# Google Translated by XNL Future Technologies

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

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

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

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

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

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

Document Name: **RGA FAQ**
Original Document: https://github.com/JustEnoughLinuxOS/linux-rga/blob/im2d/docs/RGA_FAQ.md

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

# RGA FAQ

File identification: RK-PC-YF-0002

Release version: V1.0.0

Date: 2021-06-28

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

---



Rockchip Microelectronics Co., Ltd.

Rockchip Electronics Co., Ltd.

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

Website: www.rock-chips.com

Customer service hotline: +86-4007-700-590

Customer Service Fax: +86-591-83951833

Customer service email: fae@rock-chips.com

---

**Readership**

This document is mainly applicable to the following engineers:

- Technical support engineer
- Software development engineer

**Revision history**

| date | Version | author | Modification instructions |
| --- | --- | --- | --- |
| 2021/06/28 | 1.0.0 | Yu Qiaowei | initial version |

**Table of contents**

[TOC]

# Overview

This article focuses on the RGA driver and user-mode interface librga, and summarizes some common problems encountered when calling RGA hardware to implement OSD (On Screen Display) and GUI (Graphics User Interface) graphics drawing acceleration functions on the RK platform.

# Release Notes

## Hardware version

RGA hardware is mainly divided into three versions: RGA1, RGA2, and RGA3. For specific platform mounting information, supported functions and restrictions, please refer to the "RGA IM2D API Development Guide" - Overview Chapter.

## Software version

### librga

The API version number is divided into major version number, minor version number, revised version number, and compiled version number. The four levels of version numbers correspond to different degrees of functional updates. For details, please check the "RGA IM2D API Development Guide" - API Version Description Chapter.

### RGA driver

The driver version number is divided into major version number, minor version number, revision number, and compiled version number. The four levels of version numbers

correspond to different levels of functional updates. Usually the HAL library in the released SDK matches the driver, and librga will perform the internal Verification version, developers do not need to care about this version. When the following error occurs when updating librga alone, you need to update the driver to the corresponding version.

# Log acquisition and description

## HAL layer log

### Log switch

- Android platform

The Android platform supports using attributes to configure whether librga enables HAL layer log printing:

Android 7.1 and below platforms

```
setprop sys.rga.log 1
logcat -s librga
```

Android 8.0 and above platforms

```
setprop vendor.rga.log 1
logcat -s librga
```

- Linux platform

The Linux platform does not support configuring the log switch through attributes. You need to modify the macro definitions in the following directories and recompile to enable log printing at the HAL layer:

core\NormalRgaContext.h in the source code directory

```
diff --git a/core/NormalRgaContext.h b/core/NormalRgaContext.h
index e4bf604..cbef425 100755
--- a/core/NormalRgaContext.h
+++ b/core/NormalRgaContext.h
@@ -20,7 +20,7 @@
 #define _rockchip_normal_rga_context_h_

 #ifdef LINUX
-#define __DEBUG 0
+#define __DEBUG 1

 #define ALOGE(...) { printf(__VA_ARGS__); printf("\n"); }
 #endif
```

### Log description

- General log

//当每个进程首次调用librga时，会初始化librga的单例，并打印当前的API版本号等信息
E rockchiprga: rga_api version 1.2.4_[11] (721a2f6 build: 2021-06-28 16:14:30 base: rk3566_r)

- debug log

D librga  : <<<<-------- print rgaLog -------->>>>
//以下部分为传入librga的参数打印。
D librga  : src->hnd = 0x0 , dst->hnd = 0x0 , src1->hnd = 0x0
//三个通道（src、src1、dst）传入的内存句柄的值
D librga  : src: Fd = 00 , phyAddr = 0x0 , virAddr = 0xb400007431ed6040
//src通道传入的内存类型对应的值，对应为DMA_FD、物理地址、虚拟地址。
D librga  : dst: Fd = 00 , phyAddr = 0x0 , virAddr = 0xb400007431b4f040
//dst通道传入的内存类型对应的值，对应为DMA_FD、物理地址、虚拟地址。
D librga  : src: Fd = -01 , buf = 0xb400007431ed6040, mmuFlag = 1, mmuType = 0  //src通道将配置传递的内存类型对应的值以及是否使能MMU，这里HAL层选择虚拟地址传入驱动。
D librga  : dst: Fd = -01 , buf = 0xb400007431b4f040, mmuFlag = 1, mmuType = 0  //dst通道将配置传递的内存类型对应的值以及是否使能MMU，这里HAL层选择虚拟地址传入驱动。
E librga  : blend = 0 , perpixelAlpha = 1
//混合模式以及图像格式是否本身存在Alpha值
D librga  : scaleMode = 0 , stretch = 0;
//缩放模式（RGA1）。
E librga  : rgaVersion = 3.200000  , ditherEn =0
//硬件版本号，16阶灰度图（Y4）dither使能。
D librga  : srcMmuFlag = 1 , dstMmuFlag = 1 , rotateMode = 0
//MMU使能标志位，旋转模式。
D librga  : <<<<-------- rgaReg -------->>>>
//以下为配置入驱动的参数打印。
E librga  : render_mode=0 rotate_mode=0
//RGA运行模式，旋转模式。
E librga  : src:[0,b400007431ed6040,b400007431fb7040],x-y[0,0],w-h[1280,720],vw-vh[1280,720],f=0  //src通道的内存、图像参数、格式信息。
E librga  : dst:[0,b400007431b4f040,b400007431c30040],x-y[0,0],w-h[1280,720],vw-vh[1280,720],f=0  //dst通道的内存、图像参数、格式信息。
E librga  : pat:[0,0,0],x-y[0,0],w-h[0,0],vw-vh[0,0],f=0
//pat/src1通道的内存、图像参数、格式信息，由于当前模式没有使用到该通道，所以参数均为0。

//以下部分开发者通常不用关心，为librga配置入驱动的不同模式的相关参数。
E librga  : ROP:[0,0,0],LUT[0]
//ROP模式配置，LUT表配置
E librga  : color:[0,0,0,0,0]
//colorkey配置（max color，min color），填充颜色配置(前景色配置，背景色配置，颜色填充配置)

```
E librga  : MMU:[1,0,80000521]
                                                        //MMU配置
E librga  : mode[0,0,0,0]
                                                                //palette、csc
、colorkey配置
E librga  : Full CSC : en[0]
                                                        //full csc使能标志
E librga  : gr_color_x [0, 0, 0]
                                                        //填充颜色配置，对应R、G
、B的颜色值
```

## Driver debug node

### Log switch

- Debug node address

Different RGA hardware versions open driver debugging nodes in the same way, but the debugging nodes are stored in the folders of the corresponding hardware versions:

RGA1: /sys/kernel/debug/rga_debug/rga RGA2: /sys/kernel/debug/rga2_debug/rga2

- Debugging function description

Taking RGA2 as an example, you can obtain the corresponding function description through the cat node in the corresponding directory:

```
/# cd /sys/kernel/debug/rga2_debug/
/# cat rga2
echo reg > rga2 to open rga reg MSG          //开启寄存器配置打印
echo msg > rga2 to open rga msg MSG          //开启传递参数打印
echo time > rga2 to open rga time MSG        //开启耗时打印
echo check > rga2 to open rga check flag     //打开检查 case 功能
echo stop > rga2 to stop using hardware      //停用 rga 驱动
echo int > rga2 to open interruppt MSG               //打开中断信息打印
echo slt > rga2 to open slt test                     // 进行内部 slt 测试
```

echo reg > rga2: This command switches the printing of RGA register configuration information. When this print is turned on, the configuration value of each rga working register will be printed.

echo msg> rga2: This command switches the printing of RGA upper-layer configuration parameter information. When this printing is turned on, the parameters passed by the upper layer call rga driver will be printed.

echo time> rga2: This command switches the printing of RGA work time-consuming information. When this print is turned on, the time taken for each call to rga will be printed.

echo check> rga2: This command switches the test case inside RGA. When the print is turned on, the relevant parameters will be checked every time RGA works, mainly the memory check and whether the alignment meets the requirements. If the following log is output, it means the check is passed. If the memory is out of bounds, it will cause the kernel to crash. You can confirm whether it is a problem with src data or dst data by printing the log before cash.

echo stop> rga2: This command switches the working status of RGA. When turned on, rga will return directly without working. Modulation used in some special situations.

echo int> rga2: This command switches the printing of RGA register interrupt information. When this print is turned on, the current values of the interrupt register and status baser will be printed after the RGA enters the interrupt.

echo slt> rga2: This command allows the rga driver to execute the internal SLT case to test whether the rga hardware is normal. If the log "rga slt success!!" is output, it means the function is normal.

