RGA FAQ

File No.：RK-PC-YF-0003

Current Version：V1.0.0

Finish Date：2021-06-28

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

Disclaimer

This document is provided “as is” and Fuzhou Rockchip Electronics Co. Ltd (“the company”) makes no express or implied statement or warranty as to the accuracy, reliability, completeness, merchantability, specific purpose and non-infringement of any statement, information and contents of the document. This document is for reference only.

This document may be updated without any notification due to product version upgrades or other reasons.

Brand Statement

Rockchip, RockchipTM icon, Rockchip and other Rockchip trademarks are trademarks of Fuzhou Rockchip Electronics Co., Ltd., and are owned by Fuzhou Rockchip Electronics Co., Ltd.

All other trademarks or registered trademarks mentioned in this document are owned by their respective owners.

Fuzhou Rockchip Electronics Co., Ltd.

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

Website: www.rock-chips.com

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

Customer service fax: +86-591-83951833

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

Readership

This document is intended for:

- Technical support engineers
- Software development engineers

Revision History

Date	Version	Author	Description
2021/06/28	1.0.0	Yu Qiaowei	Initial version.

Contents

1. Overview

For RGA driver and user-mode API librga, this document summarizes some common problems occurred when RGA hardware is called on RK platform to realize graph drawing acceleration with OSD (On Screen Display) and GUI (Graphics User Interface).

2. Version Description

2.1 Hardware Version

RGA hardware consists of three versions: RGA1, RGA2, and RGA3. See section Introducions in Rockchip_Developer_Guide_RGA_EN.pdf for detailed platform information, supported functions, and restrictions.

2.2 Software Version

2.2.1 librga

The API version number consists of major, minor, revision and build. The four levels of version number correspond to different levels of function update. See section API Version Description in 《RGA_API_Instruction.md》 for details.

2.2.2 RGA Driver

The driver version number consists of major, minor, revision and build. The four levels of version number correspond to different levels of function update. Usually HAL library and driver is matching in released SDK, version verification is done within librga, developers do not need to consider about version. If the following error occurs when librga is updated separately, you need to update the driver to the corresponding version.

3. Log Obtaining and Description

3.1 HAL Logs

3.1.1 Log Switch

- Android

  Android supports using Android Property to configure whether librga enables HAL log printing:

  > Android 7.1 or below

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

> Android 8.0 or later

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

- Linux

  Linux does not support configuring log switch using attributes. You need to modify the macro definition of the following directories and recompile to enable HAL log printing:

> core\NormalRgaContext.h

```
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
```

3.1.2 Log Description

- Regular Log

```
//When each process calls librga for the first time, it initializes a singleton
of librga and prints information such as the current API version number.
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 -------->>>>
//The parameters passed to librga are printed as follows.
D librga  : src->hnd = 0x0 , dst->hnd = 0x0 , src1->hnd = 0x0
//The handle passed in by the three channels (src、src1、dst).
D librga  : src: Fd = 00 , phyAddr = 0x0 , virAddr = 0xb400007431ed6040
//Parameters passed in src channel, representing DMA_FD, physical address and
virtual address respectively.
D librga  : dst: Fd = 00 , phyAddr = 0x0 , virAddr = 0xb400007431b4f040
//Parameters passed in dst channel, representing DMA_FD, physical address, and
virtual address respectively.
```

```
D librga  : src: Fd = -01 , buf = 0xb400007431ed6040, mmuFlag = 1, mmuType = 0
//The src channel configures the value for the type of memory passed and whether
or not MMU is enabled, where the HAL selects the virtual address to pass into the
driver.
D librga  : dst: Fd = -01 , buf = 0xb400007431b4f040, mmuFlag = 1, mmuType = 0
//The dst channel configures the value for the type of memory passed and whether
or not MMU is enabled, where HAL selects the virtual address to pass in the
driver.
E librga  : blend = 0 , perpixelAlpha = 1
//Blending mode and whether the image format includes an Alpha value.
D librga  : scaleMode = 0 , stretch = 0;
//scale mode (RGA1).
E librga  : rgaVersion = 3.200000  , ditherEn =0
//Hardware version, Y4 Dither enabled.
D librga  : srcMmuFlag = 1 , dstMmuFlag = 1 , rotateMode = 0
//MMU enable flag, rotation mode.
D librga  : <<<<-------- rgaReg -------->>>>
//Parameters configuration of the driver are printed as follows.
E librga  : render_mode=0 rotate_mode=0
//RGA rendering mode, rotation mode.
E librga  : src:[0,b400007431ed6040,b400007431fb7040],x-y[0,0],w-h[1280,720],vw-
vh[1280,720],f=0    //Memory, image parameters, format information of src
channel.
E librga  : dst:[0,b400007431b4f040,b400007431c30040],x-y[0,0],w-h[1280,720],vw-
vh[1280,720],f=0    //Memory, image parameters, format information of dst
channel.
E librga  : pat:[0,0,0],x-y[0,0],w-h[0,0],vw-vh[0,0],f=0
//Memory, image parameters, format information of pat/src1 channel. This channel
is not used in current mode, therefore the parameters are set to 0.
//The following are some of the parameters that developers usually don't have to
care about, which is to configure for librga different modes of the driver.
E librga  : ROP:[0,0,0],LUT[0]
//ROP and LUT configuration.
E librga  : color:[0,0,0,0,0]
//Colorkey configuration（max color, min color）, color-filling configuration
(foreground color, background color, color-filling).
E librga  : MMU:[1,0,80000521]
//MMU configuration.
E librga  : mode[0,0,0,0]
//Palette、csc、colorkey configuration.
E librga  : Full CSC : en[0]
//Full csc enabled flag.
E librga  : gr_color_x [0, 0, 0]
// Color-filling configuration, corresponding the value of R, G, B.
```

