RGA IM2D API 开发指南

文件标识：RK-PC-YF-0002

发布版本：V2.0.0

日期：2020-07-10

文件密级：□绝密 □秘密 □内部资料 ■公开

免责声明

本文档按“现状”提供，瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

读者对象

本文档主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

修订记录

日期	版本	作者	修改说明
2020/06/24	1.0.0	陈城，李煌	初始版本
2020/10/16	1.0.1	陈城，李煌，余乔伟	更新部分接口
2021/12/07	2.0.0	陈城，李煌，余乔伟	增加RGA3相关支持

目 录

1. 概述

RGA (Raster Graphic Acceleration Unit)是一个独立的2D硬件加速器，可用于加速点/线绘制，执行图像缩放、旋转、bitBlt、alpha混合等常见的2D图形操作。

1.1 设计指标

Version	Codename	Chip	Source		Destination		Pixels/Cycle	Performance w/o scale (freq = 300Mhz)
			min	max	min	max		
RGA1	Pagani	RK3066	2x2	8192x8192	2x2	2048x2048	1	≈300Mpix/s
	Jaguar Plus	RK3188						
	Beetles	RK2926/2928						
	Beetles Plus	RK3026/3028						
RGA1_plus	Audi	RK3128	2x2	8192x8192	2x2	2048x2048	1	≈300Mpix/s
	Granite	Sofia 3gr						
RGA2	Lincoln	RK3288/3288w	2x2	8192x8192	2x2	4096x4096	2	≈600Mpix/s
	Capricorn	RK3190						
RGA2-Lite0	Maybach	RK3368	2x2	8192x8192	2x2	4096x4096	2	≈520Mpix/s
	BMW	RK3366						
RGA2-Lite1	Benz	RK3228	2x2	8192x8192	2x2	4096x4096	2	≈520Mpix/s
	Infiniti	RK3228H						
	Gemini	RK3326						
	Lion	RK1808						
RGA2-Enhance	Mclaren	RK3399	2x2	8192x8192	2x2	4096x4096	2	≈600Mpix/s
	Mercury	RK1108						
	Puma	RV1126/RV1109						
	skylarkV2	RK3566/RK3568						
RGA3	Orion	RK3588	128x128	8128x8128	128x128	8128x8128	4 (by pass) 2 (scale)	≈1200Mpix/s (by pass) ≈600Mpix/s (scale)

- 预期性能为默认RGA频率下计算得出，实际运行性能表现与内存频率等相关，列表数据仅供参考。

1.2 图像格式支持

- 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 YUV422 8bit BPP8/BPP4/BPP2/BPP1	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (only for Blur/sharpness) YUV422 8bit (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 YUV422 BPP8/BPP4/BPP2/BPP1	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit (only for Blur/sharpness) YUV422 8it (only for Blur/sharpness) YUV420 8bit (only for normal Bitblt without alpha) YUV422 8bit (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 YUV422 8bit 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 YUV422 8bit
	Capricorn	RK3190		
RGA2-Lite0	Maybach	RK3368	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit YUV422 8bit 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 YUV422 8bit
	BMW	RK3366		
RGA2-Lite1	Benz	RK3228	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit YUV420 10bit YUV422 8bit YUV422 10bit 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 YUV422 8bit
	Infiniti	RK3228H		
	Gemini	RK3326		
	Lion	RK1808		
RGA2-Enhance	Mclaren	RK3399	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit YUV420 10bit	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit YUV422 8bit

			YUV422 8bit YUV422 10bit BPP8/BPP4/BPP2/BPP1 (only for color palette)	YUYV422 YUYV420
	Mercury	RK1108		
	Puma	RV1126/ RV1109	RGBA/BGRA/ARGB/ABGR8888 RGBA/BGRA/ARGB/ABGR4444 RGBA/BGRA/ARGB/ABGR5551 RGB/BGR888 RGB/BGR565 YUV420 8bit YUV420 10bit YUV422 8bit YUV422 10bit YUYV/YVYU/UYVY/VYUY422 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 YUV422 8bit YUYV/YVYU/UYVY/VYUY422 YUYV/YVYU/UYVY/VYUY420 YUV400 Y4/Y1
	skylarkV2	RK3566/RK3568		
RGA3	Orion	RK3588	RGBA/BGRA/ARGB/ABGR8888 RGB/BGR888 RGB/BGR565 YUV420 8bit (only semi-planer) YUV420 10bit (only semi-planer) YUV422 8bit (only semi-planer) YUV422 10bit (only semi-planer) YUYV/YVYU/UYVY/VYUY422	RGBA/BGRA/ARGB/ABGR8888 RGB/BGR888 RGB/BGR565 YUV420 8bit (only semi-planer) YUV420 10bit (only semi-planer) YUV422 8bit (only semi-planer) YUV422 10bit (only semi-planer) YUYV/YVYU/UYVY/VYUY422

