RGA IM2D API Instruction

File No.: RK-PC-YF-0002

Current Version: V2.1.0

Finish Date: 2022-01-20

Security Class: □Top-Secret □Secret □Internal ■Public

Fuzhou Rockchip Electronics Co., Ltd.

Address: No. 18 Building, A District, No.89,software Boulevard Fuzhou,Fujian,PRC

Website: www.rock-chips.com

Customer service tel.: +86-4007-700-590

Customer service fax: +86-591-83951833

Customer service e-mail: fae@rock-chips.com

Readership

This document is intended for:

- Technical support engineers
- Software development engineers

Revision History

Date	Version	Author	Revision Description
2020/06/24	1.0.0	Chen Cheng, Li Huang	Initial version.
2020/10/16	1.0.1	Chen Cheng, Li Huang, Yu Qiaowei	Update part of the APIs.
2021/12/07	2.0.0	Chen Cheng, Li Huang, Yu Qiaowei	Add RGA3 related support.
2022/01/20	2.1.0	Chen Cheng, Li Huang, Yu Qiaowei	- Update im2d api description. - Updated hardware index description and alignment restrictions. - Add data structure description.

Contents

RGA IM2D API Instruction

- Introductions
  - Design Index
  - Image Format Supported
  - Image Format Alignment Instructions
- API Version Description
  - Version Number Format and Update Rule
    - API Version Number
      - Format
      - Update Rule
  - Version Number Query
    - Query by Strings
    - Log Print
    - Query by API
    - Query by Property
- API Description
  - Overview
  - Obtain RGA Version and Support Information
    - querystring
  - Image Buffer Preprocessing
    - importbuffer_T
    - releasebuffer_handle
    - wrapbuffer_handle
  - Image Copy
    - imcopy
  - Image Resizing and Image Pyramid
    - imresize
    - impyramid
  - Image Cropping
    - imcrop
  - Image Rotation
    - imrotate
  - Image Mirror Flip
    - imfilp
  - Image Color Filling, Memory R, Graph Drawing
    - imfill/imreset/imdraw
  - Image Translation
    - imtranslate
  - Image Blending
    - imblend/imcomposite
  - Color Key
    - imcolorkey
  - Image Format Conversion
    - imcvtcolor
  - Pre-processing of NN Operation Points (Quantization)
    - imquantize
  - Image ROP
    - imrop
  - Image Process
    - improcess
  - Parameter Check
    - imcheck
  - Synchronous operation
    - imsync
  - Thread Context Configuration
    - imconfig
- Data Structure
  - Overview
  - Detailed Descriptions
    - rga_buffer_t
    - im_rect
    - im_opt_t
    - rga_buffer_handle_t
    - im_handle_param_t
    - im_nn_t
    - im_colorkey_range
- Test Cases and Debugging Methods
  - Test File Description
  - Debugging Method Description
  - Test Case Descriptions
    - Apply for Image Buffer
      - Graphicbuffer
      - AHardwareBuffer
    - Viewing Help Information
    - Executing Demo in Loop

Obtain RGA Version and Support Information
Code Parsing
Image Resizing
Code Parsing
Image Cropping
Code Parsing
Image Rotation
Code Parsing
Image Mirror Flip
Code Parsing
Image Color Fill
Code Parsing
Image Translation
Code Parsing
Image Copying
Code Parsing
Image Blending
Code Parsing
Image Format Conversion
Code Parsing

Introductions

RGA (Raster Graphic Acceleration Unit) is an independent 2D hardware accelerator that can be used to speed up point/line drawing, perform image resizing, rotation, bitBlt, alpha blending and other common 2D graphics operations.

Design Index

Version	Codename	Chip	Source		Destination		Function	Pixels/Cycle
			min	max	min	max		
RGA1	Pagani	RK3066	2x2	8192x8192	2x2	2048x2048	90/180/270 Rotate X/Y Mirror Crop 1/2~8 scale Alpha blend Color key Color fill ROP	1
	Jaguar Plus	RK3188						
	Beetles	RK2926/2928						
	Beetles Plus	RK3026/3028						
RGA1_plus	Audi	RK3128	2x2	8192x8192	2x2	2048x2048	90/180/270 Rotate X/Y Mirror Crop 1/2~8 scale Alpha blend Color key Color fill Color palette	1
	Granite	Sofia 3gr						
RGA2	Lincoln	RK3288/3288w	2x2	8192x8192	2x2	4096x4096	90/180/270 Rotate X/Y Mirror Crop 1/16~16 scale Alpha blend Color key Color fill Color palette ROP	2
	Capricorn	RK3190						
RGA2-Lite0	Maybach	RK3368	2x2	8192x8192	2x2	4096x4096	90/180/270 Rotate X/Y Mirror Crop 1/8~8 scale Alpha blend Color key Color fill Color palette ROP	2
	BMW	RK3366						
RGA2-Lite1	Benz	RK3228	2x2	8192x8192	2x2	4096x4096	90/180/270 Rotate X/Y Mirror Crop 1/8~8 scale Alpha blend Color key Color fill Color palette	2
	Infiniti	RK3228H						
	Gemini	RK3326						
	Lion	RK1808						
RGA2-Enhance	Mclaren	RK3399	2x2	8192x8192	2x2	4096x4096	90/180/270 Rotate X/Y Mirror Crop 1/16~16 scale Alpha blend Color key	2
	Mercury	RK1108						
	Puma	RV1126/RV1109						

	skylarkV2	RK3566/RK3568					Color fill Color palette ROP(NA for 1108/1109) NN quantize(NA for 3399/1108) osd(only 1106/1103)	
	Orion	RK3588						
	Otter	RV1106/1103						
RGA3	Orion	RK3588	128x128	8176x8176	128x128	8128x8128	90/180/270 Rotate X/Y Mirror Crop 1/8~8 scale Alpha blend Color key FBC	3 (by pass) 2 (scale)

> The expected performance is calculated at the default RGA frequency. The actual performance is affected by the memory frequency. The data above is for reference only.