3.2 Driver Debug Node

3.2.1 Log Switch

- Debugging Node Address

Different RGA hardware versions open the driver debugging node in the same way, but the debugging node is stored in the folder of the corresponding hardware version:

> /sys/kerne/debug/rkrga/debug

- Debugging Description

Take RGA2 as an example. Users can use cat node in the corresponding directory to obtain the corresponding fuction description:

```
/# cd /sys/kerne/debug/rkrga/
/# cat debug
REG [DIS]
MSG [DIS]
TIME [DIS]
INT [DIS]
CHECK [DIS]
STOP [DIS]

help:
 'echo reg > debug' to enable/disable register log printing.
 'echo msg > debug' to enable/disable message log printing.
 'echo time > debug' to enable/disable time log printing.
 'echo int > debug' to enable/disable interruppt log printing.
 'echo check > debug' to enable/disable check mode.
 'echo stop > debug' to enable/disable stop using hardware
```

> echo reg > debug：This command switches the printing of RGA register configuration information. When it is opened, the configuration of RGA register is printed.
>
> echo msg> debug：This command switches the printing of RGA register upper-layer configuration information. When it is opened, The parameters passed by the upper-level call to the RGA driver will be printed out.
>
> echo time> debug：This command switches the printing of RGA time information. When it is opened, the time taken for each rga call is printed.
>
> echo check> debug：This command switches the RGA internal test case. When it is opened, corresponding parameters are checked when RGA is working, mainly memory and alignment checks. If the following log is printed, the check is successful. If the memory exceeds the threshold, the kernel crashes. You can check whether there is a problem with src data or dst data through the print log before cash.
>
> echo stop> debug：This command switches the RGA working status. When it is opened, rga directly returns without working. A mode used in some special cases.
>
> echo int> debug：This command switches the printing of RGA register interrupt information. When it is opened, the current value of the interrupt register and state base will be printed after the RGA enters the interrupt.

> echo slt> debug：This command causes rga driver to perform an internal SLT case to test whether the RGA hardware is working well. If log “rga slt success !!” is printed, it indicates that the function is working well.

- Switch Debugging Node

The commands for opening and closing log printing are the same. Each time you run a command to switch the status (open/close), you can check whether cat debug node or the log printing function is opened or closed as expected according the log information (" open XXX "or" close XXX ") generated.

> cat debug node:

```
/# cd /sys/kernel/debug/rkrga/
/# cat debug
REG [DIS]
MSG [DIS]
TIME [DIS]
INT [DIS]
CHECK [DIS]
STOP [DIS]

help:
 'echo reg > debug' to enable/disable register log printing.
 'echo msg > debug' to enable/disable message log printing.
 'echo time > debug' to enable/disable time log printing.
 'echo int > debug' to enable/disable interruppt log printing.
 'echo check > debug' to enable/disable check mode.
 'echo stop > debug' to enable/disable stop using hardware
/# echo msg > debug
/# echo ref > debug
/# cat debug
REG [DIS]
MSG [EN]
TIME [DIS]
INT [DIS]
CHECK [DIS]
STOP [DIS]

help:
 'echo reg > debug' to enable/disable register log printing.
 'echo msg > debug' to enable/disable message log printing.
 'echo time > debug' to enable/disable time log printing.
 'echo int > debug' to enable/disable interruppt log printing.
 'echo check > debug' to enable/disable check mode.
 'echo stop > debug' to enable/disable stop using hardware
/# echo msg > debug
/# cat debug
REG [DIS]
MSG [DIS]
TIME [DIS]
INT [DIS]
CHECK [DIS]
STOP [DIS]

help:
 'echo reg > debug' to enable/disable register log printing.
```

```
 'echo msg > debug' to enable/disable message log printing.
 'echo time > debug' to enable/disable time log printing.
 'echo int > debug' to enable/disable interruppt log printing.
 'echo check > debug' to enable/disable check mode.
 'echo stop > debug' to enable/disable stop using hardware
```

