

Rockchip

(技术部, 图形框架显示平台中心)

福州瑞芯微电子有限公司

☒ 公开

☒ 正在修改

当前版本: 作者: 李煌 完成日期:

审核:

完成日期:

福州瑞芯微电子有限公司

Fuzhou Rockchips Semiconductor Co., Ltd (版本所有,翻版必究)

Linux Rga说明文档

概述

1. 介绍

和Android版本的librga功能相同, 具体的demo功能介绍以及相应配置可以参考Android版本的rga说明文档, 本文主要介绍Linux版本和Android版本不同的地方,以及Linux环境下如何搭建环境。

文档的和库的更新在以下链接: <https://github.com/lihuang1111/linux-rga>

2. 开发环境

目前已经经过测试的平台如下:

- RK3399: Centos
- RK3328: Debian
-

以上为目前较为常用的环境, 目前仅在这些环境下编译过, 并确认功能正常。

3. 开发环境搭建

(1)rk3399(centos)

需要有对应的Linux版本的固件, 查看目录/dev/下是否有rga设备节点, 如果没有rga设备节点参照以下步骤重新编译kernel.img并重新烧写。(rk3399为例)

- 在rk3399-linux.dtsi 中添加如下:

```
rga: rga@ff680000 {
    compatible = "rockchip,rga2";
    dev_mode = <1>;
    reg = <0x0 0xff680000 0x0 0x1000>;
    interrupts = <GIC_SPI 55 IRQ_TYPE_LEVEL_HIGH
0>;
    clocks = <&cru ACLK_RGA>, <&cru HCLK_RGA>,
<&cru SCLK_RGA_CORE>;
    clock-names = "aclk_rga", "hclk_rga",
"clk_rga";
    power-domains = <&power RK3399_PD_RGA>;
    dma-coherent;
    status = "okay";
};
```

- 配置

```
make ARCH=arm64 menuconfig
```

配置中按以下选项依次选择, 并打开rga2的编译选项:

```
device driver -> Graphics support -> Rockchip Misc Video driver
-> RGA2
```

- 编译kernel.img并烧写

(2)rk3328(debian)

需要有对应的Linux版本的固件, 查看目录/dev/下是否有rga设备节点, 如果没有rga设备节点参照以下步骤重新编译.

- 在rk322x.dtsi 中添加如下:

```
rga: rga@ff390000 {
    compatible = "rockchip,rga2";
    dev_mode = <1>;
    reg = <0x0 0xff390000 0x0 0x1000>;
    interrupts = <GIC_SPI 33 IRQ_TYPE_LEVEL_HIGH 0>;
    clocks = <&cru ACLK_RGA>, <&cru HCLK_RGA>, <&cru
SCLK_RGA>;
    clock-names = "aclk_rga", "hclk_rga", "clk_rga";
    status = "okay";
};
```

- 配置

```
make ARCH=arm64 menuconfig
```

- 配置中按以下选项依次选择，并打开rga2的编译选项：

```
device driver -> Graphics support -> Rockchip Misc Video  
driver -> RGA2
```

- 烧写

- 进入系统后执行如下命令挂载

```
mount /dev/mmcblk2p6 /boot
```

- PC上的/arch/arm64/boot/Image替换板子上的/boot/Image
- PC上的/arch/arm64/boot/dts/rockchip/rk3328-evb.dtb替换板子上的/boot/rk3328-evb.dtb
- 替换后reboot即可。

Rga使用说明

编译

只需要librga目录下执行make即可，对应的demo也只需要在对应demo目录执行make即可。将librga和demo解压到/opt目录，编译后生成demo可执行文件和librga.so在librga/目录下的bin，lib目录下面，根据需要将生成的librga.so拷到所要用的lib目录下，demo可执行文件可以在bin目录下直接运行或者可以拷到系统的bin目录下运行。

