Google Translated by XNL Future Technologies

Please note that I have just google translated this documentation for others to be able to use it when needed. The copyright remains at the original copyright holder and should be respected as such.

However a lot of us can't read Chinese 😉 and therefor I decided to Google translate the necessary documentation.

I needed some of this documentation because I'm working on a large project for the R36S/R36H and lots of the documentation I either needed or wanted to go through was in Chinese.

I also do realize that lots of people know how to use Google translate 😉, but I also know that needing to do so over and over can seriously disrupt your workflow. Hence these translated documents.

I also attempted to 'recover' as much as possible of the table borders, because those got lost when copying the translated output to this PDF.

WARNING: Although most of the documentation is translated quite reasonable (as far as that's possible with Auto Translate), there are still some untranslated sections/words/remarks here and there. But most of it is quite useable as it is now.

Document Name:	**RGA IM2D API Development Guide**
Original Document:	https://github.com/JustEnoughLinuxOS/linux-rga/blob/im2d/docs/RGA_API_Instruction.md

My YouTube: **https://www.youtube.com/@XNLFutureTechnologies**
My Website: **https://www.teamxnl.com**
My GitHub: **https://www.github.com/xnlfuturetechnologies**

RGA IM2D API Development Guide

File ID: RK-PC-YF-0002

Release version: V1.0.1

Date: 2020-07-10

File classification: □Top secret □Secret □Internal information ■Public

---

Rockchip Electronics Co., Ltd.

Rockchip Electronics Co., Ltd.

Address: No. 18, Area A, Software Park, Tongpan Road, Fuzhou, Fujian Province

Website: www.rock-chips.com

Customer Service Phone: +86-4007-700-590

Customer Service Fax: +86-591-83951833

Customer Service Email: fae@rock-chips.com

Target Audience

This document is mainly intended for the following engineers:

- Technical Support Engineer
- Software Development Engineer

Revision History

date	Version	author	Modification Notes
2020/06/24	1.0.0	Chen Cheng, Li Huang	Initial release
2020/10/16	1.0.1	Chen Cheng, Li Huang, Yu Qiaowei	Update some interfaces

Table of contents

[TOC]

Overview

RGA (Raster Graphic Acceleration Unit) is an independent 2D hardware accelerator that can be used to accelerate point/line drawing and perform common 2D graphics operations such as image scaling, rotation, bitBlt, alpha blending, etc.

Design indicators

Version	Codename	Chip	Source	Destination	Pixels/Cycle	Performance w/o scale
RGA1	Pagani	RK3066	8192x8192	2048x2048	1	≈300Mpix/s
	Jaguar Plus	RK3188				
	Beetles	RK2926/2928				
	Beetles Plus	RK3026/3028				
RGA1_plus	Audi	RK3128	8192x8192	2048x2048	1	≈300Mpix/s
	Granite	Sofia 3gr				
RGA2	Lincoln	RK3288/3288w	8192x8192	4096x4096	2	≈600Mpix/s
	Capricorn	RK3190				
RGA2-Enhance	Mclaren	RK3399	8192x8192	4096x4096	2	≈600Mpix/s
	Mercury	RK1108				

	Puma	RV1126/RV1109				
	skylarkV2	RK3566/RK3568				
RGA2-Lite0	Maybach	RK3368	8192x8192	4096x4096	2	≈520Mpix/s
	BMW	RK3366				
RGA2-Lite1	Benz	RK3228	8192x8192	4096x4096	2	≈520Mpix/s
	Infiniti	RK3228H				
	Gemini	RK3326				
	Lion	RK1808				

- The expected performance is calculated under the default RGA frequency. The actual operating performance is related to memory frequency, etc. The table data is for reference only.

Image format support

- Pixel Format conversion, BT.601/BT.709
- Dither operation

Version	Codename	Chip	Input Data Format	Output Data Format
RGA1	Pagani	RK3066	ARGB888/888/565/4444/5551 YUV420/YUV422 BPP8/BPP4/BPP2/BPP1	ARGB888/888/565/4444/5551 YUV420/YUV422(only for Blur/sharpness) 8bit
	Jaguar Plus	RK3188		
	Beetles	RK2926/2928		
	Beetles Plus	RK3026/3028		
RGA1_plus	Audi	RK3128	ARGB888/888/565/4444/5551 YUV420/YUV422 BPP8/BPP4/BPP2/BPP1	ARGB888/888/565/4444/5551 YUV420/YUV422 (only for Blur/sharpness) 8it YUV420/YUV422 output (only for normal Bitblt without alpha)
	Granite	Sofia 3gr		
RGA2	Lincoln	RK3288/3288w	ARGB888/888/565/4444/5551 YUV420/YUV422	ARGB888/888/565/4444/5551 YUV420/YUV422
	Capricorn	RK3190		
RGA2-Enhance	Mclaren	RK3399	ARGB888/888/565/4444/5551	ARGB888/888/565/4444/5551 YUV420/YUV422
	Mercury	RK1108		

			YUV420/YUV422 (8/10bit)	(8/10bit) YVYU422/YUYV420
	Puma	RV1126/RV1109	ARGB888/888/565/4444/5551 YUV420/YUV422 (8/10bit) YVYU422	ARGB888/888/565/4444/5551 YUV420/YUV422 (8/10bit) YUV400/Y4/Y1 YVYU422/YUYV420
	skylarkV2	RK3566/RK3568		
RGA2-Lite0	Maybach	RK3368	ARGB888/888/565/4444/5551 YUV420/YUV422	ARGB888/888/565/4444/5551 YUV420/YUV422
	BMW	RK3366		
RGA2-Lite1	Benz	RK3228	ARGB888/888/565/4444/5551 YUV420/YUV422 (8/10bit)	ARGB888/888/565/4444/5551 YUV420/YUV422 (8/10bit)
	Infiniti	RK3228H		
	Gemini	RK3326		
	Lion	RK1808		

Note: Y4 format is a grayscale image with a color scale of 2 to the fourth power, and Y400 format is a grayscale image with a color scale of 2 to the eighth power.

Image format alignment instructions

Format	Alignment
YUV420/422 YUV400/Y4 YVYU422/YUYV420	Width stride must be aligned to 4, and the rest of the parameters must be aligned to 2
YUV420/422 10bit	Width stride must be aligned to 16, and the rest of the parameters must be aligned to 2
RGB888	Width stride must be aligned to 4
RGB565	Width stride must be aligned to 2

API Version Notes

The RGA support library librga.so updates the version number according to certain rules, marking the update submission of new functions, compatibility, and problem corrections, and provides several ways to query the version number, so that developers can clearly identify whether the current library file version is suitable for the current development environment when using librga.so. For detailed version update logs, please refer to CHANGLOG.md in the source code root directory.