> log printing:

```
/# echo reg > /sys/kerne/debug/rkrga/debug
/# dmesg -c                     //For logs opened through nodes, the printing
level is KERNEL_DEBUG. You need to run the dmesg command to view the
corresponding logs on the serial port or adb.
[ 4802.344683] rga2: open rga2 reg!
/# echo reg > /sys/kernel/debug/rga2_debug/rga2
/# dmesg -c
[ 5096.412419] rga2: close rga2 reg!
```

3.2.2 Log Description

For RGA problem debugging, logs are needed to confirm work of RGA hardware. When HAL parameters are passed into the driver, the following logs describe the corresponding parameters. We usually use msg, reg or time mode for debugging.

- msg mode

```
rga2: open rga2 test MSG!                                  //Open msg log printing.
rga2: cmd is RGA2_GET_VERSION                              //Get version number,
which queries hardware version the first time each process calls librga.
rga2: cmd is RGA_BLIT_SYNC                                 //Current working mode.
rga2: render_mode:bitblt,bitblit_mode=0,rotate_mode:0   //Render_mode: display
calling interface, bitblit_mode: current blending mode（0: two-channel mode A+B-
>B, 1: three-channel mode A+B->C, rotate_mode: rotation angle.
rga2: src : y=0 uv=b4000072cc8bc040 v=b4000072cc99d040 aw=1280 ah=720 vw=1280
vh=720 xoff=0 yoff=0. format=RGBA8888 //Parameters of src channel of image data
y:fd, uv:virtual address, v: vw * vh + uv, aw and ah: actual width and height, the
actual area of image. operation, vw、vh:virtual width and height, the size of
image itself, xoff、yoff: offset in the x and y directions, format: image format.
rga2: dst : y=0 uv=b4000072cc535040 v=b4000072cc616040 aw=1280 ah=720 vw=1280
vh=720 xoff=0 yoff=0 format=RGBA8888  //Parameters of dst channel of image data.
rga2: mmu : src=01 src1=00 dst=01 els=00                   //MMU enabled flag, 0 for
close, 1 for open.
rga2: alpha : flag 0 mode0=0 mode1=0                       //Configuration of
blending.
rga2: blend mode is no blend                               //Blend mode.
rga2: yuv2rgb mode is 0                                    //Csc mode.
rga2: *** rga2_blit_sync proc ***
```

- reg mode

```
rga2: open rga2 reg!                                       //Open reg log printing.
```

```
rga2: CMD_REG                                          //Configuration of
command register.
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                                          //Configuration of full
csc register.
rga2: 00000000 00000000 00000000 00000000
rga2: 00000000 00000000 00000000 00000000
rga2: 00000000 00000000 00000000 00000000
rga2: CMD_READ_BACK_REG                                //Read back of full
command register.
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                                //Read back of full csc
register.
rga2: 00000000 00000000 00000000 00000000
rga2: 00000000 00000000 00000000 00000000
rga2: 00000000 00000000 00000000 00000000
```

- time mode

```
rga2: open rga2 test time!                             //Time Log printing is
enabled.
rga2: sync one cmd end time 2414                     //Print the RGA. hardware
time of the work,in us.
```

4. Q & A

This section introduces common questions about RGA in the form of Q&A. If the questions are not included in this section, please submit related logs and preliminary analysis information to RGA maintenance engineers.

4.1 Performance Consulting

Q1.1：How to evaluate RGA efficiency?

A1.1： When RGA performs copying, the following formula can be used to calculate the theoretical time (this function only supports data copy evaluation):

Time required for a single image copy = width × height / number of pixels that RGA can process per second

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

For example, the theoretical consuming time of copying an image of 1920 × 1080 size with RGA (frequency set at 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 consuming time depends on the type of memory used. The efficiency of different memory types from high to low is physical address > dma_fd > virtual address.

When the system is in no load, the actual time consuming of physical address is about 1.1-1.2 times of the theoretical time consuming, the actual time consuming of dma_fd is about 1.3-1.5 times of the theoretical time consuming, and the actual time consuming of virtual address is about 1.8-2.1 times of the theoretical time consuming, and is greatly affected by CPU. In general, we recommend developers to use dma_fd as the memory type passed in, which achieves great balance between accessibility and efficiency. Virtual addresses are only used as a simple and easy-to-use memory type when learning about RGA. The following table shows the actual test data of different RGA frequencies when the system is in no load on RK3566.