以下简要介绍makefile文件的配置：

- PRG_BIN_DIR 配置可执行文件的生成目录
- PRG_LIB_DIR 配置动态或者静态库的生成目录

- PRG_INC_DIR 头文件所在的目录
- EXCUTE_BIN 配置可执行文件的名字
- LD_LIBS 所需要包含的库

API使用说明

和Android版本一致，目前实现的api功能涵盖拷贝、旋转、格式转换和合成，目前仅需要关注以下2个API即可，使用如下的2个函数就能满足上述功能。

```
int RgaBlit(rga_info *src, rga_info *dst, rga_info *src1);
int rga_set_rect(rga_rect_t *rect,int x, int y, int w, int h, int
sw, int sh, int f)
```

C++接口

librga是用CPP编写的

在文件RockchipRga.h中有以下类的方法提供接口：

```
/* 初始化rga */
int RkRgaInit();
/* 申请buffer */
int RkRgaGetAllocBuffer(bo_t *bo_info, int width, int height, int
bpp);
/* 释放buffer */
int RkRgaFree(bo_t *bo_info);
/* 映射buffer到用户空间 */
int RkRgaGetMmap(bo_t *bo_info);
/* 解除映射buffer */
int RkRgaUnmap(bo_t *bo_info);
/* 获取buffer的fd */
int RkRgaGetBufferFd(bo_t *bo_info, int *fd);
/* 调用rga的接口 */
int RkRgaBlit(rga_info_t *src, rga_info_t *dst, rga_info_t *src1);
```

C接口

由于Linux上有些是C语言编写的程序，特别的引入C的接口，供C程序调用：

在文件RgaApi.h中定义了以下接口：

```

/* 初始化rga */
int c_RkRgaInit();
/* 申请buffer */
int c_RkRgaGetAllocBuffer(bo_t *bo_info, int width, int height, int
bpp);
/* 释放buffer */
int c_RkRgaFree(bo_t *bo_info);
/* 映射buffer到用户空间 */
int c_RkRgaGetMmap(bo_t *bo_info);
/* 解除映射buffer */
int c_RkRgaUnmap(bo_t *bo_info);
/* 获取buffer的fd */
int c_RkRgaGetBufferFd(bo_t *bo_info, int *fd);
/* 调用rga的接口 */
int c_RkRgaBlit(rga_info_t *src, rga_info_t *dst, rga_info_t
*src1);

```

重要的数据结构

- 额外定义的外部数据结构 bo_t，用于存放fd，handle等。

```

typedef struct bo {
    int fd;
    void *ptr;
    size_t size;
    size_t offset;
    size_t pitch;
    unsigned handle;
}bo_t;

```

- drm_rga_t 这个是封装好的用于抽象渲染区域的参数结构体，应用层需要了解具体的使用方式，和各个参数的实际意义。drm_rga_t 包含两个参数主要是 src 和 dst 的参数集合，均为 rga_rect_t 的结构

```

typedef struct drm_rga{
    rga_rect_tsrc; // 描述src图形信息
    rga_rect_tdst; // 描述dst图形信息
} drm_rga_t;

```

rga_rect_t 包含 8 个参数们用来描述图形:

```
typedef struct rga_rect {
    int xoffset;           //x 方向偏移
    int yoffset;           //y 方向偏移
    int width;             //实宽
    int height;            //实高
    int wstride;           //虚宽
    int hstride;           //虚高
    int format;            //图形格式
    int size;              //描述大小，暂时没用到，可忽略
} rga_rect_t;
```

- rga_info_t:这个变量封装的主要是描述一张图片的比较完整的信息，比如可以使用 fd、物理地址和虚拟地址来描述图片在 ddr 的存储位置。还指定图像存储的方向信息 rotation 等等。

```
typedef struct rga_info {
    int fd;                //文件描述符，用以描述存储位置
    void *virAddr;          //虚拟地址，用以描述存储位置
    void *phyAddr;          //物理地址，用以描述存储位置
    buffer_handle_t hnd;    //描述帧缓冲区
    int format;             //图片格式
    rga_rect_t rect;        //描述图形信息
    unsigned int blend;     //合成模式，用以两图像合成
    int bufferSize;        //描述 buffer 大小
    int rotation;           //描述图形旋转方向
    int color;              //一般用于 colorFill，想内存填充指定色块
    int testLog;            //RGA 早期用于 log 打印，现已利用指定属性
    int mmuFlag;            //用以描述 handle 获取内存申请的是否为物理连续

    int colorkey_en;        //使能 colorKey
    int colorkey_max;       //指定扣色最大值
    int colorkey_min;       //指定扣色最小值
    int reserve[128];       //保留位
} rga_info_t;
```