> 注：Y4格式即2的4次方色阶灰度图，Y400格式即2的8次方色阶灰度图。

1.3 图像格式对齐说明

HW_version	Format	Alignment
RGA1 RGA1_Plus RGA2 RGA2_Lite0 RGA2_Lite1 RGA2_Enhance	RGB/BGR888	width stride须4对齐
	RGB/BGR565	width stride须2对齐
	YUV420 8bit YUV422 8bit YUV400 Y4/Y1 YUYV/YVYU/UYVY/VYUY422 YUYV/YVYU/UYVY/VYUY420	width stride须4对齐，其余参数均须2对齐
	YUV420 10bit 422 10bit	width stride须16对齐，其余参数均须2对齐
RGA3	RGBA/BGRA/ARGB/ABGR8888 RGB/BGR888 RGB/BGR565	width stride须4对齐
	YUV420 8bit YUV420 10bit YUV422 8bit YUV422 10bit YUYV/YVYU/UYVY/VYUY422	width stride须4对齐，其余参数均须2对齐
	FBC mode	width stride须16对齐

2. API版本说明

RGA的支持库librga.so按照一定规则更新版本号，标识着功能新增、兼容性、问题修正的更新提交，并提供几种方式查询版本号，方便开发者在使用librga.so时可以清楚的辨别当前的库文件版本是否适合于当前的开发环境。详细版本更新日志可以查阅源码根目录下CHANGLOG.md。

2.1 版本号格式与递增规则

2.1.1 API版本号

2.1.1.1 格式

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

> 例：
>
> 1.0.0_[0]

2.1.1.2 递增规则

名称	规则
major	主版本号，当提交不向下兼容的版本。
minor	次版本号，当向下兼容的功能性API新增。
revision	修订版本号，当提交向下兼容的功能补充或致命的问题修正。
build	编译版本号，当向下兼容的问题修正。

2.2 版本号查询

2.2.1 strings命令查询：

以Android R 64位为例：

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

2.2.2 日志打印：

当每个进程首次调用RGA API时，会打印版本号。

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

2.2.3 函数接口查询

调用以下API，可以查询代码版本号、编译版本号、RGA硬件版本信息。具体使用说明可以查看 应用接口说明 章节。

```
querystring(RGA_VERSION);
```

> 字符串格式如下：
>
> RGA_api version : v1.0.0_[0]
> RGA version : RGA_2_Enhance

2.2.4 属性查询

该方式查询版本号仅Android系统支持，并且须已有进程调用RGA后，属性设置方生效。

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

3. 应用接口说明

RGA模块支持库为librga.so，通过对图像缓冲区结构体struct rga_info进行配置，实现相应的2D图形操作。为了获得更友好的开发体验，在此基础上进一步封装常用的2D图像操作接口。新的接口主要包含以下特点：

- 接口定义参考opencv/matlab中常用的2D图形接口定义，以减少二次开发的学习成本。
- 为消除RGA硬件版本差异带来的兼容问题，加入RGA query查询功能。查询内容主要包括版本信息，输入输出大分辨率及图像格式的支持。
- 对于2D图像复合操作，增加improcess接口。通过传入一系列预定义的usage执行复合操作。
- 执行图像操作之前，需要对输入输出图像缓冲区进行处理。调用wrapbuffer_T接口将输入输出图像信息填充到结构体struct rga_buffer_t，结构体中包含分辨率及图像格式等信息。

3.1 获取RGA 版本及支持信息

3.1.1 querystring

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

> 查询RGA基础信息及分辨率格式等支持情况

Parameters	Description
name	RGA_VENDOR - 厂商信息 RGA_VERSION - 版本信息 RGA_MAX_INPUT - 支持的最大输入分辨率 RGA_MAX_OUTPUT - 支持的最大输出分辨率 RGA_SCALE_LIMIT - 支持得缩放倍数 RGA_INPUT_FORMAT - 支持的输入格式 RGA_OUTPUT_FORMAT - 支持的输出格式 RGA_EXPECTED - 预期性能 RGA_ALL - 输出所有信息

Returns a string describing properties of RGA.