Test Environment：

Chip Platform	**RK3566**
System Platform	Android 11
RGA Frequency	300 M
CPU Frequency	1.8 Ghz
GPU Frequency	800 M
DDR Frequency	1056 M

Test Data：

Resolution	Memory type	Theoretical Time（us）	Actual Time（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 copying, so how to evaluate other modes?

A1.2： Currently only the formula of copying is available for evaluating use. Other modes, such as scaling and cropping, can be evaluated by taking two images of larger resolution into the copy formula to calculate the time consumption, which usually fluctuates up and down according to the size of scaling and cropping. The time consumption of the mode with no change in resolution such as blending is about 1.1-1.2 times that of the copy mode. Because of the DDR bandwidth effect in actual scenarios, it is recommended that the actual test data in the target scenario prevail in the actual evaluation.

Q1.3： Why does RGA perform poorly in certain scenarios, taking up to twice as long as running a demo?

A1.3： The bus priority of RGA in the current RK platform is the lowest. When bandwidth resources are tight, for example, in the ISP running multiplex scenario, RGA cannot read and write data in DDR timely due to bandwidth resource shortage, resulting in a large delay and performance deterioration of RGA.

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 chips such as 3399 and 1126 can be up to 400M. The RGA frequency can be improved in the following two ways:

- Set by command (temporarily modified, frequency restored upon device restart)

> Query RGA Frequency

```
cat /sys/kernel/debug/clk/clk_summary | grep rga                //Query rga
frequency.
```

> Modify RGA Frequency

```
echo 400000000 > /sys/kernel/debug/clk/aclk_rga/clk_rate        //Modify
400000000 to the needed frequency.
```