RGA调试手段

- 用于测试的输入与输出二进制文件命名规则:

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

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, 目前格式名字如下:

```
/******
```

```
HAL_PIXEL_FORMAT_RGB_565:          "rgb565";
```

```
HAL_PIXEL_FORMAT_RGB_888:          "rgb888";
```

```
HAL_PIXEL_FORMAT_RGBA_8888:        "rgba8888";
```

```
HAL_PIXEL_FORMAT_RGBX_8888:        "rgbx8888";
```

```
HAL_PIXEL_FORMAT_BGRA_8888:        "bgra8888";
```

```
HAL_PIXEL_FORMAT_YCrCb_420_SP:      "crcb420sp";
```

```
HAL_PIXEL_FORMAT_YCrCb_NV12:        "nv12";
```

```
HAL_PIXEL_FORMAT_YCrCb_NV12_VIDEO:  "nv12";
```

```
HAL_PIXEL_FORMAT_YCrCb_NV12_10:     "nv12_10";
```

```
*****/
```

例如:输入 1920x1080 RGBA8888 文件命名为:in0w1920-h1080-rgba8888.bin

- 利用二进制文件配合 demo 进行测试

1. 通过上述方式对测试的二进制文件进行命名
2. 测试时请将二进制文件放置 /data/ 目录, 这样 demo 才能找到该文件。
3. 执行 demo 后, 会输出二进制文件目录并打印 rga 耗时, 如下图:

```
dst.fd =8
cost_time=7 ms
open /data/out0w1280-h720-rgba8888.bin and write ok
threadloop
```

4. 对于数据的调试分析工具, 建议采用7yuv这个强大的工具。目前能支持很多常见的数据格式, 特别是在 rk 目前使用中几乎是 99%的使用是支持的, 并且能够在 window 和 ubuntu 上都能使用。

RGA demo 代码详细说明

目前 RGA 对外接口统一为如下接口, 可实现旋转、拷贝、格式转换和合成功能: int

RgaBlit(rga_info *src, rga_info *dst, rga_info *src1)

demo的实现主要是靠libdrm来获取buffer, fd以及相关操作, 详细的在代码注释中有说明。

Demo rgaBlit 详细代码说明

☰ 我们已对 rgaBlit 代码做了详细的注释, 以帮助理解demo实现过程.详情参见/librga/demo/rgaBlit/rgaBlit.cpp,所有demo实现均按rgaBlit代码逻辑, 并且我们通过如下方式将代码分为若干功能模块, 帮助理解及高效移植demo图形显示

```
/****** instantiation RockchipRga *****/  
/****** instantiation GraphicBufferMapper *****/  
/****** apply for src_buffer *****/  
/****** apply for dst_buffer *****/
```

demo 使用说明

demo 使用步骤

1. 修改 demo:根据需求修改对应 demo 代码
2. 编译 demo:进入对应 demo 文件夹编译即可。例如 rga 单元测试(rgaSlit)demo

```
cd xxx/demo/rgaSlit  
make  
//输出文件在 xxx/librga/bin 目录下
```

3. 执行 demo
 - a. 若 demo 需要输入 image 数据, 则需要先将 image.bin 文件放置如/data/目录
 - b. 将 demo 拷入设备, 并修改文件权限, 加入执行权限
 - c. 执行 demo, 查看打印 log
 - d. 查看输出文件, 是否与预期相符

demo 对应 log 输出如下


```

librga:RGA_GET_VERSION:1.6,1.600000 //rga 版本
ctx=0xad98a5e8,ctx->rgaFd=3 //rga 上下文
GraphicBuffer_src ok : *****
GraphicBuffer_dst ok : *****
lock buffer_src ok : *****
unlock buffer_src ok : *****
lock buffer_dst ok : *****
unlock buffer_dst ok : *****
src.fd =7 //源 image fd
dst.fd =8 //目标 image fd
cost_time=7 ms //调用 RGA 耗时
open /data/out0w1280-h720-rgba8888.bin and write ok //输出文件名及目录
threadloop