- Switch debug node

The commands for opening and closing log printing are the same. Each time you enter a command to switch the state (on/off), you can confirm whether the log printing function is as follows through the log information ("open xxx" or "close xxx") printed after entering the command. Turn on or off as expected.

```
/ # echo reg > /sys/kernel/debug/rga2_debug/rga2
/ # dmesg -c                                                    //通过节点打开的相关日志的
打印等级为KERNEL_DEBUG，需要使用dmesg命令才能在串口或者adb看到对应的日志打印。
[ 4802.344683] rga2: open rga2 reg!
/ # echo reg > /sys/kernel/debug/rga2_debug/rga2
/ # dmesg -c
[ 5096.412419] rga2: close rga2 reg!
```

**Log description**

For RGA problem debugging, you need to use logs to confirm the final work performed by the RGA hardware. When the parameters of the HAL layer are passed into the driver, the following logs will describe the corresponding parameters. Usually we use three modes of msg, reg and time for debugging.

- msg mode

```
rga2: open rga2 test MSG!
                //msg日志开启打印。
rga2: cmd is RGA2_GET_VERSION
        //获取版本号功能，每个进程第一次调用librga时会查询硬件版本。
rga2: cmd is RGA_BLIT_SYNC
                //显示当前传入的工作模式。
```

rga2: render_mode:bitblt,bitblit_mode=0,rotate_mode:0 //render_mode显示调用接口，bitblit_mode为当前混合模式（0：**双通道模式**—A+B->B， 1：**三通道模式**A+B->C），rotate_mode为旋转角度。

rga2: src : y=0 uv=b4000072cc8bc040 v=b4000072cc99d040 aw=1280 ah=720 vw=1280 vh=720 xoff=0 yoff=0 format=RGBA8888 //src**通道的**图像数据参数：y：**如有**则为fd**的**值， uv：**如有**则为虚拟地址的值， v：vw * vh + uv， aw、ah：实宽实高，即实际操作图像区域，vw、vh：**虚**宽虚高，即图像本身大小，xoff、yoff：x、y**方向的偏移量**，format：传入的图像数据格式。

rga2: dst : y=0 uv=b4000072cc535040 v=b4000072cc616040 aw=1280 ah=720 vw=1280 vh=720 xoff=0 yoff=0 format=RGBA8888 //dst**通道的**图像数据参数。

rga2: mmu : src=01 src1=00 dst=01 els=00 //MMU**使能**标志，0为关闭，1为开启。

```
rga2: alpha : flag 0 mode0=0 mode1=0
        //blend相关配置
rga2: blend mode is no blend
        //blend混合模式
rga2: yuv2rgb mode is 0
                        //csc模式
rga2: *** rga2_blit_sync proc ***
```

- reg mode

```
rga2: open rga2 reg!
                //reg日志开启打印。
rga2: CMD_REG
                        //功能寄存器配置
rga2: 00000000 00000000 00000040 000e1040
rga2: 00119440 00000000 00000500 02cf04ff
rga2: 00000000 00000000 00000000 00000000
rga2: 00000000 00000000 00000000 00000040
rga2: 000e1040 00119440 00000500 02cf04ff
rga2: 00000000 00000000 0000ff00 ffffffff
rga2: 00000007 00000000 00000000 00000101
rga2: 07a80000 00000000 07a800e4 00000000
rga2: CSC_REG
                        //full csc寄存器配置
rga2: 00000000 00000000 00000000 00000000
rga2: 00000000 00000000 00000000 00000000
rga2: 00000000 00000000 00000000 00000000
rga2: CMD_READ_BACK_REG
                        //功能寄存器回读值
rga2: 00000000 00000000 00000040 000e1040
rga2: 00119440 00000000 00000500 02cf04ff
rga2: 00000000 00000000 00000000 00000000
rga2: 00000000 00000000 00000000 00000040
rga2: 000e1040 00119440 00000500 02cf04ff
rga2: 00000000 00000000 0000ff00 ffffffff
rga2: 00000007 00000000 00000000 00000101
rga2: 07a80000 00000000 07a800e4 00000000
rga2: CSC_READ_BACK_REG
                        //full csc寄存器回读值
rga2: 00000000 00000000 00000000 00000000
rga2: 00000000 00000000 00000000 00000000
rga2: 00000000 00000000 00000000 00000000
```

- time mode

```
rga2: open rga2 test time!
                //time日志开启打印。
rga2: sync one cmd end time 2414
        //打印本次工作RGA硬件的耗时，单位为us
```

# Q&A

This section classifies and introduces the more common RGA-related issues in the form of Q&A. If the issues are not covered in this section, please sort out the relevant logs and preliminary analysis information and hand them over to the engineers who maintain the RGA module.

## Performance consulting

**Q1.1** : How to evaluate RGA efficiency?

**A1.1** : When RGA executes copy, the theoretical time consumption can be calculated by the following formula (this function only supports data copy evaluation):

The time taken to copy an image for a single time = image width × image height / number of pixels that RGA can process per second

= Image width × Image height / (Number of pixels that RGA can process per clock cycle × RGA frequency)

For example: The theoretical time required to copy a 1920 × 1080 image using RGA (frequency set to 300M) is:

RGA1: 1920 × 1080 / (1 × 300000000) = 0.006912s

RGA2: 1920 × 1080 / (2 × 300000000) = 0.003456s

RGA3: 1920 × 1080 / (4 × 300000000) = 0.001728s

The actual time consumption is related to the memory type used. The efficiency of different incoming memory types from high to low is: physical address > dma_fd > virtual address.

When the system is unloaded, the actual time consumption of physical addresses is about 1.1-1.2 times of the theoretical time consumption, the actual time consumption of using dma_fd is about 1.3-1.5 times of the theoretical time consumption, and the actual time consumption of using virtual addresses is about It takes 1.8-2.1 times the theoretical time and is greatly affected by the CPU. Generally, we recommend developers to use dma_fd as the incoming memory type, which has a good balance between easy acquisition and efficiency. The virtual address is only used as a simple and easy-to-use memory type when understanding RGA in the learning stage. The following table shows the actual test data of different RGA frequencies when the system is no-load on RK3566.