Version number format and increment rules

API version number

Format

```
major.minor.revision_[build]
```

example:

1.2.1_[1]

Incremental rule

name	rule
major	The major version number, used when submitting a backwards incompatible version.
minor	Minor version number, when backward-compatible functional APIs are added.
revision	Revision number, when submitting backwards-compatible feature additions or critical bug fixes.
build	Compile version number when backward compatibility issues are fixed.

API build version number

Format

```
(git_commit build: build_time base: build_platform)
```

example:

(be7518a build: 2021-04-29 12:01:46 base: rk3566_r)

Incremental rule

name	rule
git_commit	The code version is committed.
build_time	Compile time.
build_paltform	Chip platform (only supports Android system).

Version number query

Strings command query:

Take Android R 64-bit as an example:

```
:/# strings vendor/lib64/librga.so |grep rga_api |grep version
rga_api version 1.2.1_[1] (be7518a build: 2021-04-29 12:01:46 base:
rk3566_r)
```

Log printing:

When each process calls the RGA API for the first time, a version number is printed.

```
rockchiprga: rga_api version 1.2.1_[1] (5519100 build: 2021-04-30 15:17:33
base: rk3566_r)
```

Function interface query

By calling the following API, you can query the code version number, compilation version number, and RGA hardware version information. For specific instructions, please refer to **the Application Interface Description** section.

```
querystring(RGA_VERSION);
```

The string format is as follows:

RGA_api version: v1.2.1_[1] RGA_built version: be7518a build: 2021-04-29 12:01:46 RGA version: RGA_2_Enhance

Property query

This method of querying the version number is only supported by the Android system, and the property setting will take effect only after a process has called RGA.

```
:/# getprop |grep rga
[vendor.rga_api.version]: [1.2.1_[1]]
[vendor.rga_built.version]: [be7518a build: 2021-04-29 12:01:46]
```

Application interface description

The RGA module support library is librga.so, which implements the corresponding 2D graphics operations by configuring the image buffer structure struct rga_info. In order to obtain a more friendly development experience, the commonly used 2D image operation interface is further encapsulated on this basis. The new interface mainly includes the following features:

- The interface definition refers to the commonly used 2D graphics interface definition in opencv/matlab to reduce the learning cost of secondary development.
- In order to eliminate the compatibility issues caused by the differences in RGA hardware versions, the RGA query function is added. The query content mainly includes version information, input and output maximum resolution and image format support.
- For 2D image composite operations, the improcess interface is added. Composite operations are performed by passing in a series of predefined usages.
- Before performing image operations, the input and output image buffers need to be processed. Call the wrapbuffer_T interface to fill the input and output image information into the struct rga_buffer_t structure, which contains information such as resolution and image format.

Get RGA version and support information

---

querystring

```
const char* querystring(int name);
```

Query RGA basic information and resolution format support

Parameters	Description
name	RGA_VENDOR - Vendor informationRGA_VERSION - Version informationRGA_MAX_INPUT - Maximum supported input resolutionRGA_MAX_OUTPUT - Maximum supported output resolutionRGA_SCALE_LIMIT - Supported scaling factorsRGA_INPUT_FORMAT - Supported input formatsRGA_OUTPUT_FORMAT - Supported output formatsRGA_EXPECTED - Expected performanceRGA_ALL - Output all information

Returns a string describing properties of RGA.

Image buffer preprocessing

---

wrapbuffer_T

In the IM2D graphics library interface parameters, the input source image and output target image should support multiple types (the input parameters in the following content use the symbol 'T' to represent the supported types). Before performing the corresponding image operation, you need to call wrapbuffer_T(T) to convert the input and output image buffer types into a unified rga_buffer_t structure as the input parameter of the user API. The supported input and output image buffer types include:

Parameters(T)	Data Type	Description
virtual address	void *	Image buffer virtual address
physical address	void *	Image buffer physical address
sharedfd	int	Image buffer file descriptor
buffer handle	buffer_handle_t gralloc_drm_handle_t gralloc_drm_bo_t	Image buffer handle, including buffer address, file descriptor, resolution and format information
GraphicBuffer	GraphicBuffer	android graphic buffer
AHardwareBuffer	AHardwareBuffer	chunks of memory that can be accessed by various hardware components in the system. https://developer.android.com/ndk/reference/group/a-hardware-buffer

The performance of calling RGA with different buffer types is different. The performance ranking is as follows:

physical address > fd = buffer handle = GraphicBuffer = AHardwareBuffer > virtual address

It is generally recommended to use fd as the buffer type.

```
rga_buffer_t wrapbuffer_virtualaddr(void* vir_addr,
                                    int width,
                                    int height,
                                    int wstride = width,
                                    int hstride = height,
                                    int format);
rga_buffer_t wrapbuffer_physicaladdr(void* phy_addr,
                                     int width,
                                     int height,
                                     int wstride = width,
                                     int hstride = height,
                                     int format);
rga_buffer_t wrapbuffer_fd(int fd,
                           int width,
                           int height,
                           int wstride = width,
                           int hstride = height,
                           int format);
```

Android Only

```
rga_buffer_t wrapbuffer_GraphicBuffer(sp<GraphicBuffer> buf);
rga_buffer_t wrapbuffer_AHardwareBuffer(AHardwareBuffer *buf);
```

Returns a rga_buffer_t to desire image information.

Image scaling, image pyramid

---

imresize

```
IM_STATUS
imresize(const rga_buffer_t src,
         rga_buffer_t dst,
         double fx = 0,
         double fy = 0,
         int interpolation = INTER_LINEAR,
         int sync = 1);
```

Depending on the application scenario, you can choose to configure dst to describe the target image size for scaling, or configure the scaling factor fx/fy to achieve the effect of scaling by a specified ratio. When dst and scaling factor fx/fy are configured at the same time, the result calculated by the scaling factor fx/fy will be used as the target image size.

Interpolation Configuration is only supported by hardware version RGA1/RGA1 plus.

Note: When using the scaling factor fx/fy for scaling, formats that require width and height alignment, such as YUV, will be forced to align downward to meet the requirements. Using this function may change the expected scaling effect.