3.2 图像缓冲区预处理

3.2.1 wrapbuffer_T

> IM2D图形库接口参数中，输入源图像及输出目标图像应支持多种类型（以下内容输入参数用符号‘T’代表支持的类型）。在执行相应的图像操作之前，需要先调用wrapbuffer_T(T)将输入输出图像缓冲类型转化为统一的rga_buffer_t结构体，作为user API的输入参数。支持的输入输出图像缓冲类型具体包括：

Parameters(T)	Data Type	Description
virtual address	void *	图像缓冲区虚拟地址
physical address	void *	图像缓冲区物理地址
shared fd	int	图像缓冲区文件描述符
buffer handle	buffer_handle_t gralloc_drm_handle_t gralloc_drm_bo_t	图像缓冲区handle, 包含缓冲区地址，文件描述符，分辨率及格式等信息
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

> 不同的buffer类型调用RGA的性能是不同的，性能排序如下所示：
>
> physical address > fd = buffer handle = GraphicBuffer = AHardwareBuffer > virtual address
>
> 一般推荐使用fd作为buffer类型。

```
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 desribe image information.

3.3 图像缩放、图像金字塔

3.3.1 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);
```

> 根据不同的应用场景，可选择配置dst来描述缩放的目标图像大小，或配置缩放系数fx/fy实现缩放指定倍率的效果。同时配置dst和缩放系数fx/fy时，将采用缩放系数fx/fy计算后的结果作为目标图像大小。
>
> interpolation 仅硬件版本RGA1/RGA1 plus 可以支持配置。
>
> 注意：使用缩放系数fx/fy进行倍率缩放时，YUV等对宽高对齐有要求的格式将强制向下对齐至符合要求，使用该功能有可能会改变预期缩放效果。

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.

3.3.2 impyramid

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

> 金字塔缩放。根据direction 宽高同时做1/2 或者 2 倍的缩放。

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.

3.4 图像裁剪

3.4.1 imcrop

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

> 通过指定Rect 的大小区域执行图像裁剪。

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.

3.5 图像旋转

3.5.1 imrotate

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

> 支持图像旋转90，180，270度。

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.

3.6 图像镜像翻转

3.6.1 imfilp

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

> 支持图像做水平、垂直镜像翻转。

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.

3.7 图像颜色填充、内存赋值、图形绘制

3.7.1 imfill/imreset/imdraw

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

> 对RGBA 格式的图像的指定区域rect进行颜色填充。color参数由高到低位分别是R，G，B，A，例如，红色：color = 0xff000000.

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

> 对RGBA 格式的图像的指定区域rect内存中的内容全部设置为指定的值color。color参数由高到低位分别是R，G，B，A，例如，红色：color = 0xff000000.

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

> 对RGBA 格式的图像的指定区域rect根据指定颜色color进行绘制。color参数由高到低位分别是R，G，B，A，例如，红色：color = 0xff000000.

【注意】填充区域rect宽高需大于或等于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

Return IM_STATUS_SUCCESS on success or else negative error code.

3.8 图像平移

3.8.1 imtranslate

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

> 对图像做平移操作，移动到（x, y）坐标位置，src和dst 宽高须一致，超出部分会被裁剪。

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.

3.9 图像拷贝

3.9.1 imcopy

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

> 对图像做拷贝操作，RGA基础操作。作用与memcpy 类似。

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

Return IM_STATUS_SUCCESS on success or else negative error code.