Test **environment:**

| Chip platform | RK3566 |
|---|---|
| System platform | Android 11 |
| RGA frequency | 300M |
| CPU frequency | 1.8 Ghz |
| GPU frequency | 800M |
| DDR frequency | 1056 M |

Test **data:**

| resolution | memory type | Theoretical time consumption (us) | Actual time taken (us) |
|---|---|---|---|
| 1280 × 720 | GraphicBuffer(cache) | 1,536 | 2,620 |
| 1280 × 720 | GraphicBuffer(no cache) | 1,536 | 2,050 |
| 1280 × 720 | Drm buffer(cache) | 1,536 | 2,190 |
| 1280 × 720 | Physical address (Drm) | 1,536 | 2,000 |
| 1920 × 1080 | GraphicBuffer(cache) | 3,456 | 5,500 |
| 1920 × 1080 | GraphicBuffer(no cache) | 3,456 | 4,180 |
| 1920 × 1080 | Drm buffer(cache) | 3,456 | 4,420 |
| 1920 × 1080 | Physical address (Drm) | 3,456 | 4,100 |
| 3840 × 2160 | GraphicBuffer(cache) | 13,824 | 21,500 |
| 3840 × 2160 | GraphicBuffer(no cache) | 13,824 | 15,850 |
| 3840 × 2160 | Drm buffer(cache) | 13,824 | 16,800 |
| 3840 × 2160 | Physical address (Drm) | 13,824 | 15,600 |

**Q1.2:** The theoretical formula only provides the evaluation method of copy, so how to evaluate other modes?

**A1.2: **Currently only the copied formula is available for evaluation. In other modes such as scaling and cropping, you can use the larger resolution of the two images to bring the copied formula into the calculated time for evaluation. Usually There will be a certain fluctuation according to the size of scaling and cropping. The time-consuming mode of blending and other modes where the resolution does not change is about 1.1-1.2 times that of copy mode. Due to the impact of DDR bandwidth in specific actual scenarios, it is recommended that the actual evaluation be based on the actual test data in the target scenario.

**Q1.3:**Why does RGA perform very poorly in some scenarios and takes up to twice as long as running the demo?

**A1.3:** Because RGA has the lowest bus priority in the current RK platform, when bandwidth resources are tight, such as in a scenario where ISP runs multiple channels, RGA cannot read in time due to tight bandwidth resources. Writing data in DDR results in a large delay, which manifests as RGA performance degradation.

**Q1.4: **The efficiency of RGA cannot meet the needs of our products. Is there any way to improve it?

**A1.4:** The RGA frequency of the factory firmware of some chips is not the highest frequency. For example, the RGA frequency of 3399, 1126 and other chips can reach up to 400M. The RGA frequency can be increased in the following two ways:

- Set by command (temporary modification, the frequency will be restored when the device is restarted)

Query RGA frequency

```
cat /sys/kernel/debug/clk/clk_summary | grep rga
        //查询rga频率，其中的aclk的频率
```

Modify RGA frequency

```
echo 400000000 > /sys/kernel/debug/clk/aclk_rga/clk_rate
        //400000000修改为想要修改的频率
```

- Modify dts to modify the RGA frequency (it will still be the set frequency after restarting)

The following example shows how to modify the RGA frequency in dts on RK3288. Other platforms can modify it in the corresponding dts.

```
diff --git a/arch/arm/boot/dts/rk3288-android.dtsi
b/arch/arm/boot/dts/rk3288-android.dtsi
index 02938b0..10a1dc4 100644
--- a/arch/arm/boot/dts/rk3288-android.dtsi
+++ b/arch/arm/boot/dts/rk3288-android.dtsi
@@ -450,6 +450,8 @@
        compatible = "rockchip,rga2";
        clocks = <&cru ACLK_RGA>, <&cru HCLK_RGA>, <&cru SCLK_RGA>;
        clock-names = "aclk_rga", "hclk_rga", "clk_rga";
+       assigned-clocks = <&cru ACLK_RGA>, <&cru SCLK_RGA>;
+       assigned-clock-rates = <300000000>, <300000000>;
        dma-coherent;
 };
```

**Q1.5:** Does RGA support querying the current RGA hardware utilization (load) through commands or interfaces?

**A1.5:** The current version of the driver does not support this function. This function will be supported in the RGA3 version of the driver.

**Q1.6:** Why does calling RGA in asynchronous mode take longer than synchronous mode in some scenarios?

**A1.6:** Since the identifier of librga's asynchronous mode is currently an open device node, and a process of librga in singleton mode will only open one fd, imsync() waits for all asynchronous modes of the process to be opened. It will return after the run is finished.

**Q1.7:** Using a virtual address to call RGA for copying is more time-consuming than memcpy. Is there any way to optimize it?

**A1.7:** Generally we do not recommend using virtual addresses to call RGA, because the efficiency will be greatly reduced in scenarios with high CPU load. This is because the part of the RGA driver that converts virtual addresses into physical address page tables is controlled by the CPU. To calculate, and the process of converting virtual addresses into physical address page tables itself is very time-consuming. So usually we recommend using physical address or dma_fd to call librga.

**Q1.8:**Why is the RGA efficiency significantly lower than that of 4G when equipped with 8G DDR?

**A1.8:** Since the current MMU of RGA1/RGA2 only supports a maximum 32-bit physical address, when a buffer with a memory space larger than 4G is passed to the device equipped with 4G or above DDR, the RGA driver will pass The dma interface copies the data in the memory of the high-order address to the low-order memory reserved by swiotlb, and returns the corresponding address for RGA to read and write. After the RGA work is completed, the result is copied to the original high-order target address through dma, so The increased involvement of the CPU has seriously increased the overall working time of librga. For this situation where only RGA2/RGA1 is equipped and the device DDR is greater than 4G, it is recommended to use memory with less than 4G space when calling RGA to ensure the efficiency of RGA.

## Functional consultation

**Q2.1:**How do I know the RGA version of my current chip platform and the functions it can implement?

**A2.1:** You can check the "Overview" chapter in the "RGA IM2D API Development Guide" in the docs folder in the source code directory to learn about the RGA version and support information.

The source code paths of different systems will be different. The paths of the librga source code directory in different SDKs are as follows:

Android 7.0 or above SDK:

hardware/rockchip/librga

SDK below Android 7.0:

hardware/rk29/librga

Linux SDK:

external/linux-rga

**Q2.2:**How to call RGA to achieve hardware acceleration? Is there a demo for reference?

**A2.2: **1). For the API calling interface, you can check the "Application Interface Description" chapter in the "RGA IM2D API Development Guide" in the docs directory.

2). The demo is located in rga_im2d_demo under the sample directory. The demo internally implements most of the RGA interfaces and implements the corresponding RGA functions through command configuration. It can also be used as a tool to test whether the RGA is normal in some scenarios. It is recommended that developers who are new to RGA can directly run the demo and view the results in the early stage to understand the actual functions of RGA, then modify the parameters in the demo to implement the corresponding functions according to their own needs, and finally try to call the RGA API in their own projects alone.

**Q2.3: **Support information for RGA?

**Q2.3.1: **What formats does RGA support?

A2.3.1 **: For specific support information, you can check the "Overview" - "Image Format Support" section in the "RGA IM2D API Development Guide" to query the format support of the RGA equipped with the corresponding chip version, or you can call querystring ( RGA_INPUT_FORMAT | RGA_OUTPUT_FORMAT);** The interface queries the input and output format support of the current hardware.

**Q2.3.2: **What is the zoom ratio supported by RGA?

A2.3.2 **: For specific support information, you can check the "Overview" - "Design Indicators" section of the "RGA IM2D API Development Guide" to query the scaling ratio supported by RGA on the corresponding chip version, or you can call querystring(RGA_SCALE_LIMIT) in the code );** Interface queries the zoom ratio supported by the current hardware.

