t* src_raw, int src_stride_raw, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_uv, int dst_stride_uv, int width, int height);

Function	This API can convert a frame of BGR data into semi-planar YCbCr 420 data of BT.601 limited range.	
Params	src_raw	The virtual address of the memory where the packed BGR image data is located
	src_stride_raw	The actual span of each line of BGR image in memory
	dst_y	Virtual address of Y
	dst_stride_y	The actual span of each row of Y data in memory
	dst_uv	Virtual address of CbCr
	dst_stride_uv	The actual span of each row of CbCr data in memory
	width	The number of packed BGRs in each row of BGR image data
	height	Number of valid lines of BGR image data
Return Value	0, normal termination; others, abnormal parameter.	

RAWToJ420

int RAWToJ420(const uint8_t* src_raw, int src_stride_raw, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_u, int dst_stride_u, uint8_t* dst_v, int dst_stride_v, int width, int height);

Function	This API can convert a frame of RGB data to BT.601 full range semi-planar YCbCr 420 data.	
Params	src_raw	The virtual address of the memory where the packed RGB image data is located
	src_stride_raw	The actual span of each line of RGB image in memory
	dst_y	Virtual address of Y
	dst_stride_y	The actual span of each row of Y data in memory
	dst_u	Virtual address of Cb
	dst_stride_u	The actual span of each row of Cb data in memory
	dst_v	Virtual address of Cr
	dst_stride_v	The actual span of each row of Cr data in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

J420ToRAW

```
int J420ToRAW(const uint8_t* src_y, int src_stride_y, const uint8_t* src_u,
int src_stride_u, const uint8_t* src_v, int src_stride_v, uint8_t* dst_raw, int
dst_stride_raw, int width, int height);
```

Function	This API can convert a frame of BT.601 full range YCbCr 420 data to RGB data.	
Params	src_y	Virtual address of Y
	src_stride_y	The actual span of each row of Y data in memory
	src_u	Virtual address of Cb
	src_stride_u	The actual span of each row of Cb data in memory
	src_v	Virtual address of Cr
	src_stride_v	The actual span of each row of Cr data in memory
	dst_raw	The virtual address of the memory where the packed RGB image data is located
	dst_stride_raw	The actual span of each line of RGB image data in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

RAWToJ422

int RAWToJ422(const uint8_t* src_raw, int src_stride_raw, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_u, int dst_stride_u, uint8_t* dst_v, int dst_stride_v, int width, int height);

Function	This API can convert a frame of RGB data to BT.601 full range YCbCr 422 data.	
Params	src_raw	The virtual address of the memory where the packed RGB image data is located
	src_stride_raw	The actual span of each line of RGB image in memory
	dst_y	Virtual address of Y
	dst_stride_y	The actual span of each row of Y data in memory
	dst_u	Virtual address of Cb
	dst_stride_u	The actual span of each row of Cb data in memory
	dst_v	Virtual address of Cr
	dst_stride_v	The actual span of each row of Cr data in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

J422ToRAW

int J422ToRAW(const uint8_t* src_y, int src_stride_y, const uint8_t* src_u, int src_stride_u, const uint8_t* src_v, int src_stride_v, uint8_t* dst_raw, int dst_stride_raw, int width, int height);

Function	This API can convert a frame of BT.601 full range YCbCr 422 data to RGB data.	
Params	src_y	Virtual address of Y
	src_stride_y	The actual span of each row of Y data in memory
	src_u	Virtual address of Cb
	src_stride_u	The actual span of each row of Cb data in memory
	src_v	Virtual address of Cr
	src_stride_v	The actual span of each row of Cr data in memory
	dst_raw	The virtual address of the memory where the packed RGB image data is located
	dst_stride_raw	The actual span of each line of RGB image data in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

RGB24ToJ422

int RGB24ToJ422(const uint8_t* src_rgb24, int src_stride_rgb24, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_u, int dst_stride_u, uint8_t* dst_v, int dst_stride_v, int width, int height);

Function	This API can convert a frame of BGR data to BT.601 full range YCbCr 422 data.	
Params	src_rgb24	The virtual address of the memory where the packed BGR image data is located
	src_stride_rgb24	The actual span of each line of BGR image in memory
	dst_y	Virtual address of Y
	dst_stride_y	The actual span of each row of Y data in memory
	dst_u	Virtual address of Cb
	dst_stride_u	The actual span of each row of Cb data in memory
	dst_v	Virtual address of Cr
	dst_stride_v	The actual span of each row of Cr data in memory
	width	The number of packed BGRs in each row of BGR image data
	height	Number of valid lines of BGR image data
Return Value	0, normal termination; others, abnormal parameter.	

J422ToRGB24

int J422ToRGB24(const uint8_t* src_y, int src_stride_y, const uint8_t* src_u, int src_stride_u, const uint8_t* src_v, int src_stride_v, uint8_t* dst_rgb24, int dst_stride_rgb24, int width, int height);

Function	This API can convert a frame of BT.601 full range YCbCr 422 data to BGR data.	
Params	src_y	Virtual address of Y
	src_stride_y	The actual span of each row of Y data in memory
	src_u	Virtual address of Cb
	src_stride_u	The actual span of each row of Cb data in memory
	src_v	Virtual address of Cr
	src_stride_v	The actual span of each row of Cr data in memory
	dst_rgb24	The virtual address of the memory where the packed BGR image data is located
	dst_stride_rgb24	The actual span of each row of BGR image data in memory
	width	The number of packed BGRs in each line of RGB image data
	height	Number of valid lines of BGR image data
Return Value	0, normal termination; others, abnormal parameter.	

RAWToJ444

int RAWToJ444(const uint8_t* src_raw, int src_stride_raw, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_u, int dst_stride_u, uint8_t* dst_v, int dst_stride_v, int width, int height);

Function	This API can convert a frame of RGB data to BT.601 full range YCbCr 444 data	
Params	src_row	The virtual address of the memory where the packed RGB image data is located
	src_stride_row	The actual span of each line of RGB image in memory
	dst_y	Virtual address of Y
	dst_stride_y	The actual span of each row of Y data in memory
	dst_u	Virtual address of Cb
	dst_stride_u	The actual span of each row of Cb data in memory
	dst_v	Virtual address of Cr
	dst_stride_v	The actual span of each row of Cr data in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

J444ToRAW

int J444ToRAW(const uint8_t* src_y, int src_stride_y, const uint8_t* src_u, int src_stride_u, const uint8_t* src_v, int src_stride_v, uint8_t* dst_raw, int dst_stride_raw, int width, int height);

Function	This API can convert a frame of BT.601 full range YCbCr 444 data to RGB data.	
Params	src_y	Virtual address of Y
	src_stride_y	The actual span of each row of Y data in memory
	src_u	Virtual address of Cb
	src_stride_u	The actual span of each row of Cb data in memory
	src_v	Virtual address of Cr
	src_stride_v	The actual span of each row of Cr data in memory
	dst_raw	The virtual address of the memory where the packed RGB image data is located
	dst_stride_raw	The actual span of each line of RGB image data in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

RGB24ToJ444

int RGB24ToJ444(const uint8_t* src_rgb24, int src_stride_rgb24, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_u, int dst_stride_u, uint8_t* dst_v, int dst_stride_v, int width, int height);

Function	This API can convert a frame of BGR data to BT.601 full range YCbCr 444 data.	
Params	src_rgb24	The virtual address of the memory where the packed BGR image data is located
	src_stride_rgb24	The actual span of each line of BGR image in memory
	dst_y	Virtual address of Y
	dst_stride_y	The actual span of each row of Y data in memory
	dst_u	Virtual address of Cb
	dst_stride_u	The actual span of each row of Cb data in memory
	dst_v	Virtual address of Cr
	dst_stride_v	The actual span of each row of Cr data in memory
	width	The number of packed BGRs in each row of BGR image data
	height	Number of valid lines of BGR image data
Return Value	0, normal termination; others, abnormal parameter.	

J444ToRGB24

int J444ToRGB24(const uint8_t* src_y, int src_stride_y, const uint8_t* src_u, int src_stride_u, const uint8_t* src_v, int src_stride_v, uint8_t* dst_rgb24, int dst_stride_rgb24, int width, int height);

Function	This API can convert a frame of BT.601 full range YCbCr 444 data to BGR data.	
Params	src_y	Virtual address of Y
	src_stride_y	The actual span of each row of Y data in memory
	src_u	Virtual address of Cb
	src_stride_u	The actual span of each row of Cb data in memory
	src_v	Virtual address of Cr
	src_stride_v	The actual span of each row of Cr data in memory
	dst_rgb24	The virtual address of the memory where the packed BGR image data is located
	dst_stride_rgb24	The actual span of each row of BGR image data in memory
	width	The number of packed BGRs in each line of RGB image data
	height	Number of valid lines of BGR image data
Return Value	0, normal termination; others, abnormal parameter.	

H420ToJ420

int H420ToJ420(const uint8_t* src_y, int src_stride_y, const uint8_t* src_u, int src_stride_u, const uint8_t* src_v, int src_stride_v, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_u, int dst_stride_u, uint8_t* dst_v, int dst_stride_v, int width, int height);

Function	This API can convert a frame of BT.709 limited range YCbCr 420 data to BT.601 full range data. It can be used as a preprocessing function before jpeg encoding.	
Params	src_y	Virtual address of Y
	src_stride_y	The actual span of each row of Y data in memory
	src_u	Virtual address of Cb
	src_stride_u	The actual span of each row of Cb data in memory
	src_v	Virtual address of Cr
	src_stride_v	The actual span of each row of Cr data in memory
	dst_y	Virtual address of Y
	dst_stride_y	The actual span of each row of Y data in memory
	dst_u	Virtual address of Cb
	dst_stride_u	The actual span of each row of Cb data in memory
	dst_v	Virtual address of Cr
	dst_stride_v	The actual span of each row of Cr data in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

I420ToJ420

int I420ToJ420(const uint8_t* src_y, int src_stride_y, const uint8_t* src_u, int src_stride_u, const uint8_t* src_v, int src_stride_v, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_u, int dst_stride_u, uint8_t* dst_v, int dst_stride_v, int width, int height);

Func-tion	This API can convert a frame of BT.601 limited range YCbCr 420 data to BT.601 full range data. It can be used as a preprocessing function before jpeg encoding.	
Params	src_y	Virtual address of Y
	src_stride_y	The actual span of each row of Y data in memory
	src_u	Virtual address of Cb
	src_stride_u	The actual span of each row of Cb data in memory
	src_v	Virtual address of Cr
	src_stride_v	The actual span of each row of Cr data in memory
	dst_y	Virtual address of Y
	dst_stride_y	The actual span of each row of Y data in memory
	dst_u	Virtual address of Cb
	dst_stride_u	The actual span of each row of Cb data in memory
	dst_v	Virtual address of Cr
	dst_stride_v	The actual span of each row of Cr data in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Re-turn Value	0, normal termination; others, abnormal parameter.	

NV12ToJ420

int NV12ToJ420(const uint8_t* src_y, int src_stride_y, const uint8_t* src_uv, int src_stride_uv, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_u, int dst_stride_u, uint8_t* dst_v, int dst_stride_v, int width, int height);

Function	This API can convert a frame of BT.601 limited range semi-plannar YCbCr 420 data into BT.601 full range data. It can be used as a preprocessing function before jpeg encoding.	
Params	src_y	Virtual address of Y
	src_stride_y	The actual span of each row of Y data in memory
	src_uv	Virtual address of CbCr
	src_stride_uv	The actual span of each row of CbCr data in memory
	dst_y	Virtual address of Y
	dst_stride_y	The actual span of each row of Y data in memory
	dst_u	Virtual address of Cb
	dst_stride_u	The actual span of each row of Cb data in memory
	dst_v	Virtual address of Cr
	dst_stride_v	The actual span of each row of Cr data in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

NV21ToJ420

int NV21ToJ420(const uint8_t* src_y, int src_stride_y, const uint8_t* src_vu, int src_stride_vu, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_u, int dst_stride_u, uint8_t* dst_v, int dst_stride_v, int width, int height);

Function	This API can convert a frame of BT.601 limited range semi-plannar YCbCr 420 data into BT.601 full range data. It can be used as a preprocessing function before jpeg encoding.	
Params	src_y	Virtual address of Y
	src_stride_y	The actual span of each row of Y data in memory
	src_vu	Virtual address of CrCb
	src_stride_vu	The actual span of each row of CrCb data in memory
	dst_y	Virtual address of Y
	dst_stride_y	The actual span of each row of Y data in memory
	dst_u	Virtual address of Cb
	dst_stride_u	The actual span of each row of Cb data in memory
	dst_v	Virtual address of Cr
	dst_stride_v	The actual span of each row of Cr data in memory
	width	The number of packed RGBs in each line of RGB image data
	height	Number of valid lines of RGB image data
Return Value	0, normal termination; others, abnormal parameter.	

I444ToNV12

int I444ToNV12(const uint8_t* src_y, int src_stride_y, const uint8_t* src_u, int src_stride_u, const uint8_t* src_v, int src_stride_v, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_uv, int dst_stride_uv, int width, int height);

Function	This API can convert one frame of YCbCr 444 data to semi-plannar YCbCr 420 data, and available in full range and limited range. It is not involve in color space conversion and can be used flexibly.	
Param	src_y	The virtual address of Y component of the source image
	src_stride_y	The actual span of each row of Y data in memory
	src_u	The virtual address of Cb component of the source image
	src_stride_u	The actual span of each row of Cb data in memory
	src_v	The virtual address of Cr component of the source image
	src_stride_v	The actual span of each row of Cr data in memory
	dst_y	The virtual address of Y component of the destination image
	dst_stride_y	The actual span of each row of Y data in memory
	dst_uv	The virtual address of CbCr component of the destination image
	dst_stride_uv	The actual span of each row of CbCr data in memory
	width	Number of pixels in each line of image data
	height	number of valid lines of image data pixels
Return Value	0, normal termination; others, abnormal parameter.	

I422ToNV12

int I422ToNV12(const uint8_t* src_y, int src_stride_y, const uint8_t* src_u, int src_stride_u, const uint8_t* src_v, int src_stride_v, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_uv, int dst_stride_uv, int width, int height);

Function	This API can convert one frame of YCbCr 422 data to semi-plannar YCbCr 420 data ,and available in full range and limited range. It is not involve in color space conversion and can be used flexibly.	
Param	src_y	The virtual address of Y component of the source image
	src_stride_y	The actual span of each row of Y data in memory
	src_u	The virtual address of Cb component of the source image
	src_stride_u	The actual span of each row of Cb data in memory
	src_v	The virtual address of Cr component of the source image
	src_stride_v	The actual span of each row of Cr data in memory
	dst_y	The virtual address of Y component of the destination image
	dst_stride_y	The actual span of each row of Y data in memory
	dst_uv	The virtual address of CbCr component of the destination image
	dst_stride_uv	The actual span of each row of CbCr data in memory
	width	Number of pixels in each line of image data
	height	number of valid lines of image data pixels
Return Value	0, normal termination; others, abnormal parameter.	

I400ToNV12

int I400ToNV12(const uint8_t* src_y, int src_stride_y, uint8_t* dst_y, int dst_stride_y, uint8_t* dst_uv, int dst_stride_uv, int width, int height);

Function	This API can convert one frame of YCbCr 400 data to semi-plannar YCbCr 420 data, and available in full range and limited range. It is not involve in color space conversion and can be used flexibly.	
Param	src_y	The virtual address of Y component of the source image
	src_stride_y	The actual span of each row of Y data in memory
	dst_y	The virtual address of Y component of the destination image
	dst_stride_y	The actual span of each row of Y data in memory
	dst_uv	The virtual address of CbCr component of the destination image
	dst_stride_uv	The actual span of each row of CbCr data in memory
	width	Number of pixels in each line of image data
	height	number of valid lines of image data pixels
Return Value	0, normal termination; others, abnormal parameter.	