3.10 图像合成

3.10.1 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使用A+B -> B 的图像双通道合成模式， 将前景图像（srcA通道）与背景图像（dst通道）根据配置的混合模型执行对应的Alpha叠加计算，并将合成结果输出至dst通道上。

```
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使用A+B -> C 的图像三通道合成模式，将前景图像（srcA通道）与背景图像（srcB通道）根据配置的混合模型执行对应的Alpha叠加计算，并将合成结果输出至dst通道上。

两种图像合成模式中mode 可以配置不同的**Porter-Duff**混合模型：

> 说明Porter-Duff混合模型前，先做出如下定义：
>
> - S -标识两个混合图像中的源图像，即前景图像，为souce的缩写。
> - D -标识两个混合图像中的目标图像，即背景图像，为destination的缩写。
> - R -标识两个图像混合的结果，为result的缩写。
> - c -标识像素的颜色，即（RGBA）的RGB部分，描述图像本身色彩，为color的缩写。（注意，Porter-Duff混合模型中的色彩值（RGB）均为左乘后的结果，即原始色彩与透明度的乘积，如色彩值未左乘则需要进行预乘（Xc = Xc * Xa）操作。）
> - a -标识像素的透明度，即（RGBA）的A部分，描述图像本身的透明度，为Alpha的缩写。
> - f -标识作用于**C**或者**A**上的因子，为factor的缩写。
>
> Porter-Duff混合模型的核心公式如下：
>
> Rc = Sc * Sf + Dc * Df;
>
> 即： 结果色 = 源色彩 * 源因子 + 目标色彩 * 目标因子。
>
> Ra = Sa * Sf + Da * Df;
>
> 即： 结果透明度 = 源透明度 * 源因子 + 目标透明度 * 目标因子。

RGA支持以下几种混合模型：

> 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]；

【注意】图像合成模式不支持YUV格式之间合成，imblend函数dst图像不支持YUV格式，imcomposite函数srcB图像不支持YUV格式。

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模式 IM_ALPHA_BLEND_DST —— DST模式 IM_ALPHA_BLEND_SRC_OVER —— SRC OVER模式 IM_ALPHA_BLEND_DST_OVER —— DST OVER模式 IM_ALPHA_BLEND_PRE_MUL —— 预乘使能，当需要预乘时须将该标识与其他模式标识进行或处理，再赋值给mode
sync	**[optional]** wait until operation complete

Return IM_STATUS_SUCCESS on success or else negative error code.

3.11 色键（Color Key）

3.11.1 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技术是对源图像进行预处理，将符合色键过滤条件的像素的alpha分量置零，其中所述色键过滤条件为非透明的颜色值，并将预处理后的源图像与目标图像进行alpha混合模式。
>
> 该模式仅支持在源图像（src）区域的图像上针对设定的颜色范围实现Color Key功能，并叠加在目标图像（dst）区域上。

> IM_ALPHA_COLORKEY_NORMAL为正常模式，即在设定的颜色范围内的颜色作为过滤条件，在该色彩范围内的像素点Alpha分量清零，IM_ALPHA_COLORKEY_INVERTED则反之。

Parameters	Range	Description
max	0x0 ~ 0xFFFFFFFF	需要消去/抠取的颜色范围最大值，排列为ABGR
min	0x0 ~ 0xFFFFFFFF	需要消去/抠取的颜色范围最小值，排列为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

Return IM_STATUS_SUCCESS on success or else negative error code.

3.12 图像格式转换

3.12.1 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)
```

> 格式转换功能，具体格式支持根据soc有不同请查阅图像格式支持章节。
>
> 格式可以通过rga_buffer_t 设置，也可以通过sfmt/dfmt分别配置源图像及输出图像格式。

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.

3.13 NN运算点前处理

3.13.1 imquantize

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

> 目前仅RV1126 / RV1109上支持。NN运算点前处理，图像RGB 三个通道可以分开单独配置offset以及scale。

公式：

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

参数范围：

Parameters	Range	Description
scale	0 ~ 3.99	10bit，从左往右，高位2个bit 表示整数部分，低位8bit表示小数部分
offset	-255 ~ 255	9bit，从左往右，高位表示符号位，地位表示0～255的偏移量

parameter	Description
src	**[required]** input image
dst	**[required]** output image
nn_info	**[required]** rga_nn_t结构体对RGB三个通道offset及scale进行单独配置 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

3.14 ROP 与或非运算

3.14.1 imrop

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

> 对两个图形做ROP 与或非运算

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

3.15 图像处理

3.15.1 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 图像复合操作函数，其他API都是基于此API开发，improcess 可以实现更复杂的复合操作。
>
> 图像操作通过usage 的方式进行配置。