**Q2.3.3: **What is the maximum resolution supported by RGA?

A2.3.3 **: For specific support information, you can check the "Overview" - "Design Indicators" section of the "RGA IM2D API Development Guide" to query the maximum input and output resolution supported by RGA on the corresponding chip version, or you can call it in the code. querystring(RGA_MAX_INPUT | RGA_MAX_OUTPUT);** The interface queries the maximum input and output resolution supported by the current hardware.

**A2.3:** Generally speaking, if you have any questions about RGA support, you can check the "RGA IM2D API Development Guide", which will provide a more detailed introduction to RGA support information.

**Q2.4:** What are the differences between the new version of librga and the old version of librga? How to tell the difference?

**A2.4:** In the current RK platform, there are two versions of librga:

The old version of librga has currently stopped supporting and maintaining. The main symptom is that the SDK released before November 2020 is equipped with the old version of librga. Some chip platforms such as RK3399 Linux SDK SDK released before June 2021 ( V2.5 or below) is also an old version of librga. This version of librga cannot perfectly fit the newer driver, and may cause problems such as color deviation and format abnormalities. Mixed use is not recommended. If you need to use a newer kernel, it is recommended to update. The new version of librga, and vice versa, the kernel needs to be updated to match.

The new version of librga is currently the version that is mainly supported and maintained. The main feature is the addition of **the im2d_api** directory under the source code directory. This version integrates with the old version of librga and launches a simple and easy-to-use IM2D API. It can also be called the IM2D version of librga. The new version of librga not only supports the new IM2D API, but also supports the old version of RockchipRga interface and C_XXX interface. For specific API call instructions, please refer to the "RGA IM2D API Development Guide". This version adds a new software management version number that can be queried through **the querystring(RGA_VERSION);** interface.

Generally, for the function support of some old and new versions of librga, it is generally recommended to update the overall SDK to avoid dependency problems. It is strongly not recommended to use the new version of librga with the old driver or the old version of librga with the new kernel. In some scenarios, there will be obvious errors.

**Q2.5:** Does RGA have alignment restrictions?

**A2.5:** Different formats have different alignment requirements. The RGA hardware itself fetches the data of each line of the image in a word (world) aligned manner, that is, 4 bytes and 32 bits. For example, the RGBA format itself has a single pixel storage size of 32 (4 × 8) bits, so there is no alignment requirement; the RGB565 format has a storage size of 16 (5 + 6 + 5) bits, so 2 alignment is required; the RGB888 format has a storage size of 24 (8 × 3) bit, so this format requires 4 alignments to meet the 32-bit access requirements of RGA hardware; YUV format storage is relatively special, and its own arrangement requirements require 2 alignments. The Y channel single pixel storage size is 8 bits, and the UV channel is based on 420/422 Determines the storage size of each four pixels, so the Y channel of the YUV format requires 4 alignments to meet the hardware access requirements of RGA, so the YUV format requires 4 alignments; the principles of alignment requirements for other formats not mentioned are the same. Note that the alignment in this question refers to the alignment requirements of width stride. The actual width, height, and offset of the YUV format itself also require 2 alignment due to the characteristics of the format itself. For specific alignment restrictions, please view the "Overview" - "Image Format Alignment Instructions" section in the "RGA IM2D API Development Guide".

**Q2.6:** Can RGA support drawing multiple rectangular areas at one time, or perform multiple operations? How does RGA work?

**A2.6: **RGA can only work sequentially on hardware, that is, the completion of one configured task and the start of the next configured task. Therefore, you cannot draw multiple rectangular areas at one time. You can configure the work that needs to be done by RGA to the underlying driver through async mode. RGA will store the work in a work queue managed by the driver itself and complete it in order. When the upper layer needs to process this buffer, it calls **imsync()** to determine whether the RGA hardware has completed the work.

**Q2.7:** Can RGA's fill function support YUV format?

**A2.7:** Old versions of librga are not supported. Only new versions of librga that include the following submissions are supported. If there is no such submission, please try updating the SDK to the latest version.

```
commit 8c526a6bb9d0e43b293b885245bb53a3fa8ed7f9
Author: Yu Qiaowei <cerf.yu@rock-chips.com>
Date:   Wed Dec 23 10:57:28 2020 +0800

    Color fill supports YUV format as input source.

    Signed-off-by: Yu Qiaowei <cerf.yu@rock-chips.com>
    Change-Id: I0073c31d770da513f81b9b64e4c27fee2650f30b
```

This function is consistent with the RGB color fill call. You fill the color by configuring the RGB value that needs to be filled. The difference is that the output result can be set to YUV format.

**Q2.8: **Does RGA support YUYV format?

**A2.8:** The old version of librga (here refers to the librga in the SDK released before October 2020) is not supported. Only the new version of librga (the version with **im2d_api** directory in the source code directory) is included The librga versions after the following submissions are supported. If there is no such submission, please try updating the SDK to the latest version.

```
commit db278db815d147c0ff7a80faae0ea795ceffd341
Author: Yu Qiaowei <cerf.yu@rock-chips.com>
Date:   Tue Nov 24 19:50:17 2020 +0800

    Add support for Y4/YUV400/YUYV in imcheck().

    Signed-off-by: Yu Qiaowei <cerf.yu@rock-chips.com>
    Change-Id: I3cfea7c8bb331b65b5bc741956da47924eeda6e1
```

**Q2.9: **Does RGA support scaling of grayscale image input and output?

**A2.9:** The old version of librga (here refers to the librga in the SDK released before October 2020) is not supported. Only the new version of librga (the version with the **im2d_api** directory in the source code directory) 1.2. Only version 2 supports grayscale image input. If the librga version is lower than this version, please try updating the SDK to the latest version. Since the RGA hardware itself does not support the grayscale image format, the

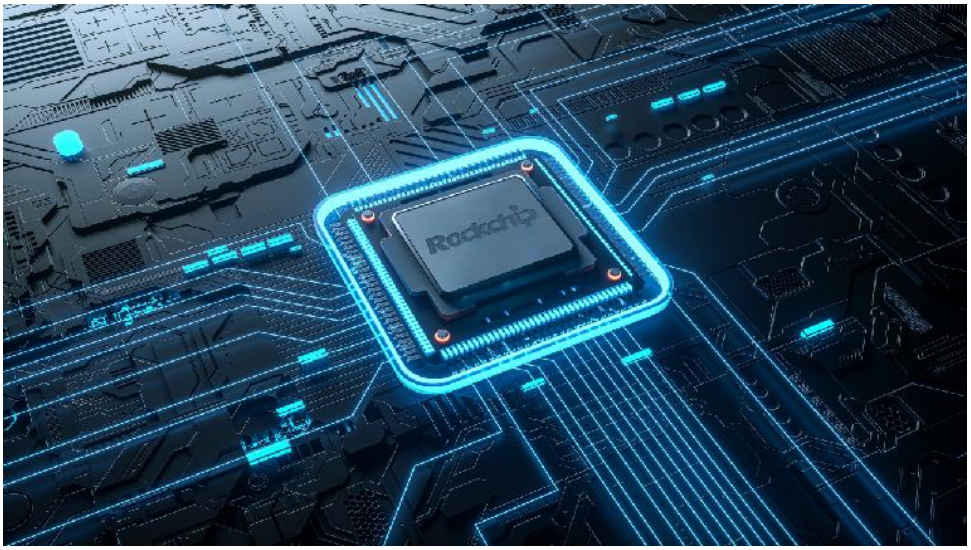
format used for the grayscale image here is **RK_FORMAT_Y400** , which is represented by the YUV format without UV channels. YUV with only Y channel is a 256-level grayscale image.