Image Format Supported

- Pixel Format conversion, BT.601/BT.709/BT.2020(only RGA3)
- Dither operation

Version	Codename	Chip	Input Data Format	Output Data Format
RGA1	Pagani	RK3066	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (planar/semi-planar) YUV422 8bit (planar/semi-planar) BPP8/BPP4/BPP2/BPP1	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (planar/semi-planar, only for Blur/sharpness) YUV422 8bit (planar/semi-planar, only for Blur/sharpness)
	Jaguar Plus	RK3188		
	Beetles	RK2926/2928		
	Beetles Plus	RK3026/3028		
RGA1_plus	Audi	RK3128	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (planar/semi-planar) YUV422 8bit (planar/semi-planar) BPP8/BPP4/BPP2/BPP1	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (planar/semi-planar, only for normal Bitblt without alpha) YUV422 8bit (planar/semi-planar, only for normal Bitblt without alpha)
	Granite	Sofia 3gr		
RGA2	Lincoln	RK3288/3288w	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (planar/semi-planar) YUV422 8bit (planar/semi-planar) BPP8/BPP4/BPP2/BPP1 (only for color palette)	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (planar/semi-planar) YUV422 8bit (planar/semi-planar)
	Capricorn	RK3190		
RGA2-Lite0	Maybach	RK3368	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (planar/semi-planar) YUV422 8bit (planar/semi-planar) BPP8/BPP4/BPP2/BPP1 (only for color palette)	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (planar/semi-planar) YUV422 8bit (planar/semi-planar)
	BMW	RK3366		
RGA2-Lite1	Benz	RK3228	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (planar/semi-planar) YUV422 8bit (planar/semi-planar) YUV420 10bit (planar/semi-planar) YUV422 10bit (planar/semi-planar) BPP8/BPP4/BPP2/BPP1 (only for color palette)	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (planar/semi-planar) YUV422 8bit (planar/semi-planar)
	Infiniti	RK3228H		
	Gemini	RK3326		
	Lion	RK1808		
RGA2-Enhance	Mclaren	RK3399	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888

			RGB/BGR565 YUV420 8bit (planar/semi-planar) YUV422 8bit (planar/semi-planar) YUV420 10bit (planar/semi-planar) YUV422 10bit (planar/semi-planar) BPP8/BPP4/BPP2/BPP1 (only for color palette)	RGB/BGR565 YUV420 8bit (planar/semi-planar/packed) YUV422 8bit (planar/semi-planar/packed)
	Mercury	RK1108		
	Puma	RV1126/ RV1109	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (planar/semi-planar) YUV422 8bit (planar/semi-planar/packed) YUV420 10bit (planar/semi-planar) YUV422 10bit (planar/semi-planar) BPP8/BPP4/BPP2/BPP1 (only for color palette)	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (planar/semi-planar/packed) YUV422 8bit (planar/semi-planar/packed) YUV400 Y4/Y1
	skylarkV2	RK3566/RK3568		
	Orion	RK3588		
	Otter	RV1106/1103		
RGA3	Orion	RK3588	RGBA/BGRA/ARGB/ABGR8888 RGB/BGR888 RGB/BGR565 YUV420 8bit (semi-planar) YUV422 8bit (semi-planar/packed) YUV420 10bit (semi-planar) YUV422 10bit (semi-planar)	RGBA/BGRA/ARGB/ABGR8888 RGB/BGR888 RGB/BGR565 YUV420 8bit (semi-planar) YUV422 8bit (semi-planar/packed) YUV420 10bit (semi-planar) YUV422 10bit (semi-planar)

> Note: Y4 format is 2 to the 4th power grayscale, Y400 format is 2 to the 8th power grayscale.

Image Format Alignment Instructions

Version	Byte_stride	Format	Alignment
RGA1 RGA1_Plus	4	RGBA/BGRA/ARGB/ABGR8888	width stride no alignment requirements
		RGB/BGR888	width stride must be aligned to 4
		RGB/BGR565 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551	width stride must be aligned to 2
		YUV420 8bit (planar/semi-planar) YUV422 8bit (planar/semi-planar)	width stride must be aligned to 4，x_offset、y_offset、width、height、height stride must be aligned to 2
RGA2 RGA2_Lite0 RGA2_Lite1 RGA2_Enhance	4	RGBA/BGRA/ARGB/ABGR8888	width stride no need to align
		RGB/BGR888	width stride must be aligned to 4
		RGB/BGR565 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551	width stride must be aligned to 2
		YUV420 8bit (planar/semi-planar/packed) YUV422 8bit (planar/semi-planar/packed) YUV400 Y4/Y1	width stride must be aligned to 4，x_offset、y_offset、width、height、height stride must be aligned to 2
		YUV420 10bit (planar/semi-planar) YUV422 10bit (planar/semi-planar)	width stride must be aligned to 16，x_offset、y_offset、width、height、height stride must be aligned to 2
RGA3	16	RGBA/BGRA/ARGB/ABGR8888	width stride must be aligned to 4
		RGB/BGR888	width stride must be aligned to 16
		RGB/BGR565	width stride must be aligned to 8
		YUV420 8bit (semi-planar) YUV422 8bit (semi-planar/packed)	width stride must be aligned to 16，x_offset、y_offset、width、height、height stride must be aligned to 2
		YUV420 10bit YUV422 10bit	width stride must be aligned to 64，x_offset、y_offset、width、height、height stride must be aligned to 2
		FBC mode	In addition to the alignment required for the formats mentioned above，width、height must be aligned to 16

> Alignment requirement formula: lcm(bpp，byte_stride * 8) / pixel_stride.
>
> When loaded with multiple versions of hardware, chip platform constraints according to the most strict alignment requirements.

API Version Description

RGA's support library librga.so updates the version number according to certain rules, indicating the submission of new features, compatibility, and bug fixes, and provides several ways to query the version number, so that developers can clearly determine whether the current library version is suitable for the current development environment when using librga.so. Detailed version update logs can be found in CHANGLOG.md in the root directory of source code.

Version Number Format and Update Rule

API Version Number

Format

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

> example:
>
> 1.0.0_[0]

Update Rule

Name	Rule
major	Major version number, when submitting a version that is not backward compatible.
minor	Minor version number, when the functional API with backward compatibility is added.
revision	Revision version number, when submitting backward compatible feature additions or fatal bug fixes.
build	Compile version number, when backward compatibility issue is fixed.

