RTOS G2D

开发指南

版本号: 1.0

发布日期: 2021.4.10

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.7.29	AWA1639	1. 添加初版
1.0	2021.4.10	AWA1693	1. 添加输出宽度限制说明

目　录

插　图

1 概述

1.1 文档简介

介绍 Sunxi 平台 RTOS 上 G2D 驱动 hal 的一般使用方法及调试接口，为开发与调试提供参考。

1.2 适用范围

表 1-1: 适用产品列表

产品名称	内核版本	驱动文件
V833	Melis	rtos-hal/hal/source/g2d_rcq/
F133	Melis	rtos-hal/hal/source/g2d_rcq/

1.3 目标读者

G2D 驱动及应用层的开发/维护人员。

2 模块功能特性介绍

G2D 驱动主要实现图像旋转/数据格式/颜色空间转换, 以及图层合成功能 (包括包括 alpha、colorkey、rotate、mirror、rop、maskblt) 等图形加速功能。

2.1 支持的 format

G2D_FORMAT_ARGB8888/G2D_FORMAT_ARGB8888/G2D_FORMAT_ABGR8888/
G2D_FORMAT_RGBA8888/G2D_FORMAT_BGRA8888/G2D_FORMAT_XRGB8888,
G2D_FORMAT_XBGR8888/G2D_FORMAT_RGBX8888/G2D_FORMAT_BGRX8888/
G2D_FORMAT_RGB888/G2D_FORMAT_BGR888/G2D_FORMAT_RGB565,
G2D_FORMAT_BGR565/G2D_FORMAT_ARGB4444/G2D_FORMAT_ABGR4444/
G2D_FORMAT_RGBA4444/G2D_FORMAT_BGRA4444/G2D_FORMAT_ARGB1555,
G2D_FORMAT_ABGR1555/G2D_FORMAT_RGBA5551/G2D_FORMAT_BGRA5551/
G2D_FORMAT_ARGB2101010/G2D_FORMAT_ABGR2101010,
G2D_FORMAT_RGBA1010102/G2D_FORMAT_BGRA1010102

G2D_FORMAT_IYUV422_V0Y1U0Y0,
G2D_FORMAT_IYUV422_Y1V0Y0U0,
G2D_FORMAT_IYUV422_U0Y1V0Y0,
G2D_FORMAT_IYUV422_Y1U0Y0V0,
G2D_FORMAT_YUV422UVC_V1U1V0U0,
G2D_FORMAT_YUV422UVC_U1V1U0V0,
G2D_FORMAT_YUV422_PLANAR,
G2D_FORMAT_YUV420UVC_V1U1V0U0,
G2D_FORMAT_YUV420UVC_U1V1U0V0,
G2D_FORMAT_YUV420_PLANAR,
G2D_FORMAT_YUV411UVC_V1U1V0U0,
G2D_FORMAT_YUV411UVC_U1V1U0V0,
G2D_FORMAT_YUV411_PLANAR,
G2D_FORMAT_Y8,

G2D_FORMAT_YVU10_P010,
G2D_FORMAT_YVU10_P210,
G2D_FORMAT_YVU10_444,
G2D_FORMAT_YUV10_444,

2.2 图层 size

图层的 size 相关的参数有 Image size、source rect 以及 dest rect。

- Image size 指图片的 buffer 属性，可以理解为图片的原始完整的大小；
- source rect 是指图片 clip 区域的位置与尺寸（G2D 驱动支持 clip 完整的图片，也可以 clip 图片中某一块区域）；
- dest rect 则为图片 dest Image 在显示屏幕中的位置与尺寸。如果是 Stretchblt，source rect 与 dest rect 的宽高可以不一样，其他工作模式的图层，这二者应该一致。

如下图所示，左图区域为完整的图片尺寸，淡绿色矩形区域则为图片 clip 区域，即 source rect；右图橙黄色区域为 dest image，little dog 区域则为 source rect 拷贝到 dest image 区域的 dest rect。