```

RGA 配置 log

相关参数打印在normal/NormalRgaContext.h 设置 __DEBUG 宏定义为1

```

librga: <<<<----- print rgaLog ----->>>>
librga: src->hnd = 0xb4b190e0 , dst->hnd = 0xb4b19150
librga: srcFd = 07 , phyAddr = 0 , virAddr = 0
librga: dstFd = 08 , phyAddr = 0 , virAddr = 0
librga: srcBuf = b475c000 , dstBuf = b43d8000
librga: blend = ff0405 , perpixelAlpha = 1
librga: scaleMode = 0 , stretch = 0;
librga: rgaVersion = 1.600000 , ditherEn =0
librga: srcMmuFlag = 1 , dstMmuFlag = 1 , rotateMode = 0
librga: <<<<----- rgaReg ----->>>>
librga: render_mode=0 rotate_mode=0
librga: src:[0,b475c000,b483d000],x-y[0,0],w-h[1280,720],vw-
vh[1280,720],f=0
librga: dst:[0,b43d8000,b44b9000],x-y[0,0],w-h[1280,720],vw-
vh[1280,720],f=0
librga: pat:[0,0,0],x-y[0,0],w-h[0,0],vw-vh[0,0],f=0
librga: ROP:[0,11,0],LUT[0]
librga: color:[0,0,0,0,0]
librga: MMU:[1,0,80000521]
librga: mode[0,0,0,0,0]
rgaBlit : cost_time=7 ms

```

demo 功能说明

rgaSlt

- 功能介绍

rgaSlit 用于单元测试，验证目标环境是否满足 RGA 调用要求。可在验证其他 demo 前先执行此 demo 确保 RGA 环境没有问题。

- demo 说明

该 demo 调用 RGA 最基本的 copy 操作，将 src 的 RGBA8888 格式数据从 src 地址 copy 到 dst 目标地址，并将 dst 数据与 src 数据进行比对，若完全相同则单元测试通过。若不满足要求，则需对环境进行检查。详情见以下模块代码。

```
int size = dstWidth * dstHeight * 4;
unsigned int *pstd = (unsigned int *)bo_src.ptr;
unsigned int *pnw = (unsigned int *)bo_dst.ptr;
int errCount = 0;
int rightCount = 0;
printf("[ num : srcInfo dstInfo ] \n");
//从src与dst的首地址逐按像素(4字节)比较
for (int i = 0; i < size / 4; i++) {
    //若不相等，则打印错误位置及数值，并记录错误数
    if (*pstd != *pnw) {
        printf("[X%.8d:0x%x 0x%x] ", i, *pstd, *pnw);
        if (i % 4 == 0)
            printf("\n");
        errCount++;
        //若相等，则记录正确数
    } else {
        if (i % (640*1024) == 0)
            printf("[Y%.8d:0x%.8x 0x%.8x]\n", i, *pstd, *pnw);
        rightCount++;
    }
    pstd++;
    pnw++;
    //错误数大于64停止检查
    if (errCount > 64)
        break;
}

printf("errCount=%d,rightCount=%d\n", errCount, rightCount);
if(errCount != 0)
    printf("rga slit err !! \n");
else
    printf("rga slit sucess !! \n");
```

- 若单元测试通过，则会有如下打印

```
[ num : srcInfo dstInfo ]
[Y000000000:0xff190200 0xff190200]
[Y00655360:0xff9b2f00 0xff9b2f00]
errCount=0,rightCount=921600
rga slt sucess !!
```