1.5 SOPHGO JPEG Usage Guidelines

1.5.1 Introduction

The JPEG module is a functional component within the BM1688 product series, capable of encoding and decoding various image formats. This module includes encoding, decoding, mirroring, rotation and ROI functions, and supports various YUV formats. It supports a resolution range from 16x16 to 32768x32768, and the encoding and decoding performance is also excellent through hardware acceleration.

1.5.2 JPEG data structure description

BmJpuLogLevel

This type indicates the log level in the jpeg dynamic library.

BmJpuLogLevel The type definition is as follows:

```
typedef enum
{
    BM_JPU_LOG_LEVEL_ERROR   = 0,
    BM_JPU_LOG_LEVEL_WARNING = 1,
    BM_JPU_LOG_LEVEL_INFO    = 2,
    BM_JPU_LOG_LEVEL_DEBUG   = 3,
    BM_JPU_LOG_LEVEL_LOG     = 4,
    BM_JPU_LOG_LEVEL_TRACE   = 5
} BmJpuLogLevel;
```

BmJpuImageFormat

This type indicates the format of the image.

BmJpuImageFormat The type definition is as follows:

```
typedef enum
{
    BM_JPU_IMAGE_FORMAT_YUV420P = 0,
    BM_JPU_IMAGE_FORMAT_YUV422P,
    BM_JPU_IMAGE_FORMAT_YUV444P,
    BM_JPU_IMAGE_FORMAT_NV12,
    BM_JPU_IMAGE_FORMAT_NV21,
    BM_JPU_IMAGE_FORMAT_NV16,
    BM_JPU_IMAGE_FORMAT_NV61,
    BM_JPU_IMAGE_FORMAT_GRAY,
    BM_JPU_IMAGE_FORMAT_RGB  /* for opencv */
} BmJpuImageFormat;
```

BmJpuColorFormat

This type indicates the pixel format of the image.

BmJpuColorFormat The type definition is as follows:

```
typedef enum
{
    /* planar 4:2:0; if the chroma_interleave parameter is 1, the corresponding␣
↪format is NV12, otherwise it is I420 */
    BM_JPU_COLOR_FORMAT_YUV420          = 0,
    /* planar 4:2:2; if the chroma_interleave parameter is 1, the corresponding␣
↪format is NV16 */
    BM_JPU_COLOR_FORMAT_YUV422_HORIZONTAL = 1,
    /* 4:2:2 vertical, actually 2:2:4 (according to the JPU docs); no corresponding␣
↪format known for the chroma_interleave=1 case */
    /* NOTE: this format is rarely used, and has only been seen in a few JPEG files */
    BM_JPU_COLOR_FORMAT_YUV422_VERTICAL   = 2,
    /* planar 4:4:4; if the chroma_interleave parameter is 1, the corresponding␣
```

(continues on next page)

(continued from previous page)

```
↪format is NV24 */
  BM_JPU_COLOR_FORMAT_YUV444        = 3,
  /* 8-bit greayscale */
  BM_JPU_COLOR_FORMAT_YUV400        = 4,
  /* RGBP */
  BM_JPU_COLOR_FORMAT_RGB           = 5,
  /* BUTT */
  BM_JPU_COLOR_FORMAT_BUTT
} BmJpuColorFormat;
```

- **BM_JPU_COLOR_FORMAT_YUV420**

  YUV420 format. Corresponds to NV12 format when cbcr_interleave=1.
- **BM_JPU_COLOR_FORMAT_YUV422_HORIZONTAL**

  YUV422 format. Corresponds to NV16 format when cbcr_interleave=1.
- **BM_JPU_COLOR_FORMAT_YUV422_VERTICAL**

  YUV422 format(actually arranged in 2:2:4), a format defined by JPU, is not commonly used.
- **BM_JPU_COLOR_FORMAT_YUV444**

  YUV444 format. Corresponds to NV24 format when cbcr_interleave=1.
- **BM_JPU_COLOR_FORMAT_YUV400**

  YUV400 format. Corresponding to grayscale image, only has Y component.
- **BM_JPU_COLOR_FORMAT_RGB**

  RGB format, not used yet.
- **BM_JPU_COLOR_FORMAT_BUTT**

  Undefined format, used to set default values and exception judgments.

BmJpuChromaFormat

This type indicates the chroma format.

BmJpuChromaFormat The type definition is as follows:

```
typedef enum
{
  BM_JPU_CHROMA_FORMAT_CBCR_SEPARATED = 0,
  BM_JPU_CHROMA_FORMAT_CBCR_INTERLEAVE = 1,
  BM_JPU_CHROMA_FORMAT_CRCB_INTERLEAVE = 2
} BmJpuChromaFormat;
```

- **BM_JPU_CHROMA_FORMAT_CBCR_SEPARATED**

  Cb is separated from Cr and is not interleaved.
- **BM_JPU_CHROMA_FORMAT_CBCR_INTERLEAVE**

  Cb and Cr are intertwined.
- **BM_JPU_CHROMA_FORMAT_CRCB_INTERLEAVE**

  Cb and Cr are intertwined.

BmJpuRotateAngle

Angle of rotation.

BmJpuRotateAngle The type definition is as follows:

```
typedef enum
{
  BM_JPU_ROTATE_NONE = 0,
  BM_JPU_ROTATE_90 = 90,
  BM_JPU_ROTATE_180 = 180,
  BM_JPU_ROTATE_270 = 270
} BmJpuRotateAngle;
```

- **BM_JPU_ROTATE_NONE**

  No rotation.
- **BM_JPU_ROTATE_90**

  Rotate 90 degrees counterclockwise.
- **BM_JPU_ROTATE_180**

  Rotate 180 degrees counterclockwise.
- **BM_JPU_ROTATE_270**

  Rotate 270 degrees counterclockwise.

BmJpuMirrorDirection

Mirror direction.

BmJpuMirrorDirection The type definition is as follows:

```
typedef enum
{
  BM_JPU_MIRROR_NONE = 0,
  BM_JPU_MIRROR_VER = 1,
  BM_JPU_MIRROR_HOR = 2,
```

(continues on next page)

(continued from previous page)

```
    BM_JPU_MIRROR_HOR_VER = 3
} BmJpuMirrorDirection;
```

- **BM_JPU_MIRROR_NONE**

  No rotation.

- **BM_JPU_MIRROR_VER**

  Vertical mirroring.

- **BM_JPU_MIRROR_HOR**

  Horizontal mirroring.

- **BM_JPU_MIRROR_HOR_VER**

  Vertical and horizontal mirroring.

BmJpuFramebuffer

This structure is used to describe a frame of YUV data in BMAPI.

BmJpuFramebuffer The structure is defined as follows:

```
typedef struct {
    unsigned int y_stride;
    unsigned int cbcr_stride;
    bm_device_mem_t *dma_buffer;
    size_t y_offset;
    size_t cb_offset;
    size_t cr_offset;

    void *context;
    int already_marked;
    void *internal;
} BmJpuFramebuffer;
```

- **y_stride**

  The step size of the lum (Y) component.

- **cbcr_stride**

  The step size of the chrominance (Cb, Cr) components.

- **dma_buffer**

  A block of device memory used to store YUV data, allocated by bmlib.

- **y_offset**

  The offset of the starting address of the Y component relative to the physical address in dma_buffer.

- **cb_offset**

  The offset of the starting address of the Cb component relative to the physical address in dma_buffer.

- **cr_offset**

  The offset of the Cr component starting address relative to the physical address in dma_buffer.

- **context**

  Used to save decoding context information.

- **already_marked**

  Mark whether the current frame can be output.

- **internal**

  Internal data, users decide how to use it.

BmJpuFramebufferSizes

This structure is used to record the framebuffer information after alignment

BmJpuFramebufferSizes The structure is defined as follows:

```
typedef struct{
  unsigned int aligned_frame_width, aligned_frame_height;
  unsigned int y_stride, cbcr_stride;
  unsigned int y_size, cbcr_size;
  unsigned int total_size;
  BmJpuImageFormat image_format;
} BmJpuFramebufferSizes;
```

- **aligned_frame_width**

  The width after alignment.

- **aligned_frame_height**

  The height after alignment.

- **y_stride**

  The step size of the luma(Y)component.

- **cbcr_stride**

  The step size of the chrominance(Cb, Cr)components.

- **y_size**

  The total data size of the Y component, in byte.

- **cbcr_size**

  The total data size of Cb and Cr components, unit: byte.

- **total_size**

  The total data amount of the framebuffer, unit: byte.

- **image_format**

  Image format, refer to BmJpuImageFormat.

BmJpuEncodedFrame

Encoded jpeg information. (not used)

BmJpuEncodedFrame The structure is defined as follows:

```
typedef struct{
  uint8_t *data;
  size_t data_size;
  void *acquired_handle;
  void *context;
  uint64_t pts, dts;
} BmJpuEncodedFrame;
```

- **data**

  Encoded jpeg data.

- **data_size**

  The length of encoded jpeg data.

- **acquired_handle**

  When bm_jpu_jpeg_enc_encode, the user-defined acquire_output_buffer function pointer.

- **context**

  User-defined context , It will not be used in dynamic libraries.

- **pts, dts**

  User-defined timestamp.

BmJpuRawFrame

Information of the original yuv image used for encoding.

BmJpuRawFrame The structure is defined as follows:

```
typedef struct{
  BmJpuFramebuffer *framebuffer;
  void *context;
  uint64_t pts, dts;
} BmJpuRawFrame;
```

- **BmJpuFramebuffer**

  For encoding yuv frames, refer to BmJpuFramebuffer.

- **context**

  User-defined context , It will not be used in dynamic libraries.

- **pts, dts**

  User-defined timestamp.

BmJpuDecOpenParams

This structure is used to define the parameters for opening the decoder, in other words, to set the properties of the decoder(the size of the received image, the size of the decoding buffer, etc.).

BmJpuDecOpenParams The structure is defined as follows:

```
typedef struct
{
  /* These are necessary with some formats which do not store the width
  * and height in the bitstream. If the format does store them, these
  * values can be set to zero. */
  unsigned int min_frame_width;
  unsigned int min_frame_height;
  unsigned int max_frame_width;
  unsigned int max_frame_height;

  BmJpuColorFormat color_format;
  /* If this is 1, then Cb and Cr are interleaved in one shared chroma
  * plane, otherwise they are separated in their own planes.
  * See the BmJpuColorFormat documentation for the consequences of this. */
  int chroma_interleave;

  /* 0: no scaling; n(1-3): scale by 2^n; */
  unsigned int scale_ratio;

  /* The DMA buffer size for bitstream */
```

(continues on next page)

(continued from previous page)

```
    int bs_buffer_size;
#ifdef _WIN32
    uint8_t *buffer;
#else
    uint8_t *buffer __attribute__((deprecated));
#endif

    int device_index;

    int rotationEnable;
    BmJpuRotateAngle rotationAngle;
    int mirrorEnable;
    BmJpuMirrorDirection mirrorDirection;

    int roiEnable;
    int roiWidth;
    int roiHeight;
    int roiOffsetX;
    int roiOffsetY;

    bool framebuffer_recycle;
    size_t framebuffer_size;

    bool bitstream_from_user;
    bm_jpu_phys_addr_t bs_buffer_phys_addr;
    bool framebuffer_from_user;
    int framebuffer_num;
    bm_jpu_phys_addr_t *framebuffer_phys_addrs;

    int timeout;
    int timeout_count;
} BmJpuDecOpenParams;
```

- **min_frame_width**

  The minimum width supported by the decoder.

- **min_frame_height**

  The minimum height supported by the decoder.

- **max_frame_width**

  The maximum width supported by the decoder.

- **max_frame_height**

  The maximum height supported by the decoder.

- **color_format**

  The encoding format of the input image. (Deprecated by BM1688)

- **chroma_interleave**

Identification option for the storage method of chroma components, which can be interleaved or separated. (Deprecated by BM1688)

- **scale_ratio**

  Used to specify the scaling ratio when decoding a video. It determines whether the image is resized (i.e. scaled) during the decoding process. If the value is 0, it means no scaling; if the value is between 1 and 3 (inclusive), it means that the image is scaled by 2 to the power of n, where n is the value of scale_ratio.

- **bs_buffer_size**

  Indicates the size of the code stream buffer, which records the byte size required to store the input image. When the user applies for device memory for the bitstream externally, the buffer size needs to be 1024 aligned.

- **buffer**

  Used to store the specific content of the input image. (Deprecated by BM1688)

- **device_index**

  Decode device ID.

- **rotationEnable**

  Identification of whether to perform rotation operation, 0 means no rotation, 1 means rotation.

- **rotationAngle**

  Refer to BmJpuRotateAngle.

- **mirrorEnable**

  Identification of whether to perform mirroring operation, 0 means no mirroring, 1 means mirroring.

- **mirrorDirection**

  Refer to BmJpuMirrorDirection.

- **roiEnable**

  Whether to set ROI (region of interest).

- **roiWidth**

  The width of the ROI.

- **roiHeight**

  The height of the ROI.

- **roiOffsetX**

  The horizontal offset of ROI relative to upper left corner of image.

- **roiOffsetY**

  The vertical offset of ROI relative to upper left corner of image.
- **framebuffer_recycle**

  Whether to reuse framebuffer. (Deprecated by BM1688)
- **framebuffer_size**

  Set the size of the framebuffer, unit: byte.
- **bitstream_from_user**

  Whether the device memory of jpeg data is set by user.
- **bs_buffer_phys_addr**

  Device memory address of jpeg data set by user.
- **framebuffer_from_user**

  Whether the device memory of the decoded yuv image is set by user.
- **framebuffer_num**

  The number of framebuffer_num. (Deprecated by BM1688)
- **framebuffer_phys_addrs**

  The pointer of the device memory address of the decoded yuv image set by the user.
- **timeout**

  User sets the decoding timeout (milliseconds). If not set, the default value is 20000ms..
- **timeout_count**

  User sets the number of decoding timeout retries. (Deprecated by BM1688)

BmJpuDecoder

This structure defines the decoder inside BMAPI, including decoder handle, device ID, framebuffer allocated for decoding, and other information.

BmJpuDecoder The structure is defined as follows:

```
struct _BmJpuDecoder
{
  unsigned int device_index;
  DecHandle handle;
  bm_device_mem_t *bs_dma_buffer;
  uint8_t *bs_virt_addr;
  uint64_t bs_phys_addr;
  int chroma_interleave;
```

(continues on next page)

(continued from previous page)

```
    int scale_ratio;
    unsigned int old_jpeg_width;
    unsigned int old_jpeg_height;
    BmJpuColorFormat old_jpeg_color_format;
    unsigned int num_framebuffers, num_used_framebuffers;
    FrameBuffer *internal_framebuffers;
    BmJpuFramebuffer *framebuffers;
    BmJpuDecFrameEntry *frame_entries;
    DecInitialInfo initial_info;
    int initial_info_available;
    DecOutputInfo dec_output_info;
    int available_decoded_frame_idx;
    bm_jpu_dec_new_initial_info_callback initial_info_callback;
    void *callback_user_data;
    int framebuffer_recycle;
    int channel_id;
    FramebufferList *fb_list_head;
    FramebufferList *fb_list_curr;
    int timeout;
};
typedef struct _BmJpuDecoder BmJpuDecoder;
```

- **device_index**

  Decode device ID.

- **handle**

  The decoder handle.

- **bs_dma_buffer**

  A piece of device memory used to store the code stream to be decoded, allocated by bmlib.

- **bs_virt_addr**

  The virtual address that stores the code stream to be decoded.

- **bs_phys_addr**

  Store the physical address of the code stream to be decoded.

- **chroma_interleave**

  Indicates whether the chroma components are arranged crosswise.

- **scale_ratio**

  Indicates the image zoom ratio. This parameter controls both the horizontal and vertical zoom levels of the image.