- Modify dts to modify RGA frequency（The frequency is still set after restart）

> The following example shows how to change RGA frequency in dts of RK3288. Users can change RGA frequency in the corresponding dts of other platforms.

```
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 the RGA support querying the current RGA hardware utilization (load) through commands or interfaces?

A1.5： It is not supported in the current version of driver. RGA3 version of driver will support this feature.

Q1.6： Why are RGA calls in asynchronous mode slower than those in synchronous mode in some scenarios?

A1.6： Because the current identifier of the asynchronous mode of librga is open device node, while the singleton mode of librga only opens one fd per process, imsync() will return after all the asynchronous modes of the process have been run.

Q1.7： The time cousuming when using virtual address to call RGA for copying is higher than memcpy, is there a way to optimize?

A1.7： In general, we do not recommend using virtual addresses to call RGA, because the efficiency of calling RGA is greatly reduced in scenarios with high CPU load. This is because the conversion of virtual addresses to physical address page tables is calculated by CPU in the RGA driver, and the conversion of virtual addresses to physical address page tables itself is time-consuming. Therefore, it is generally recommended to call librga using a physical address or dma_fd.

Q1.8： When carrying 8G DDR, why is RGA efficiency worse than 4G?

A1.8： Since the current RGA1/RGA2 MMU only supports a maximum of 32 bits of physical address, therefore, with devices equipped with DDR of 4G or more, when a buffer with memory greater than 4G is passed to RGA, the RGA driver copies the data from the memory with the highest address to the memory reserved by swiotlb through the DMA interface and returns the corresponding address for RGA to read and write. After the work is finished, the result is copied to the previous high target address through dma, so the CPU involvement was increased, leading to a serious increase in the working time of the librga. If only RGA2/RGA1 is configured and the DDR of the device is greater than 4 GB, you are advised to use less than 4 GB memory when calling RGA to ensure RGA efficiency.

4.2 Functions Consulting

Q2.1： How do I know what version of RGA is available on my current chip platform and what functions are available?

A2.1： See docs/RGA_API_Instruction_EN - Overview for RGA version and support information.

Different systems have different source code paths. librga source code directory paths in different SDKS are as follows:

Android 7.0 and above SDK：

hardware/rockchip/librga

Android 7.0 and below SDK：

hardware/rk29/librga

Linux SDK：

external/linux-rga

Q2.2： How to call RGA for hardware acceleration? Can there be a demo for reference?

A2.2： 1). For API call interface, see docs/RGA_API_Instruction_EN - API.

2). Demo is located in sample/rga_im2d_demo. The demo internally implements most RGA API and implements corresponding RGA functions through command. It can also be used as a tool to test whether RGA works properly in some scenarios. It is recommended that developers who are learning about RGA for the first time run the demo and get the results to understand the actual functions of RGA, modify parameters in the demo to implement corresponding functions according to their own needs, and finally try to call RGA API in their own projects.

Q2.3： Support information of RGA?

Q2.3.1： What format is supported by RGA?

A2.3.1： For detailed support information, see RGA_API_Instruction_EN - Overview - Image Format Supported to check the format support information of RGA for the corresponding chip version. Users can also call **querystring(RGA_INPUT_FORMAT | RGA_OUTPUT_FORMAT);** to query the supported input and output formats of current hardware.

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

A2.3.2： For detailed support information, see RGA_API_Instruction_EN - Overview - Design Index to query scaling ratio supported by RGA for the corresponding chip version. Users can also call **querystring(RGA_SCALE_LIMIT);** to query the scaling ratio supported by current hardware.

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

A2.3.3： For detailed support information, see RGA_API_Instruction_EN - Overview - Design Index to query the max input and output resolution supported by RGA for the corresponding chip version. Users can also call **querystring(RGA_MAX_INPUT | RGA_MAX_OUTPUT);** to query the max input/output resolution supported by current hardware.

A2.3： In general, if you have any questions about RGA support, please refer to RGA_API_Instruction_EN, which provides detailed instructions on RGA support information.

Q2.4： How does the new version of librga differ from the old one and how to tell?

A2.4： There are currently two versions of librga on RK platform:

The support and maintenance of the old version librga has been stopped. The main feature is that the SDK released before November 2020 is loaded with the old version librga. Some chip platforms, such as RK3399 Linux SDK released before June 2021 (V2.5 and below), are also with the old version librga. This version of librga cannot perfectly fit newer drivers and may have color deviation, abnormal format and other problems, so it is not recommended to use it together. When using a newer kernel, users are recommended to update the new version librga, and when using a newer version librga , kernel should be updated to match.

The new version librga is currently the main version of support and maintenance, the main feature is to add **im2d_api** directory under source directory. This version integrates with the old version librga, and introduces a simple and easy to use IM2D API, also called IM2D librga. The new version librga supports not only the new IM2D API, but also RockchipRga and C_XXX interfaces of older version. For details about API, see RGA_API_Instruction_EN. This version adds the software management version number, which can be queried through **querystring(RGA_VERSION);**.

Generally, for librga support information of old and new version, it is recommended to update the overall SDK to avoid dependency problems. It is strongly not recommended to use the new version librga with old driver or the old version librga with new kernel. It may cause obvious errors in certain scenario.

Q2.5： Does the RGA have alignment requirements?

A2.5： Different formats have different alignment requirements， the RGA hardware itself fetches the data of each line of the image in a word aligned manner， that is 4 bytes/32 bits. For example, the RGBA format itself has a single pixel storage size of 32 bit (4 × 8bit), so there is no alignment requirement. RGB565 format storage size is 16 bit (5bit + 6bit + 5bit), so it needs 2 alignment; RGB888 format storage size is 24 bit (3 × 8bit), so the format needs 4 alignment to meet the 32bit fetching requirement of RGA hardware; YUV format storage is relatively special, its own alignment requirement needs 2 alignment, Y channel single pixel storage size is 8bit, so YUV format needs 4 alignment to meet the 32bit fetching requirement of RGA hardware. UV channel according to 420/422 to determine the storage size of each four pixels, so the YUV format Y channel needs 4 alignment to meet the RGA hardware fetching requirements, then the YUV format needs 4 alignment; other unmentioned format alignment requirements are similar in principle. Note that the alignment in the question refers to the alignment requirements of width stride, the actual width and height of YUV format itself, offset due to the characteristics of the format itself is also required 2 alignment. See the RGA_API_Instruction_EN, "Overview" - "Image Format Alignment Instructions" for specific alignment restrictions.

Q2.6： Can RGA support drawing more than one rectangular at a time, or performing multiple operations? How does RGA work?