Version Number Query

Query by Strings

Take Android R 64bit as example:

```
:/# strings vendor/lib64/librga.so |grep rga_api |grep version
rga_api version 1.0.0_[0]
```

Log Print

Version number is printed when each process first calls RGA API.

```
rockchiprga: rga_api version 1.0.0_[0]
```

Query by API

Call the following API to query the code version number, compilation version number, and RGA hardware version information. For details, see **API Description**.

```
querystring(RGA_VERSION);
```

> String format is as follows:
>
> RGA_api version : v1.0.0_[0]
> RGA version : RGA_2_Enhance

Query by Property

This method is supported only by Android system, and the property takes effect only after an existing process calls RGA.

```
:/# getprop |grep rga
[vendor.rga_api.version]: [1.0.0_[0]]
```

API Description

RGA library librga.so realizes 2D graphics operations through the image buffer structure rga_info configuration. In order to obtain a better development experience, the common 2D image operation API is further encapsulated. The new API mainly contains the following features:

- API definitions refer to common 2D graphics API definitions in opencv/matlab, reducing the learning cost of secondary development.

- To eliminate compatibility problems caused by RGA hardware version differences, RGA query is added. Query mainly includes version information, large resolution and image format support.
- Add improcess API for 2D image compound operations. Compound operations are performed by passing in a set of predefined usage.
- Before performing image operation, the input and output image buffers need to be processed. The wrapbuffer_T API is called to pass the input and output image information to rga_buffer_t, which contains information such as resolution and image format.

Overview

The software support library provides the following API, asynchronous mode only supports C++ implementation.

- **querystring**: Query information about the RGA hardware version and supported functions of chip platform, return a string.
- **importbuffer_T**: Import external buffer into RGA driver to achieve hardware fast access to discontinuous physical addresses (dma_fd, virtual address).
- **releasebuffer_handle**: Remove reference and map of the external buffer from inside the RGA driver.
- **wrapbuffer_handle** Quickly encapsulate the image buffer structure (rga_buffer_t) .
- **imcopy**: Call RGA for fast image copy.
- **imresize**: Call RGA for fast image resize.
- **impyramind**: Call RGA for fast image pyramid.
- **imcrop**: Call RGA for fast image cropping.
- **imrotate**: Call RGA for fast image rotation.
- **imflip**: Call RGA for fast image flipping.
- **imfill**: Call RGA for fast image filling.
- **imtranslate**: Call RGA for fast image translation.
- **imblend**: Call RGA for double channel fast image blending.
- **imcomposite**: Call RGA for three-channel fast image composite.
- **imcolorkey**: Call RGA for fast image color key.
- **imcvtcolor**: Call RGA for fast image format conversion.
- **imquantize**: Call RGA for fast image operation point preprocessing (quantization).
- **imrop**: Call RGA for fast image ROP.
- **improcess**: Call RGA for fast image compound processing.
- **imcheck**: Verify whether the parameters are valid and whether the current hardware supports the operation.
- **imsync**: Synchronize task completion status in asynchronous mode.
- **imconfig**: Add default configuration to current thread context.

Obtain RGA Version and Support Information

querystring

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

> Query RGA basic information and resolution format.

Parameters	Description
name	RGA_VENDOR - vendor information RGA_VERSION - RGA version RGA_MAX_INPUT - max input resolution RGA_MAX_OUTPUT - max output resolution RGA_BYTE_STRIDE - stride alignment requirements RGA_SCALE_LIMIT - scale limit RGA_INPUT_FORMAT - input formats supported RGA_OUTPUT_FORMAT - output formats supported RGA_EXPECTED - expected performance RGA_ALL - print all information

Returns a string describing properties of RGA.

Image Buffer Preprocessing

importbuffer_T

> For external memory that requires RGA processing, you can use importbuffer_T to map physical address of buffer to RGA driver and obtain the corresponding address of buffer, facilitating the subsequent stable and fast RGA call to complete the work.

Parameters(T)	Data Type	Description
virtual address	void *	image buffer virtual address
physical address	uint64_t	contiguous physical address of image buffer
fd	int	image buffer DMA file descriptor
GraphicBuffer handle	buffer_handle_t	image buffer handle, containing buffer address, file descriptor, resolution and format
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

> Performance varies when different buffer types call RGA, and the performance order is shown below:
>
> physical address > fd > virtual address
>
> The recommended buffer type is fd.

```
IM_API rga_buffer_handle_t importbuffer_fd(int fd, im_handle_param_t *param);
IM_API rga_buffer_handle_t importbuffer_virtualaddr(void *va, im_handle_param_t *param);
IM_API rga_buffer_handle_t importbuffer_physicaladdr(uint64_t pa, im_handle_param_t *param);
```

Parameter	Description
fd/va/pa	**[required]** external buffer
param	**[required]** configure buffer parameters

```
IM_API rga_buffer_handle_t importbuffer_GraphicBuffer_handle(buffer_handle_t hnd);
IM_API rga_buffer_handle_t importbuffer_GraphicBuffer(sp<GraphicBuffer> buf);
IM_API rga_buffer_handle_t importbuffer_AHardwareBuffer(AHardwareBuffer *buf);
```

Parameter	Description
hnd/buf	**[required]** external buffer

```
IM_API rga_buffer_handle_t importbuffer_fd(int fd, int width, int height, int format);
IM_API rga_buffer_handle_t importbuffer_virtualaddr(void *va, int width, int height, int format);
IM_API rga_buffer_handle_t importbuffer_physicaladdr(uint64_t pa, int width, int height, int format);
```

Parameter	Description
fd/va/pa	**[required]** external buffer
width	**[required]** pixel width stride of the image buffer
height	**[required]** pixel height stride of the image buffer
format	**[required]** pixel format of the image buffer

Returns rga_buffer_handle_t to describe the memory handle.

releasebuffer_handle

> After finishing calling RGA using external memory, you need to call releasebuffer_handle through memory handle to remove the mapping and binding between buffer and RGA driver, and release the resource inside RGA driver.

```
IM_API IM_STATUS releasebuffer_handle(rga_buffer_handle_t handle);
```

Return IM_STATUS_SUCCESS on success or else negative error code.

wrapbuffer_handle

> In IM2D library API parameters, input image and output image should support multiple types, which mainly include memory, image format, image width and height. Before performing corresponding image operation, you need to call wrapbuffer_handle to convert the input and output image parameters into rga_buffer_t structure as the input parameter of user API.

```
rga_buffer_t wrapbuffer_handle(rga_buffer_handle_t handle,
                               int width,
                               int height,
                               int wstride = width,
                               int hstride = height,
                               int format);
```

Parameter	Description
handle	**[required]** RGA buffer handle
width	**[required]** pixel width of the image that needs to be processed
height	**[required]** pixel height of the image that needs to be processed
wtride	**[optional]** pixel width stride of the image
hstride	**[optional]** pixel width stride of the image
format	**[required]** pixel format

Returns a rga_buffer_t to desribe image information.