Parameters	Description
src	**[required]** input image
dst	**[required]** output image; it has the size dsize (when it is non-zero) or the size computed from src.size(), fx, and fy; the type of dst is the same as of src.
fx	**[optional]** scale factor along the horizontal axis; when it equals 0, it is computed as: fx = (double) dst.width / src.width
fy	**[optional]** scale factor along the vertical axis; when it equals 0, it is computed as: fy = (double) dst.height / src.height
interpolation	**[optional]** interpolation method: INTER_NEAREST - a nearest-neighbor interpolation INTER_LINEAR - a bilinear interpolation (used by default) INTER_CUBIC - a bicubic interpolation over 4x4 pixel neighborhood
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code.

impyramid

```
IM_STATUS impyramid (const rga_buffer_t src,
                     rga_buffer_t dst,
                     IM_SCALE direction)
```

Pyramid scaling. Scale the width and height by 1/2 or 2 times according to the direction.

Parameters	Description
src	**[required]** input image
dst	**[required]** output image;
direction	**[required]** scale mode IM_UP_SCALE —— up scale IM_DOWN_SCALE —— down scale

Return IM_STATUS_SUCCESS on success or else negative error code.

Image Cropping

---

imcrop

```
IM_STATUS imcrop(const rga_buffer_t src,
                 rga_buffer_t dst,
                 im_rect rect,
                 int sync = 1);
```

Performs image cropping by specifying a Rect size region.

Parameter	Description
src	**[required] **input image
dst	**[required]** output image
rect	**[required]** crop region x - upper-left x coordinate y - upper-left y coordinate width - region width height - region height
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code.

Image Rotation

---

imrotate

```
IM_STATUS imrotate(const rga_buffer_t src,
                   rga_buffer_t dst,
                   int rotation,
                   int sync = 1);
```

Support image rotation of 90, 180, 270 degrees.

Parameter	Description
src	**[required]** input image
dst	**[required]** output image
rotation	**[required]** rotation angle: 0 IM_HAL_TRANSFORM_ROT_90 IM_HAL_TRANSFORM_ROT_180 IM_HAL_TRANSFORM_ROT_270
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code.

Image mirror flip

---

imfilp

```
IM_STATUS imflip (const rga_buffer_t src,
                  rga_buffer_t dst,
                  int mode,
                  int sync = 1);
```

Supports horizontal and vertical image flipping.

Parameter	Description
src	**[required]** input image
dst	**[required]** output image
mode	**[optional]** flip mode: 0 IM_HAL_TRANSFORM_FLIP_H IM_HAL_TRANSFORM_FLIP_V
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code.

Image color filling, memory assignment, graphics drawing

---

imfill/imreset/imdraw

```
IM_STATUS imfill(rga_buffer_t buf,
                 im_rect rect,
                 int color = 0x00000000,
                 int sync = 1);
```

Fill the specified area rect of the RGBA format image with color. The color parameters are R, G, B, A from high to low. For example, red: color = 0xff000000.

```
IM_STATUS imreset(rga_buffer_t buf,
                  im_rect rect,
                  int color = 0x00000000,
                  int sync = 1);
```

Sets all the contents of the memory in the specified area rect of the RGBA format image to the specified value color. The color parameters are R, G, B, A from high to low. For example, red: color = 0xff000000.

```
IM_STATUS imdraw(rga_buffer_t buf,
                 im_rect rect,
                 int color = 0x00000000,
                 int sync = 1);
```

Draw the specified area rect of the RGBA format image according to the specified color color. The color parameters are R, G, B, A from high to low. For example, red: color = 0xff000000.

【Note】The width and height of the filling area rect must be greater than or equal to 2

Parameter	Description
src	**[required]** input image

Parameter	Description
dst	**[required]** output image
rect	**[required]** image region to fill specified color width and height of rect must be greater than or equal to 2
color	**[required]** fill with color, default=0x00000000
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code.

Image translation

---

imtranslate

```
IM_STATUS imtranslate(const rga_buffer_t src,
                      rga_buffer_t dst,
                      int x,
                      int y,
                      int sync = 1)
```

Translate the image to the (x, y) coordinate position. The width and height of src and dst must be consistent. The excess part will be cropped.

Parameter	Description
src	**[required]** Input image
dst	**[required]** output image
x	**[optional]** horizontal translation
y	**[optional]** vertical translation
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code.

Image Copy

---

imcopy

```
IM_STATUS imcopy(const rga_buffer_t src,
                 rga_buffer_t dst,
                 int sync = 1);
```

Copy the image, a basic RGA operation. Its function is similar to memcpy.

Parameter	Description
src	**[required]** input image

Parameter	Description
dst	**[required]** output image
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code.

Image synthesis

---

imblend/imcomposite

```
IM_STATUS imblend(const rga_buffer_t srcA,
                  rga_buffer_t dst,
                  int mode = IM_ALPHA_BLEND_SRC_OVER,
                  int sync = 1);
```

RGA uses the dual-channel image synthesis mode of A+B -> B, performs corresponding Alpha superposition calculations on the foreground image (srcA channel) and the background image (dst channel) according to the configured mixing model, and outputs the synthesis result to the dst channel.

```
IM_STATUS imcomposite(const rga_buffer_t srcA,
                      const rga_buffer_t srcB,
                      rga_buffer_t dst,
                      int mode = IM_ALPHA_BLEND_SRC_OVER,
                      int sync = 1);
```

RGA uses the three-channel image synthesis mode of A+B -> C to perform the corresponding Alpha superposition calculation on the foreground image (srcA channel) and the background image (srcB channel) according to the configured mixing model, and outputs the synthesis result to the dst channel.

In the two image synthesis modes, mode can be configured with different **Porter-Duff hybrid models** :

Before explaining the Porter-Duff mixing model, the following definitions are made:

- S - **identifies the source image in the two mixed images** , that is, the foreground image, which is the abbreviation of souce.
- D - **identifies the target image in the two mixed images** , that is, the background image, which is the abbreviation of destination.
- R - **identifies the result of mixing two images** , which is the abbreviation of result.
- c - **identifies the color of the pixel** , that is, the RGB part of (RGBA), describes the color of the image itself, and is the abbreviation of color. (**Note** that the color values (RGB) in the Porter-Duff mixture model are all the results of pre-multiplication, that is, the product of the original color and transparency. If the color value is not pre-multiplied, a pre-multiplication (Xc = Xc * Xa) operation is required.)

- a - **identifies the transparency of the pixel** , that is, the A part of (RGBA), which describes the transparency of the image itself and is the abbreviation of Alpha.
- f - **identifies the factor acting on C or A** , which is the abbreviation of factor.