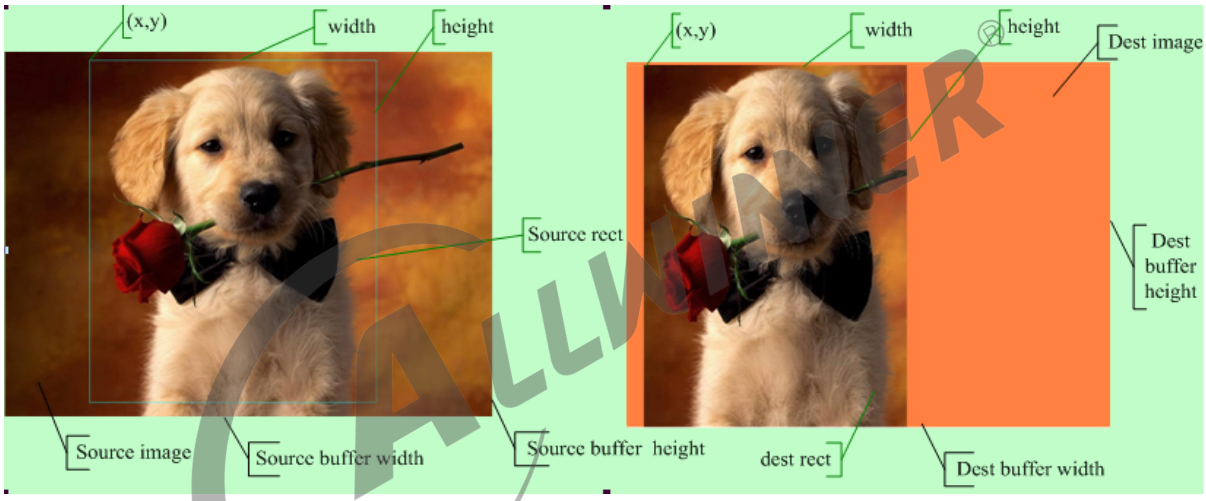

图 2-1: clip size 示意图

2.3 矩形填充 (fill color rectgngle)

填充矩形区域功能可以实现对某块区域进行预订的颜色值填充，如下图就填充了 0xFF0080FF 的 ARGB 值，该功能还可以通过设定数据区域大小实现画点和直线，同时也可以通过设定 flag 实现一种填充颜色和目标做 alpha 运算。

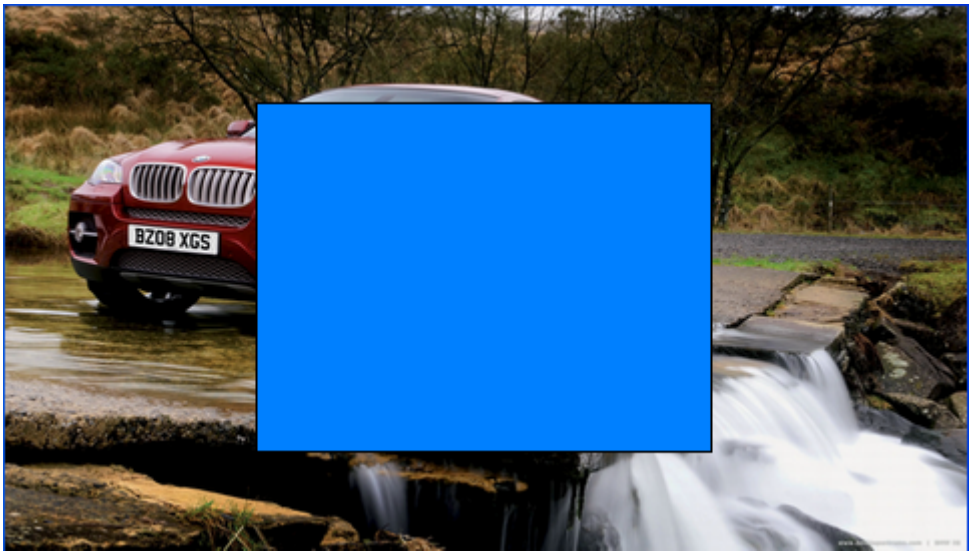

图 2-2: fill rectangle 示意图

2.4 旋转和镜像 (rotate and mirror)

旋转镜像主要是实现如下 Horizontal、Vertical、Rotate180°、Mirror45°、Rotate90°、Mirror135°、Rotate270°7 种操作。