Image Copy

imcopy

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

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

Parameter	Description
src	**[required]** input image
dst	**[required]** output image
sync	**[optional]** wait until operation complete
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

Return IM_STATUS_SUCCESS on success or else negative error code.

Image Resizing and 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);
```

> According to different scenario, you can choose to configure dst to describe the output image size of resizing, or configure the scale factor fx/fy to resize at a specified ratio. When dst and fx/fy are configured at the same time, the calculated result of fx/fy is used as the output image size.
>
> Only hardware version RGA1/RGA1 plus supports interpolation configuration.
>
> Note: When resizing with fx/fy, format such as YUV that requires width and height alignment will force downward alignment to meet the requirements. Using this feature may affect the expected resizing 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
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

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 by 1/2 or twice, depending on the direction width and height.

Parameters	Description
src	**[required]** input image
dst	**[required]** output image;
direction	**[required]** scale mode IM_UP_SCALE —— up scale IM_DOWN_SCALE —— down scale
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

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);
```

> Perform image clipping by specifying the size of the Rect 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
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

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 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
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

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);
```

> Support image to do horizontal, vertical mirror flip.

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
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

Return IM_STATUS_SUCCESS on success or else negative error code.

Image Color Filling, Memory R, Graph Drawing

imfill/imreset/imdraw

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

> Color-fill specified region rect of RGBA image. Color parameters from high to low are respectively R，G，B，A. For example, red: color = 0xff000000.

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

> For RGBA image, set the memory of specified region rect to specified color. Color parameters from high to low are respectively R，G，B，A. For example, red: color = 0xff000000.

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

> Paint the specified region rect of RGBA image with specified color. Color parameters from high to low are respectively R，G，B，A. For example, red: color = 0xff000000.

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

Parameter	Description
src	**[required]** input image
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
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

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)
```

> Image translation. Move to (x, y) position, the width and height of src and dst must be the same, the excess part will be clipped.

Parameter	Description
src	**[required]**input image
dst	**[required]** output image
x	**[optional]** horizontal translation
y	**[optional]** vertical translation
sync	**[optional]** wait until operation complete
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

Return IM_STATUS_SUCCESS on success or else negative error code.

Image Blending

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 A+B -> B image two-channel blending mode to perform Alpha superposition for foreground image (srcA channel) and background image (dst channel) according to the configured blending model, and output the blending result to 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 A+B -> C image three-channel blending mode to perform Alpha superposition for foreground image (srcA channel) and background image (srcB channel) according to the configured blending model, and output the blending result to dst channel.

mode in the two image blending modes can be configured with different **Porter-Duff blending model**:

> Before describing the Porter-Duff blending model, give the following definitions:
>
> - S -**Marks the source image in two blended images**，namely the foreground image, short for source.
> - D -**Marks the destination image in two blended images**，namely the background image, short for destination.
> - R -**Marks the result of two images blending**，short for result.
> - c -**Marks the color of the pixel**，the RGB part of RGBA, which describes the color of the image itself, short for color.（**Note**，Color values (RGB) in the Porter-Duff blending model are all left-multiplied results, that is, the product of original color and transparency. If the color values are not left-multiplied, pre-multiplied operations（Xc = Xc * Xa） are required.)
> - a -**Marks the Alpha of the pixel**，Namely the A part of RGBA, describe the transparency of the image itself, short for Alpha.
> - f -**Marks factors acting on C or A**，short for factor.
>
> The core formula of Porter-Duff blending model is as follows:
>
> Rc = Sc * Sf + Dc * Df;

> that is： result color = source color * source factor + destination color * destination factor.
>
> Ra = Sa * Sf + Da * Df;
>
> that is： result Alpha = source Alpha * source factor + destination Alpha * destination factor.

RGA supports following blending 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】Image blending model does not support the YUV format image blending, the YUV format is not support in dst image of imblend , the YUV format is not support in srcB image of imcomposite.

Parameter	Description
srcA	**[required]** input image A
srcB	**[required]** input image B
dst	**[required]** output image
mode	**[optional]** blending mode: IM_ALPHA_BLEND_SRC —— SRC mode IM_ALPHA_BLEND_DST —— DST mode IM_ALPHA_BLEND_SRC_OVER —— SRC OVER mode IM_ALPHA_BLEND_DST_OVER —— DST OVER mode IM_ALPHA_BLEND_PRE_MUL —— Enable premultipling. When premultipling is required, this identifier must be processed with other mode identifiers, then assign to mode
sync	**[optional]** wait until operation complete
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

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)
```

> Color Key is to preprocesses the source image, zeros the alpha component of pixels that meet the Color Key filtering conditions, wherein the Color Key filtering conditions are non-transparent color values, and performs the alpha blending mode between the preprocessed source image and the destination image.
>
> This mode only supports the Color Key operation on the source image (src) region of the image for the set Color range, and overlays on the destination image (dst) region.

> IM_ALPHA_COLORKEY_NORMAL is the normal mode, that is, the colors in the set color range are used as the filtering condition. The Alpha components of pixels in this color range are set zero; IM_ALPHA_COLORKEY_INVERTED is inverted.

Parameters	Range	Description
max	0x0 ~ 0xFFFFFFFF	The max color range to cancel/scratch, arranged as ABGR
min	0x0 ~ 0xFFFFFFFF	The min color range to cancel/scratch, arranged as ABGR

parameter	Description
src	**[required]** input image
dst	**[required]** output image
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
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

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)
```

> Image format conversion, specific format support varies according to soc, please refer to the **Image Format Supported** section.
>
> The format can be set by rga_buffer_t, or configure the input image and output image formats respectively by 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
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

Return IM_STATUS_SUCCESS on success or else negative error code.