**Q2.10:** Why does the ROP code on RK3399 have no corresponding effect when executed on RV1126?

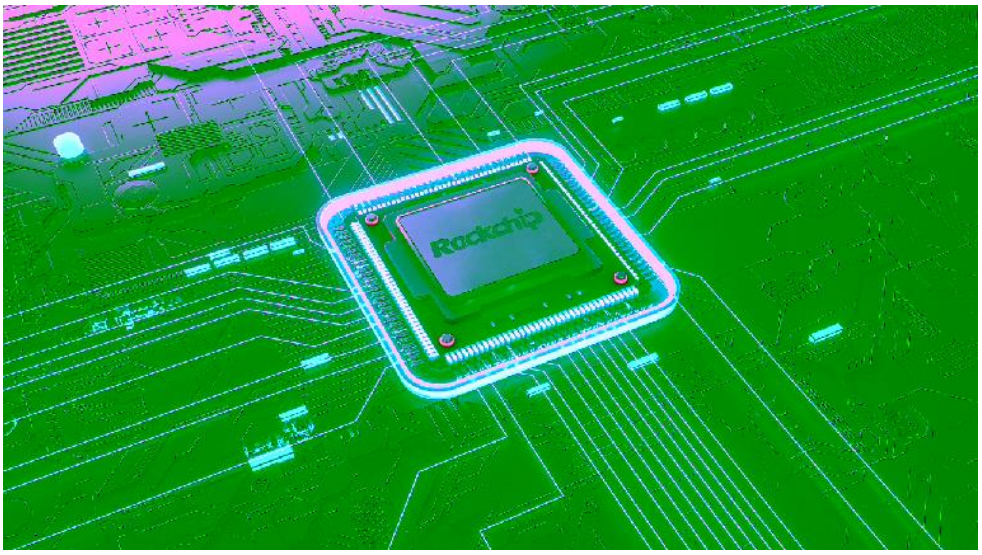
**A2.10: **Although the RGAs installed on RK3399 and RV1126 are both RGA2-ENHANCE, their small versions are different. The ROP function has been cut out on RV1126. For specific function support, please check "RGA IM2D API Development Guide" or call **querystring(RGA_FEATURE)** in the code ; the interface implements the query support function.

**Q2.11: **Other functions are normal when using RGA, but serious color difference (pinkish and greenish) occurs only when converting RGB and YUV formats. What is the reason?

Expectation:




Result:

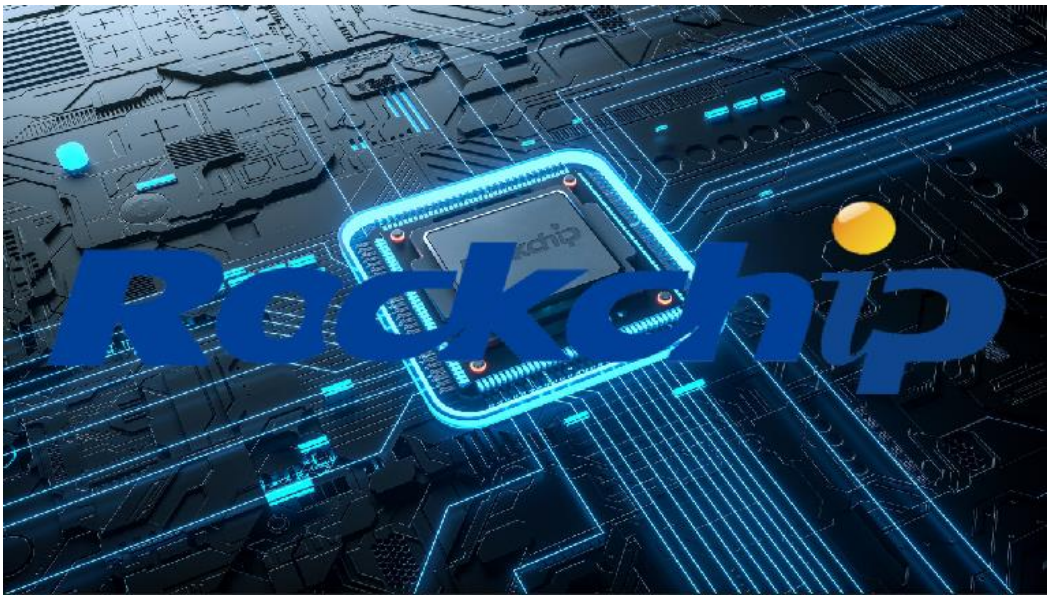


**A2.11:** This phenomenon is usually caused by the mismatch between librga and the kernel. For detailed version instructions, please view **A2.4** . This problem usually occurs after some SDKs released before November 2020 use librga obtained from github. Librga is updated on github to a new version of librga, which does not match the older version of the RGA driver. Some configurations of the color gamut space have changed, so there will be an obvious color shift.

There are two solutions to this problem. One is to update the SDK or RGA driver and keep librga and the driver matching. The second is if you do not need the functions of the new version of librga, you can use the librga that comes with the SDK. That's it.

**Q2.12:** How does RGA implement OSD superimposed subtitles?

Expectation:



**A2.12:** If the output result is in RGB format, it can be implemented through the **imblend()** interface. Usually the src over mode is selected to superimpose the src channel

image on the dst channel image; if the output result is in YUV format, you can Implemented through **the imcomposite()** interface, usually select dst over' mode, superimpose the image of the src1 channel on the image of the src channel, and then output it to the dst channel.

The overlay principle of this function is **the Porter-Duff hybrid model** . For details, please view the "Application Interface Description" - "Image Synthesis" section in the "RGA IM2D API Development Guide".

The reason why RGA requires different configurations for different output formats is that RGA2 has 3 image channels - src, src1/pat, and dst. The src channel supports YUV2RGB conversion, the src1/pat and dst channels only support RGB2YUV conversion, and the superposition within RGA needs to be performed in RGB format, so in order to ensure that the RGB image is superimposed on the YUV image, src must be used as the superimposed background image YUV , src1 is used as the superimposed foreground image RGB, and the mixed RGB image is finally converted into YUV format and output by the dst channel.

**Q2.13:** Why is there a brightness or numerical difference in the output when calling RGA to convert YUV format and RGB format?

**A2.13:**The causes of this phenomenon can be roughly divided into two types:

1). When the YUV and RGB mutual conversion configurations are the same, some pixel values will be slightly different (usually the difference is 1). This is due to the accuracy of the formula when the RGA hardware implements the CSC function. The decimals of the CSC formulas of RGA1 and RGA2 The bit precision is 8 bits, and the decimal place precision of RGA3's CSC formula is 10 bits. Due to the precision here, some operation results will have an error of ±1 after rounding.

**2). When the CSC mode configured when converting RGB2YUV and YUV2RGB is different, the default CSC mode of RGB2YUV and YUV2RGB in the new version of librga is BT.601-limit _range. When the corresponding color_space_mode** member variable is incorrectly configured , the color gamut Different configurations of space will lead to larger changes in mutual conversion. In the old version of librga, RGB2YUV defaults to BT.601-full_range, and YUV2RGB defaults to BT.709-limit_range. Since the color gamut space configurations of the two conversions are different, there will be major changes in the mutual conversion.

**Q2.14: **How to configure the color gamut space during format conversion in librga?

**A2.14:**Both versions of librga support the color gamut space when configuring format conversion.

1). In the new version of librga, you can refer to the "Application Interface Description" - "Image Format Conversion" section in the "RGA IM2D API Development Guide" and focus on configuring the mode parameters.