A2.6： RGA can only work sequentially on hardware, that is, one configured task ends and the next configured begins. Therefore, instead of drawing multiple rectangular at a time, async mode can be used to configure the work of RGA to the underlying driver. RGA will store the work in a work queue managed by driver and complete them in sequence. When the upper layer needs to process the buffer, it calls **imsync()** to determine if the RGA hardware has completed its work.

**Q2.7：* Does the fill function of RGA support YUV format?

A2.7： Older versions librga do not support YUV format. Only newer versions librga with the following submission support this format. If there is no such submission please try to update 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 the same as the RGB color fill API, which fills the color by configuring RGB value of the color, except that the output can be set to YUV format.

Q2.8： Does RGA support YUYV format?

A2.8： Older version librga (librga in the SDK released before October 2020) do not support YUYV format, only newer versions librga (with ** im2d_API ** in the source directory) with the following submission support this format. If there is no such submission, please try to update 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 input and output?

A2.9： Older version librga (librga in the SDK released before October 2020) do not support this format, only newer version 1.2.2 of librga (with ** im2d_API ** in the source directory) supports grayscale input. If the librga version is lower than this, please try to update SDK to the latest version. Since the RGA hardware itself does not support grayscale format, the grayscale format used here is **RK_FORMAT_Y400**, which is represented as YUV format without UV channel. YUV with only Y channel is 256-order grayscale.

Q2.10： Why does ROP code of RK3399 run on RV1126 without corresponding results?

A2.10： Although RGA on both RK3399 and RV1126 is RGA2-ENHANCE, their sub versions are different, and ROP function has been cut out of RV1126. For detailed function support information, see RGA_API_Instruction_EN or call **querystring(RGA_FEATURE);** to query support functions.

Q2.11： What is the reason for serious color difference (too pink or too green) in RGB and YUV format conversion, while other functions of RGA are normal.

Expectations：

Actual result：

A2.11： This is usually caused by a mismatch between librga and kernel. For detailed version description, see **A2.4**. The problem usually occurs after librga available on Github is used in SDK released before November 2020. Librga on Github is of new version, which does not match the older version RGA driver. Here, some configurations about color space have been changed, which causes the obvious color deviation.

There are two solutions to this problem: one is to update the SDK or RGA driver and keep librga matching with the driver; the other is to use the librga provided with SDK if the functions only available in the new version librga are not needed.

Q2.12： How does RGA implement OSD overlay subtitle?

Expectations：

A2.12： If the output is in RGB format, **imblend()** can be used to implement this, usually select src over mode, and the src channel image is overlaid on the dst channel image. If the output is in YUV format, ** imcomposite ()** can be used to implement this, usually select dst over mode, the src1 channel image is overlaid on the src channel image, and then output to the dst channel.

The blending principle of this function is **Porter-Duff blending model**. For details, see RGA_API_Instruction_EN - API - Image Blending.

The reason RGA requires different configurations for different output formats is that RGA2 has three image channels: src, src1/pat, and dst, in which src channel supports YUV2RGB conversion, src1/pat and dst channel only supports RGB2YUV conversion. The blending inside RGA needs to be performed in RGB format. Therefore, in order to ensure that RGB images are overlaid on YUV images, src must be used as the overlaid background image YUV. Src1 is used as the overlaid RGB foreground image, and the blended RGB image is finally converted into YUV format output by dst channel.

Q2.13： Why brightness or numerical difference exists when RGA is called to implement YUV and RGB format conversion?

A2.13： The reasons can be roughly divided into two kinds:

1). When YUV and RGB interconversion configuration are the same, some pixel values will be slightly different (usually 1), which is caused by the formula accuracy difference when RGA hardware implements CSC function. The decimal accuracy of CSC formula of RGA1 and RGA2 is 8bit, and that of RGA3 is 10bit. In this case, the accuracy of some calculations will have ±1 error when the results are rounded.

2). When the CSC modes configured for RGB2YUV and YUV2RGB conversion are different, the default CSC modes of RGB2YUV and YUV2RGB in the new version librga is BT.601-limit _range. When the corresponding **color_space_mode** member variable is incorrectly configured, the different configurations of the color space will result in large changes in interconversion. In the old version librga, RGB2YUV is BT.601-full_range by default, and YUV2RGB is BT.709-limit_range by default. Due to the different color space configuration of the two kinds of conversions, there are great changes in interconversion.

Q2.14： How to configure the color space for format conversion in librga?

A2.14： Both versions of librga support configuring the color space for format conversion.

1). In the new version librga, see to the RGA_API_Instruction - API - Image Format Conversion, and focus on configuring the mode parameter.

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

Format Conversion	**Color Space**	**Parameters**
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： Why does calling RGA to perform alpha overlay have no effect?

A2.15： Check whether the alpha value of the two input images is both 0xFF. When the alpha value of the foreground image in the overlay is 0xFF, the result is that the foreground image directly overwrites the background image. The result looks like there is no effect, but in fact it is a correct result.

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

Foreground Image：（Black and white and rockchip alpha is 0x00）

Expectations：

Actual Results：

A2.16： In normal configuration mode, default color value has been pre-multiplied by the corresponding alpha value, while the color value of the original image read directly has not been pre-multiplied by alpha value, so we need to add an extra flag bit when calling imblend to indicate that the color value of the image processed does not need to be pre-multiplied by alpha value. For details of calling method, see RGA_API_Instruction_EN - API - Image Blending.

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

A2.17： Yes, please refer to RGA_API_Instruction - API - Image process, and refer to the implementation of other IM2D API to understand the use of **improcess()**.

Q2.18： When RGA is called to perform image rotation, the output image is stretched?

Expectations：

Actual Result：

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

Q2.19： RGB888 output scaling results show that the image is slanted and has black lines?

Input（1920 × 1080）：

Output（1282 × 720）：

A2.19： This problem is caused by alignment requirement, virtual width of RGB888 format needs 4 alignment, please check the configured image parameters. For alignment requirement, see **Q2.5**.

Q2.20： What cause the error that in some system processes, the output of RGA is fuzzy?

A2.20： Usually RGA exception does not cause the phenomenon of fuzzy screen, when this problem occurs, users need to figure out whether the problem is RGA problem. In some system processes, users need to confirm whether the RGA input data is abnormal, you can call **fwrite()** to write memory data to file. before calling RGA, and check whether the source data is normal. If you're not familiar with how to write files, see the implementation of the **output_buf_data_to_file()** function in the **core/ rgautils.cpp** directory.

4.3 HAL Error

4.3.1 IM2D_API Error

Q3.1.1： How to deal with the error of imcheck()?