Pre-processing of NN Operation Points (Quantization)

imquantize

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

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

Formula:

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

Parameters range:

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

Parameter	Description
src	**[required]** input image
dst	**[required]** output image
nn_info	**[required]** rga_nn_t struct configures the offset and scale of the three RGB channels respectively 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
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

Return IM_STATUS_SUCCESS on success or else negative error code

Image ROP

imrop

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

> Perform ROP, and, or, not operations on two images

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
release_fence_fd	**[optional]**Used in async mode, as a parameter of imsync()

Return IM_STATUS_SUCCESS on success or else negative error code

Image Process

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 compound operation. Other APIs are developed based on this API, improcess can achieve more complex compound operations.
>
> Image processes are configured by usage.

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

usage definitions:

```
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;
```

```
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 acquire_fence_fd,
                    int *release_fence_fd,
                    im_opt_t *opt,
                    int usage)
```

> RGA image compound operation. Other APIs are developed based on this API, improcess can achieve more complex compound operations.
>
> Image processes are configured by usage.

Parameter	Description
src	**[required]** input imageA
dst	**[required]** output image
pat	**[required]** input imageB
srect	**[required]** src crop region
drect	**[required]** dst crop region
prect	**[required]** pat crop region
acquire_fence_fd	**[required]** Used in async mode, run the job after waiting foracquire_fence signal
release_fence_fd	**[required]** Used in async mode, as a parameter of imsync()
opt	**[required]** operation options typedef struct im_opt { int color; im_colorkey_range colorkey_range; im_nn_t nn; int rop_code; int priority; int core; } im_opt_t;
usage	**[required]** image operation usage

Parameter Check

imcheck

```
IM_API IM_STATUS imcheck(const rga_buffer_t src, const rga_buffer_t dst,
                         const im_rect src_rect, const im_rect dst_rect,
                         const int mode_usage);
IM_API IM_STATUS imcheck_composite(const rga_buffer_t src, const rga_buffer_t dst, const rga_buffer_t pat,
                                   const im_rect src_rect, const im_rect dst_rect, const im_rect pat_rect,
                                   const int mode_usage);
```

> After RGA parameters are configured, users can use this API to verify whether the current parameters are valid and determine whether the hardware supports them based on the current hardware conditions.
>
> Users are advised to use this API only during development and debugging to avoid performance loss caused by multiple verification.

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

Return IM_STATUS_NOERROR on success or else negative error code.

Synchronous operation

imsync

```
IM_STATUS imsync(int fence_fd);
```

> RGA asynchronous mode requires this API to be called, passing the returned release_fence_fd as parameter.
>
> Other API enable asynchronous call mode when sync is set to 0, which is equivalent to glFlush in opengl. Further calls to imsync is equivalent to glFinish.

Parameter	Description
fence_fd	**[required]** fence_fd to wait

Return IM_STATUS_SUCCESS on success or else negative error code.

Thread Context Configuration

imconfig

```
IM_STATUS  imconfig(IM_CONFIG_NAME name, uint64_t value);
```

> The context for the current thread is configured through different configuration name, which will be the default configuration for the thread.
>
> The thread context configuration has a lower priority than the parameter configuration of the API. If no related parameters are configured for API, the local call uses the default context configuration. If related parameters are configured for API, the call uses the API parameter configuration.

Parameter	Description
name	**[required]** context config name： IM_CONFIG_SCHEDULER_CORE —— Specify the task processing core IM_CONFIG_PRIORITY —— Specify the task priority IM_CHECK_CONFIG —— Check enable
value	**[required]** config value IM_CONFIG_SCHEDULER_CORE : IM_SCHEDULER_RGA3_CORE0 IM_SCHEDULER_RGA3_CORE1 IM_SCHEDULER_RGA2_CORE0 IM_SCHEDULER_RGA3_DEFAULT IM_SCHEDULER_RGA2_DEFAULT IM_CONFIG_PRIORITY: 0 ~ 6 IM_CHECK_CONFIG: TRUE FALSE

> Note：Permissions of priority and core are very high. Improper operations may cause system crash or deadlock. Therefore, users are advised to configure them only during development and debugging. Users are not advised to perform this configuration in actual product

Return IM_STATUS_SUCCESS on success or else negative error code

Data Structure

This section describes the data structures involved in API in detail.

Overview

Data structure	Description
rga_buffer_t	image buffer information
im_rect	the actual operating area of the image
im_opt_t	image manipulation options
rga_buffer_handle_t	RGA driver image buffer handle
im_handle_param_t	buffer parameters of image to be imported
im_context_t	default context for the current thread
im_nn_t	operation point preprocessing parameters
im_colorkey_range	Colorkey range

Detailed Descriptions

rga_buffer_t

- **descriptions**

Buffer information of image with single channel.

- **path**

im2d_api/im2d.h

- **definitions**

```
typedef struct {
    void* vir_addr;                 /* virtual address */
    void* phy_addr;                 /* physical address */
    int fd;                         /* shared fd */
    rga_buffer_handle_t handle;     /* buffer handle */

    int width;                      /* width */
    int height;                     /* height */
    int wstride;                    /* wstride */
    int hstride;                    /* hstride */
    int format;                     /* format */

    int color_space_mode;           /* color_space_mode */
    int global_alpha;               /* global_alpha */
    int rd_mode;
} rga_buffer_t;
```

Parameter	Description
vir_addr	Virtual address of image buffer.
phy_addr	Contiguous physical address of the image buffer.
fd	File descriptor of image buffer DMA.
handle	Import handle corresponding to the image buffer of the RGA driver.
width	The width of the actual operating area of image,in pixels.
height	The height of the actual operating area of image,in pixels.
wstride	The stride of the image width, in pixels.
hstride	The stride of the image height, in pixels.
format	Image format.
color_space_mode	Image color space mode.
global_alpha	Global Alpha configuration.
rd_mode	The mode in which the current channel reads data.

- **Note**

Simply selects either one of vir_addr、phy_addr、fd、handle as the description of image buffer, if multiple values are assigned, only one of them is selected as the image buffer description according to the default priority, which is as follows：handle > phy_addr > fd > vir_addr.

im_rect

- **descriptions**

Describes the actual operation area of image with single channel.