- **old_jpeg_width**

  Indicates the width of the decoded image of the previous frame.

- **old_jpeg_height**

  Represents the height of the decoded image of the previous frame.
- **old_jpeg_color_format**

  Indicates the encoding format of the decoded image of the previous frame.
- **num_framebuffers**

  Indicates the number of decoding buffers that have been allocated.
- **num_used_framebuffers**

  Indicates the number of decoding buffers used.
- **internal_framebuffers**

  The decoder is internally registered to the JPU framebuffer for interacting with the JPU.
- **framebuffers**

  The entire framebuffer allocated internally by the decoder.
- **frame_entries**

  The decoder’ s internal decoding buffer control structure specifies the current frame’ s PTS (presentation time), DTS (decoding time), and usage status.
- **initial_info**

  Used to record some initial configuration of the decoder.
- **initial_info_available**

  Used to indicate whether the decoder has completed initialization.
- **dec_output_info**

  Used to record the output information of the decoder.
- **available_decoded_frame_idx**

  Indicates the decoding frame number currently in use.
- **initial_info_callback**

  A callback function used to initialize the framebuffer. The framebuffer will be reallocated when the decoder input code stream changes.
- **callback_user_data**

  Used to store user-defined data passed in the above callback function.
- **framebuffer_recycle**

  A flag used to indicate whether the framebuffer is reused. In the case of multiplexing, the same decoder instance can be used to decode code streams of different resolutions and formats.

- **channel_id**

  Indicates the decoder channel ID.

- **fb_list_head**

  Indicates the head node in the framebuffer linked list. The decoder uses a linked list to store the currently used framebuffers, which will be released when a flush or close operation is performed. You can also use the **bm_jpu_jpeg_dec_frame_finished** interface to release a frame.

- **fb_list_curr**

  Indicates the current node in the framebuffer linked list. Used to add newly generated framebuffer to the linked list.

- **timeout**

  Timeout set by the user.

BmJpuJPEGDecoder

This structure defines the externally provided decoder, including decoder handle, device ID, framebuffer allocated for decoding and other information.

BmJpuJPEGDecoder The structure is defined as follows:

```
typedef struct _BMJpuJPEGDecoder
{
  BmJpuDecoder *decoder;

  bm_jpu_phys_addr_t bitstream_buffer_addr;
  size_t bitstream_buffer_size;
  unsigned int bitstream_buffer_alignment;

  BmJpuDecInitialInfo initial_info;

  BmJpuFramebuffer *framebuffers;
  bm_jpu_phys_addr_t *framebuffer_addrs;
  unsigned int num_framebuffers;
  unsigned int num_extra_framebuffers;
  BmJpuFramebufferSizes calculated_sizes;

  BmJpuRawFrame raw_frame;
  int device_index;

  void *opaque;

  int rotationEnable;
  BmJpuRotateAngle rotationAngle;
  int mirrorEnable;
  BmJpuMirrorDirection mirrorDirection;
```

(continues on next page)

(continued from previous page)

```
    bool framebuffer_recycle;
    bool bitstream_from_user;
    bool framebuffer_from_user;

}BmJpuJPEGDecoder;
```

- **decoder**

  Decoder defined inside BMAPI.

- **bitstream_buffer_addr**

  The user applies externally to store a piece of device memory for the code stream to be decoded. (Effective in BmJpuDecOpenParams, deprecated by BM1688)

- **bitstream_buffer_size**

  Indicates the memory size of the above code stream, unit: byte. (Effective in BmJpuDecOpenParams, deprecated by BM1688)

- **bitstream_buffer_alignment**

  Indicates the alignment requirement of the above code stream, unit: byte. (Deprecated by BM1688)

- **initial_info**

  Used to record some initial configuration of the decoder.

- **framebuffers**

  Used to record the address and size of the framebuffer in the decoder. (Deprecated by BM1688)

- **framebuffer_addrs**

  Device memory requested by the user for storing the framebuffer in the decoder. (Effective in BmJpuDecOpenParams, deprecated by BM1688)

- **num_framebuffers**

  The total number of framebuffer frames required for decoding.

- **num_extra_framebuffers**

  The number of extra frames in the framebuffer required for decoding, usually 0.

- **calculated_sizes**

  Record the framebuffer size information after alignment.

- **raw_frame**

  Represents the original frame data, used to store the original data and timestamp of the image. (Deprecated by BM1688)

- **device_index**

  Indicates decoding device ID.

- **opaque**

  Undefined data, it is up to the user to decide how to use it.

- **rotationEnable**

  Indicates whether to perform rotation operation. 0 means no rotation and 1 means rotation.

- **rotationAngle**

  Refer to BmJpuRotateAngle.

- **mirrorEnable**

  Indicates whether to perform mirroring operation. 0 means no mirroring and 1 means mirroring.

- **mirrorDirection**

  Refer to BmJpuMirrorDirection.

- **framebuffer_recycle**

  A flag used to indicate whether the framebuffer is reused. In the case of multiplexing, the same decoder instance can be used to decode streams with different resolutions and formats.

- **bitstream_from_user**

  The user applies for device memory for jpeg data. (Effective in BmJpuDecOpenParams, deprecated by BM1688)

- **framebuffer_from_user**

  The user applies for the device memory of the decoded yuv image. (Effective in BmJpuDecOpenParams, deprecated by BM1688)

BmJpuJPEGDecInfo

This structure records the decoded YUV data information and is used by users to obtain decoded data.

BmJpuJPEGDecInfo The structure is defined as follows:

```
typedef struct
{
  /* Width and height of JPU framebuffers are aligned to internal boundaries.
   * The frame consists of the actual image pixels and extra padding pixels.
   * aligned_frame_width / aligned_frame_height specify the full width/height
   * including the padding pixels, and actual_frame_width / actual_frame_height
```

(continues on next page)

(continued from previous page)

```
    * specify the width/height without padding pixels. */
    unsigned int aligned_frame_width, aligned_frame_height;
    unsigned int actual_frame_width, actual_frame_height;

    /* Stride and size of the Y, Cr, and Cb planes. The Cr and Cb planes always
    * have the same stride and size. */
    unsigned int y_stride, cbcr_stride;
    unsigned int y_size, cbcr_size;

    /* Offset from the start of a framebuffer's memory, in bytes. Note that the
    * Cb and Cr offset values are *not* the same, unlike the stride and size ones. */
    unsigned int y_offset, cb_offset, cr_offset;

    /* Framebuffer containing the pixels of the decoded frame. */
    BmJpuFramebuffer *framebuffer;

    BmJpuImageFormat image_format;

    bool framebuffer_recycle;
    size_t framebuffer_size;
} BmJpuJPEGDecInfo;
```

- **aligned_frame_width**

  The width of the frame data after alignment.

- **aligned_frame_height**

  The height of the frame data after alignment.

- **actual_frame_width**

  The width of the original image.

- **actual_frame_height**

  The height of the original image.

- **y_stride**

  The step size of the Y component.

- **cbcr_stride**

  The step size of Cb and Cr components.

- **y_size**

  The total data amount of Y component, unit: byte.

- **cbcr_size**

  The total data size of Cb and Cr components, unit: byte.

- **y_offset**

The offset of the Y component starting address relative to the physical address in the framebuffer.

- **cb_offset**

  The offset of the Cb component start address relative to the physical address in the framebuffer.

- **cr_offset**

  The offset of the Cr component start address relative to the physical address in the framebuffer.

- **framebuffer**

  Used to record information related to decoded YUV data.

- **image_format**

  Image format.

- **framebuffer_recycle**

  A flag used to indicate whether the framebuffer is reused. In the case of multiplexing, the same decoder instance can be used to decode streams with different resolutions and formats.

- **framebuffer_size**

  The framebuffer memory size that needs to be applied for in framebuffer_recycle mode, unit: byte.

BmJpuEncoder

This structure defines the encoder inside BMAPI, including encoder handle, device ID, output bitstream and other information.

BmJpuEncoder The structure is defined as follows:

```
typedef struct _BmJpuEncoder
{
    unsigned int device_index;
    EncHandle handle;

    bm_device_mem_t *bs_dma_buffer;
    uint8_t *bs_virt_addr;

    BmJpuColorFormat color_format;
    unsigned int frame_width, frame_height;

    BmJpuFramebuffer *framebuffers;

    int channel_id;
```

(continues on next page)

(continued from previous page)

```
    int timeout;
} BmJpuEncoder;
```

- **device_index**

Encoder device ID.

- **handle**

Encoder handle.

- **bs_dma_buffer**

A piece of device memory used to store the output code stream, allocated by bmlib.

- **bs_virt_addr**

Store the virtual address of the output code stream.

- **color_format**

The encoding format of the output code stream.

- **frame_width**

The width of the output code stream.

- **frame_height**

The height of the output code stream.

- **framebuffers**

The framebuffer used inside the encoder.

- **channel_id**

Internal channel id.

- **timeout**

Encoder timeout.

BmJpuJPEGEncoder

This structure defines the encoder provided to the outside, including encoder handle, device ID, output code stream and other information.

BmJpuJPEGEncoder The structure is defined as follows:

```
typedef struct _BmJpuJPEGEncoder
{
    BmJpuEncoder *encoder;

    bm_jpu_phys_addr_t bitstream_buffer_addr;
```

(continues on next page)

(continued from previous page)

```
    size_t bitstream_buffer_size;
    unsigned int bitstream_buffer_alignment;

    BmJpuEncInitialInfo initial_info;

    unsigned int frame_width, frame_height;

    BmJpuFramebufferSizes calculated_sizes;

    unsigned int quality_factor;

    BmJpuImageFormat image_format;

    int device_index;

    int rotationEnable;
    BmJpuRotateAngle rotationAngle;
    int mirrorEnable;
    BmJpuMirrorDirection mirrorDirection;

    bool bitstream_from_user;
} BmJpuJPEGEncoder;
```

- **encoder**

Encoder defined inside BMAPI.

- **bitstream_buffer_addr**

Device memory address of jpeg data requested externally by the user.

- **bitstream_buffer_size**

Size of jpeg data requested externally by the user.

- **bitstream_buffer_alignment**

Indicates the alignment requirement of the above bitstream, unit: byte.

- **initial_info**

Used to record the initial configuration of the encoder’ s framebuffer.

- **frame_width**

Input image width.

- **frame_height**

Input image height.

- **calculated_sizes**

Records the aligned framebuffer size information.

- **quality_factor**

Encoding quality.

- **image_format**

Input image encoding format.

- **device_index**

Indicates the encoding device ID.

- **rotationEnable**

Flag to indicate whether to perform rotation operation, 0 means no rotation, 1 means rotation.

- **rotationAngle**

Refer to BmJpuRotateAngle.

- **mirrorEnable**

Flag to indicate whether to perform mirror operation, 0 means no mirror, 1 means mirror.

- **mirrorDirection**

Refer to BmJpuMirrorDirection.

- **bitstream_from_user**

Whether the device memory of jpeg data is requested by the user externally. (Abandoned by BM1688)

BmJpuJPEGEncParams

This structure defines the encoding configuration parameters and the configurable interface function for obtaining output data.

BmJpuJPEGEncParams The structure is defined as follows:

```
typedef struct
{
   /* Frame width and height of the input frame. These are the actual sizes;
   * they will be aligned internally if necessary. These sizes must not be
   * zero. */
   unsigned int frame_width, frame_height;

   /* Quality factor for JPEG encoding. 1 = best compression, 100 = best quality.
   * This is the exact same quality factor as used by libjpeg. */
   unsigned int quality_factor;

   /* Image format of the input frame. */
   BmJpuImageFormat image_format;

   /* Functions for acquiring and finishing output buffers. See the
```

(continues on next page)

(continued from previous page)

```
    * typedef documentations in bmjpuapi.h for details about how
    * these functions should behave. */
    BmJpuEncAcquireOutputBuffer acquire_output_buffer;
    BmJpuEncFinishOutputBuffer finish_output_buffer;

    /* Function for directly passing the output data to the user
    * without copying it first.
    * Using this function will inhibit calls to acquire_output_buffer
    * and finish_output_buffer. */
    BmJpuWriteOutputData write_output_data;

    /* User supplied value that will be passed to the functions:
    * acquire_output_buffer, finish_output_buffer, write_output_data */
    void *output_buffer_context;

    int rotationEnable;
    BmJpuRotateAngle rotationAngle;
    int mirrorEnable;
    BmJpuMirrorDirection mirrorDirection;

    /* Identify the output data is in device memory or system memory */
    int bs_in_device;

    int timeout;
    int timeout_count;

    /* Optional: User supplied device memory for following encode,
     * will replace the bitstream buffer in encoder */
    bm_jpu_phys_addr_t bs_buffer_phys_addr;
    int bs_buffer_size;
} BmJpuJPEGEncParams;
```

- **frame_width**

Output code stream width.

- **frame_height**

Output code stream height.

- **quality_factor**

Coding quality, optional 1 (highest compression rate) ~ 100 (best image quality).

- **image_format**

Output image encoding format.

- **acquire_output_buffer**

Callback function used to obtain the output buffer of the encoded code stream.

- **finish_output_buffer**

Callback function used to release the above buffer.

- **write_output_data**

Callback function used to specify the output method of the encoded code stream, such as: write to a file or write to a specified memory address, mutually exclusive with the above two interfaces.

- **output_buffer_context**

Context used to save output data.

- **rotationEnable**

Flag for whether to perform rotation operation, 0 means no rotation, 1 means rotation.

- **rotationAngle**

Refer to BmJpuRotateAngle.

- **mirrorEnable**

Whether to perform mirroring operation, 0 means no mirroring, 1 means mirroring.

- **mirrorDirection**

Refer to BmJpuMirrorDirection.

- **bs_in_device**

The encoded jpeg data is output to the device memory. (BM1688 deprecated)

- **timeout**

The user sets the encoder timeout (ms), the default is 20000ms.

- **timeout_count**

The user sets the number of timeout retries. (BM1688 deprecated)

- **bs_buffer_phys_addr**

The user sets the device memory address of the jpeg data.

- **bs_buffer_size**

The user sets the size of the jpeg data.

1.5.3 JPEG interface description

bm_jpu_dec_load

This interface opens the specified decoding device node according to the passed ID, and can manage memory allocation through bmlib.

Interface form:

```
BmJpuDecReturnCodes bm_jpu_dec_load(int device_index);
```

Parameter description:

- **device_index**

Decoding device ID.

Return value description:

- BM_JPU_DEC_RETURN_CODE_OK: Success
- Others: Failure

bm_jpu_dec_unload

This interface releases the specified decoding device node.

Interface format:

```
BmJpuDecReturnCodes bm_jpu_dec_unload(int device_index);
```

Parameter description:

- **device_index**

Decoding device ID.

Return value description:

- BM_JPU_DEC_RETURN_CODE_OK: Success
- Others: Failure

bm_jpu_jpeg_dec_open

This interface opens a decoder and configures the decoding channel according to the passed parameters.

Interface type:

```
BmJpuDecReturnCodes bm_jpu_jpeg_dec_open(BmJpuJPEGDecoder **jpeg_
↪decoder,
                    BmJpuDecOpenParams *open_params,
                    unsigned int num_extra_framebuffers)
```

Parameter description:

- **jpeg_decoder**

Secondary pointer to the decoder, initialized inside the interface.

- **open_params**

Configuration parameters when opening the decoder.

- **num_extra_framebuffers**

The number of framebuffers that need to be applied for additionally. (Abandoned by BM1688)

Return value description:

- BM_JPU_DEC_RETURN_CODE_OK: Success
- Others: Failure

bm_jpu_jpeg_dec_close

This interface is used to close the decoder and release resources.

Interface form:

```
BmJpuDecReturnCodes bm_jpu_jpeg_dec_close(BmJpuJPEGDecoder *jpeg_
↪decoder);
```

Parameter description:

- **jpeg_decoder**

An already opened decoder.