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
usage	**[optional]** image operation usage

usage 参照定义：

```
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;
```

3.16 同步操作

3.16.1 imsync

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

> RGA异步模式需要调用该接口等待操作完成。
>
> 其他API 将 sync 设置为0，效果相当于opengl中的 glFlush，如果进一步调用imsync 可以达到glFinish的效果。

3.17 线程上下文配置

3.17.1 imconfig

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

> 通过不同的配置名对当前线程的上下文进行配置，该上下文将作为该线程的默认配置。
>
> 线程上下文的配置优先级低于接口的传参配置。如果接口传参配置未配置相关参数，则本地调用使用上下文默认配置完成本地调用；如果接口传参配置相关参数，以接口传参的配置完成本次调用。

parameter	Description
name	**[required]** context config name： IM_CONFIG_SCHEDULER_CORE —— 指定任务处理核心 IM_CONFIG_PRIORITY —— 任务优先级 IM_CHECK_CONFIG —— 校验使能
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

Return IM_STATUS_SUCCESS on success or else negative error code

4. 测试用例及调试方法

> 为了让开发者更加快捷的上手上述的新接口，这里通过运行demo和对demo源码的解析以加速开发者对API的理解和运用。

4.1 测试文件说明

> 用于测试的输入与输出二进制文件需提前准备好，在/sample/sample_file目录下，存放着默认的RGBA8888格式的源图像文件可以直接使用。
>
> Android系统须将源图片存储在设备/data/目录下，Linux系统须将源图储存在设备/usr/data目录下，文件命名规则如下：

```
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
```

参数解释如下：

> 输入文件为 in， 输出文件为 out
> --->第一个%d 是文件的索引， 一般为 0， 用于区别格式及宽高完全相同的文件
> --->第二个%d 是宽的意思， 这里的宽一般指虚宽
> --->第三个%d 是高的意思， 这里的高一般指虚高
> --->第四个%s 是格式的名字。
>
> 预置测试的部分常用图像格式如下，其他格式对应字符串名可以查看rgaUtils.cpp中查看：

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"

> demo中默认的输入图像文件分辨率为1280x720，格式为RGBA8888， 则须在/data或/usr/data目录下提前准备好名为in0w1280-h720-rgba8888.bin的源图像文件，图像合成模式还须额外在/data或/usr/data目录下提前准备好名为in1w1280-h720-rgba8888.bin的源图像文件。

4.2 调试方法说明

> 运行demo后打印日志如下（以图像拷贝为例）：
>
> Android中打印日志如下：

```
# 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
```

> Linux系统中打印日志如下：

```
# 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
```

> 当需要查看RGA运行更加详细的日志时，Android系统可以通过设置属性vendor.rga.log（Android 8及以下是sys.rga.log）来打开RGA配置log打印：

```
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
```

> Linux系统中需要打开代码core/NormalRgaContext.h，将__DEBUG设置为1，重新编译即可

```
#ifdef LINUX

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

> 一般打印log如下，可将此log上传至RedMine，由RK有关工程师分析：
>
> Android系统中打印日志如下：

```
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]
```

> Linux系统打印日志如下：

```
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]
```

4.3 测试用例说明

- 测试路径位于librga源码目录下 sample/im2d_api_demo ，开发者可以根据需求修改demo的配置，建议第一次运行demo使用默认配置。
- 测试用例的编译不同的平台编译是不同的，Android平台可以使用 ‘mm’ 命令进行编译，linux平台上在使用cmake编译librga.so时会在同目录下生成对应的测试用例。
- 将对应的测试用例编译后生成的可执行文件通过adb传入设备，添加执行权限，执行demo，查看打印log。
- 查看输出文件，检查是否与预期相符。

4.3.1 申请图像缓冲

> demo中提供了两种buffer用于RGA合成——Graphicbuffer、AHardwareBuffer。这两种buffer通过宏USE_AHARDWAREBUFFER区分。

```
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
```

4.3.1.1 Graphicbuffer

> 主要通过三个函数完成Graphicbuffer的初始化、填充/清空、填充rga_buffer_t结构体。

```
    /*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);
```

4.3.1.2 AHardwareBuffer

> 主要通过三个函数完成AHardwareBuffer的初始化、填充/清空、填充rga_buffer_t结构体。