2). In the old version of librga, you need to modify the librga source code, the value of yuvToRgbMode in Normal/NormaRga.cpp, the corresponding parameters are as follows:

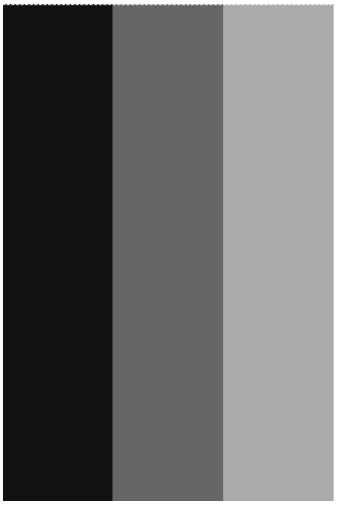
| Convert format | color gamut space | parameter |
| --- | --- | --- |
| YUV2RGB | BT.601-full_range | yuvToRgbMode = 0x1 << 0; |
| YUV2RGB | BT.601-limit_range | yuvToRgbMode = 0x2 << 0; |
| YUV2RGB | BT.709-limit_range | yuvToRgbMode = 0x3 << 0; |
| RGB2YUV | BT.601-full_range | yuvToRgbMode = 0x2 << 4; |
| RGB2YUV | BT.601-limit_range | yuvToRgbMode = 0x1 << 4; |
| RGB2YUV | BT.709-limit_range | yuvToRgbMode = 0x3 << 4; |

**Q2.15:** Calling RGA to perform alpha overlay, why does it have no effect?

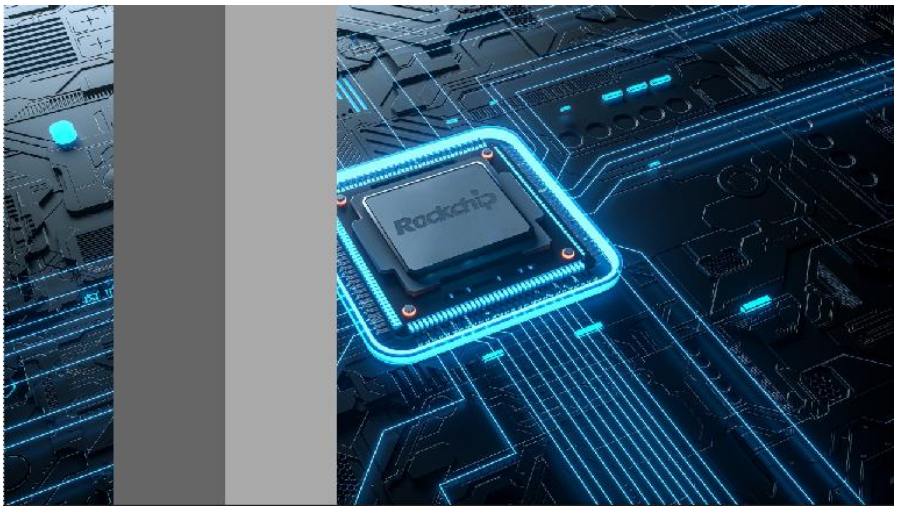
**A2.15:** Check whether the alpha values of the two input images are both 0xFF. When the alpha value of the overlaying foreground image is 0xFF, the result is that the foreground image is directly overlaid on the background image. It looks like The result looks like it has no effect, but it is actually a normal result.

**Q2.16:** Call RGA to perform alpha overlay. The alpha value of the foreground image is 0x0. Why is the result not fully transparent?

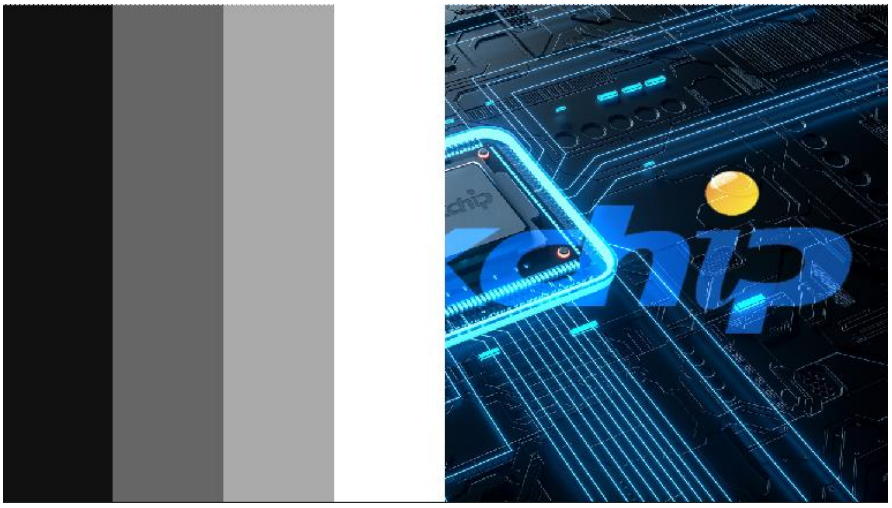
Foreground image: (black and white and rockchip alpha is 0x00)



Expectation:




Result:



**A2.16:** Our normal configuration mode is that the default color value has been premultiplied by the corresponding alpha value, and the color value of the original image read directly has not been premultiplied by the alpha value, so it needs to be called When imblend, an additional flag bit is added to indicate that the image color value in this processing does not need to be pre-multiplied by the alpha value. For the specific calling method, please refer to the "Application Interface Description" - "Image Synthesis" section in the "RGA IM2D API Development Guide".

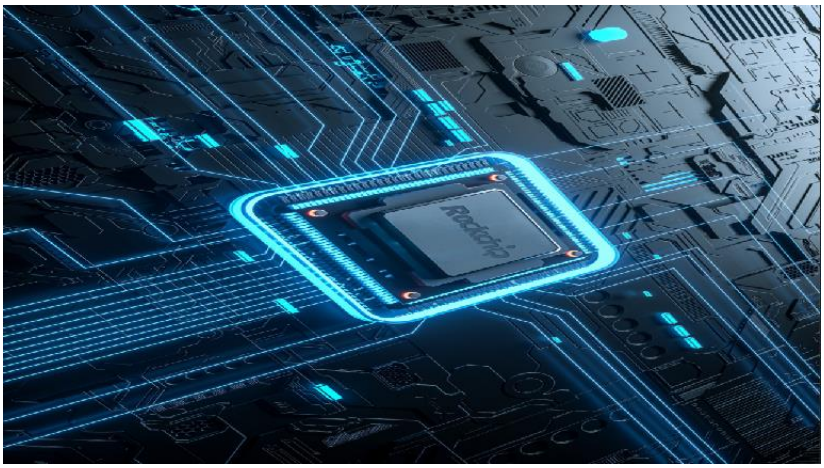
**Q2.17:** Can the IM2D API implement multiple functions with one RGA call?

**A2.17:** Yes, for details, please refer to the "Application Interface Description" - "Image Processing" section in the "RGA IM2D API Development Guide", and refer to the implementation of other interfaces of the IM2D API to understand the usage of **improcess()** .