Return value description:

- BM_JPU_DEC_RETURN_CODE_OK: Success
- Others: Failure

bm_jpu_jpeg_dec_decode

This interface performs decoding operations.

Interface form:

```
BmJpuDecReturnCodes bm_jpu_jpeg_dec_decode(
  BmJpuJPEGDecoder *jpeg_decoder,
  uint8_t const *jpeg_data,
  size_t const jpeg_data_size
  int timeout,
  int timeout_count);
```

Parameter description:

- **jpeg_decoder**

An already opened decoder.

- **jpeg_data**

Image data to be decoded.

- **jpeg_data_size**

Image data size to be decoded.

- **timeout**

Decoder timeout.

- **timeout_count**

Number of decoder timeout retries. (Abandoned by BM1688)

Return value description:

- BM_JPU_DEC_RETURN_CODE_OK: Success
- Others: Failure

bm_jpu_jpeg_dec_get_info

This interface obtains decoding information from the decoder.

Interface form:

```
void bm_jpu_jpeg_dec_get_info(
  BmJpuJPEGDecoder *jpeg_decoder,
  BmJpuJPEGDecInfo *info);
```

Parameter description:

- **jpeg_decoder**

An already opened decoder.

- **info**

A data structure used to store decoding information.

Return value description:

- None

bm_jpu_jpeg_dec_frame_finished

This interface releases a framebuffer after decoding.

Interface form:

```
BmJpuDecReturnCodes bm_jpu_jpeg_dec_frame_finished(
  BmJpuJPEGDecoder *jpeg_decoder,
  BmJpuFramebuffer *framebuffer);
```

Parameter description:

- **jpeg_decoder**

An already opened decoder.

- **framebuffer**

A framebuffer that has been decoded.

Return value description:

- BM_JPU_DEC_RETURN_CODE_OK: Success
- Others: Failure

bm_jpu_jpeg_dec_flush

This interface is used to refresh the decoder and release all decoded framebuffers.

Interface form:

```
BmJpuDecReturnCodes bm_jpu_jpeg_dec_flush(BmJpuJPEGDecoder *jpeg_
↪decoder);
```

Parameter description:

- **jpeg_decoder**

An already opened decoder.

Return value description:

- BM_JPU_DEC_RETURN_CODE_OK: Success
- Others: Failure

bm_jpu_enc_load

This interface opens the specified encoding device node according to the passed ID, and can manage memory allocation through bmlib.

Interface form:

```
BmJpuEncReturnCodes bm_jpu_enc_load(int device_index);
```

Parameter description:

- **device_index**

Encoding device ID.

Return value description:

- BM_JPU_ENC_RETURN_CODE_OK: Success
- Others: Failure

bm_jpu_enc_unload

This interface releases the specified encoding device node.

Interface format:

```
BmJpuEncReturnCodes bm_jpu_enc_unload(int device_index);
```

Parameter description:

- **device_index**

Encoding device ID.

Return value description:

- BM_JPU_ENC_RETURN_CODE_OK: Success
- Others: Failure

bm_jpu_jpeg_enc_open

This interface opens an encoder and applies for a bitstream buffer of a specified size.

Interface format:

```
BmJpuEncReturnCodes bm_jpu_jpeg_enc_open(
    BmJpuJPEGEncoder **jpeg_encoder,
    bm_jpu_phys_addr_t bs_buffer_phys_addr,
    int bs_buffer_size,
    int device_index
    );
```

Parameter description:

- **jpeg_encoder**

Secondary pointer to the encoder, initialized inside the interface.

- **bs_buffer_phys_addr**

Memory address of the bitstream buffer device requested externally by the user.

- **bs_buffer_size**

The size of the bistream buffer. If the input is 0, 5MB is requested by default.

- **device_index**

Encoding device ID.

Return value description:

- BM_JPU_ENC_RETURN_CODE_OK: Success
- Others: Failure

bm_jpu_jpeg_enc_close

This interface is used to close the encoder and release resources.

Interface form:

```
BmJpuEncReturnCodes bm_jpu_jpeg_enc_close(BmJpuJPEGEncoder *jpeg_
↪encoder);
```

Parameter description:

- **jpeg_encoder**

An already opened encoder

Return value description:

- BM_JPU_ENC_RETURN_CODE_OK: Success
- Others: Failure

bm_jpu_jpeg_enc_encode

This interface performs encoding operations.

Interface format:

```
BmJpuEncReturnCodes bm_jpu_jpeg_enc_encode(
    BmJpuJPEGEncoder *jpeg_encoder,
    BmJpuFramebuffer const *framebuffer,
    BmJpuJPEGEncParams const *params,
    void **acquired_handle,
    size_t *output_buffer_size
    );
```

Parameter description:

- **jpeg_encoder**

An already opened encoder.

- **framebuffer**

Input frame data.

- **params**

Encoding related parameters.

- **acquired_handle**

The location for storing encoded data, specified by the user. If NULL, it is output through the write_output_data interface.

- **output_buffer_size**

The size of the output data buffer, in bytes.

Return value description:

- BM_JPU_ENC_RETURN_CODE_OK: Success
- Others: Failure

acquire_output_buffer

This interface is used to obtain buffer receiving encoded data.

Interface form:

```
void* acquire_output_buffer(void *context, unsigned int size, void **acquired_
↪handle);
```

Parameter description:

- **context**

Output buffer context.

- **size**

Output buffer size, unit: byte.

- **acquired_handle**

Output buffer start address.

finish_output_buffer

This interface is used to release the buffer acquired by the above interface.

Interface format:

```
void finish_output_buffer(void *context, void *acquired_handle);
```

Parameter Description:

- **context**

Output buffer context.

- **acquired_handle**

Output buffer start address.

1.5.4 JPEG test case description

bmjpegdec

For code, please refer to example/jpeg_dec_test.c

Parameter Description

```
usage:  bmjpegdec [option]
option:
    -i input_file
    -o output_file
    -n loop_num
    -c crop function(0:disable  1:enable crop)
    -g rotate (default 0) [rotate mode[1:0]  0:No rotate  1:90  2:180  3:270] [rotator␣
↪mode[2]:vertical flip] [rotator mode[3]:horizontal flip]
    -s scale (default 0) -> 0 to 3
    -r roi_x,roi_y,roi_w,roi_h
```

Single-pass decoding

```

```

bmjpegdec -i JPEG_1920x1088_yuv420_planar.jpg -o out_1920x1088_yuv420_planar.yuv -n 1

Single-pass loop decoding

```

```

bmjpegdec -i JPEG_1920x1088_yuv420_planar.jpg -o out_1920x1088_yuv420_planar.yuv -n 10

bmjpegenc

For code, please refer to example/jpeg_enc_test.c

Parameter description

```
usage:  bmjpegenc [option]
option:
    -f pixel format: 0.YUV420(default); 1.YUV422; 2.YUV444; 3.YUV400. (optional)
    -w actual width
    -h actual height
    -y luma stride (optional)
    -c chroma stride (optional)
    -v aligned height (optional)
    -i input file
    -o output file
    -n loop num
```

(continues on next page)

(continued from previous page)

```
    -g rotate (default 0) [rotate mode[1:0]  0:No rotate  1:90  2:180  3:270] [rotator␣
↪mode[2]:vertical flip] [rotator mode[3]:horizontal flip]
```

Single-channel encoding

```
bmjpegenc -f 0 -w 1920 -h 1088 -i JPEG_1920x1088_yuv420_planar.yuv -o JPEG_
↪1920x1088_yuv420_planar.jpg -n 1
```

Single-pass loop encoding

```
bmjpegenc -f 0 -w 1920 -h 1088 -i JPEG_1920x1088_yuv420_planar.yuv -o JPEG_
↪1920x1088_yuv420_planar.jpg -n 10000
```

Rotate

```
bmjpegenc -f 0 -w 1920 -h 1088 -i JPEG_1920x1088_yuv420_planar.yuv -o JPEG_
↪1920x1088_yuv420_planar.jpg -n 1 -g 1
```

Mirroring

```
bmjpegenc -f 0 -w 1920 -h 1088 -i JPEG_1920x1088_yuv420_planar.yuv -o JPEG_
↪1920x1088_yuv420_planar.jpg -n 1 -g 4
```

bmjpegmulti

Please refer to example/jpeg_multi_test.c for the code.

32-channel decoding

```
First, write the configuration file multi.lst,
the content is as follows: (the first line indicates the number of channels and the␣
↪number of cycles for each channel, and each subsequent line indicates the␣
↪configuration of each channel)
 32 100
 JPEG_352x288_yuv420_planar.jpg 32 1 0 0
 JPEG_352x288_yuv420_planar.jpg 32 1 0 0
 ...(Repeat until line 32)

Then execute bmjpegmulti -f multi.lst
option input 30
```

32-channel decoding

```
First, write the configuration file enc.cfg, the content is as follows:

 YUV_SRC_IMG JPEG_352x288_yuv420_planar.yuv
 FRAME_FORMAT 0
 PICTURE_WIDTH 352
```

(continues on next page)

(continued from previous page)

```
 PICTURE_HEIGHT 288
 IMG_FORMAT 0

Then write the configuration file multi.lst,
the content is as follows: (the first line indicates the number of channels and the␣
↪number of cycles for each channel, and each subsequent line indicates the␣
↪configuration of each channel)

 32 1000000
 enc.cfg 32 0 0 0
 enc.cfg 32 0 0 0
 ...(Repeat until line 32)

Then execute bmjpegmulti -f multi.lst
option input 30
```

Single-channel YUV400 encoding

```
bmjpegmulti -t 1 -i JPEG_1920x1088_yuv400_planar.yuv -o OUT400.jpg -w 1920 -
↪h 1088 -s 4 -f 0 -n 4
```

Single-channel YUV420P encoding

```
bmjpegmulti -t 1 -i JPEG_1920x1088_yuv420_planar.yuv -o OUT420_planar.jpg -
↪w 1920 -h 1088 -s 0 -f 0 -n 4
```

Single-channel YUV422P encoding

```
bmjpegmulti -t 1 -i JPEG_1920x1088_yuv422_planar.yuv -o OUT422_planar.jpg -
↪w 1920 -h 1088 -s 1 -f 0 -n 4
```

1.6 SOPHGO Video Decoder Usage Guidelines

1.6.1 Introduction

Overview

The VDEC module provides the corresponding interface to drive the video decoding hardware to realize the video decoding function.

The BM1688 VDEC module supports H.264 and H.265 decoding, and supports decoding 16 channels of 1080P video at 30fps at the same time.

Definitions and Abbreviations

abbreviation	meaning
VPU	Video Processing Unit
VDEC	Video Decoder
core	core
BitStream	Input stream data
Frame	Frame
Buffer	Buffer
channel	channel

1.6.2 VDEC Data Type Introduction

BMVidDecRetStatus

Defines the decoder error return value type.

Enumeration definition

```
typedef enum
{
   BM_ERR_VDEC_INVALID_CHNID = -27,
   BM_ERR_VDEC_ILLEGAL_PARAM,
   BM_ERR_VDEC_EXIST,
   BM_ERR_VDEC_UNEXIST,
   BM_ERR_VDEC_NULL_PTR,
   BM_ERR_VDEC_NOT_CONFIG,
   BM_ERR_VDEC_NOT_SUPPORT,
   BM_ERR_VDEC_NOT_PERM,
   BM_ERR_VDEC_INVALID_PIPEID,
   BM_ERR_VDEC_INVALID_GRPID,
   BM_ERR_VDEC_NOMEM,
   BM_ERR_VDEC_NOBUF,
   BM_ERR_VDEC_BUF_EMPTY,
   BM_ERR_VDEC_BUF_FULL,
   BM_ERR_VDEC_SYS_NOTREADY,
   BM_ERR_VDEC_BADADDR,
   BM_ERR_VDEC_BUSY,
   BM_ERR_VDEC_SIZE_NOT_ENOUGH,
   BM_ERR_VDEC_INVALID_VB,
   BM_ERR_VDEC_ERR_INIT,
   BM_ERR_VDEC_ERR_INVALID_RET,
   BM_ERR_VDEC_ERR_SEQ_OPER,
   BM_ERR_VDEC_ERR_VDEC_MUTEX,
   BM_ERR_VDEC_ERR_SEND_FAILED,
   BM_ERR_VDEC_ERR_GET_FAILED,
   BM_ERR_VDEC_BUTT,
   BM_ERR_VDEC_FAILURE
}BMVidDecRetStatus;
```

Parameter meaning

- **BM_ERR_VDEC_INVALID_CHNID**: Invalid channel id.
- **BM_ERR_VDEC_ILLEGAL_PARAM**: Illegal parameter.
- **BM_ERR_VDEC_NULL_PTR**: Null pointer.
- **BM_ERR_VDEC_NOBUF**: Invalid buffer.
- **BM_ERR_VDEC_BUF_FULL**: Buffer empty.
- **BM_ERR_VDEC_BUF_FULL**: Buffer full.
- **BM_ERR_VDEC_BUSY**: Decoder busy.
- **BM_ERR_VDEC_FAILURE**: General decoder error.

BmVpuDecStreamFormat

Code stream format.

Enumeration definition

```
typedef enum{
  BMDEC_AVC      = 0,
  BMDEC_HEVC     = 12,
}BmVpuDecStreamFormat;
```

Parameter meaning

- **BMDEC_AVC**: AVC bitstream.
- **BMDEC_HEVC**: HEVC bitstream.

BmVpuDecSkipMode

Set frame skipping mode

Enumeration definition

```
typedef enum {
  BMDEC_FRAME_SKIP_MODE        = 0,
  BMDEC_SKIP_NON_REF_NON_I   = 1,
  BMDEC_SKIP_NON_I           = 2,
}BmVpuDecSkipMode;
```

Parameter meaning

- **BMDEC_FRAME_SKIP_MODE**: Disable frame skip mode.
- **BMDEC_SKIP_NON_REF_NON_I**: Enable frame skip mode and skip video frames except reference frames and I frames

- **BMDEC_SKIP_NON_I**: Enable frame skip mode and skip video frames except I frames

BmVpuDecDMABuffer

Stores VPU buffer information

Structure definition

```
typedef struct {
  unsigned int  size;
  u64       phys_addr;
  u64       virt_addr;
} BmVpuDecDMABuffer;
```

Parameter meaning

- **size**: The size of the buffer.
- **phys_addr**: The physical address of the buffer.
- **phys_addr**: The virtual address of the buffer.

BmVpuDecOutputMapType

Set the type of output data.

Enumeration definition

```
typedef enum {
  BMDEC_OUTPUT_UNMAP,
  BMDEC_OUTPUT_TILED = 100,
  BMDEC_OUTPUT_COMPRESSED,
} BmVpuDecOutputMapType;
```

Parameter meaning

- **BMDEC_OUTPUT_UNMAP**: Output yuv data.
- **BMDEC_OUTPUT_COMPRESSED**: Output compressed mode data.

BmVpuDecBitStreamMode

Set VPU decoding mode

Enumeration definition

```
typedef enum {
  BMDEC_BS_MODE_INTERRUPT = 0,
  BMDEC_BS_MODE_RESERVED,
  BMDEC_BS_MODE_PIC_END = 2,
} BmVpuDecBitStreamMode;
```

Parameter meaning

- **BMDEC_BS_MODE_INTERRUPT**: Use stream mode decoding, and send it to the decoder when the input buffer is full.
- **BMDEC_BS_MODE_PIC_END**: Use frame mode decoding, and send it to the decoder as soon as a frame of data is obtained. It is necessary to parse the code stream in advance.