The core formula of the Porter-Duff mixture model is as follows:

Rc = Sc * Sf + Dc * Df;

That is: Result color = source color * source factor + target color * target factor.

Ra = Sa * Sf + Da * Df;

That is: Result transparency = source transparency * source factor + target transparency * target factor.

RGA supports the following hybrid models:

SRC:

Sf = 1, Df = 0;

[Rc, Ra] = [Sc, Sa];

DST:

Sf = 0, Df = 1;

[Rc, Ra] = [Dc, Da];

SRC_OVER:

Sf = 1, Df = (1 - Sa);

[Rc, Ra] = [Sc + (1 - Sa) * Dc, Sa + (1 - Sa) * Da];

DST_OVER:

Sf = (1 - Da), Df = 1;

[Rc, Ra] = [Sc * (1 - Da) + Dc, Sa * (1 - Da) + Da];

[Note] The image synthesis mode does not support synthesis between YUV formats. The imblend function dst image does not support the YUV format, and the imcomposite function srcB image does not support the YUV format.

Parameter	Description
srcA	**[required]** input image A
srcB	**[required]** input image B

Parameter	Description
dst	**[required]** output image
mode	**[optional]** blending mode: IM_ALPHA_BLEND_SRC —— SRC modeIM_ALPHA_BLEND_DST —— DST modeIM_ALPHA_BLEND_SRC_OVER —— SRC OVER modeIM_ALPHA_BLEND_DST_OVER —— DST OVER modeIM_ALPHA_BLEND_PRE_MUL —— Premultiplication enabled. When premultiplication is required, this flag must be processed with other mode flags and then assigned to the mode.
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code.

Color Key

---

imcolorkey

```
IM_STATUS imcolorkey(const rga_buffer_t src,
                     rga_buffer_t dst,
                     im_colorkey_range range,
                     int mode = IM_ALPHA_COLORKEY_NORMAL,
                     int sync = 1)
```

The Color Key technique preprocesses the source image, sets the alpha component of pixels that meet the color key filtering condition to zero, wherein the color key filtering condition is a non-transparent color value, and performs an alpha blending mode on the preprocessed source image and the target image.

This mode only supports the Color Key function for the set color range on the image in the source image (src) area and superimposes it on the destination image (dst) area.

IM_ALPHA_COLORKEY_NORMAL is the normal mode, that is, the color within the set color range is used as the filtering condition, and the alpha component of the pixels within the color range is cleared to zero, while IM_ALPHA_COLORKEY_INVERTED is the opposite.

Parameters	Range	Description
max	0x0 ~ 0xFFFFFFFF	The maximum value of the color range that needs to be eliminated/extracted, arranged as ABGR
min	0x0 ~ 0xFFFFFFFF	The minimum value of the color range that needs to be eliminated/extracted, arranged as ABGR

parameter	Description
src	**[required]** input image
dst	**[required]** output image

parameter	Description
range	**[required]** Target color range typedef struct im_colorkey_range { int max; int min; } im_colorkey_value;
Mode	**[required]** Color Key mode: IM_ALPHA_COLORKEY_NORMAL IM_ALPHA_COLORKEY_INVERTED
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code.

Image format conversion

---

imcvtcolor

```
IM_STATUS imcvtcolor(rga_buffer_t src,
                     rga_buffer_t dst,
                     int sfmt,
                     int dfmt,
                     int mode = IM_COLOR_SPACE_DEFAULT,
                     int sync = 1)
```

Format conversion function. The specific format support varies depending on the soc. Please refer to the **image format support** section.

The format can be set via rga_buffer_t, or the source and output image formats can be configured separately via sfmt/dfmt.

parameter	Description
src	**[required]** input image
dst	**[required]** output image
sfmt	**[optional]** source image format
dfmt	**[optional]** destination image format
Mode	**[optional]** color space mode: IM_YUV_TO_RGB_BT601_LIMIT IM_YUV_TO_RGB_BT601_FULL IM_YUV_TO_RGB_BT709_LIMIT IM_RGB_TO_YUV_BT601_LIMIT IM_RGB_TO_YUV_BT601_FULL IM_RGB_TO_YUV_BT709_LIMIT
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code.

NN operation point pre-processing

---

imquantize

```
IM_STATUS imquantize(const rga_buffer_t src,
                     rga_buffer_t dst,
                     rga_nn_t nn_info,
                     int sync = 1)
```

Currently only supported on RV1126 / RV1109. NN operation point pre-processing, image RGB three channels can be configured with offset and scale separately.

formula:

```
dst = 【(src + offset) * scale 】
```

Parameter range:

Parameters	Range	Description
scale	0 ~ 3.99	10 bits, from left to right, the high 2 bits represent the integer part, the low 8 bits represent the decimal part
offset	-255 ~ 255	9 bits, from left to right, the high bit indicates the sign bit, and the low bit indicates the offset from 0 to 255

parameter	Description
src	**[required]** input image
dst	**[required]** output image
nn_info	**[required]** The rga_nn_t structure configures the offset and scale of the three RGB channels separately typedef struct rga_nn { int nn_flag; int scale_r; int scale_g; int scale_b; int offset_r; int offset_g; int offset_b; } rga_nn_t;
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code

ROP AND or NOT operation

---

imrop

```
IM_STATUS imquantize(const rga_buffer_t src,
                     rga_buffer_t dst,
                     int rop_code,
                     int sync = 1)
```

Perform ROP AND/OR operations on two graphs

parameter	Description
src	**[required]** input image
dst	**[required]** output image
rop_code	**[required]** rop code mode IM_ROP_AND : dst = dst **AND** src; IM_ROP_OR : dst = dst **OR** src IM_ROP_NOT_DST : dst = **NOT** dst IM_ROP_NOT_SRC : dst = **NOT** src IM_ROP_XOR : dst = dst **XOR** src IM_ROP_NOT_XOR : dst = **NOT** (dst **XOR** src)
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code

Image Processing

---

improcess

```
IM_STATUS improcess(rga_buffer_t src,
                    rga_buffer_t dst,
                    rga_buffer_t pat,
                    im_rect srect,
                    im_rect drect,
                    im_rect prect,
                    int usage)
```

RGA image composite operation function. Other APIs are developed based on this API. improcess can implement more complex composite operations.

Image operations are configured via usage.