**Q2.18:** When calling RGA to perform image rotation, the resulting image is stretched?

Expectation:

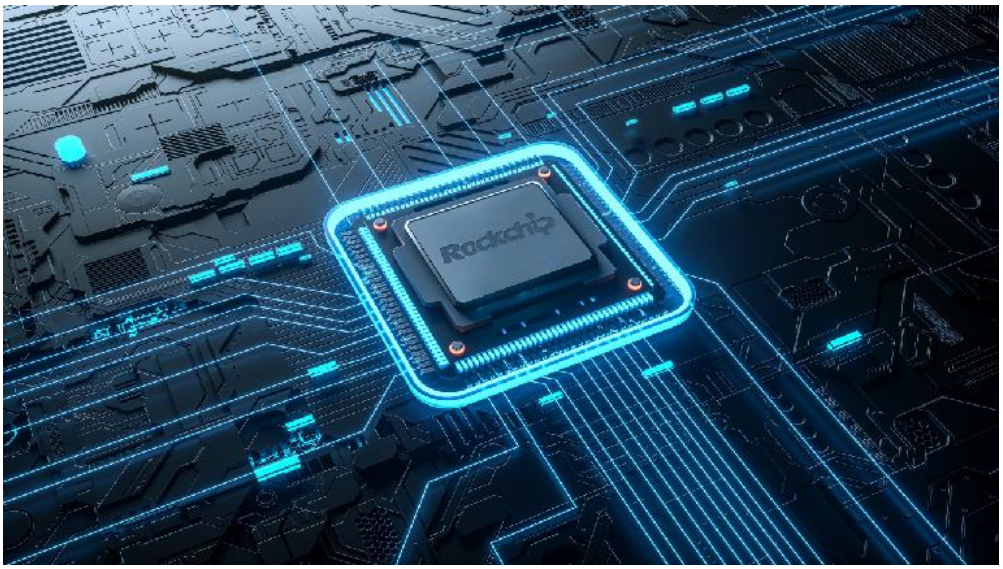
Result:



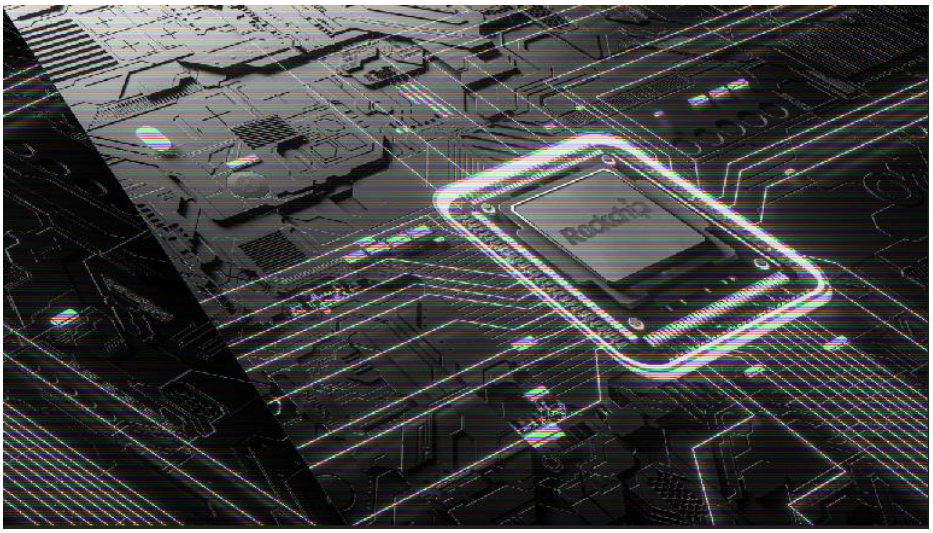
**A2.18:** When rotating 90° or 270°, if you do not want RGA to perform scaling, you should exchange the width and height of the image. Otherwise, the RGA driver will default to the rotation + scaling behavior to perform the work. The result The performance is the effect of stretching.

**Q2.19: **After RGB888 output scaling, the result shows that the image is slanted and has black lines?

Original image (1920 × 1080):




Result (1282 × 720):




**A2.19:** This problem is caused by alignment restrictions. The virtual width of RGB888 format requires 4 alignments. Please check the configured image parameters. For alignment restrictions, please refer to the answer to **Q2.5** .

**Q2.20:** In some system processes, the output result of calling RGA is spent. What is the reason for this?

**A2.20: **Usually RGA anomalies will not cause image corruption. Generally, when encountering this kind of problem, you need to first locate whether the problem is caused by RGA. In some system processes, you need to first confirm the source of the input RGA. To check whether the data is abnormal, you can call **fwrite()** to write the data in the memory to a file before calling RGA to check whether the source data is normal. If you are not familiar with the method of writing files, you can refer to the implementation part of **the output_buf_data_to_file()** function in **core/RgaUtils.cpp** in the source directory .

## HAL layer error

**IM2D_API error report**

**Q3.1.1: **imcheck() returns an error, how to deal with it?

```
check error! Invalid parameters: dst, Error yuv not align to 2,
rect[x,y,w,h] = [0, 0, 1281, 720], wstride = 1281, hstride = 720, format =
0xa00(nv12)
output support format : RGBA_8888 RGB_888 RGB_565 RGBA_4444 RGBA_5551
YUV420/YUV422 YUV420_10bit/YUV422_10bit YUYV420 YUYV422 YUV400/Y4
```

**A3.1.1: **imcheck() interface is used as a verification interface for calling librga. It will determine whether the parameters of the data structure to be passed to librga are correct, whether the function is supported, whether hardware restrictions are triggered, etc. You can use imcheck() )'s return error value is passed into **IMStrError()** as a parameter. The returned string is detailed error information. You can confirm which conditional restrictions are triggered or parameter errors based on the error prompts.

If an error is reported in the question, it is due to the limitation of YUV format alignment. The image width 1281 here is not aligned with 2, so the verification fails.

**RockchipRga interface error report**

**Q3.2.1:** How to deal with the "Try to use uninit rgaCtx=(nil)" error?

**A3.2.1:** The principle of this error is that the called interface finds that the librga module has not been initialized, and the error is returned. The error reported in the current version is usually due to the fact that some older codes that call RGA still use the RgaInit/RgaDeInit/c_RkRgaInit/c_RkRgaDeInit interface to manage the initialization of the RGA module by themselves. The current version of the interface uses the singleton mode. When it is abnormally DeInit , the error will appear, just remove the Init/DeInit related calls in the calling code.

**Q3.2.2:** What is the reason for the error "RgaBlit(1027) RGA_BLIT fail: Not a typewriter"?

**A3.2.2:** This error is usually caused by parameter errors. It is recommended to check whether the scaling factor and virtual width are less than the sum of the real width and the offset in the corresponding direction, and whether the alignment meets the requirements. It is recommended that new development projects use the IM2D API, which has a more comprehensive detection and error reporting mechanism to facilitate developers to save a lot of debugging time.

**Q3.2.3:** Why does the "RgaBlit(1349) RGA_BLIT fail: Bad file descriptor" exception error return?

**A3.2.3:** This error is an ioctl error. The fd identifying the currently passed in device node is invalid. Please try to update librga or confirm whether the RGA initialization process has been modified.

**Q3.2.4:** What is the reason for the "RgaBlit(1360) RGA_BLIT fail: Bad address" error?

**A3.2.4: **This error is usually caused by a problem with the memory address of the src/src1/dst channel passed into the kernel (commonly out of bounds). You can refer to this document "Log Acquisition and Description" - "Driver Debugging Node" " section, turn on the driver log and locate the faulty memory.

**Q3.2.5:** What are the errors "err ws[100,1280,1280]" and "Error srcRect" reported in the log?

**A3.2.5: **This error is an obvious parameter error, "err ws" means the virtual width (width stride) parameter is abnormal, and the parameters in the subsequent "[]" are [x_offeset, width, width_stride], here Since the sum of the offset in the X direction and the width of the actual operation area is greater than the virtual width, librga thinks there is a problem with the virtual width and returns an error. Here just change the virtual width to 1380 or the real width (width) to 1180.