BmVpuDecPixFormat

Set the format of output data

Enumeration definition

```
typedef enum
{
  BM_VPU_DEC_PIX_FORMAT_YUV420P = 0,
  BM_VPU_DEC_PIX_FORMAT_YUV422P = 1,
  BM_VPU_DEC_PIX_FORMAT_YUV444P = 3,
  BM_VPU_DEC_PIX_FORMAT_YUV400  = 4,
  BM_VPU_DEC_PIX_FORMAT_NV12    = 5,
  BM_VPU_DEC_PIX_FORMAT_NV21    = 6,
  BM_VPU_DEC_PIX_FORMAT_NV16    = 7,
  BM_VPU_DEC_PIX_FORMAT_NV24    = 8,
  BM_VPU_DEC_PIX_FORMAT_COMPRESSED = 9,
  BM_VPU_DEC_PIX_FORMAT_COMPRESSED_10BITS = 10,
} BmVpuDecPixFormat;
```

Parameter meaning

- **BM_VPU_DEC_PIX_FORMAT_YUV420P**: Output YUV420P data
- **BM_VPU_DEC_PIX_FORMAT_YUV422P**: Output YUV422P data, BM1688 does not support
- **BM_VPU_DEC_PIX_FORMAT_YUV444P**: Output YUV444P data, BM1688 does not support

- **BM_VPU_DEC_PIX_FORMAT_YUV400**: Output YUV400 data, BM1688 does not support
- **BM_VPU_DEC_PIX_FORMAT_NV12**: Output NV12 data
- **BM_VPU_DEC_PIX_FORMAT_NV21**: Output NV21 data, BM1688 Not supported
- **BM_VPU_DEC_PIX_FORMAT_NV16**: Output NV16 data, BM1688 does not support
- **BM_VPU_DEC_PIX_FORMAT_NV24**: Output NV24 data, BM1688 does not support
- **BM_VPU_DEC_PIX_FORMAT_COMPRESSED**: Output compressed format data
- **BM_VPU_DEC_PIX_FORMAT_COMPRESSED_10BITS**: Output 10bits compressed format data, BM1688 does not support

BMDecStatus

Used to indicate the status of the decoder.

Enumeration definition

```
typedef enum {
  BMDEC_UNCREATE,
  BMDEC_UNLOADING,
  BMDEC_UNINIT,
  BMDEC_INITING,
  BMDEC_WRONG_RESOLUTION,
  BMDEC_FRAMEBUFFER_NOTENOUGH,
  BMDEC_DECODING,
  BMDEC_FRAME_BUF_FULL,
  BMDEC_ENDOF,
  BMDEC_STOP,
  BMDEC_HUNG,
  BMDEC_CLOSE,
  BMDEC_CLOSED,
} BMDecStatus;
```

Parameter meaning

Currently only the following states are valid:

- **BMDEC_UNINIT**: The decoder is not initialized
- **BMDEC_INITING**: The decoder is initializing
- **BMDEC_WRONG_RESOLUTION**: The set resolution does not match
- **BMDEC_FRAMEBUFFER_NOTENOUGH**: The allocated Frame Buffer is insufficient
- **BMDEC_DECODING**: The decoder is decoding
- **BMDEC_STOP**: The decoder has finished decoding

BMVidDecParam

BMVidDecParam is used to set the initialization parameters of the decoder. Before calling the interface bmvpu_dec_create, you need to create a BMVidDecParam object and initialize it.

Structure definition

```
typedef struct {
   BmVpuDecStreamFormat      streamFormat;
   BmVpuDecOutputMapType     wtlFormat;
   BmVpuDecSkipMode          skip_mode;
   BmVpuDecBitStreamMode     bsMode;
   int                       enableCrop;
   BmVpuDecPixFormat         pixel_format;
   int                       secondaryAXI;
   int                       mp4class;
   int                       frameDelay;
   int                       pcie_board_id;
   int                       pcie_no_copyback;
   int                       enable_cache;
   int                       perf;
   int                       core_idx;
   int                       cmd_queue_depth;
   int                       decode_order;
   int                       picWidth;
   int                       picHeight;
   int                       timeout;
   int                       timeout_count;
   int                       extraFrameBufferNum;
   int                       min_framebuf_cnt;
   int                       framebuf_delay;
   int                       streamBufferSize;
   BmVpuDecDMABuffer*        bitstream_buffer;
   BmVpuDecDMABuffer*        frame_buffer;
   BmVpuDecDMABuffer*        Ytable_buffer;
   BmVpuDecDMABuffer*        Ctable_buffer;
   int                       reserved[13];
} BMVidDecParam;
```

Parameter meaning

- **streamFormat**:

Set the input bitstream type, 0 is H.264 (AVC), 12 is H.265 (HEVC).

- **wtlFormat**:

Set the output data type.

If set to 0, the corresponding yuv data is output according to the yuv type;

If set to 101, the compressed fbc data is output.

- **skip_mode**:

Set the frame skipping mode.

- **bsMode**:

Set the decoder working mode.

- **pixel_format**:

Output image format.

- **secondaryAXI**:

This parameter is no longer required in BM1688.

The SDK will automatically select whether to enable the secondary AXI function based on the bitstream type.

- **decode_order**

Set the decoder frame output order. 0, output frames in display order; 1, output frames in decoding order.

- **timeout**:

Decoding timeout.

- **timeout_count**:

The number of decoding timeout retries.

- **extraFrameBufferNum**:

Set the number of Frame Buffers.

- **min_framebuf_cnt**:

The minimum number of Frame Buffers required for the input code stream.

- **framebuf_delay**:

The number of Frame Buffers required for decoding delayed frames.

- **streamBufferSize**:

Set the buffer size of the input code stream.

If set to 0, the default buffer size is 0x700000.

- **bitstream_buffer**:

Input code stream buffer information.

- **frame_buffer**:

Frame Buffer buffer information.

- **Ytable_buffer**:

Compression mode Y table buffer information.

- **Ctable_buffer**:

Compression mode C table buffer information.

BMVidStream

Pass the code stream data to the decoder through the BMVidStream object.

Structure definition

```
typedef struct BMVidStream {
  unsigned char* buf;
  unsigned int   length;
  unsigned char* header_buf;
  unsigned int   header_size;
  unsigned char* extradata;
  unsigned int   extradata_size;
  unsigned char  end_of_stream;
  u64            pts;
  u64            dts;
} BMVidStream;
```

Parameter meaning

- **buf, length**:

The address and size of the BitStream Buffer.

- **end_of_stream**:

The end of the bitstream. When the bitstream is read, this flag needs to be set to 1.

- **pts, dts**:

Timestamp

- In BM1688, header and extradata data are no longer accepted. In order to be consistent with the previous product, the above variables still exist.

BMVidFrame

BMVidFrame is used to receive frame information output by the decoder.

Structure definition

```
typedef struct BMVidFrame {
  BmVpuDecPicType    picType;
  unsigned char*     buf[8];
  int                stride[8];
  unsigned int       width;
  unsigned int       height;
  BmVpuDecLaceFrame  interlacedFrame;
  int                lumaBitDepth;
```

(continues on next page)

(continued from previous page)

```
    int             chromaBitDepth;
    BmVpuDecPixFormat   pixel_format;
    int             endian;
    int             sequenceNo;
    int             frameIdx;
    u64             pts;
    u64             dts;
    int             colorPrimaries;
    int             colorTransferCharacteristic;
    int             colorSpace;
    int             colorRange;
    int             chromaLocation;
    int             size;
    unsigned int        coded_width;
    unsigned int        coded_height;
} BMVidFrame;
```

Parameter meaning

- **picType**:

Indicates the type of the current Frame. The corresponding relationship is as follows:

Table 1.2: picType corresponding relationship

picType	Type
0	I picture
1	P picture
2	B picture
4	IDR picture

- **buf:**
  The address where the output data is stored. The corresponding meanings of each channel are as follows:

Table 1.3: buf corresponding relationship

channel	YUV420P	NV12/NV21	FBC
0	Y component virtual address	Y component virtual address	/
1	Cb component virtual address	CbCr component virtual address	/
2	Cr component virtual address	/	/
3	/	/	/
4	Y component physical address	Y component physical address	Y component physical address
5	Cb component physical address	CbCr component physical address	Cb component physical address
6	Cr component physical address	/	Y table component physical address
7	/	/	Cb table component physical address

- **stride:**

  Corresponding to buf, it stores the width of the corresponding channel, which is the width after alignment.

  For FBC data, stride stores slightly different data.

  Channels 0 and 4 store the width of the Y component;

  Channels 1 and 5 store the width of the Cb component;

  Channels 2 and 6 store the length of the Y table;

  Channels 3 and 7 store the length of the Cb table.

- **width:**

  Stores the width of the Frame.

- **height:**

  Stores the height of the Frame.

- **frameFormat:**

  Stores the format of the image.

- **interlacedFrame:**

  Image scanning method.

- **lumaBitDepth:**

  Depth of luminance data.

- **chromaBitDepth:**

  Depth of chrominance data.

- **cbcrInterleave:**
  Indicates the storage method of chrominance components.

  If cbcrInterleave is 0, Cb and Cr components are stored in different memory spaces;

  If cbcrInterleave is 1, Cb and Cr components are stored in the same memory space.

- **nv21:**
  Indicates the storage format of YUV data, which is only valid when cbcrInterleave=1.

  nv21=0, stored in Nv12 format;

  nv21=1, stored in NV21 format.

- **endian:**
  Indicates the segment order of the frame buffer.

  endian=0, stored in little-endian mode;

  endian=1, stored in big-endian mode;

  endian=2, stored in 32-bit little-endian mode;

  endian=3, stored in 32-bit big-endian mode.

- **sequenceNo:**
  Indicates the state of the code stream sequence. When the code stream sequence changes, the value of sequenceNo will be accumulated.

- **frameIdx:**
  The index of the image frame buffer. Used to indicate the position of the frame buffer in the decoder.

- **pts:**
  Display timestamp.

- **dts:**
  Decoding timestamp.

- **size:**
  The size of the frame buffer.

- **coded_width:**
  The width of the picture used for display.

- **coded_height:**
  The height of the picture used for display.

- **compressed_mode:**
  Indicates the format of the decoder output data.

CropRect

CropRect is used to save the cropping information of the image.

Structure definition

```
typedef struct {
  unsigned int left;
  unsigned int top;
  unsigned int right;
  unsigned int bottom;
} CropRect;
```

Parameter meaning

- **left:**
  The horizontal offset of the upper left corner of the cropping box relative to the pixel origin.
- **top:**
  The vertical offset of the upper left corner of the cropping box relative to the pixel origin.
- **right:**
  The horizontal offset of the lower right corner of the cropping box relative to the pixel origin.
- **bottom:**
  The vertical offset of the lower right corner of the cropping box relative to the pixel origin.

BMVidStreamInfo

BMVidStreamInfo is used to receive the information of input code stream.

Structure definition

```
typedef struct BMVidStreamInfo {
  int       picWidth;
  int       picHeight;
  int       fRateNumerator;
  int       fRateDenominator;
  CropRect     picCropRect;
  int       mp4DataPartitionEnable;
  int       mp4ReversibleVlcEnable;
  int       mp4ShortVideoHeader;
  int       h263AnnexJEnable;
  int       minFrameBufferCount;
  int       frameBufDelay;
  int       normalSliceSize;
```

(continues on next page)

(continued from previous page)

```
  int       worstSliceSize;
  int       maxSubLayers;
  int       profile;
  int       level;
  int       tier;
  int       interlace;
  int       constraint_set_flag[4];
  int       direct8x8Flag;
  int       vc1Psf;
  int       isExtSAR;
  int       maxNumRefFrmFlag;
  int       maxNumRefFrm;
  int       aspectRateInfo;
  int       bitRate;
  int       mp2LowDelay;
  int       mp2DispVerSize;
  int       mp2DispHorSize;
  unsigned int userDataHeader;
  int       userDataNum;
  int       userDataSize;
  int       userDataBufFull;
  int       chromaFormatIDC;
  int       lumaBitdepth;
  int       chromaBitdepth;
  int       seqInitErrReason;
  int       warnInfo;
  unsigned int sequenceNo;
} BMVidStreamInfo;
```

Parameter meaning

- **picWidth:**
  Picture width.
- **picHeight:**
  Picture height.
- **fRateNumerator:**
  Frame rate fraction numerator.
- **fRateDenominator:**
  Frame rate fraction denominator.
- **picCropRect:**
  Cropping information, for details, please refer to CropRect.
- **minFrameBufferCount:**
  The minimum number of frame buffers required by the decoder.
- **frameBufDelay:**
  The maximum display frame buffer delay used to buffer decoded image re-ordering.

- **profile:**
  H.264/H.265 profile index.
- **level:**
  H.264/H.265 level index.
- **interlace:**
  When interlace=1, the bitstream is decoded as progressive frames or interlaced frames; otherwise, it is decoded as progressive frames.
- **bitRate:**
  The bit rate when the BitStream is written.
- **lumaBitdepth:**
  The depth of the luminance data.
- **chromaBitdepth:**
  The depth of the chrominance data.
- **sequenceNo:**
  Indicates the state of the bitstream sequence. When the bitstream sequence changes, the value of sequenceNo will be accumulated.

1.6.3 VDEC API Introduction

bmvpu_dec_create

bmvpu_dec_create Used to create a decoder channel.

Interface form

```
BMVidDecRetStatus bmvpu_dec_create (
  BMVidCodHandle* pVidCodHandle,
  BMVidDecParam decParam );
```

Parameter Description

- **pVidCodHandle:**
  Output parameters. Decoder handle. When the decoder is successfully created, a handle will be returned. The handle can be used to perform subsequent operations on the decoder.
- **decParam:**
  Input parameters. Decoder initialization parameter.

Return value

When the decoder is successfully created, 0 will be returned, otherwise the corresponding error code will be returned.

bmvpu_dec_decode

Use bmvpu_dec_decode to send the BitStream to VDEC for decoding. BM1688 supports two decoding modes, INTERRUPT and PIC_END.

> **INTERRUPT mode** is to send fixed-size bitstream data each time according to the pre-set BitStream Buffer size, and there is no concept of frames.
>
> **PIC_END mode** is to pre-parse the position of a frame of data according to the H.264/H.265 protocol, and only transmit the BitStream corresponding to a frame of data to the decoder each time. Therefore, when creating a decoder, it is necessary to reasonably consider the value of streamBufferSize.

Interface form

```
BMVidDecRetStatus bmvpu_dec_decode (
  BMVidCodHandle vidCodHandle,
  BMVidStream vidStream );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.
- **vidStream:**
  Input parameters. BitStream address and size.

Return value

When the bitstream is successfully sent to the decoder, it will return 0, otherwise it will return the corresponding error code. Returning an error does not mean that the decoder is working abnormally. If BM_ERR_VDEC_BUF_FULL is returned, you need to check the decoder status and try to send the BitStream again.

bmvpu_dec_get_output

bmvpu_dec_get_output This interface is used to obtain the output data of the decoder.

Interface format

```
BMVidDecRetStatus bmvpu_dec_get_output (
  BMVidCodHandle vidCodHandle,
  BMVidFrame *bmFrame );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.

- **bmFrame:**
  Output parameters. Used to receive the output data of the decoder.

Return value

When the decoded data is successfully obtained, 0 will be returned, otherwise the corresponding error code will be returned. Returning an error code does not mean that the decoder is working abnormally. The following situations may cause the error code to be returned:

1. No BitStream data is input;
2. Decoding is not completed and the output data is not ready;
3. The decoder is closed;
4. The decoder is abnormal.

bmvpu_dec_clear_output

When the Frame Buffer is no longer used, you need to call bmvpu_dec_clear_output to release it. Otherwise, the decoder may work abnormally due to insufficient Frame Buffer.