Parameter	Description
src	**[required]** input imageA
dst	**[required]** output image
Pat	**[required]** input imageB
srect	**[optional]** src crop region
drect	**[optional]** dst crop region
prect	**[optional]** pat crop region

Parameter	Description
usage	**[optional]** image operation usage

usage Refer to definition:

```
typedef enum {
    /* Rotation */
    IM_HAL_TRANSFORM_ROT_90     = 1 << 0,
    IM_HAL_TRANSFORM_ROT_180    = 1 << 1,
    IM_HAL_TRANSFORM_ROT_270    = 1 << 2,
    IM_HAL_TRANSFORM_FLIP_H     = 1 << 3,
    IM_HAL_TRANSFORM_FLIP_V     = 1 << 4,
    IM_HAL_TRANSFORM_FLIP_H_V   = 1 << 5,
    IM_HAL_TRANSFORM_MASK       = 0x3f,

    /*
     * Blend
     * Additional blend usage, can be used with both source and target
configs.
     * If none of the below is set, the default "SRC over DST" is applied.
     */
    IM_ALPHA_BLEND_SRC_OVER     = 1 << 6,     /* Default, Porter-Duff "SRC
over DST" */
    IM_ALPHA_BLEND_SRC          = 1 << 7,     /* Porter-Duff "SRC" */
    IM_ALPHA_BLEND_DST          = 1 << 8,     /* Porter-Duff "DST" */
    IM_ALPHA_BLEND_SRC_IN       = 1 << 9,     /* Porter-Duff "SRC in DST"
*/
    IM_ALPHA_BLEND_DST_IN       = 1 << 10,    /* Porter-Duff "DST in SRC"
*/
    IM_ALPHA_BLEND_SRC_OUT      = 1 << 11,    /* Porter-Duff "SRC out DST"
*/
    IM_ALPHA_BLEND_DST_OUT      = 1 << 12,    /* Porter-Duff "DST out SRC"
*/
    IM_ALPHA_BLEND_DST_OVER     = 1 << 13,    /* Porter-Duff "DST over SRC"
*/
    IM_ALPHA_BLEND_SRC_ATOP     = 1 << 14,    /* Porter-Duff "SRC ATOP" */
    IM_ALPHA_BLEND_DST_ATOP     = 1 << 15,    /* Porter-Duff "DST ATOP" */
    IM_ALPHA_BLEND_XOR          = 1 << 16,    /* Xor */
    IM_ALPHA_BLEND_MASK         = 0x1ffc0,

    IM_ALPHA_COLORKEY_NORMAL    = 1 << 17,
    IM_ALPHA_COLORKEY_INVERTED  = 1 << 18,
    IM_ALPHA_COLORKEY_MASK      = 0x60000,

    IM_SYNC                     = 1 << 19,
    IM_ASYNC                    = 1 << 26,
    IM_CROP                     = 1 << 20,    /* Unused */
    IM_COLOR_FILL               = 1 << 21,
    IM_COLOR_PALETTE            = 1 << 22,
    IM_NN_QUANTIZE              = 1 << 23,
    IM_ROP                      = 1 << 24,
    IM_ALPHA_BLEND_PRE_MUL      = 1 << 25,
} IM_USAGE;
```

Synchronous Operation

---

imsync

```
IM_STATUS imsync(void);
```

RGA asynchronous mode needs to call this interface to wait for the operation to complete.

Other APIs set sync to 0, which is equivalent to glFlush in OpenGL. If you further call imsync, you can achieve the effect of glFinish.

Test cases and debugging methods

In order to help developers get started with the new API more quickly, we will run the demo and parse the demo source code to speed up developers' understanding and use of the API.

Test file description

The input and output binary files for testing need to be prepared in advance. In the /sample/sample_file directory, the default RGBA8888 format source image files are stored and can be used directly.

The Android system must store the source image in the device's /data/ directory, and the Linux system must store the source image in the device's /usr/data directory. The file naming rules are as follows:

```
in%dw%d-h%d-%s.bin
out%dw%d-h%d-%s.bin
```

示例：

`1280×720 RGBA8888的输入图像： in0w1280-h720-rgba8888.bin`

`1280×720 RGBA8888的输出图像： out0w1280-h720-rgba8888.bin`

The parameters are explained as follows:

The input file is in, the output file is out ---> the first %d is the file index, usually 0, used to distinguish files with exactly the same format and width and height ---> the second %d means width, where width generally refers to virtual width ---> the third %d means height, where height generally refers to virtual height ---> the fourth %s is the name of the format.

Some common image formats for preset tests are as follows. For other formats, the corresponding string names can be found in rgaUtils.cpp:

format (Android)	format (Linux)	name
HAL_PIXEL_FORMAT_RGB_565	RK_FORMAT_RGB_565	"rgb565"
HAL_PIXEL_FORMAT_RGB_888	RK_FORMAT_RGB_888	"rgb888"
HAL_PIXEL_FORMAT_RGBA_8888	RK_FORMAT_RGBA_8888	"rgba8888"

format (Android)	format (Linux)	name
HAL_PIXEL_FORMAT_RGBX_8888	RK_FORMAT_RGBX_8888	"rgbx8888"
HAL_PIXEL_FORMAT_BGRA_8888	RK_FORMAT_BGRA_8888	"bgra8888"
HAL_PIXEL_FORMAT_YCrCb_420_SP	RK_FORMAT_YCrCb_420_SP	"crcb420sp"
HAL_PIXEL_FORMAT_YCrCb_NV12	RK_FORMAT_YCbCr_420_SP	"nv12"
HAL_PIXEL_FORMAT_YCrCb_NV12_VIDEO	/	"nv12"
HAL_PIXEL_FORMAT_YCrCb_NV12_10	RK_FORMAT_YCbCr_420_SP_10B	"nv12_10"

The default input image file resolution in the demo is 1280x720 and the format is RGBA8888. You need to prepare a source image file named in0w1280-h720-rgba8888.bin in the /data or /usr/data directory. For the image synthesis mode, you also need to prepare a source image file named in1w1280-h720-rgba8888.bin in the /data or /usr/data directory.