说明 RGA 测试环境 pass，RGA 可以正常调用并运行。

- 若单元测试不通过，则会有如下打印

```
[ num : srcInfo dstInfo ]
[X000000000:0xff190200 0xff000000]
[X000000001:0xff190200 0xff010101] [X00
[X000000005:0xff170200 0xff010101] [X00
[X000000009:0xff1c0200 0xff010101] [X00
[X000000013:0xff1c0200 0xff010101] [X00
[X000000017:0xff1a0200 0xff010101] [X00
[X000000021:0xff1a0200 0xff010101] [X00
[X000000025:0xff1b0200 0xff010101] [X00
[X000000029:0xff1a0200 0xff010101] [X00
[X000000033:0xff230200 0xff010101] [X00
[X000000037:0xff190200 0xff010101] [X00
[X000000041:0xff1a0200 0xff010101] [X00
[X000000045:0xff190200 0xff010101] [X00
[X000000049:0xff1b0200 0xff010101] [X00
[X000000053:0xff210200 0xff010101] [X00
[X000000057:0xff1d0200 0xff010101] [X00
[X000000061:0xff1b0200 0xff010101] [X00
errCount=65,rightCount=0
rga slt err !!
```

说明 RGA 不能正常使用，请将 rga 配置 log 上传 redmine，确认问题原因。

rgaBlit

- 功能介绍

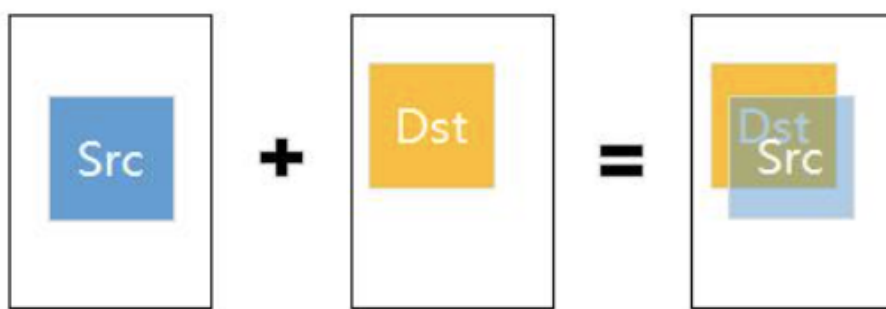
Image Src 与 Dst 做合成，通过 Src.blend 的设置来选择混合模式，Src.blend 解释如下: Blend:[23:16] = globalAlpha 值，范围为 0x00 ->0xff，不使用则须设置为 0xFF[16:0] = 合成模式，共有三种模式，如下:

blend = 0xFF0100:不混合，直接覆盖



blend = 0xFF0105:该 Src 的颜色已经做过 alpha 预乘, 因此混合方式为 $\text{src} + (1 - \text{src.a}) \text{dst}$

blend = 0xFF0405:该 Src 的颜色未做过预乘, 按 $\text{src.a} * \text{src} + (1 - \text{src.a}) * \text{dst}$ 的方式混合



blend = 0x30xxxx:globalAlpha 值不为 0xFF, 则引入 globalAlpha 值, 可将整张 image 的进行 globalAlpha 运算, 先对 Src.a 的值根据 globalAlpha 值重新计算。

$\text{Src.a} = \text{src.a} \times \text{globalAlpha}$

再通过判断合成模式进行 alpha 运算。

0x00 -> 0xFF :全透明 -> 全不透明

- demo说明

配置好 Src 与 Dst 数据参数后, 只需要在 RGA 设置模式中加入如下代码即可:

```

/***** set the rga_mod *****/
src.blend = 0xff0105;

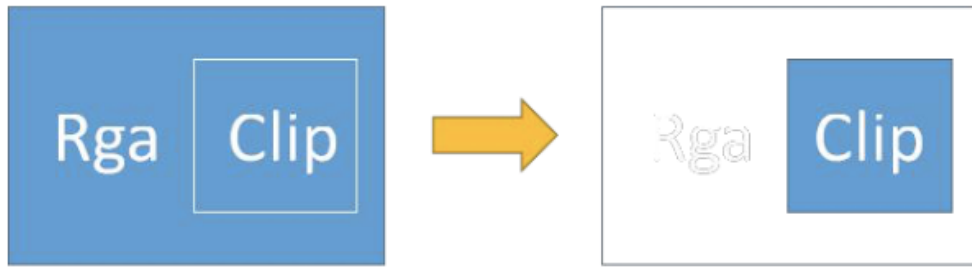
```