Interface form

```
BMVidDecRetStatus bmvpu_dec_clear_output (
  BMVidCodHandle vidCodHandle,
  BMVidFrame *frame );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.
- **frame:**
  Input parameters. The address of the bmFrame object to be released.

Return Value

When the frame is released successfully, 0 will be returned, otherwise the corresponding error code will be returned.

bmvpu_dec_flush

When all BitStreams are sent to the decoder, it does not mean that the decoding is complete. You still need to wait for the decoder to output all the frames. Use bmvpu_dec_flush to flush the remaining frames.

Interface form

```
BMVidDecRetStatus bmvpu_dec_flush (
  BMVidCodHandle vidCodHandle );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.

Return Value

When all Frames are taken out, 0 will be returned.

bmvpu_dec_get_all_frame_in_buffer

Also used to get the remaining frame data in the decoder after the code stream is transmitted. The difference from bmvpu_dec_flush is that bmvpu_dec_get_all_frame_in_buffer will not block.

Interface form

```
BMVidDecRetStatus bmvpu_dec_get_all_frame_in_buffer (
  BMVidCodHandle vidCodHandle );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.

Return value

When all the decoders are taken out of the Frame, 0 will be returned.

bmvpu_dec_delete

When the decoding task is completed, call bmvpu_dec_delete to destroy the decoder and release the resources occupied by the decoder

Interface form

```
BMVidDecRetStatus bmvpu_dec_delete (
  BMVidCodHandle vidCodHandle );
```

Parameter Description

- **vidCodHandle:**
  Input parameter. Decoder handle. You need to create a decoder before using this interface.

Return value

After the decoder is destroyed successfully, it will return 0, otherwise it will return the corresponding error code.

bmvpu_dec_get_caps

bmvpu_dec_get_caps is used to obtain decoder information, mainly the bitstream information sent to the decoder. The information that can be obtained can refer to the definition of the BMVidStreamInfo structure.

Interface form

```
BMVidDecRetStatus bmvpu_dec_get_caps (
  BMVidCodHandle vidCodHandle,
  BMVidStreamInfo *streamInfo );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.
- **streamInfo:**
  Output parameters. Receive decoder information.

Return value

If the decoder information is successfully obtained, 0 is returned, otherwise the corresponding error code is returned.

bmvpu_dec_get_status

bmvpu_dec_get_status is used to obtain the working status of the decoder. The currently supported decoder status can refer to the definition of BMDec-Status.

Interface form

```
BMDecStatus bmvpu_dec_get_status (
  BMVidCodHandle vidCodHandle );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.

Return Value

If the decoder status is successfully obtained, the corresponding status will be returned, otherwise an error code will be returned.

bmvpu_dec_get_all_empty_input_buf_cnt

Used to obtain the number of idle BitStream Buffers.

Interface Form

```
int bmvpu_dec_get_all_empty_input_buf_cnt (
  BMVidCodHandle vidCodHandle );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.

Return Value

If the number of BitStreams is successfully obtained, the number of free buffers is returned, otherwise an error code is returned.

bmvpu_dec_get_stream_buffer_empty_size

Used to obtain the size of the unused space in the BitStream Buffer.

Interface Form

```
int bmvpu_dec_get_stream_buffer_empty_size (
  BMVidCodHandle vidCodHandle );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.

Return Value

If successful, it returns the size of the unused space in the BitStream Buffer, otherwise it returns an error code.

bmvpu_dec_get_pkt_in_buf_cnt

Used to obtain the number of occupied BitStream Buffers.

Interface Form

```
int bmvpu_dec_get_pkt_in_buf_cnt (
  BMVidCodHandle vidCodHandle );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.

Return value

If the number of BitStreams is successfully obtained, the number of occupied Buffers is returned, otherwise an error code is returned.

bmvpu_dec_dump_stream

Used to save input stream data. Currently, only INTERRUPT mode data is supported. If it is PIC_END mode, the saved data may not be playable (some non-display frame information is missing). This interface has been written into bmvpu_dec_decode. This function can be enabled by setting the environment variable BMVPU_DEC_DUMP_NUM.

Interface form

```
BMVidDecRetStatus bmvpu_dec_dump_stream (
  BMVidCodHandle vidCodHandle,
  BMVidStream vidStream );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.
- **vidStream:**
  Input parameters. BitStream address and size.

Return value

If the code stream is saved successfully, it returns 0, otherwise it returns an error code.

bmvpu_dec_get_core_idx

Used to get the core id of the VPU. For BM1688, there are two decoding cores, core 0 and core 1.

Interface form

```
int bmvpu_dec_get_core_idx (
  BMVidCodHandle handle );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.

Return value

If successful, it returns the core id of the VPU, otherwise it returns an error code.

bmvpu_dec_get_inst_idx

Used to obtain the channel id of VDEC. For BM1688, each core can apply for up to 32 channels.

Interface form

```
int bmvpu_dec_get_inst_idx (
  BMVidCodHandle vidCodHandle );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.

Return Value

If successful, returns the channel id of VDEC, otherwise returns an error code.

bmvpu_dec_get_device_fd

Used to obtain the device node id corresponding to the VDEC channel.

Interface form

```
int bmvpu_dec_get_device_fd (
  BMVidCodHandle vidCodHandle );
```

Parameter Description

- **vidCodHandle:**
  Input parameters. Decoder handle. You need to create a decoder before using this interface.

Return Value

If successful, it returns the id of the device node, otherwise it returns an error code.

1.6.4 VDEC API test routine

A dedicated test routine bm_test is provided for VDEC. The VDEC API is called to decode the local H.264/H.265 bitstream file.

bm_test provides the following configurable parameters

- **-h**
  Check the prompt information to see the optional parameters of bm_test and the corresponding parameter descriptions.
- **-v**
  Set the LOG level. 0: none; 1: error; 2: warning; 3: info; 4: trace. The default value is 1.
- **-c**
  Set the output verification method. 0: No output verification; 1: Directly compare yuv data; 2: Compare the md5 value of the output yuv.

  Before turning on output verification, you need to prepare the reference file in advance and set the comparison frequency through –comp-skip. The default value is 0.

- **-n**
  Set the number of decoding rounds. The default value is 1.
- **-m**
  Set the decoding mode. 0: Set to INTERRUPT mode; 2: Set to PIC_END mode.
- **–input**
  Set the input file.
- ****–cbcr_interleave ****
  Set whether Cb and Cr are interleaved, 0 means not interleaved, 1 means interleaved.
- **–nv21**
  Only valid when –cbcr_interleave is set to 1. Set to 0 to output in NV12 format; set to 1 to output in NV21 format.
- **–stream-type**
  Set the input stream type. 0: H.264; 12: H.265.
- **–ref-yuv**
  Set the reference file path. When output verification is enabled, this parameter must be set, otherwise output verification will not be performed.
- **–instance**
  Set the number of decoding threads.
- **–write_yuv**
  Set the number of output frames. This parameter is only used to set the number of frames that need to be written to the file, not the number of decoded frames.
- **–wtl-format**
  Set the output mode. 0: Output YUV; 1: Output FBC.
- **–read-block-len**
  Set the input buffer size. The default value is 0x80000.

Example 1

Use INTERRUPT mode to decode the H.264 code stream and output it in yuv420p format.

```
bm_test -c 0 -n 1 -m 0 --input wkc_h264_100.264 --cbcr_interleave 0 --instance 1
```

Example 2

Use PIC_END mode to decode H.265 code stream and output it in yuv420p format.

```
bm_test -c 0 -n 1 -m 2 --stream-type 12 --input wkc_h265_100.265 --cbcr_interleave 0 --
↪instance 1
```

Example 3

Use INTERRUPT mode to decode H.264 code stream, use nv12 format for output, and start 4 decoding threads.

```
bm_test -c 0 -n 1 -m 0 --input wkc_h264_100.264 --cbcr_interleave 1 --nv21 0 --instance 4
```

Example 4

Use INTERRUPT mode to decode H.264 code stream, use nv12 format for output, and enable yuv comparison.

```
bm_test -c 1 -n 1 -m 0 --input wkc_h264_100.264 --cbcr_interleave 1 --nv21 0 \
--ref-yuv wkc_h264_100.yuv --instance 1
```

Example 5

Use PIC_END mode to decode H.265 code stream, use nv12 format for output, enable md5 comparison, and write the first 10 frames to the file.

```
bm_test -c 1 -n 1 -m 2 --stream-type 12 --input wkc_h265_100.265 --cbcr_interleave 0 \
--ref-yuv wkc_h265_100.yuv --instance 1 --write_yuv 10
```

1.7 SOPHGO Video Encoder User Guide

1.7.1 Introduction

The Encoder module mainly includes video encoding. This module includes all interfaces open to users and detailed explanations of their parameters

1.7.2 Data structure description

1. BmVpuEncReturnCodes

```
typedef enum
{
  BM_VPU_ENC_RETURN_CODE_OK = 0,
  BM_VPU_ENC_RETURN_CODE_ERROR,
  BM_VPU_ENC_RETURN_CODE_INVALID_PARAMS,
  BM_VPU_ENC_RETURN_CODE_INVALID_HANDLE,
  BM_VPU_ENC_RETURN_CODE_INVALID_FRAMEBUFFER,
  BM_VPU_ENC_RETURN_CODE_INSUFFICIENT_FRAMEBUFFERS,
  BM_VPU_ENC_RETURN_CODE_INVALID_STRIDE,
  BM_VPU_ENC_RETURN_CODE_WRONG_CALL_SEQUENCE,
  BM_VPU_ENC_RETURN_CODE_TIMEOUT,
  BM_VPU_ENC_RETURN_CODE_RESEND_FRAME,
  BM_VPU_ENC_RETURN_CODE_ENC_END,
  BM_VPU_ENC_RETURN_CODE_END
} BmVpuEncInitialInfo;
```

Parameter Description

- BM_VPU_ENC_RETURN_CODE_OK

The operation was completed successfully

- BM_VPU_ENC_RETURN_CODE_ERROR

Generic error code, used as a generic error when other error return codes do not match

- BM_VPU_ENC_RETURN_CODE_INVALID_PARAMS

Invalid input parameters

- BM_VPU_ENC_RETURN_CODE_INVALID_HANDLE

Invalid VPU encoder handle, internal error, may be a library error, please report such errors

- BM_VPU_ENC_RETURN_CODE_INVALID_FRAMEBUFFER

Invalid frame buffer information, usually occurs when a BmVpuFramebuffer structure containing invalid values is passed to the bmvpu_enc_register_framebuffers() function

- BM_VPU_ENC_RETURN_CODE_INSUFFICIENT_FRAMEBUFFERS

Failed to register framebuffers for encoding because not enough framebuffers were provided to the bmvpu_enc_register_framebuffers() function

- BM_VPU_ENC_RETURN_CODE_INVALID_STRIDE

Invalid stride value, for example, one of the stride values of the framebuffer is invalid

- BM_VPU_ENC_RETURN_CODE_WRONG_CALL_SEQUENCE

Function called at an inappropriate time

- BM_VPU_ENC_RETURN_CODE_TIMEOUT

Operation timed out

- BM_VPU_ENC_RETURN_CODE_RESEND_FRAME

Repeated frame delivery

- BM_VPU_ENC_RETURN_CODE_ENC_END

Encoding ends

- BM_VPU_ENC_RETURN_CODE_END

Encoding ends

2. BmVpuEncOutputCodes

```
typedef enum
{
  BM_VPU_ENC_OUTPUT_CODE_INPUT_USED = 1 << 0,
  BM_VPU_ENC_OUTPUT_CODE_ENCODED_FRAME_AVAILABLE = 1 << 1,
  BM_VPU_ENC_OUTPUT_CODE_CONTAINS_HEADER = 1 << 2
} BmVpuEncOutputCodes;
```

Parameter Description

- BM_VPU_ENC_OUTPUT_CODE_INPUT_USED

Indicates that the input data has been used. If this flag is not set, the encoder has not used the input data yet, so feed it to the encoder again until this flag is set or an error is returned

- BM_VPU_ENC_OUTPUT_CODE_ENCODED_FRAME_AVAILABLE

Indicates that a fully encoded frame is now available. The encoded_frame parameter passed to bmvpu_enc_encode() contains information about this frame

- BM_VPU_ENC_OUTPUT_CODE_CONTAINS_HEADER

Indicates that the data in the encoded frame also contains header information, such as SPS/PSS for h.264. The header information is always placed at the beginning of the encoded data, and this flag will not be set if BM_VPU_ENC_OUTPUT_CODE_ENCODED_FRAME_AVAILABLE is not set

3. BmVpuEncHeaderDataTypes

```
typedef enum
{
  BM_VPU_ENC_HEADER_DATA_TYPE_VPS_RBSP = 0,
  BM_VPU_ENC_HEADER_DATA_TYPE_SPS_RBSP,
  BM_VPU_ENC_HEADER_DATA_TYPE_PPS_RBSP
} BmVpuEncHeaderDataTypes;
```

Parameter Description

- BM_VPU_ENC_HEADER_DATA_TYPE_VPS_RBSP

RBSP (Raw Byte Sequence Payload) data type of Video Parameter Set (VPS)

- BM_VPU_ENC_HEADER_DATA_TYPE_SPS_RBSP

RBSP data type of Sequence Parameter Set (SPS)

- BM_VPU_ENC_HEADER_DATA_TYPE_PPS_RBSP

RBSP data type of Picture Parameter Set (PPS)

4. BmVpuCodecFormat

```
typedef enum
{
  BM_VPU_CODEC_FORMAT_H264 = 0,
  BM_VPU_CODEC_FORMAT_H265
} BmVpuCodecFormat;
```

Parameter Description

- BM_VPU_CODEC_FORMAT_H264

Encoding type h264

- BM_VPU_CODEC_FORMAT_H265

Encoding type h265

5. BmVpuEncPixFormat

```
typedef enum
{
  BM_VPU_ENC_PIX_FORMAT_YUV420P = 0,
  BM_VPU_ENC_PIX_FORMAT_YUV422P  = 1,
  BM_VPU_ENC_PIX_FORMAT_YUV444P  = 3,
  BM_VPU_ENC_PIX_FORMAT_YUV400 = 4,
  BM_VPU_ENC_PIX_FORMAT_NV12  = 5,
```

(continues on next page)

(continued from previous page)

```
    BM_VPU_ENC_PIX_FORMAT_NV16 = 6,
    BM_VPU_ENC_PIX_FORMAT_NV24 = 7,
    BM_VPU_ENC_PIX_FORMAT_NV21 = 8
} BmVpuEncPixFormat;
```

Parameter Description

- Encoder input yuv format

  Currently only supports nv12, nv21, yuv420p

6. BMVpuEncGopPreset

```
typedef enum
{
    BM_VPU_ENC_GOP_PRESET_ALL_I = 1,
    BM_VPU_ENC_GOP_PRESET_IPP   = 2,
    BM_VPU_ENC_GOP_PRESET_IBBB  = 3,
    BM_VPU_ENC_GOP_PRESET_IBPBP = 4,
    BM_VPU_ENC_GOP_PRESET_IBBBP  = 5,
    BM_VPU_ENC_GOP_PRESET_IPPPP = 6,
    BM_VPU_ENC_GOP_PRESET_IBBBB = 7,
    BM_VPU_ENC_GOP_PRESET_RA_IB = 8
} BMVpuEncGopPreset;
```

Parameter Description

- BM_VPU_ENC_GOP_PRESET_ALL_I

  All I frame mode gopsize=1

  - BM_VPU_ENC_GOP_PRESET_IPP

    All IP frame mode gopsize=1

    - BM_VPU_ENC_GOP_PRESET_IBBB

    All IB frame mode gopsize=1

    - BM_VPU_ENC_GOP_PRESET_IBPBP

    All IBP frame mode gopsize=2

    - BM_VPU_ENC_GOP_PRESET_IBBBP

    All IBP frame mode gopsize=4

    - BM_VPU_ENC_GOP_PRESET_IPPPP

    All IP frame mode gopsize=4

    - BM_VPU_ENC_GOP_PRESET_IBBBB

    All IB frame mode gopsize=4

- BM_VPU_ENC_GOP_PRESET_RA_IB

  Random IB frame mode gopsize=8

7. BMVpuEncMode