```
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： The imcheck() API serves as the verification API to call librga, which determines whether the parameters of the data structure to be passed to librga are correct, whether the function is supported, whether the hardware restrictions are triggered, etc. You can pass the error value of imcheck() as an argument to **IMStrError()** and the string returned is a detailed error message. which can be used to confirm which conditions were triggered or which parameters were wrong.

The error in this problem is caused by the alignment limitation of YUV format. Here, the width 1281 of the image is not 2 aligned, so the verification fails.

4.3.2 RockchipRga API Error

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

A3.2.1： The error is caused by the fact that called API finds that librga module has not been initialized and returns an error. In the current version, the error is usually caused by some older code still uses RgaInit/RgaDeInit/c_RkRgaInit/c_RkRgaDeInit interface to manage the initialization of RGA module, and when the singleton mode used by the current version of API is abnormal DeInit, the error will occur. Users just need to remove the Init/DeInit related calls in the code.

Q3.2.2： What causes the error “RgaBlit(1027) RGA_BLIT fail: Not a typewriter”?

A3.2.2： This error is usually caused by parameter errors. You are advised to check the scaling factor, whether virtual width is smaller than the sum of actual width and the offset in the corresponding direction, and whether the alignment meets requirements. It is recommended that new developed projects use IM2D API, which has a more comprehensive error detection mechanism, and is convenient for developers.

Q3.2.3： What causes the error “RgaBlit(1349) RGA_BLIT fail: Bad file descriptor”?

A3.2.3： This error is an ioctl error, indicating that the current fd passed to device node is invalid. Please try to update librga or confirm whether the RGA initialization process has been modified.

Q3.2.4： What causes the error “RgaBlit(1360) RGA_BLIT fail: Bad address”?

A3.2.4： The error is usually caused by a problem with the memory address of the src/src1/dst channel passed into the kernel (commonly out-of-bouns).See "Log Obtaining and Description" - "Driver Debug Node" in this document to open driver logging and locate the faulty memory.

Q3.2.5： What cause the log error “err ws[100,1280,1280]”、”Error srcRect“?

A3.2.5： The error is an obvious parameter error. “err ws” represents width stride parameter error. The parameters in the following "[]" are [X_offeset, width, width_stride] respectively.Here, because the sum of offset in X direction and width of the actual operation area is larger than the width stride, librga thinks there is a problem with the width stride and returns an error. Change the width stride to 1380 or width to 1180.

After this error occurs, the following parameters are printed in logcat:

```
E librga  : err ws[100,1280,1280]
//Represent the width stride error.
E librga  : [RgaBlit,731]Error srcRect
//Represent the src channel error.
E rockchiprga: fd-vir-phy-hnd-format[0, 0xb400006eb6ea9040, 0x0, 0x0, 0]
//Input address (fd, virtual address, physical address, handle) of  corresponding
src channel.
E rockchiprga: rect[100, 0, 1280, 720, 1280, 720, 1, 0]
//Image parameters of corresponding src channel are: X direction offset, Y
direction offset, width of the actual operation area, height of the actual
operation area, image width (virtual width), image height (virtual height), image
format, size ( parameters currently not used) respectively.
E rockchiprga: f-blend-size-rotation-col-log-mmu[0, 0, 0, 0, 0, 0, 1]
//Represents the mode configuration in the call.
E rockchiprga: fd-vir-phy-hnd-format[0, 0xb400006eb2ea6040, 0x0, 0x0, 0]
//Parameters of corresponding dst channel.
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
//Error information.
```

4.4 Kernel Errors

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

A4.1： This error is usually caused by the fact that the actual memory size of virtual address is smaller than the memory size needed (that is, the memory needed for the image of current channel calculated according to parameters of image) when the RGA is called using the virtual address. Just check the size of the buffer. In some scenarios where the application and the call are not performed together, users can memset the size of image before calling RGA, to confirm whether the problem is caused by insufficient memory size.