即可将 RGA 设置成 Blit 模式。

rgaClip

- 功能介绍

从一张 1920x1080 的 RGBA8888 中裁剪出任意比例的矩形区域，如下图：

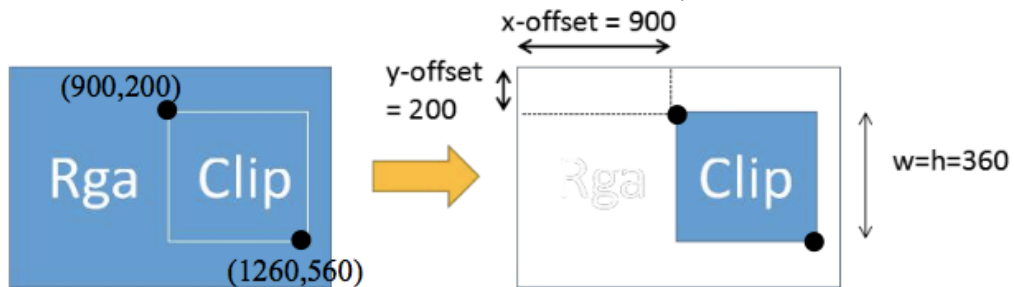


- demo说明

同样是调用 `rga_set_rect` 接口，只是参数配置上需要多留意。

```
static inline int rga_set_rect(rga_rect_t *rect, int x, int y, int w, int h, int sw, int sh, int f)
```

现在要做到从 720P 图像中裁剪出一块 360x360 的区域，如下图：



```
Src_info:1280x720, srcFormat = RK_FORMAT_RGBA_8888  
Dst_info:360x360, dstFormat = RK_FORMAT_RGBA_8888
```

在设置 `rect_info` 模块设置如下：

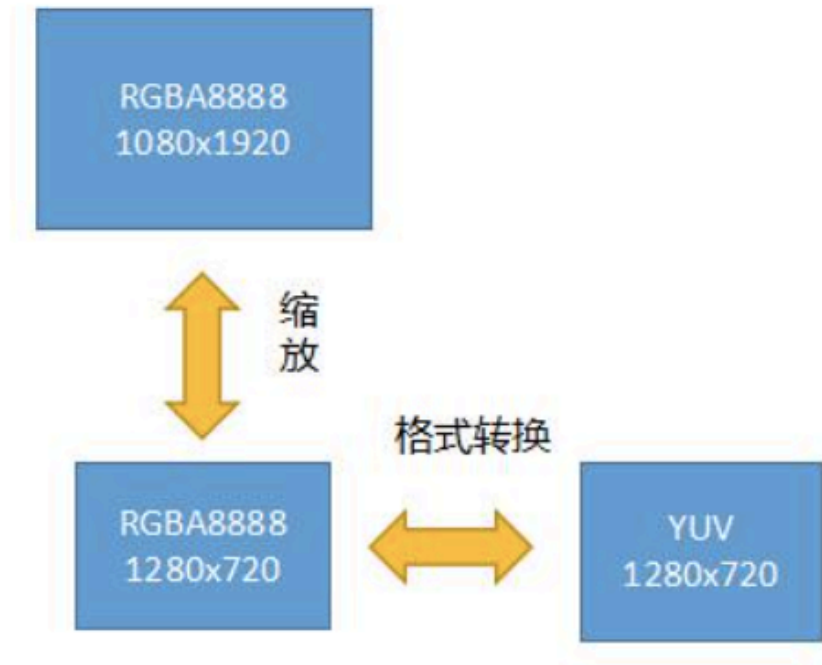
```
/****** set the rect_info *****/  
rga_set_rect(&src.rect, 900, 200, 360, 360, 1280, 720, srcFormat);  
rga_set_rect(&dst.rect, 0, 0, 360, 360, 360, 360, dstFormat);
```