- **path**

im2d_api/im2d.h

- **definitions**

```
typedef struct {
    int x;        /* upper-left x */
    int y;        /* upper-left y */
    int width;    /* width */
    int height;   /* height */
} im_rect;
```

Parameters	Description
x	The starting abscissa of the actual operation area of the image, in pixels.
y	The starting ordinate of the actual operating area of an image, in pixels.
width	The width of the actual operating area of the image, in pixels.
height	The height of the actual operating area of the image, in pixels.

- **Note**

The actual operating area cannot exceed the image size, i.e（x + width）<= wstride，（y + height）<= hstride。

im_opt_t

- **description**

Describes operation options of current image.

- **path**

im2d_api/im2d.h

- **definitions**

```
typedef struct im_opt {
    int color;                          /* color, used by color fill */
    im_colorkey_range colorkey_range;   /* range value of color key */
    im_nn_t nn;
    int rop_code;

    int priority;
    int core;
} im_opt_t;
```

Parameter	Description
color	Image color-fill configuration.
colorkey_range	Colorkey range configuration.
nn	Operation point preprocessing (quantization) configuration.
rop_code	ROP operation code configuration.
priority	Current task priority configuration.
core	Specify the hardware core of current task.

- **Note**

Permissions of priority and core are very high. Improper operations may cause system crash or deadlock. Therefore, users are advised to configure them only during development and debugging.Users are not advised to perform this configuration in actual product.

rga_buffer_handle_t

- **description**

RGA driver image buffer handle.

- **path**

include/rga.h

- **definitions**

```
typedef int rga_buffer_handle_t;
```

- **Note**

null

im_handle_param_t

- **description**

Describe parameters of the image buffer to be imported.

- **path**

im2d_api/im2d.h

include/rga.h

- **definitions**

```
typedef struct rga_memory_parm im_handle_param_t;

struct rga_memory_parm {
    uint32_t width_stride;
    uint32_t height_stride;
    uint32_t format;
};
```

Parameter	Description
width_stride	Describes the horizontal stride of the image buffer to be imported, in pixels.
height_stride	Describes the vertical stride of the image buffer to be imported, in pixels.
format	Describes the format of the buffer of the image to be imported.

- **Note**

If the actual size of buffer memory is smaller than the configured size, the importbuffer_T API error occurs.

im_nn_t

- **description**

Parameter of operation point preprocessing (quantization).

- **path**

im2d_api/im2d.h

- **definitions**

```
typedef struct im_nn {
    int scale_r;                /* scaling factor on R channal */
    int scale_g;                /* scaling factor on G channal */
    int scale_b;                /* scaling factor on B channal */
    int offset_r;               /* offset on R channal */
    int offset_g;               /* offset on G channal */
    int offset_b;               /* offset on B channal */
} im_nn_t;
```

Parameter	Description
scale_r	Scaling factor on red channel.
scale_g	Scaling factor on green channel.
scale_b	Scaling factor on blue channel.
offset_r	Offset on red channel.
offset_g	Offset on green channel.
offset_b	Offset on blue channel.

- **Note**

null

im_colorkey_range

- **description**

Colorkey range.

- **path**

im2d_api/im2d.h

- **definitions**

```
typedef struct {
    int max;                    /* The Maximum value of the color key */
    int min;                    /* The minimum value of the color key */
} im_colorkey_range;
```

Parameter	Description
max	The Maximum value of the color key.
min	The minimum value of the color key.

- **Note**

null

Test Cases and Debugging Methods

> In order to make developers get started with the above new API more quickly, here by running demo and parsing the source code to help developers to understand and use the API.

Test File Description

> Input and output binary file for testing should be prepared in advance. The default source image file in RGBA8888 format is stored in directory /sample/sample_file.
>
> In Android system, the source image should be stored in /data/ directory of device，in Linux system, the source image should be stored in/usr/data directory of device. The file naming rules are as follows:

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

Example:
1280x720 RGBA8888 input image: in0w1280-h720-rgba8888.bin
1280x720 RGBA8888 output image: out0w1280-h720-rgba8888.bin
```

Parameter descriptions：

> The input is in ,the output is out.
> --->The first%d is the index of files, usually 0, used to distinguish files of the same format, width and height.
> --->The second%d is width, usually indicates virtual width.
> --->The third%d is height, usually indicates virtual height.
> --->The fourth%s is the name of format.
>
> Some common image formats for preset tests are as follows. You can view names of other formats 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"
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 resolution of input image of demo is 1280x720, format is RGBA8888, in0w1280-h720-rgba8888.bin source image should be prepared in advance in the /data or /usr/data directory, in1w1280-h720-rgba8888.bin source image should be additionally prepared in advance in the /data or /usr/data directory in image blending mode.

Debugging Method Description

> After running demo, print log as follows (in image copying, for example):
>
> Log is printed in Android system as follows:

```
# rgaImDemo --copy

librga:RGA_GET_VERSION:3.02,3.020000                    //RGA version
ctx=0x7ba35c1520,ctx->rgaFd=3                           //RGA context
Start selecting mode
im2d copy ..                                            //RGA running mode
GraphicBuffer check ok
GraphicBuffer check ok
lock buffer ok
open file ok                                            //src file status, if there is no corresponding file
in /data/ directory, an error will be reported here
unlock buffer ok
lock buffer ok
unlock buffer ok
copying .... successfully                               //indicates successful running
open /data/out0w1280-h720-rgba8888.bin and write ok      //output filename and directory
```

> Log is printed in Linux system as follows:

```
# rgaImDemo --copy

librga:RGA_GET_VERSION:3.02,3.020000                    //RGA version
ctx=0x2b070,ctx->rgaFd=3                                //RGA context
Rga built version:version:1.00
Start selecting mode
im2d copy ..                                            //RGA running mode
open file                                               //src file status, if there is no corresponding file
in /usr/data/ directory, an error will be reported here
copying .... Run successfully                           //indicates successful running
open /usr/data/out0w1280-h720-rgba8888.bin and write ok //output filename and directory
```

> To view more detailed logs of RGA running, the Android system can enable RGA configuration log print by setting vendor.rga.log (Android 8 and below is sys.rga.log):

```
setprop vendor.rga.log 1        enable RGA log print
logcat -s librga                enable and filter log print
setprop vendor.rga.log 0         cancel RGA log print
```

> In Linux system, you should open core/NormalRgaContext.h, set __DEBUG to 1 and recompile.