Usually after this type of error is reported, some corresponding parameters will be printed in logcat:

```
E librga  : err ws[100,1280,1280]
                                                    //标识单签虚宽存在问题
E librga  : [RgaBlit,731]Error srcRect
                                              //标识是src通道报错
E rockchiprga: fd-vir-phy-hnd-format[0, 0xb400006eb6ea9040, 0x0, 0x0, 0]
        //对应src通道的输入地址（fd、虚拟地址、物理地址、handle）。
E rockchiprga: rect[100, 0, 1280, 720, 1280, 720, 1, 0]
                                  //对应src通道的图像参数依次为：x方向偏移、y方向偏移
、实际操作区域的宽、实际操作区域的高、图像宽（虚高）、图像高（虚高）、图像格式、size（目
前没有使用到的参数）。
E rockchiprga: f-blend-size-rotation-col-log-mmu[0, 0, 0, 0, 0, 0, 1]
                //标识着本次调用中的模式配置。
E rockchiprga: fd-vir-phy-hnd-format[0, 0xb400006eb2ea6040, 0x0, 0x0, 0]
        //对应dst通道的参数
E rockchiprga: rect[0, 0, 1920, 1080, 1920, 1080, 1, 0]
E rockchiprga: f-blend-size-rotation-col-log-mmu[0, 0, 0, 0, 0, 0, 1]
E rockchiprga: This output the user patamaters when rga call blit fail
                //报错信息
```

## kernel layer error

**Q4.1:** What causes the error "RGA2 failed to get vma, result = 32769, pageCount = 65537"?

**A4.1:** This error is usually caused when calling RGA using a virtual address. The actual memory of the virtual address is smaller than the actual required memory size (that is, how much memory is needed for the image of the current channel based on the image parameters). Just check The size of the buffer is sufficient. In some scenarios where the application and the call are not in the same place, you can execute memset to determine the size of the corresponding image before calling RGA to confirm whether the problem is caused by insufficient memory size.

After correcting the error, you can usually confirm which channel's memory problem is caused by "rga2 map src0 memory failed". As shown in this example, the buffer size actually applied for for the src channel is only half of the required size of the image. , so this error was triggered.

**Q4.2:** "rga2_reg_init, [868] set mmu info error" What is the reason for the MMU error?

**A4.2:** This error is characterized by an error in converting fd/virtual address to physical address page table. It is usually a problem with the requested memory size, which is the same as Q4.1.

**Q4.3:** When "rga: dma_buf_get fail fd[328]" reports this kind of error, it generally refers to some abnormality in the buffer?

**Q4.3:** This error is reported when fd passes through the dma interface in the kernel. It is recommended to check the process of applying for fd, and verify that fd is available outside librga before calling RGA.

**Q4.4:** "RGA2 failed to get pte, result = -14, pageCount = 112", "rga2_reg_init, [868] set mmu info error" After checking according to **Q4.1** and **Q4.2** , the same error is reported here. The physical address allocated by DRM is used, and the virtual address mapped by mmap is passed to RGA. Memset is normal. What is the reason?

**A4.4:** This problem is a problem of the allocator DRM itself. DRM itself believes that after the user state obtains the physical address, normally the kernel state does not need the virtual address, so it will be allocated when the buffer is allocated. Release the corresponding kmap. Only releasing kmap will not affect the mapping and use of virtual addresses in user mode. However, when the virtual address of this buffer user mode is passed to the RGA driver and the driver performs conversion query of the physical address page table, due to The buffer's kmap has been released, or the corresponding page table entry cannot be queried, or the wrong address is directly accessed, causing the kernel to crash.

For this scenario, DRM provides an interface flag bit. The user can judge whether the user mode wants DRM to release kmap, that is, whether to consider passing the mapped virtual address to the kernel for use:

```
(1) drm buffer申请选项增加ROCKCHIP_BO_ALLOC_KMAP定义。
+       /* keep kmap for cma buffer or alloc kmap for other type memory */
+   ROCKCHIP_BO_ALLOC_KMAP  = 1 << 4,
(2) 申请drm内存时，增加新增的drm buffer选项ROCKCHIP_BO_ALLOC_KMAP。
        struct drm_mode_create_dumb arg;
        ...
-       arg.flags = ROCKCHIP_BO_CONTIG;
+       arg.flags = ROCKCHIP_BO_CONTIG | ROCKCHIP_BO_ALLOC_KMAP;
        //ROCKCHIP_BO_ALLOC_KMAP仅与ROCKCHIP_BO_CONTIG共同使用时有效。
        ret = drmIoctl(drm_fd, DRM_IOCTL_MODE_CREATE_DUMB, &arg);
```

And confirm whether the kernel contains the following submissions. If not, please update the SDK:

```
commit 1a81ee3e2d3726b9382ff2c48d08f4d837bc0143
```

```
Author: Sandy Huang <hjc@rock-chips.com>
Date:   Mon May 10 16:52:04 2021 +0800

    drm/rockchip: gem: add flag ROCKCHIP_BO_ALLOC_KMAP to assign kmap

    RGA need to access CMA buffer at kernel space, so add this flag to keep
kernel
    line mapping for RGA.

    Change-Id: Ia59acee3c904a495792229a80c42f74ae34200e3
    Signed-off-by: Sandy Huang <hjc@rock-chips.com>
```

**Q4.5:** "rga: Rga err irq! INT[701],STATS[1]" What causes the interrupt error when calling RGA?

**A4.5:** This problem usually occurs when the RGA hardware encounters a problem and returns abnormally during execution. There are many reasons for the abnormality. Common ones include memory out-of-bounds and abnormal configuration. It is recommended that if you encounter this problem, you should first check whether the incoming memory will go out of bounds.

**Q4.6:** What usually causes the hardware timeout error "rga: Rga sync pid 1001 wait 1 task done timeout"?

**A4.6:** There are many reasons for hardware timeout errors, which can be checked according to the following situations:

1). Check the overall process and confirm that no other module or application holds a lock on the buffer or is abnormally occupied. When the same buffer is occupied abnormally by other modules, RGA cannot read and write data normally, exceeding the 2000ms limit of the driver design. After the threshold is exceeded, an exception will be returned and an error will be printed.

2). Check the DDR bandwidth and utilization of the current system. Since the bus priority of RGA is low, when the DDR load is full, if the RGA is not completed within 2000ms, the driver will return abnormally and print the error.

3). Confirm whether there have been errors from other IP modules before the RGA timeout error, such as ISP, vpu, etc. When there is a problem with the hardware on the same bus, the RGA may not work properly. The driver waits for more than 2000ms. , an exception will be returned and an error will be printed.

4). Confirm the current RGA frequency (you can refer to the RGA frequency related operations in **Q1.4** ). In some scenarios, the module on the same bus may be down-converted and affect the RGA frequency. The RGA frequency decreases, resulting in overall performance degradation. , unable to complete the work within 2000ms, the driver will return abnormally and print an error report.

5). Some chip RGA is overclocked to a higher frequency. At this time, the RGA frequency increases but the voltage does not increase, which will cause the overall performance of the RGA to decrease significantly, resulting in the inability to complete the work within the specified threshold, thus causing the driver to return abnormally and print. Report an error. In

this scenario, it is recommended that developers modify the RGA frequency to a normal frequency. Overclocking will have an impact on the stability and service life of the overall chip. This behavior is strongly not recommended.

6). No problems are found in the above scenarios. You can try to write the data in the target memory to the file after the RGA times out and return an error. Check whether the RGA has written part of the data. If it has written part of the data, please reconfirm. In scenarios 1-5, this phenomenon is obviously caused by insufficient RGA performance; if the target memory is not written with data by RGA, collect the corresponding log information and related experimental processes, and contact the engineer who maintains the RGA module.