rgaCopyXXXXX

- 功能介绍

通过设置将图像数据从 Src 搬移到 Dst，支持的模式包括但不限于如下列举：

- rgaCopyScale图像任意比例缩放
- rgaCopyXXToXX图像格式转换



- demo说明

- rgaCopyScale 说明

```

//一张 1280x720 图片缩放到 1920x1080
Src_info:1280x720, srcFormat = RK_FORMAT_RGBA_8888
Dst_info:1920x1088, dstFormat = RK_FORMAT_RGBA_8888
  
```

在设置 rect_info 模块设置如下:

```

/***** set the rect_info *****/
rga_set_rect(&src.rect, 0,0,1280,720,1280,720,srcFormat);
rga_set_rect(&dst.rect, 0,0,1920,1088,1920,1088,dstFormat);
//demo 中是通过设定
srcWidth, srcHeight, dstWidth, dstHeight, Format 的值并传递到
//rga_set_rect(...)接口, 效果一致。
  
```

备注:缩放可以任意尺寸, 只要指定了合法的宽高。

- rgaCopyXXXTXXXX说明

```
//一张 1280x720 的 XXX 格式图片需要转化为 XXXX 格式  
rgaCopyRgbaToYuv //RGBA8888格式转换为YUV_NV12格式  
Src_info:1280x720, srcFormat = RK_FORMAT_RGBA_8888  
Dst_info:1280x720, dstFormat = RK_FORMAT_YCrCb_420_SP
```

在设置 rect_info 模块设置如下:

```
/****** set the rect_info *****/  
rga_set_rect(&src.rect, 0,0,1280,720,1280,720,srcFormat);  
rga_set_rect(&dst.rect, 0,0,1280,720,1280,720,dstFormat);
```

备注:只要配置源 image 格式与目标 image 格式后调用 RGA 接口即可

rgaMirror

- 功能介绍

【实现效果】图像进行水平翻转，如下图:



- demo说明

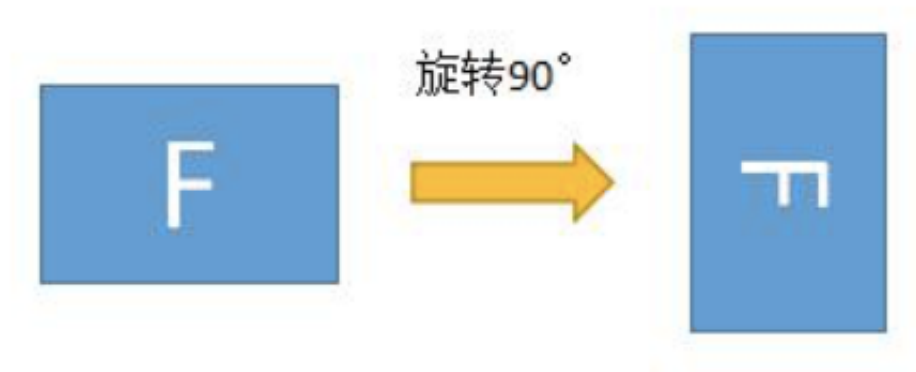
```
//仅需在 rga_mod 模块设置将 src.rotation 值设置为对应宏值即可。  
Src_info:1280x720, srcFormat = RK_FORMAT_RGBA_8888  
Dst_info:1280x720, dstFormat = RK_FORMAT_RGBA_8888
```

在 set_rga_mod 模块设置如下

```
/****** set the rga_mod *****/  
src.rotation = HAL_TRANSFORM_FLIP_H; //水平翻转  
//src.rotation = HAL_TRANSFORM_FLIP_H; //垂直翻转
```

rgaRotation

- 功能介绍 将图像进行顺时针旋转，旋转角度可设置为 90°、180°、270°。旋转 90 度如下图：



- demo说明

```
//1280x720 图片顺时针旋转 90 度  
Src_info:1280x720,  srcFormat = RK_FORMAT_RGBA_8888  
Dst_info:720x1280,  dstFormat = RK_FORMAT_RGBA_8888
```