Debugging method description

After running the demo, the printed log is as follows (taking image copy as an example):

The print log in Android is as follows:

```
# rgaImDemo --copy

librga:RGA_GET_VERSION:3.02,3.020000                                    //RGA版本
ctx=0x7ba35c1520,ctx->rgaFd=3
        //RGA上下文
Start selecting mode
im2d copy ..
                        //RGA运行模式
GraphicBuffer check ok
GraphicBuffer check ok
lock buffer ok
open file ok
                        //src文件的状态，如果/data/目录下没有对应文件这里会报错
unlock buffer ok
lock buffer ok
unlock buffer ok
copying .... successfully
               //标志运行成功
open /data/out0w1280-h720-rgba8888.bin and write ok          //输出文件名以及目录
```

The print log in Linux system is as follows:

```
# rgaImDemo --copy

librga:RGA_GET_VERSION:3.02,3.020000                                    //RGA版本
ctx=0x2b070,ctx->rgaFd=3
                //RGA上下文
Rga built version:version:1.00
Start selecting mode
im2d copy ..
                        //RGA运行模式
open file
                                //src文件的状态，如果/usr/data/目录下没有对应文件这里会报错
copying .... Run successfully
        //标志运行成功
open /usr/data/out0w1280-h720-rgba8888.bin and write ok      //输出文件名以及目录
```

When you need to view more detailed logs of RGA operation, the Android system can enable RGA configuration log printing by setting the property vendor.rga.log (sys.rga.log for Android 8 and below):

```
setprop vendor.rga.log 1            打开RGA log打印
logcat -s librga                            开启并过滤log打印
setprop vendor.rga.log 0            关闭RGA log打印
```

In Linux system, you need to open the code core/NormalRgaContext.h, set __DEBUG to 1, and recompile.

```
#ifdef LINUX

-#define __DEBUG 0
+#define __DEBUG 1
```

The general printed log is as follows, which can be uploaded to RedMine for analysis by RK engineers:

The print log in the Android system is as follows:

```
 D librga  : <<<<-------- print rgaLog -------->>>>
 D librga  : src->hnd = 0x0 , dst->hnd = 0x0
 D librga  : srcFd = 11 , phyAddr = 0x0 , virAddr = 0x0
 D librga  : dstFd = 15 , phyAddr = 0x0 , virAddr = 0x0
 D librga  : srcBuf = 0x0 , dstBuf = 0x0
 D librga  : blend = 0 , perpixelAlpha = 1
 D librga  : scaleMode = 0 , stretch = 0;
 D librga  : rgaVersion = 3.020000  , ditherEn =0
 D librga  : srcMmuFlag = 1 , dstMmuFlag = 1 , rotateMode = 0
 D librga  : <<<<-------- rgaReg -------->>>>
 D librga  : render_mode=0 rotate_mode=0
 D librga  : src:[b,0,e1000],x-y[0,0],w-h[1280,720],vw-vh[1280,720],f=0
 D librga  : dst:[f,0,e1000],x-y[0,0],w-h[1280,720],vw-vh[1280,720],f=0
 D librga  : pat:[0,0,0],x-y[0,0],w-h[0,0],vw-vh[0,0],f=0
```

```
 D librga  : ROP:[0,0,0],LUT[0]
 D librga  : color:[0,0,0,0,0]
 D librga  : MMU:[1,0,80000521]
 D librga  : mode[0,0,0,0]
```

The Linux system prints the log as follows:

```
render_mode=0 rotate_mode=0
src:[0,a681a008,a68fb008],x-y[0,0],w-h[1280,720],vw-vh[1280,720],f=0
dst:[0,a6495008,a6576008],x-y[0,0],w-h[1280,720],vw-vh[1280,720],f=0
pat:[0,0,0],x-y[0,0],w-h[0,0],vw-vh[0,0],f=0
ROP:[0,0,0],LUT[0]
color:[0,0,0,0,0]
MMU:[1,0,80000521]
mode[0,0,0,0,0]
gr_color_x [0, 0, 0]
gr_color_x [0, 0, 0]
```

Test case description

- The test path is located in sample/im2d_api_demo in the librga source directory. Developers can modify the demo configuration according to their needs. It is recommended to use the default configuration when running the demo for the first time.
- The compilation of test cases is different for different platforms. The Android platform can be compiled using the 'mm' command. When using cmake to compile librga.so on the Linux platform, the corresponding test cases will be generated in the same directory.
- Transfer the executable file generated by compiling the corresponding test case to the device through adb, add execution permissions, execute the demo, and view the print log.
- Review the output file to check whether it matches your expectations.

Apply for image buffer

The demo provides two buffers for RGA synthesis: Graphicbuffer and AHardwareBuffer. These two buffers are distinguished by the macro USE_AHARDWAREBUFFER.

目录：librga/samples/im2d_api_demo/Android.mk
（line +15）

```
ifeq (1,$(strip $(shell expr $(PLATFORM_SDK_VERSION) \> 25)))
/*USE_AHARDWAREBUFFER为1则使用AHardwareBuffer，为0使用Graphicbuffer*/
LOCAL_CFLAGS += -DUSE_AHARDWAREBUFFER=1
endif
```

Graphicbuffer

Graphicbuffer initialization, filling/clearing, and filling the rga_buffer_t structure are mainly completed through three functions.

```
/*传入src/dst的宽、高、图像格式，初始化Graphicbuffer*/
src_buf = GraphicBuffer_Init(SRC_WIDTH, SRC_HEIGHT, SRC_FORMAT);
dst_buf = GraphicBuffer_Init(DST_WIDTH, DST_HEIGHT, DST_FORMAT);

/*通过枚举值FILL_BUFF/EMPTY_BUFF，执行填充/清空Graphicbuffer*/
GraphicBuffer_Fill(src_buf, FILL_BUFF, 0);
if(MODE == MODE_BLEND)
        GraphicBuffer_Fill(dst_buf, FILL_BUFF, 1);
else
        GraphicBuffer_Fill(dst_buf, EMPTY_BUFF, 1);

/*填充rga_buffer_t结构体:src、dst*/
src = wrapbuffer_GraphicBuffer(src_buf);
dst = wrapbuffer_GraphicBuffer(dst_buf);
```

AHardwareBuffer

The initialization, filling/clearing, and filling of the rga_buffer_t structure of AHardwareBuffer are mainly completed through three functions.

```
/*传入src/dst的宽、高、图像格式，初始化AHardwareBuffer*/
AHardwareBuffer_Init(SRC_WIDTH, SRC_HEIGHT, SRC_FORMAT, &src_buf);
AHardwareBuffer_Init(DST_WIDTH, DST_HEIGHT, DST_FORMAT, &dst_buf);

/*通过枚举值FILL_BUFF/EMPTY_BUFF，执行填充/清空AHardwareBuffer*/
AHardwareBuffer_Fill(&src_buf, FILL_BUFF, 0);
if(MODE == MODE_BLEND)
        AHardwareBuffer_Fill(&dst_buf, FILL_BUFF, 1);
else
        AHardwareBuffer_Fill(&dst_buf, EMPTY_BUFF, 1);

/*填充rga_buffer_t结构体:src、dst*/
src = wrapbuffer_AHardwareBuffer(src_buf);
dst = wrapbuffer_AHardwareBuffer(dst_buf);
```