```
#ifdef LINUX

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

> Generally, the printed log is as follows, which can be uploaded to RedMine for analysis by relevant engineers of RK:
>
> Log is printed in Android system 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]
```

> Log is printed in Linux system 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 Descriptions

- The test path is sample/im2d_API_demo. Developers can modify the demo configuration as required. It is recommended to use the default configuration when running demo for the first time.
- The compilation of test cases varies on different platforms. On the Android platform, the 'mm' command can be used to compile the test cases. On the Linux platform, when librga.so is compiled using cmake, the corresponding test cases will be generated in the same directory
- Import the executable file generated by compiling the corresponding test case into the device through adb, add the execution permission, execute demo, and check the printed log.
- Check the output file to see if it matches your expectations.

Apply for Image Buffer

> The demo provides two types of buffer for RGA synthesis: Graphicbuffer and AHardwareBuffer. The two buffers are distinguished by the macro USE_AHARDWAREBUFFER.

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

ifeq (1,$(strip $(shell expr $(PLATFORM_SDK_VERSION) \> 25)))
/*if USE_AHARDWAREBUFFER is set to 1 then use AHardwareBuffer, if USE_AHARDWAREBUFFER is set to 0 then use Graphicbuffer*/
LOCAL_CFLAGS += -DUSE_AHARDWAREBUFFER=1
endif
```

Graphicbuffer

> Graphicbuffer is initialized, filled/emptied, and filling rga_buffer_t structure through three functions.

```
    /*Passing in width, height, and image formats of src/dst and initialize Graphicbuffer*/
    src_buf = GraphicBuffer_Init(SRC_WIDTH, SRC_HEIGHT, SRC_FORMAT);
    dst_buf = GraphicBuffer_Init(DST_WIDTH, DST_HEIGHT, DST_FORMAT);

    /*Fill/empty Graphicbuffer by enumerating FILL_BUFF/EMPTY_BUFF*/
    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);

    /*Fill rga_buffer_t structure: src、dst*/
    src = wrapbuffer_GraphicBuffer(src_buf);
    dst = wrapbuffer_GraphicBuffer(dst_buf);
```

AHardwareBuffer

> AHardwareBuffer is initialized, filled/emptied, and filling rga_buffer_t structure through three functions.

```
    /*Passing in width, height, and image formats of src/dst and initialize AHardwareBuffer*/
    AHardwareBuffer_Init(SRC_WIDTH, SRC_HEIGHT, SRC_FORMAT, &src_buf);
    AHardwareBuffer_Init(DST_WIDTH, DST_HEIGHT, DST_FORMAT, &dst_buf);

    /*Fill/empty AHardwareBuffer by enumerating FILL_BUFF/EMPTY_BUFF*/
    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);

    /*Fill rga_buffer_t structure: src、dst*/
    src = wrapbuffer_AHardwareBuffer(src_buf);
    dst = wrapbuffer_AHardwareBuffer(dst_buf);
```

Viewing Help Information

> Run the following command to obtain the help information about the test case:

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

> You can use demo according to the help information. The following information is printed:

```
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.
============================================================================================
```

> Parameter parsing is in the directory /librga/demo/im2d_api_demo/args.cpp.

Executing Demo in Loop

> Run the following command to loop demo. The loop command must precede all parameters. The number of cycles are of the type int and the default interval is 200ms.

```
rgaImDemo -w6 --copy
rgaImDemo --while=6 --copy
```

Obtain RGA Version and Support Information

> Run the following command to obtain the version and support information:

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

> If there is no input options, all versions and support information of 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.
```

Code Parsing

> According to parameters of main() to print different information.

```
    /*Convert the parameters of main() into QUERYSTRING_INFO enumeration values*/
    IM_INFO = (QUERYSTRING_INFO)parm_data[MODE_QUERYSTRING];
    /*Print the string returned by querystring(), which is the required information*/
    printf("\n%s\n", querystring(IM_INFO));
```

Image Resizing

> Use the following command to test image resizing.

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

> This function must be filled withoptions as follows:

```
options:
    =up                 image resolution scale up to 1920x1080
    =down           image resolution scale down to 720x480
```

Code Parsing

> According to the parameters (up/down) of main() to choose to up(720p->1080p) or down(720p->480p), that is, for different scenarios, the buffer is re-initialized, emptied, or fills rga_buffer_t structure, and the rga_buffer_t structure that stores src and dst image data is passed to imresize().

```
    switch(parm_data[MODE_RESIZE])
    {
        /*scale up the image*/
        case IM_UP_SCALE :
            /*re-initialize Graphicbuffer to corresponding resolution 1920x1080*/
            dst_buf = GraphicBuffer_Init(1920, 1080, DST_FORMAT);
            /*empty the buffer*/
            GraphicBuffer_Fill(dst_buf, EMPTY_BUFF, 1);
            /*refill rga_buffer_t structure that stores dst data*/
            dst = wrapbuffer_GraphicBuffer(dst_buf);
            break;

        case IM_DOWN_SCALE :
            /*re-initialize Graphicbuffer to corresponding resolution 720x480**/
            dst_buf = GraphicBuffer_Init(720, 480, DST_FORMAT);
            /*empty the buffer*/
            GraphicBuffer_Fill(dst_buf, EMPTY_BUFF, 1);
            /*refill rga_buffer_t structure that stores dst data*/
            dst = wrapbuffer_GraphicBuffer(dst_buf);
            break;
    }
    /*pass src and dst of rga_buffer_t structure to imresize()*/
    STATUS = imresize(src, dst);
    /*print running status according to IM_STATUS enumeration values*/
    printf("resizing .... %s\n", imStrError(STATUS));
```

Image Cropping

> Test image cropping using the following command.

```
rgaImDemo --crop
```

> Options are not available for this feature. By default, crop the image within the coordinate LT(100,100)，RT(400,100)，LB(100,400)，RB(400,400).

Code Parsing

> Assign the size of clipped area in 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().