图像旋转后需要交换宽高，才能保证图像不变形。在设置 rect_info 模块设置如下：

```
/****** set the rect_info *****/  
rga_set_rect(&src.rect, 0,0,1280,720,1280,720,srcFormat);  
rga_set_rect(&dst.rect, 0,0,720,1280,720,1280,dstFormat);
```

同时在 set_rga_mod 模块设置如下

```
/****** set the rga_mod *****/  
src.rotation = HAL_TRANSFORM_ROT_90; //旋转 90°  
//src.rotation = HAL_TRANSFORM_ROT_180; //旋转 180°  
//src.rotation = HAL_TRANSFORM_ROT_270; //旋转 270°
```

备注:三个旋转方向可供选择

rgaUserSpace

- 功能介绍

通过 malloc()申请 buffer 内存空间，区别于通过drm获取 buffer 地址。

- demo 说明

Demo 仅仅修改 buffer 获取方式，通过 malloc 方式获取 buffer 首地址，实现较简单，可通过查看 demo 源代码学习。此获取 buffer 方法可应用于以上所有 demo，有需要可以采用这种方式。

```
/****** apply for buffer *****/
src = malloc(srcWidth * srcHeight * 4);
if (!src)
    return -ENOMEM;
/****** apply for buffer *****/
dst = malloc(dstWidth * dstHeight * 4);
if (!dst) {
    free(src);
    return -ENOMEM;
}
```

rgaColorFill

- 功能介绍

对指定 dst 内存进行色块填充，目前仅支持 RGBA 格式。

- demo说明

仅需要参照 demo 修改 dst.color 的值即可向内存填充指定色块。

```
/****** set the rga_mod *****/
dst.color = 0xffff0000; //填充蓝色 , alpha=0xff
//dst.color = 0xff00ff00; //填充绿色 , alpha=0xff
//dst.color = 0xff0000ff; //填充红色 , alpha=0xff
```

统一接口 RgaBlit 详细源代码说明

关于统一接口 RgaBlit的代码实现过程，可通过提供的librga源代码进行阅读，我们已经对该接口做了详细的注释说明，帮助维护代码会应用开发。

文件目录:./librga/normal/NormalRga.cpp

测试数据使用说明

RockchipFileOps.cpp 这个文件主要是获取文件的方式。

比如一张 rgba8888 的数据，就会被命名为 in%dw%d-h%d-%s.bin(第一%d 是获取索引，可能有两张一模一样的rgba数据，第二个是宽，第三个是高，第四个%就是 rgba8888==>in0w1920-h1080-rgba8888.bin)。

RGA 使用场景

1. rga速度的计算

RGA1:理论上每个时钟能执行 1 个像素点【注意:是像素点】，则 300m 的 aclk【只需参考 aclk 有关系】每秒能处理 3001000, 000 个像素点。

RGA2:理论上每个时钟能执行 2 个像素点【注意:是像素点】，则 300m 的 aclk【只需参考 aclk 有关系】每秒能处理 3001000, 000*2 个像素点。

那么 1920x1080 的像素点需要这么长的理论时间:

```
Rga1: 2073600 / (300 * 1000, 000)= 0.006912s  
Rga2: 2073600 / (300 * 1000, 000*2)= 0.003456s
```

2. rga的功能

- 旋转，如果需要处理内存比较大的数据，可以使用rga做加速，通常比cpu快
- 拷贝，如果需要处理内存比较大的数据，可以使用rga做加速，通常比cpu快，也能减少cpu的负荷。
- 缩放，带有算法的缩放，cpu 的处理时间更慢，rga 有很好的优势
- 合成，这个用 cpu 基本上是无解的，有解时间也太长了
- 清内存，对某个内存清除某种特定的颜色，比如 RGBA8888(0x80808080)
- 格式转换
- 图像裁剪
- 指定颜色范围内的抠色处理(未加入demo，如需请咨询rk相关工程师)