```
    /*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);
```

4.3.2 查看帮助信息

> 使用如下命令获取测试用例帮助信息

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

> 运行成功后，便可以根据帮助信息使用demo，打印信息如下：

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

> 所有的参数解析在目录/librga/demo/im2d_api_demo/args.cpp中。

4.3.3 循环执行**demo**

> 使用如下命令循环执行示例demo，循环命令必须在所有参数之前，循环次数为int型，默认每次循环间隔200ms。

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

4.3.4 获取RGA版本及支持信息

> 使用如下命令获取版本及支持信息：

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

> 该命令有可选options，没有options则默认视为选择=all，可选options如下：

```
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
```

4.3.4.1 代码解析

> 根据main()传参决定打印出的不同信息。

```
    /*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));
```

4.3.5 图像缩放

> 使用如下命令进行图像缩放测试

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

> 该功能必须填入可选options，可选options如下：

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

4.3.5.1 代码解析

> 根据main()传参（up/down）决定放大或是缩小，即针对不同场景，重新初始化、清空buffer，填充rga_buffer_t结构体，并将最终的存储src、dst图像数据的rga_buffer_t结构体传入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));
```

4.3.6 图像裁剪

> 使用如下命令测试图像裁剪

```
rgaImDemo --crop
```

> 该功能无可选options，默认裁剪坐标LT(100,100)，RT(400,100)，LB(100,400)，RB(400,400)内的图像。

4.3.6.1 代码解析

> 将需要裁剪的大小在存储src矩形数据的im_rect结构体中赋值，并将存储src、dst图像数据的rga_buffer_t结构体传入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));
```

4.3.7 图像旋转

> 使用如下命令测试图像旋转

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

> 该功能必须填入可选options，可选options如下：

```
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
```

4.3.7.1 代码解析

> 根据main()传参（90/180/270）决定旋转角度，并将传参转化为IM_USAGE枚举值，与存储src、dst图像数据的rga_buffer_t结构体一同传入imrotate()。

```
    /*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));
```

4.3.8 图像镜像翻转

> 使用如下命令测试镜像翻转

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

> 该功能必须填入可选options，可选options如下：

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

4.3.8.1 代码解析

> 根据main函数传参（H/V）决定镜像翻转方向，并将传参转化为IM_USAGE枚举值，与存储src、dst图像数据的rga_buffer_t结构体一同传入imflip()。

```
    /*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));
```

4.3.9 图像颜色填充

> 使用如下命令测试颜色填充

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

该功能必须填入可选options，默认填充颜色在坐标LT(100,100)，RT(400,100)，LB(100,400)，RB(400,400)内的图像，可选options如下：

```
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
```

4.3.9.1 代码解析

> 根据main函数传参（bule/green/red）决定填充颜色，将需要填充的大小在存储dst矩形数据的im_rect结构体中赋值，并将传参转化为对应颜色的16进制数，与存储dst图像数据的rga_buffer_t结构体一同传入imfill()。

```
    /*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));
```

4.3.10 图像平移

> 使用如下命令测试图像平移操作

```
rgaImDemo --translate
```

> 该功能无可选options，默认顶点（左上角坐标）平移至(300,300)，即向右平移300个像素，再向下平移300个像素。

4.3.10.1 代码解析

> 将需要平移的偏移量在存储src矩形数据的im_rect结构体中赋值，并将存储src、dst图像数据的rga_buffer_t结构体传入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));
```

4.3.11 图像拷贝

> 使用如下命令测试图像拷贝

```
rgaImDemo --copy
```

> 该功能无可选options，默认拷贝分辨率为1280x720，格式为RGBA8888的图像。

4.3.11.1 代码解析

> 将存储src、dst图像数据的rga_buffer_t结构体传入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));
```

4.3.12 图像合成

> 使用如下命令测试图像合成

```
rgaImDemo --blend
```

> 该功能无可选options，默认合成模式为 IM_ALPHA_BLEND_DST 模式。

4.3.12.1 代码解析

> 将存储src、dst图像数据的rga_buffer_t结构体传入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));
```

4.3.13 图像格式转换

> 使用如下命令测试图像格式转换

```
rgaImDemo --cvtcolor
```

> 该功能无可选options，默认将分辨率为1280x720的图像从RGBA8888格式转换为NV12格式。

4.3.13.1 代码解析

将需要转换的格式在rga_buffer_t的成员变量format中赋值，并将存储src、dst图像数据的rga_buffer_t结构体传入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));
```