View help information

Use the following command to get test case help information

```
rgaImDemo -h
rgaImDemo --help
rgaImDemo
```

After running successfully, you can use the demo according to the help information. The printed information is as follows:

```
rk3399_Android10:/ # rgaImDemo
librga:RGA_GET_VERSION:3.02,3.020000
ctx=0x7864d7c520,ctx->rgaFd=3
```

```
================================================================================
==================
   usage: rgaImDemo [--help/-h] [--while/-w=(time)] [--querystring/--
querystring=<options>]
                    [--copy] [--resize=<up/down>] [--crop] [--
rotate=90/180/270]
                    [--flip=H/V] [--translate] [--blend] [--cvtcolor]
                    [--fill=blue/green/red]
         --help/-h     Call help
         --while/w     Set the loop mode. Users can set the number of
cycles by themselves.
         --querystring You can print the version or support information
corresponding to the current version of RGA according to the options.
                       If there is no input options, all versions and
support information of the current version of RGA will be printed.
                       <options>:
                       vendor           Print vendor information.
                       version          Print RGA version, and
librga/im2d_api version.
                       maxinput         Print max input resolution.
                       maxoutput        Print max output resolution.
                       scalelimit       Print scale limit.
                       inputformat      Print supported input formats.
                       outputformat     Print supported output formats.
                       expected         Print expected performance.
                       all              Print all information.
         --copy        Copy the image by RGA.The default is 720p to 720p.
         --resize      resize the image by RGA.You can choose to up(720p-
>1080p) or down(720p->480p).
         --crop        Crop the image by RGA.By default, a picture of
300*300 size is cropped from (100,100).
         --rotate      Rotate the image by RGA.You can choose to rotate
90/180/270 degrees.

         --flip        Flip the image by RGA.You can choice of horizontal
flip or vertical flip.
         --translate   Translate the image by RGA.Default translation
(300,300).
         --blend       Blend the image by RGA.Default, Porter-Duff 'SRC
over DST'.
         --cvtcolor    Modify the image format and color space by RGA.The
default is RGBA8888 to NV12.
         --fill        Fill the image by RGA to blue, green, red, when you
set the option to the corresponding color.
================================================================================
==================
```

All parameter parsing is in directory/librga/demo/im2d_api_demo/args.cpp.

Loop execution demo

Use the following command to execute the demo in a loop. The loop command must be before all parameters. The number of loops is int type. The default interval between each loop is 200ms.

```
rgaImDemo -w6 --copy
```

```
rgaImDemo --while=6 --copy
```

Get RGA version and support information

Use the following command to obtain version and support information:

```
rgaImDemo --querystring
rgaImDemo --querystring=<options>
```

This command has optional options. If there are no options, the default is to select = all. The optional options are as follows:

```
options：
        =vendor                         打印厂商信息
        =version                        打印版本信息
        =maxinput                       打印支持的最大输入分辨率
        =maxoutput                      打印支持的最大输出分辨率
        =scalelimit                     打印支持的缩放倍数
        =inputformat            打印支持的输入格式
        =outputformat           打印支持的输出格式
        =expected                       打印预期性能
        =all                            打印所有信息
```

Code Analysis

Different information is printed according to the parameters passed to main().

```
        /*将main()传参转化为QUERYSTRING_INFO枚举值*/
        IM_INFO = (QUERYSTRING_INFO)parm_data[MODE_QUERYSTRING];
        /*打印querystring()返回的字符串，即所需要的信息*/
        printf("\n%s\n", querystring(IM_INFO));
```

Image Scaling

Use the following command to test image scaling

```
rgaImDemo --resize=up
rgaImDemo --resize=down
```

This function must fill in the optional options. The optional options are as follows:

```
options：
        =up                                     图像分辨率放大至1920x1080
        =down                           图像分辨率缩小至720x480
```

Code Analysis

Zoom in or out according to the parameters (up/down) passed by main(), that is, reinitialize and clear the buffer for different scenes, fill the rga_buffer_t structure, and pass the final rga_buffer_t structure storing the src and dst image data to imresize().

```
        switch(parm_data[MODE_RESIZE])
        {
        /*放大图像*/
                case IM_UP_SCALE :
            /*重新初始化Graphicbuffer为分辨率1920x1080对应大小*/
                        dst_buf = GraphicBuffer_Init(1920, 1080,
DST_FORMAT);
            /*清空buffer*/
                        GraphicBuffer_Fill(dst_buf, EMPTY_BUFF, 1);
            /*重新填充存储dst数据的rga_buffer_t结构体*/
                        dst = wrapbuffer_GraphicBuffer(dst_buf);
                        break;

                case IM_DOWN_SCALE :
            /*重新初始化Graphicbuffer为分辨率1920x1080对应大小*/
                        dst_buf = GraphicBuffer_Init(720, 480, DST_FORMAT);
            /*清空buffer*/
                        GraphicBuffer_Fill(dst_buf, EMPTY_BUFF, 1);
            /*重新填充存储dst数据的rga_buffer_t结构体*/
                        dst = wrapbuffer_GraphicBuffer(dst_buf);
                        break;
        }
        /*将rga_buffer_t格式的结构体src、dst传入imresize()*/
        STATUS = imresize(src, dst);
        /*根据返回的IM_STATUS枚举值打印运行状态*/
        printf("resizing .... %s\n", imStrError(STATUS));
```

Image Cropping

Use the following command to test image cropping

```
rgaImDemo --crop
```

This function has no optional options. By default, the image within the coordinates LT(100,100), RT(400,100), LB(100,400), and RB(400,400) are cropped.