Usually by “rga2 map src0 memory failed”, the channel with memory problems can be confirmed, as shown in this case, the src channel triggered this error due to the actual application of buffer size only half the size of required for the image.

Q4.2： What causes the error ”rga2_reg_init, [868] set mmu info error“?

A4.2： This error represents a fd or virtual address conversion to physical address page table error, usually due to the size of the applied memory, the same as Q4.1.

Q4.3： Error “rga：dma_buf_get fail fd[328]“ usually refers to what exception occurs in the buffer?

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

Q4.4： What cause the error “RGA2 failed to get pte, result = -14, pageCount = 112”、”rga2_reg_init, [868] set mmu info error“? After checking accroding to **Q4.1** 、**Q4.2**, the error remains the same. In this case, the physical address allocated by DRM is used. The memset passed to RGA through virtual address mapped by MMAP is correct.

A4.4： This problem is caused by the allocator DRM itself. The DRM itself judges that the user mode obtains the physical address, the kernel mode usually does not need the virtual address, so the corresponding kmap will be released when allocating buffer. Releasing kmap will not affect the virtual address mapping and use in the user mode. However, when the virtual address in user mode of this buffer was passed into the RGA driver and the driver perform conversion query of the physical address page table, the kernel crashes because kmap of the buffer has been released, or the corresponding page table entry can not be queried, or the wrong address is accessed.

For this scenario, DRM provides an interface flag bit for users to figure out whether the user mode wants DRM to release kmap, that is, whether to pass the mapped virtual address to kernel:

```
(1) drm buffer application options add ROCKCHIP_BO_ALLOC_KMAP definition.
+   /* keep kmap for cma buffer or alloc kmap for other type memory */
+   ROCKCHIP_BO_ALLOC_KMAP  = 1 << 4,
(2) When applying for drm memory, add drm buffer option 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 is valid only when used together with
ROCKCHIP_BO_CONTIG.
    ret = drmIoctl(drm_fd, DRM_IOCTL_MODE_CREATE_DUMB, &arg);
```

And confirm whether the kernel contains the following submission, if not, please update 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： What causes the error “rga：Rga err irq! INT[701],STATS[1]”?

A4.5： This problem usually occurs when an exception occurs during RGA hardware execution. There are many reasons for the exception, such as memory out-of-bounds and abnormal configuration. If this problem occurs, you are advised to check whether the memory passed in is out of bounds.

Q4.6： What causes the error “rga: Rga sync pid 1001 wait 1 task done timeout”?

A4.6： There are many reasons for the hardware timeout error. You can rectify the fault as follows:

1). Check the overall process and ensure that no other modules or applications are locking or abnormally occupying the buffer. If the same buffer is abnormally occupied by other modules, RGA cannot read and write data properly. If the work cannot be completed within 2000ms, the driver returns with exception and report the error message.

2). Check the DDR bandwidth and utilization of current system. Because the bus priority of RGA is low, when the DDR load is full, if RGA is not completed within 2000ms, the driver returns with exception and report the error message.

3). Check whether other IP modules, such as ISP and vpu, have reported an error before the RGA timeout error occurs. If the hardware on the same bus is faulty, the RGA may fail to work properly. If the work cannot be completed within 2000ms, the driver returns with exception and report the error message.

4). Check current RGA frequency (see RGA frequency related operations in **Q1.4**). In some scenarios, the module on the same bus may lower the frequency thus affect RGA frequency. RGA frequency decrease will lead to the overall performance decline, if the work cannot be completed within 2000ms, the driver returns with exception and report the error message.

5). RGA of some chips is overclocked to a higher frequency, at which case RGA frequency rises but the voltage does not, leading to the overall performance of RGA decreases significantly and the work cannot be completed within the specified threshold. As a result,the driver returns with exception and report the error message. In this scenario, developers are advised to change the RGA frequency to proper frequency. Overclocking will affect the stability and service life of the overall chip, so this behavior is strongly not recommended.

6). If no error is found in any of the above scenarios, try to write the data in the target memory to file after an RGA timeout error is reported, and check whether part of the data is written to RGA. If some data is written to RGA, reconfirm scenarios 1 to 5. This is obviously caused by insufficient RGA performance. If no data is written to the target memory by RGA, collect corresponding log information and related experiments, and contact RGA maintenance engineers.