```
    /*The coordinates of the clipped vertex are determined by x and y, the size of the clipped area is
determined by width and height*/
    src_rect.x      = 100;
    src_rect.y      = 100;
    src_rect.width  = 300;
    src_rect.height = 300;

    /*pass src and dst of src_rect structure and rga_buffer_t structure format to imcrop()*/
    STATUS = imcrop(src, dst, src_rect);
    /*print the running status according to the returned IM_STATUS enumeration values*/
    printf("cropping .... %s\n", imStrError(STATUS));
```

Image Rotation

> Test image rotation using the following command.

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

> This function must be filled with options, which are as follows:

```
options:
    =90                 Image rotation by 90°, exchange the width and height of output image resolution.
    =180                Image rotation by 180°, output image resolution unchanged.
    =270                Image rotation by 270°, exchange the width and height of output image resolution.
```

Code Parsing

> According to the arguments (up/down) of main() to choose the rotation degrees(90/180/270). IM_USAGE enumeration transformed from arguments values, together with the rga_buffer_t structure that stores src and dst image data is passed to imrotate().

```
    /*convert the parameters of main() into IM_USAGE enumeration values*/
    ROTATE = (IM_USAGE)parm_data[MODE_ROTATE];

    /*pass both IM_USAGE enumeration value that identifies the rotation degrees and src and dst of
rga_buffer_t structure format to imrotate()*/
    STATUS = imrotate(src, dst, ROTATE);
    /*print the running status according to the returned IM_STATUS enumeration values*/
    printf("rotating .... %s\n", imStrError(STATUS));
```

Image Mirror Flip

> Use the following command to test mirror flipping

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

> This function must be filled with options, which are as follows:

```
options:
    =H                  Image horizontal mirror flip.
    =V                  Image vertical mirror flip.
```

Code Parsing

> According to the arguments (H/V) of main() to choose the flipped direction, transform the arguments to IM_USAGE enumeration values, and the rga_buffer_t structure that stores src and dst image data is passed to imflip().

```
    /*convert the parameters of main() into IM_USAGE enumeration values*/
    FLIP = (IM_USAGE)parm_data[MODE_FLIP];

    /*pass both IM_USAGE enumeration value that identifies the flipped direction and src and dst of
rga_buffer_t structure format to imflip()*/
    STATUS = imflip(src, dst, FLIP);
    /*print the running status according to the returned IM_STATUS enumeration value*/
    printf("flipping .... %s\n", imStrError(STATUS));
```

Image Color Fill

> Use the following command to test the color fill.

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

This function must be filled with options. By default, fill the color of image within the coordinate LT(100,100), RT(400,100), LB(100,400), RB(400,400), options are as follows:

```
options:
    =blue               Fill the image color as blue.
    =green          Fill the image color as green.
    =red                Fill the image color as red.
```

Code Parsing

> The filled color is determined according to the (blue/green/red) parameters of main(), and the size to be filled is assigned to the im_rect structure that stores the dst rectangle data, and the passed parameter is converted to the hexadecimal number of the corresponding color, which is passed to imfill() along with rga_buffer_t that stores the dst image data.

```
    /*Convert parameter of main() to hexadecimal number of the corresponding color*/
    COLOR = parm_data[MODE_FILL];

    /*The coordinates of clipping vertex are determined by x and y, and size of color-filled area is
determined by width and height*/
    dst_rect.x      = 100;
    dst_rect.y      = 100;
    dst_rect.width  = 300;
    dst_rect.height = 300;

    /*Pass dst_rect of im_rect format and hexadecimal number of the corresponding color together with src and
dst of rga_buffer_t format to imfill().*/
    STATUS = imfill(dst, dst_rect, COLOR);
    /*print the running status according to the returned IM_STATUS enumeration value*/
    printf("filling .... %s\n", imStrError(STATUS));
```

Image Translation

> Use the following command to test image translation.

```
rgaImDemo --translate
```

> This feature has no options. By default, the vertex (upper-left coordinate) is shifted to (300,300), that is, shifted 300 pixels to the right and 300 pixels down.

Code Parsing

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

```
    /*The coordinates of vertices of translated image are determined by x and y*/
    src_rect.x = 300;
    src_rect.y = 300;

    /*pass the src_rect of im_rect format along with src and dst of rga_buffer_t format into imtranslate()*/
    STATUS = imtranslate(src, dst, src_rect.x, src_rect.y);
    /*print the running status according to the returned IM_STATUS enumeration value*/
    printf("translating .... %s\n", imStrError(STATUS));
```

Image Copying

> Use the following command to test image copying.

```
rgaImDemo --copy
```

> This feature has no options. The default copy resolution is 1280x720 and the format is RGBA8888.

Code Parsing

> Passing rga_buffer_t that stores src and dst image data to imcopy().

```
    /*passing src and dst of rga_buffer_t format to imcopy()*/
    STATUS = imcopy(src, dst);
    /*print the running status according to the returned IM_STATUS enumeration value*/
    printf("copying .... %s\n", imStrError(STATUS));
```

Image Blending

> Use the following command to test image blending.

```
rgaImDemo --blend
```

> This feature has no options. By default, the blending mode is IM_ALPHA_BLEND_DST.

Code Parsing

> Passing rga_buffer_t that stores src and dst image data to imblend().

```
    /*passing src and dst of rga_buffer_t format to imblend()*/
    STATUS = imblend(src, dst);
    /*print the running status according to the returned IM_STATUS enumeration value*/
    printf("blending .... %s\n", imStrError(STATUS));
```

Image Format Conversion

> Use the following command to test image format conversion.

```
rgaImDemo --cvtcolor
```

> This feature has no options. By default, images with resolution of 1280x720 will be converted from RGBA8888 to NV12.

Code Parsing

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

```
    /*Assign the format in the format variable member*/
    src.format = HAL_PIXEL_FORMAT_RGBA_8888;
    dst.format = HAL_PIXEL_FORMAT_YCrCb_NV12;

    /*passing the format to be converted and src and dst of rga_buffer_t format to  imcvtcolor()*/
    STATUS = imcvtcolor(src, dst, src.format, dst.format);
    /*print the running status according to the returned IM_STATUS enumeration value*/
    printf("cvtcolor .... %s\n", imStrError(STATUS));
```