Code Analysis

Assign the size to be cropped to the im_rect structure that stores the src rectangle data, and pass the rga_buffer_t structure that stores the src and dst image data to imcrop().

```
        /*这里通过x、y确定裁剪顶点的坐标，width、height确定裁剪区域大小*/
        src_rect.x      = 100;
        src_rect.y      = 100;
```

```
        src_rect.width  = 300;
        src_rect.height = 300;

        /*将im_rect格式的结构体src_rect与rga_buffer_t格式的结构体src、dst传入
imcrop()*/
        STATUS = imcrop(src, dst, src_rect);
        /*根据返回的IM_STATUS枚举值打印运行状态*/
        printf("cropping .... %s\n", imStrError(STATUS));
```

Image Rotation

Use the following command to test image rotation

```
rgaImDemo --rotate=90
rgaImDemo --rotate=180
rgaImDemo --rotate=270
```

This function must fill in the optional options. The optional options are as follows:

```
options：
        =90                                    图像旋转90°，输出图像分辨率宽高交
换
        =180                                图像旋转180°，输出图像分辨率不变
        =270                                图像旋转270°，输出图像分辨率宽高交换
```

Code Analysis

The rotation angle is determined based on the parameters passed in main() (90/180/270), and the parameters are converted into IM_USAGE enumeration values and passed to imrotate() together with the rga_buffer_t structure that stores the src and dst image data.

```
        /*将main()传参转化为IM_USAGE枚举值*/
        ROTATE = (IM_USAGE)parm_data[MODE_ROTATE];

        /*将标识旋转角度的IM_USAGE枚举值与rga_buffer_t格式的结构体src、dst一同传入
imrotate()*/
        STATUS = imrotate(src, dst, ROTATE);
        /*根据返回的IM_STATUS枚举值打印运行状态*/
        printf("rotating .... %s\n", imStrError(STATUS));
```

Image mirror flip

Use the following command to test the mirror flip

```
rgaImDemo --flip=H
rgaImDemo --flip=V
```

This function must fill in the optional options. The optional options are as follows:

```
options：
        =H                                      图像水平镜像翻转
        =V                                      图像垂直镜像翻转
```

Code Analysis

The image flip direction is determined according to the parameter (H/V) passed by the main function, and the parameter is converted into an IM_USAGE enumeration value and passed to imflip() together with the rga_buffer_t structure that stores the src and dst image data.

```
        /*将main()传参转化为IM_USAGE枚举值*/
        FLIP = (IM_USAGE)parm_data[MODE_FLIP];

        /*将标识镜像反转方向的IM_USAGE枚举值与rga_buffer_t格式的结构体src、dst一同
传入imflip()*/
        STATUS = imflip(src, dst, FLIP);
        /*根据返回的IM_STATUS枚举值打印运行状态*/
        printf("flipping .... %s\n", imStrError(STATUS));
```

Image color fill

Use the following command to test the color filling

```
rgaImDemo --fill=blue
rgaImDemo --fill=green
rgaImDemo --fill=red
```

This function must be filled with optional options. The default fill color is the image within the coordinates LT (100, 100), RT (400, 100), LB (100, 400), RB (400, 400). The optional options are as follows:

```
options：
        =blue                                   图像颜色填充为蓝色
        =green                                  图像颜色填充为绿色
        =red                                    图像颜色填充为红色
```

Code Analysis

The fill color is determined according to the parameters (bule/green/red) passed by the main function, the size to be filled is assigned to the im_rect structure that stores the dst rectangle data, and the parameter is converted into a hexadecimal number of the corresponding color and passed to imfill() together with the rga_buffer_t structure that stores the dst image data.

```
        /*将main()传参转化为对应颜色的16进制数*/
        COLOR = parm_data[MODE_FILL];
```

```
        /*这里通过x、y确定裁剪顶点的坐标，width、height确定填充颜色区域大小*/
        dst_rect.x      = 100;
        dst_rect.y      = 100;
        dst_rect.width  = 300;
        dst_rect.height = 300;

        /*将im_rect格式的结构体dst_rect、对应颜色的16进制数与rga_buffer_t格式的结构体src、dst一同传入imfill()*/
        STATUS = imfill(dst, dst_rect, COLOR);
        /*根据返回的IM_STATUS枚举值打印运行状态*/
        printf("filling .... %s\n", imStrError(STATUS));
```

Image translation

Use the following command to test the image translation operation

```
rgaImDemo --translate
```

This function has no optional options. The default vertex (upper left corner coordinate) is translated to (300,300), that is, translated 300 pixels to the right and then 300 pixels downward.

Code Analysis

Assign the offset to be translated to the im_rect structure that stores the src rectangle data, and pass the rga_buffer_t structure that stores the src and dst image data to imtranslate().

```
        /*这里通过x、y确定平移后图像的顶点的坐标*/
        src_rect.x = 300;
        src_rect.y = 300;

        /*将im_rect格式的结构体src_rect与rga_buffer_t格式的结构体src、dst一同传入imtranslate()*/
        STATUS = imtranslate(src, dst, src_rect.x, src_rect.y);
        /*根据返回的IM_STATUS枚举值打印运行状态*/
        printf("translating .... %s\n", imStrError(STATUS));
```

Image Copy

Test the image copy using the following command

```
rgaImDemo --copy
```

This function has no options. By default, the image with a resolution of 1280x720 and a format of RGBA8888 is copied.

Code Analysis

Pass the rga_buffer_t structure storing the src and dst image data to imcopy().

```
/*rga_buffer_t格式的结构体src、dst传入imcopy()*/
STATUS = imcopy(src, dst);
/*根据返回的IM_STATUS枚举值打印运行状态*/
printf("copying .... %s\n", imStrError(STATUS));
```

Image synthesis

Use the following command to test image synthesis

```
rgaImDemo --blend
```

This function has no optional options, and the default compositing mode is IM_ALPHA_BLEND_DST.

Code Analysis

Pass the rga_buffer_t structure storing the src and dst image data to imblend().

```
/*rga_buffer_t格式的结构体src、dst传入imblend()*/
STATUS = imblend(src, dst);
/*根据返回的IM_STATUS枚举值打印运行状态*/
printf("blending .... %s\n", imStrError(STATUS));
```

Image format conversion

Use the following command to test image format conversion

```
rgaImDemo --cvtcolor
```

This function has no optional options and converts images with a resolution of 1280x720 from RGBA8888 format to NV12 format by default.

Code Analysis

Assign the format to be converted to the member variable format of rga_buffer_t, and pass the rga_buffer_t structure storing the src and dst image data to imcvtcolor().

```
/*将转换前后的格式赋值给对应的rga_buffer_t结构体的成员变量format*/
src.format = HAL_PIXEL_FORMAT_RGBA_8888;
dst.format = HAL_PIXEL_FORMAT_YCrCb_NV12;

/*将需要转换的格式与rga_buffer_t格式的结构体src、dst一同传入imcvtcolor()*/

STATUS = imcvtcolor(src, dst, src.format, dst.format);
/*根据返回的IM_STATUS枚举值打印运行状态*/
printf("cvtcolor .... %s\n", imStrError(STATUS));
```