```
typedef enum
{
    BM_VPU_ENC_CUSTOM_MODE = 0,
    BM_VPU_ENC_RECOMMENDED_MODE  = 1,
    BM_VPU_ENC_BOOST_MODE  = 2,
    BM_VPU_ENC_FAST_MODE = 3
} BMVpuEncMode;
```

Parameter Description

- BM_VPU_ENC_CUSTOM_MODE

  Custom mode

  - BM_VPU_ENC_RECOMMENDED_MODE

  Recommended mode (slow encoding speed, highest quality)

  - BM_VPU_ENC_BOOST_MODE

  Boost mode (normal encoding speed, normal quality)

  - BM_VPU_ENC_FAST_MODE

  Fast mode (high encoding speed, low quality)

8. BmVpuMappingFlags

```
typedef enum
{
    BM_VPU_MAPPING_FLAG_WRITE = 1 << 0,
    BM_VPU_MAPPING_FLAG_READ  = 1 << 1
} BmVpuMappingFlags;
```

Parameter Description

- BM_VPU_MAPPING_FLAG_WRITE

  Writable permission flag

  - BM_VPU_MAPPING_FLAG_READ

  Readable permission flag

9. BmVpuEncH264Params

```
typedef struct
{
  int enable_transform8x8;
} BmVpuEncH264Params;
```

参数说明

- enable_transform8x8

  Enable 8x8 intra prediction and 8x8 transform. Default value is 1

10. BmVpuEncH265Params

```
typedef struct
{
  int enable_tmvp;
  int enable_wpp;
  int enable_sao;
  int enable_strong_intra_smoothing;
  int enable_intra_trans_skip;
  int enable_intraNxN;
} BmVpuEncH265Params;
```

Parameter Description

- enable_tmvp

Enable temporal motion vector prediction. Default value is 1

- enable_wpp

Enable wavefront parallel processing for linear buffer mode. Default value is 0

- enable_sao

Enable SAO if set to 1; disable if set to 0. Default value is 1

- enable_strong_intra_smoothing

Enable strong intra-frame smoothing for areas with a small number of AC coefficients to prevent artifacts. Default value is 1

- enable_intra_trans_skip

Enable transform skipping for intra CUs. Default value is 0

- enable_intraNxN

Enable intra NxN PUs. Default value is 1

11. BmVpuEncOpenParams

```
typedef struct
{
    BmVpuCodecFormat codec_format;
    BmVpuEncPixFormat color_format;
    uint32_t frame_width;
    uint32_t frame_height;
    uint32_t timebase_num;
    uint32_t timebase_den;
    uint32_t fps_num;
    uint32_t fps_den;
    int64_t  bitrate;
    uint64_t vbv_buffer_size;
    int cqp;
    BMVpuEncMode enc_mode;
    int max_num_merge;
    int enable_constrained_intra_prediction;
    int enable_wp;
    int disable_deblocking;
    int offset_tc;
    int offset_beta;
    int enable_cross_slice_boundary;
    int enable_nr;
    union
    {
        BmVpuEncH264Params h264_params;
        BmVpuEncH265Params h265_params;
    };
    int soc_idx;
    BMVpuEncGopPreset gop_preset;
    int intra_period;
    int bg_detection;
    int mb_rc;
    int delta_qp;

    /* minimum QP for rate control
    * Default value is 8 */
    int min_qp;
    /* maximum QP for rate control
    * Default value is 51 */
    int max_qp;

    /* roi encoding flag
    * Default value is 0 */
    int roi_enable;
    /* set cmd queue depth
      * Default value is 4
      * the value must 1 <= value <= 4*/
    int cmd_queue_depth;
```

(continues on next page)

(continued from previous page)

```
    int timeout;
    int timeout_count;

    BmVpuEncBufferAllocFunc buffer_alloc_func;
    BmVpuEncBufferFreeFunc buffer_free_func;
    void *buffer_context;
} BmVpuEncOpenParams;
```

Parameter Description

- BmVpuCodecFormat codec_format

  Specifies the encoding format of the encoded data to be generated

- BmVpuEncPixFormat color_format

  Specifies the image format used by the incoming frame

- uint32_t frame_width

  The width of the incoming frame in pixels, no alignment required

- uint32_t frame_height

  The height of the incoming frame in pixels, no alignment required

- uint32_t timebase_num

  The time base, given as a fraction

- uint32_t timebase_den

  The time denominator, given as a fraction

- uint32_t fps_num

  The frame rate, given as a fraction

- uint32_t timebase_den

  The frame rate denominator, given as a fraction

- int64_t bitrate

  The bit rate in bps. If set to 0, rate control is disabled and constant quality mode is used. The default value is 100000

- int cqp

  Quantization parameter for constant quality mode

- int enc_mode

  0: Custom mode

  1: Recommended encoder parameters (slow encoding speed, highest quality)

  2: Boost mode (normal encoding speed, normal quality)

3: Fast mode (high encoding speed, low quality)

- int max_num_merge

  Number of merge candidates in RDO (1 or 2). 1: Improve encoding performance, 2: Improve the quality of the encoded image. The default value is 2

- int enable_constrained_intra_prediction

  Enable constrained intra prediction. If set to 1, it is enabled; if set to 0, it is disabled. The default value is 0

- int enable_wp

  Enable weighted prediction. The default value is 1

- int disable_deblocking

  If set to 1, disable the deblocking filter. If set to 0, it remains enabled. Default value is 0

- int offset_tc

  Alpha/Tc offset of deblocking filter. Default value is 0

- int offset_beta

  Beta offset of deblocking filter. Default value is 0

- int enable_cross_slice_boundary

  Enable cross-slice boundary filtering in intra-loop filtering. Default value is 0

- int enable_nr

  Enable cross-slice boundary filtering in intra-loop filtering. Default value is 0

- int h264_params

  H.264 encoder parameters. (union, select one from h264_params and h265_params)

- int h265_params

  H.265 encoder parameters. (union, select one from h264_params and h265_params)

- int soc_idx

  For PCIe mode only. For SOC mode, this value must be 0. Default value is 0

- int gop_preset

  1: all I, all Intra, gopsize = 1

  2: I-P-P, consecutive P, cyclic gopsize = 1

  3: I-B-B-B, consecutive B, cyclic gopsize = 1

  4: I-B-P-B-P, gopsize = 2

5: I-B-B-B-P, gopsize = 4

6: I-P-P-P-P, consecutive P, cyclic gopsize = 4

7: I-B-B-B-B, consecutive B, cyclic gopsize = 4

8: Random Access, I-B-B-B-B-B-B-B-B, cyclic gopsize = 8

1, 2, 3, 6, 7 for low latency. Default is 5

- int intra_period

  Intra picture period within GOP size. Default is 28

- int bg_detection

  Enable background detection. Default is 0

- int mb_rc

  Enable MB/CU level rate control. Default is 1

- int delta_qp

  Maximum delta QP for rate control. Default is 5

- int min_qp

  Minimum QP for rate control. Default is 8

- int max_qp

  Maximum QP for rate control. Default is 51

- int roi_enable

  ROI encoding flag. The default value is 0

- int cmd_queue_depth

  Encoding queue depth, which can improve encoder performance and consume a certain amount of physical memory

- int timeout

  Encoding timeout, the default is 1000ms (i.e. VPU_WAIT_TIMEOUT) (bm1688 is an asynchronous interface, no timeout setting is required)

- int timeout_count

  Number of encoding timeout retries, the default is 40 (i.e. VPU_MAX_TIMEOUT_COUNTS) (bm1688 is an asynchronous interface, no timeout_count setting is required)

- BmVpuEncBufferAllocFunc buffer_alloc_func

  Buffer memory allocation function interface

- BmVpuEncBufferFreeFunc buffer_free_func

  Buffer memory release function interface

- void* buffer_context

  Buffer context information

12. BmVpuEncInitialInfo

```
typedef struct
{
  uint32_t min_num_rec_fb;
  uint32_t min_num_src_fb;
  uint32_t framebuffer_alignment;
  BmVpuFbInfo rec_fb;
  BmVpuFbInfo src_fb;
} BmVpuEncInitialInfo;
```

Parameter Description

- min_num_src_fb

Minimum recommended frame buffer number. Allocating less than this number may affect encoding quality

- rec_fb

Minimum number of input YUV data frames. Allocating less than this number may affect encoding

- framebuffer_alignment

Alignment requirements for physical frame buffer addresses

- rec_fb

Frame buffer size information for reconstruction. Including width, height and other information

- src_fb

Input YUV data width and height information

13. BmCustomMapOpt

```
typedef struct
{
  int roiAvgQp;
  int customRoiMapEnable;
  int customLambdaMapEnable;
  int customModeMapEnable;
  int customCoefDropEnable;
  bmvpu_phys_addr_t addrCustomMap;
} BmCustomMapOpt;
```

Parameter Description

- roiAvgQp

Average QP of ROI mapping

- customRoiMapEnable

Whether to enable ROI mapping

- customLambdaMapEnable

Whether to enable Lambda mapping

- customModeMapEnable

Whether to specify CTU to use inter-frame coding, otherwise skip

- customCoefDropEnable

For each CTU, whether to set the TQ coefficient to all 0, and CTUs with all 0 coefficients will be discarded

- addrCustomMap

The starting address of the custom mapping buffer

14. BmVpuEncParams

```
typedef struct
{
   int skip_frame;
   int forcePicTypeEnable;
   int forcePicType;
   BmVpuEncAcquireOutputBuffer acquire_output_buffer;
   BmVpuEncFinishOutputBuffer finish_output_buffer;
   void* output_buffer_context;
   BmCustomMapOpt* customMapOpt;
} BmVpuEncParams;
```

Parameter Description

- skip_frame

The default value is 0, which disables skip frame generation. If set to 1, the VPU ignores the given original frame and generates a "skip frame" which is a copy of the previous frame. This skip frame is encoded as a P frame

- forcePicTypeEnable

Whether to force the specified encoding frame type

- forcePicType

Force the specified encoding frame type (I frame, P frame, B frame, IDR frame, CRA frame), only valid when forcePicTypeEnable is 1

- acquire_output_buffer

Function for acquiring output buffers

- finish_output_buffer

Function for releasing output buffers

- output_buffer_context

User-provided value passed to the above functions

- customMapOpt

Pointer to custom mapping options

15. BmVpuEncoder

```
/* BM VPU Encoder structure. */
typedef struct
{
  void* handle;

  int soc_idx; /* The index of Sophon SoC.
          * For PCIE mode, please refer to the number at /dev/bm-sophonxx.
          * For SOC mode, set it to zero. */
  int core_idx; /* unified index for vpu encoder cores at all Sophon SoCs */

  BmVpuCodecFormat codec_format;
  BmVpuEncPixFormat pix_format;

  uint32_t frame_width;
  uint32_t frame_height;

  uint32_t fps_n;
  uint32_t fps_d;

  int first_frame;

  int rc_enable;
  /* constant qp when rc is disabled */
  int cqp;

  /* DMA buffer for working */
  BmVpuEncDMABuffer*  work_dmabuffer;

  /* DMA buffer for bitstream */
  BmVpuEncDMABuffer* bs_dmabuffer;

  unsigned long long bs_virt_addr;
  bmvpu_phys_addr_t bs_phys_addr;
```

(continues on next page)

(continued from previous page)

```
    /* DMA buffer for frame data */
    uint32_t    num_framebuffers;
    void * /*VpuFrameBuffer**/  internal_framebuffers;
    BmVpuFramebuffer* framebuffers;

    /* TODO change all as the parameters of bmvpu_enc_register_framebuffers() */
    /* DMA buffer for colMV */
    BmVpuEncDMABuffer*  buffer_mv;

    /* DMA buffer for FBC luma table */
    BmVpuEncDMABuffer*  buffer_fbc_y_tbl;

    /* DMA buffer for FBC chroma table */
    BmVpuEncDMABuffer*  buffer_fbc_c_tbl;

    /* Sum-sampled DMA buffer for ME */
    BmVpuEncDMABuffer*  buffer_sub_sam;

    uint8_t* headers_rbsp;
    size_t  headers_rbsp_size;

    BmVpuEncInitialInfo initial_info;

    int timeout;
    int timeout_count;

    /* internal use */
    void *video_enc_ctx;
} BmVpuEncoder;
```

Parameter Description

- handle

  Encoder handle

- soc_idx

  Index of the Sophon SoC. For PCIE mode, refer to the number in /dev/bm-sophonxx. For SOC mode, set it to zero

- core_idx

  Unified index of the VPU encoder core in all Sophon SoCs

- codec_format

  Video codec format used by the encoder

- color_format

  Image format used by the incoming frame

- frame_width

Width of the incoming frame in pixels

- frame_height

  Height of the incoming frame in pixels

- fps_n

  Numerator of the frame rate.

- fps_d

  Frame rate denominator

- first_frame

  Is it the first frame?

- rc_enable

  Is rate control enabled?

- cqp

  When rate control is disabled, use a constant quantization parameter QP

- work_dmabuffer

  DMA buffer for encoder work

- bs_dmabuffer

  DMA buffer for storing bitstream

- bs_virt_addr

  Virtual address of bitstream

- bs_phys_addr

  Physical address of bitstream

- num_framebuffers

  Number of framebuffers

- internal_framebuffers

  Framebuffers inside the encoder

- framebuffers

  Framebuffers

- buffer_mv

  DMA buffer for storing motion vectors

- buffer_fbc_y_tbl

  DMA buffer for storing FBC brightness table

- buffer_fbc_c_tbl

  DMA buffer for storing FBC chroma table
- buffer_sub_sam

  Sub-sampling DMA buffer for ME
- headers_rbsp

  Frame header RBSP data
- headers_rbsp_size

  Frame header RBSP data size
- initial_info

  Encoder initial information
- timeout

  Encoding timeout, default is 1000ms (ie VPU_WAIT_TIMEOUT)
- timeout_count

  Encoding timeout retries, default is 40 (ie VPU_MAX_TIMEOUT_COUNTS)
- video_enc_ctx

  Encoding context information, internal use

16. BmVpuFbInfo

Parameter Description

- width

Frame width, aligned to the 16-pixel boundary required by the VPU

- height

Frame height, aligned to the 16-pixel boundary required by the VPU

- y_stride

Aligned stride size of the Y component, in bytes

- c_stride

Aligned stride size of the Cb and Cr components, in bytes (optional)

- y_size

Y component DMA buffer size, in bytes

- c_size

Cb and Cr component DMA buffer size, in bytes

- size

Total size of the frame buffer DMA buffer, in bytes. This value includes the size of all channels

17. BmVpuEncodedFrame

```
typedef struct
{
    uint8_t *data;
    size_t data_size;
    BmVpuFrameType frame_type;
    void *acquired_handle;
    void *context;
    uint64_t pts;
    uint64_t dts;
    int src_idx;
    bmvpu_phys_addr_t u64CustomMapPhyAddr;
    int avg_ctu_qp;
} BmVpuEncodedFrame;
```

Parameter Description

- uint8_t *data

  When decoding, data must point to a memory block containing bitstream data, not used by the encoder

- size_t data_size

  The size of the encoded data. When encoding, it is set by the encoder to indicate the size of the acquired output block in bytes

- BmVpuFrameType frame_type

  The frame type of the encoded frame (I, P, B, etc.). Filled by the encoder. Only used by the encoder

- void* acquired_handle

  The handle generated by the user-defined **acquire_output_buffer** function when encoding. Only used by the encoder

- void* context

  A user-defined pointer. The encoder will not change this value. This pointer and the pointer to the corresponding raw frame will have the same value, passed in the encoder

- uint64_t pts

User-defined presentation timestamp. Like the context pointer, the encoder just passes it to the associated raw frame and does not actually change its value

- uint64_t dts

  User defined decoding timestamp. Like the pts pointer, the encoder just passes it to the associated raw frame and does not actually change its value

- int src_idx

  Index of the raw frame

- bmvpu_phys_addr_t u64CustomMapPhyAddr

  Start address of the user-defined mapping option

- int avg_ctu_qp

  Average CTU QP (Quantization Parameter)

18. BmVpuEncDMABuffer

```
typedef struct
{
  unsigned int  size;
  uint64_t    phys_addr;
  uint64_t    virt_addr;
  int         enable_cache;
  int         dmabuf_fd;
} BmVpuEncDMABuffer;
```

Parameter Description

- size

  Physical memory size

- phys_addr

  Physical memory address

- virt_addr

  Physical memory virtual address after mmap

- enable_cache

  Whether to enable cache

- dmabuf_fd

  File handle, user cannot modify it, just pass it through

19. BmVpuRawFrame

```
typedef struct
{
   BmVpuFramebuffer *framebuffer;
   void *context;
   uint64_t pts;
   uint64_t dts;
} BmVpuRawFrame;
```

Parameter Description

- BmVpuFramebuffer *framebuffer

  Frame buffer for raw frames

- void* context

  User-defined pointer. The encoder does not change this value. This pointer and the pointer to the corresponding coded frame will have the same value, passed in the encoder

- uint64_t pts

  User-defined presentation timestamp. Like the context pointer, the encoder just passes it to the associated coded frame and does not actually change its value

- uint64_t dts

  User-defined decoding timestamp. Like the pts pointer, the encoder just passes it to the associated coded frame and does not actually change its value

20. BmVpuFramebuffer

```
typedef struct
{
   BmVpuEncDMABuffer *dma_buffer;
   int       myIndex;
   unsigned int y_stride;
   unsigned int cbcr_stride;
   unsigned int width;
   unsigned int height;
   size_t y_offset;
   size_t cb_offset;
   size_t cr_offset;
   int already_marked;
   void *internal;
   void *context;
} BmVpuFramebuffer;
```

Parameter Description

- BmVpuEncDMABuffer *dma_buffer

  Physical memory for storing YUV data
- int myIndex

  YUV index, set by the user, used to release YUV data
- unsigned int y_stride

  Y channel aligned size
- unsigned int cbcr_stride

  UV channel aligned size
- unsigned int width

  Encoded YUV image width
- unsigned int height

  Encoded YUV image height
- size_t y_offset

  Y channel offset. Relative to the start position of the buffer, specifies the starting offset of each component. Specified in bytes
- size_t cb_offset

  U channel offset
- size_t cr_offset

  V channel offset
- int already_marked

  Set to 1 if the frame buffer has been marked as used in the encoder. For internal use only. Do not read or write from the outside
- void* internal

  Internal implementation-defined data. Do not modify
- void* context

  User-defined pointer, the encoder will not modify this value. Usage is determined by the user, for example, it can be used to identify the serial number of the frame buffer containing this frame in the encoder

1.7.3 API extension description

1. char const * bmvpu_enc_error_string(BmVpuEncReturnCodes code)

Function	Returns a detailed description of the encoding error code
Input Parameters	BmVpuEncReturnCodes code Encoding error code

2. int bmvpu_enc_get_core_idx(int soc_idx)

Function	Get the unique index of the VPU encoder core on the specified Sophon SoC
Input Parameters	int soc_idx Device index number

3. int bmvpu_enc_load(int soc_idx)

Function	Loads the encoding module of the video processing unit (VPU) on the Sophon device.
Input Parameters	int soc_idx Device index number
Function Description	The number of calls to unload() and load() must be consistent. Before performing any other operations on the encoder, the encoder must be loaded. Likewise, an encoder must not be unloaded until all encoder activity has completed, including opening an encoder instance.

4. int bmvpu_enc_unload(int soc_idx)

Function	Unload encoder
Input Parameters	int soc_idx Device index number

5. void bmvpu_enc_get_bitstream_buffer_info(size_t *size, uint32_t *alignment)

Function	This function obtains the bitstream buffer size (size) and the required alignment value required by the encoder
Input Parameters	size_t *size - The size of the physical memory block required for the code stream buffer uint32_t *alignment - Alignment of the physical memory block required by the code stream buffer
Return Value	Returns the alignment and size of the physical memory block required by the encoder' s code stream buffer
Function Description	Need to be called before bmvpu_enc_open The user must allocate a DMA buffer of at least this size and its physical address must be aligned according to the alignment value.

6. void bmvpu_enc_set_default_open_params(BmVpuEncOpenParams *open_params, BmVpuCodecFormat codec_format)

Function	Set the default variables of the encoder to pass parameters when the encoder is initialized
Input Parameters	BmVpuEncOpenParams *open_params - Returns the encoder parameters. BmVpuCodecFormat codec_format - Selection of encoder and decoder, h264 and h265
Return Value	None
Function Description	Need to be called before bmvpu_enc_open If the caller only wants to modify a few member variables (or not modify them at all), you can call this function

7. int bmvpu_fill_framebuffer_params(BmVpuFramebuffer *fb, BmVpuFbInfo *info, bm_device_mem_t *fb_dma_buffer, int fb_id, void *context)

Function	Fill the fields of the buffer structure BmVpuFramebuffer structure according to the data obtained from bmvpufbinfo, and also set the specified DMA buffer and context pointer
Input Parameters	BmVpuFramebuffer *fb - An actual buffer, which is set by receiving information in info, such as buffer index, pointer, frame width and height, and Y, U, V components offset in the frame buffer. BmVpuFbInfo *info - Stores information about buffer settings bm_device_mem_t *fb_dma_buffer - The pointer points to the starting address of the actual DMA buffer. Through this pointer, the program can directly access and operate the data in the DMA buffer without copying or processing through the central processor. int fb_id - Buffer Index void *context - Context information
Return Value	BM_SUCESS - Allocation successful else - Allocation Failure
Function Description	None

8. int bmvpu_enc_open(BmVpuEncoder **encoder, BmVpuEncOpenParams *open_params, BmVpuDMABuffer *bs_dmabuffer, BmVpuEncInitialInfo *initial_info)

Function	Open a new encoder instance, set encoder parameters and start receiving video frames
Input Parameters	BmVpuEncOpenParams *open_params - Various parameters of the encoder BmVpuEncoder **encoder - A secondary pointer to the encoder instance, which receives the properties of the encoder and some settings of the video frame, such as device id, buffer settings, frame rate, width and height, etc. BmVpuDMABuffer *bs_dmabuffer - Pointer to the code stream buffer, using the code stream buffer that has been allocated previously BmVpuEncInitialInfo *initial_info - Initialization information of the encoder, returning to the user the minimum number and size of frame buffers required by the encoder
Return Value	BM_SUCESS - Open successfully else - Open Failed
Function Description	Before calling, make sure that BmVpuEncOpenParams and BmVpuDMABuffer are not empty.

9. int bmvpu_enc_close(BmVpuEncoder *encoder)

Function	Close the encoder instance
Input Parameters	BmVpuEncoder *encoder - Video Encoder Instance
Return Value	BM_SUCESS - Closed successfully else - Close Failed
Function Description	Multiple attempts to close the same instance result in undefined behavior

10. int bmvpu_enc_encode(BmVpuEncoder *encoder, BmVpuRawFrame const *raw_frame, BmVpuEncodedFrame *encoded_frame, BmVpuEncParams *encoding_params, uint32_t *output_code)

Function	Encodes the given original input frame using the given encoding parameters. encoded_frame is filled with information about the resulting encoded output frame. bm1688 encoding uses asynchronous interface, this interface is not supported
Input Parameters	BmVpuEncoder *encoder - Video Encoder Instance BmVpuRawFrame const *raw_frame - Original video frame, including frame data, timestamp, etc. BmVpuEncodedFrame *encoded_frame - Encoded video frames, including frame data, frame type, timestamp, etc. BmVpuEncParams *encoding_params - Parameters for encoding uint32_t *output_code - Return output status code
Return Value	BM_SUCESS - Encoding successful else - Encoding failed
Function Description	The encoded frame data itself is stored in buffers allocated by the user-supplied functions (set to acquire_output_buffer and finish_output_buffer function pointers in encoding_params).

11. int bmvpu_enc_send_frame(BmVpuEncoder *encoder, BmVpuRawFrame const *raw_frame, BmVpuEncParams *encoding_params)

Function	Encodes the given original input frame using the given encoding parameters. New interface in bm1688
Input Parameters	BmVpuEncoder *encoder - Video encoder instance BmVpuRawFrame const *raw_frame - Raw video frames, including frame data, timestamps, etc BmVpuEncParams *encoding_params - Parameters used for encoding
Return value	BM_SUCESS - Data sent successfully else - Failed to send data

10. int bmvpu_enc_get_stream(BmVpuEncoder *encoder, BmVpuEncodedFrame *encoded_frame, BmVpuEncParams *encoding_params)

Function	The encoded bitstream data is obtained and stored in encoded_frame New interface in bm1688
Input Parameters	BmVpuEncoder *encoder - Video encoder instance BmVpuEncodedFrame *encoded_frame - The encoded video frame, including frame data, frame type, timestamp, etc BmVpuEncParams *encoding_params - Parameters used for encoding
Return Value	BM_SUCESS - Acquired successfully else - Acquisition failure
Function Description	The encoded frame data themselves are stored in buffers allocated by the user-supplied functions (which are set to the acquire_output_buffer and finish_output_buffer function Pointers in encoding_params)

11. int bmvpu_enc_dma_buffer_allocate(int vpu_core_idx, BmVpuEncDMABuffer *buf, unsigned int size)

Function	Allocate device memory according to the size specified by the user
Input Parameters	int vpu_core_idx - Input parameters, specify the index of the core where the encoder is located BmVpuEncDMABuffer *buf - Output parameters. After the function is executed, the phys_addr, size, and enable_cache member variables of the structure will be filled. unsigned int size- Input parameters, in bytes, specify the required buffer size
Return Value	BM_SUCESS - Close Success else - Close Failed
Function Description	None

12. int bmvpu_enc_dma_buffer_deallocate(int vpu_core_idx, BmVpuEncDMABuffer *buf)

Function	Release device memory allocated by **bmvpu_enc_dma_buffer_allocate** function
Input Parameters	int vpu_core_idx - Input parameters, specify the index of the core where the encoder is located BmVpuEncDMABuffer *buf - Output parameters. After the function is executed, the phys_addr, size, and enable_cache member variables of the structure will be filled.
Return Value	BM_SUCESS - Close Success else - Close Failed
Function Description	None

13. int bmvpu_enc_dma_buffer_attach(int vpu_core_idx, uint64_t paddr, unsigned int size)

Function	Bind the device memory address requested by the user through other methods other than the **bmvpu_enc_dma_buffer_allocate** function to the encoder
Input Parameters	int vpu_core_idx - Input parameters, specify the index of the core where the encoder is located uint64_t paddr - Input parameters, the device memory address requested by the user through other methods other than the **bmvpu_enc_dma_buffer_allocate** function unsigned int size Input parameters, memory size of the block device (byte)
Return Value	BM_SUCESS - Close Success else - Close Failed
Function Description	None

14. int bmvpu_enc_dma_buffer_deattach(int vpu_core_idx, uint64_t paddr, unsigned int size)

Function	Unbind the device memory bound by the user through the **bmvpu_enc_dma_buffer_attach** function
Input Parameters	int vpu_core_idx - Input parameters, specify the index of the core where the encoder is located uint64_t paddr - Input parameters, the device memory address requested by the user through other methods other than the **bmvpu_enc_dma_buffer_allocate** function unsigned int size Input parameters, memory size of the block device (byte)
Return Value	BM_SUCESS - Close Success else - Close Failed
Function Description	None

15. int bmvpu_dma_buffer_map(int vpu_core_idx, BmVpuEncDMABuffer *buf, int port_flag)

Function	Map the device memory requested on the corresponding core to the system memory
Input Parameters	int vpu_core_idx - Input parameters, specify the index of the core where the encoder is located BmVpuEncDMABuffer *buf - Input parameters to specify the address, size, and other information of the device memory int port_flag Input parameters, configure **mmap** memory to be readable (BM_VPU_MAPPING_FLAG_READ) or writable (BM_VPU_MAPPING_FLAG_WRITE)
Return Value	BM_SUCESS - Close Success else - Close Failed
Function Description	None

16. int bmvpu_dma_buffer_unmap(int vpu_core_idx, BmVpuEncDMABuffer *buf)

Function	Unmap device memory mapped on a core
Input Parameters	int vpu_core_idx - Input parameters, specify the index of the core where the encoder is located BmVpuEncDMABuffer *buf - Input parameters to specify the address, size, and other information of the device memory
Return Value	BM_SUCESS - Close Success else - Close Failed
Function Description	None

17. int bmvpu_enc_dma_buffer_flush(int vpu_core_idx, BmVpuEncDMABuffer *buf)

Function	Perform a flush operation on allocated device memory
Input Parameters	int vpu_core_idx - Input parameters, specify the index of the core where the encoder is located BmVpuEncDMABuffer *buf - Input parameters. Before calling, the user must at least fill in the phys_addr and size member variables of the structure.
Return Value	BM_SUCESS - Close Success else - Close Failed
Function Description	None

18. int bmvpu_enc_dma_buffer_invalidate(int vpu_core_idx, BmVpuEncDMABuffer *buf)

Function	Perform invalid operation on allocated device memory
Input Parameters	int vpu_core_idx - Input parameters, specify the index of the core where the encoder is located BmVpuEncDMABuffer *buf - Input parameters. Before calling, the user must at least fill in the phys_addr and size member variables of the structure.
Return Value	BM_SUCESS - Close Success else - Close Failed
Function Description	None

19. uint64_t bmvpu_enc_dma_buffer_get_physical_address(BmVpuEncDMABuffer *buf)

Function	Returns the address of allocated device memory
Input Parameters	BmVpuEncDMABuffer *buf - Input parameters, allocated device memory
Return Value	Physical address of allocated device memory
Function Description	None

20. unsigned int bmvpu_enc_dma_buffer_get_size(BmVpuEncDMABuffer *buf)

Function	Returns the size of the allocated device memory
Input Parameters	BmVpuEncDMABuffer *buf - Input parameters
Return Value	Size of allocated device memory
Function Description	None

21. int bmvpu_enc_upload_data(int vpu_core_idx, const uint8_t* host_va, int host_stride, uint64_t vpu_pa, int vpu_stride, int width, int height)

Function	Transfer data to the device memory address allocated using **bmvpu_enc_dma_buffer_allocate()**
Input Parameters	int vpu_core_idx - Input parameters, specify the index of the core where the encoder is located const uint8_t *host_va - Input parameters, host virtual address of data to be transmitted int host_stride - Input parameters, data span on host side uint64_t vpu_pa - Input parameters, target physical address of transmitted data int vpu_stride - Input parameters, data span on device side int width - Input parameters, data width int height - Input parameters, data height
Return Value	BM_SUCESS - Success else - Failure
Function Description	Only supports PCIe mode

22. int bmvpu_enc_download_data(int vpu_core_idx, uint8_t* host_va, int host_stride, uint64_t vpu_pa, int vpu_stride, int width, int height)

Function	Transfer data from the device memory address allocated by **bmvpu_enc_dma_buffer_allocate()** to the host
Input Parameters	int vpu_core_idx - Input parameters, specify the index of the core where the encoder is located const uint8_t *host_va - Input parameters, host virtual address of data to be transmitted int host_stride - Input parameters, data span on host side uint64_t vpu_pa - Input parameters, target physical address of transmitted data int vpu_stride - Input parameters, data span on device side int width - Input parameters, data width int height - Input parameters, data height
Return Value	BM_SUCESS - Success else - Failure
Function Description	Only supports PCIe mode