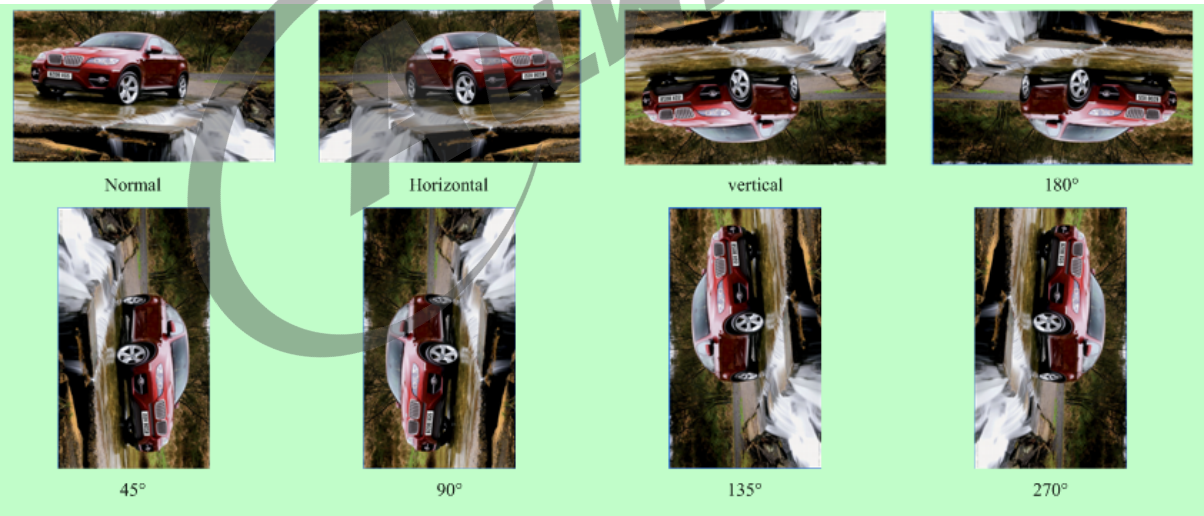

图 2-3: rotate and mirror 示意图

2.5 alpha blending

不同的图层之间可以做 alpha blending。Alpha 分为 pixel alpha、plane alpha、multi alpha 三种：

- pixel alpha 意为每个像素自带有一个专属 alpha 值。
- plane alpha 则是一个图层中所有像素共用一个 globe alpha 值。
- multi alpha 则每个像素在代入 alpha 运算时的值为 globe alpha*pixel alpha，可以通过 G2D 驱动接口的 flag 去控制。

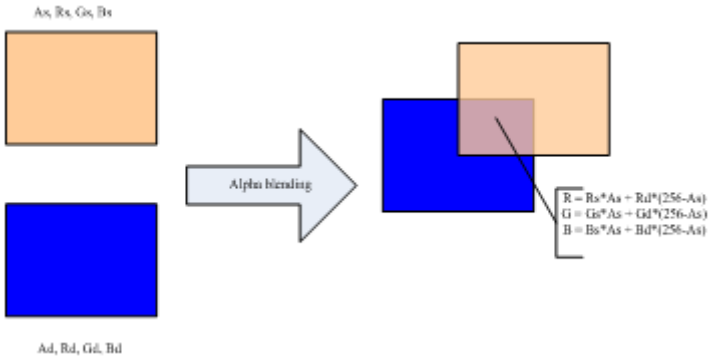

图 2-4: alpha blending 示意图

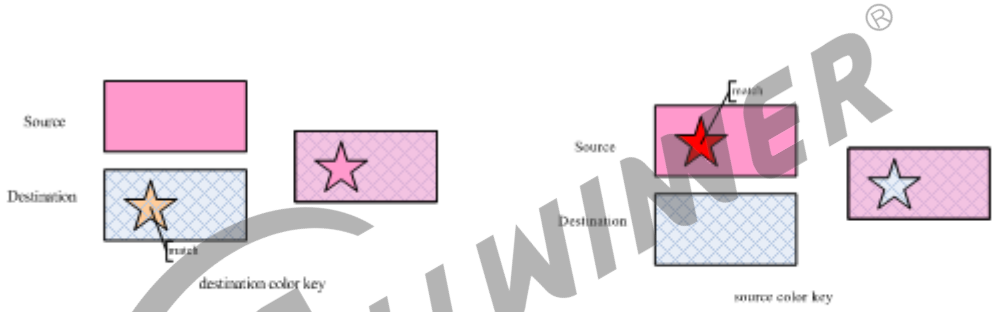

图 2-5: alpha blending 示意图

2.6 colorkey

不同 image 之间可以做 colorkey 效果，效果如下。

- 左图中 destination 的优先级高于 source，destination 中 match 部分（橙色五角星部分），则被选择透过，显示为 source 与 destination 做 alpha blending 后的效果图。
- 右图中 source 的优先级高于 destination，则 source 中 match 部分（深红色五角星部分），则被选择透过，直接显示 destination 与 source 做 alpha blending 后的效果图。

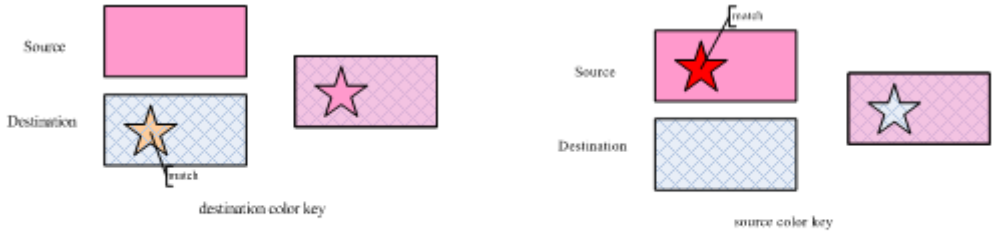

图 2-6: colorkey 示意图

2.7 缩放 (Stretchblt)

Stretchblt 主要是把 source 按照 destination 的 size 进行缩放，并最终与 destination 做 alpha blending、colorkey 等运算或直接旋转镜像后拷贝到目标，此接口在 1.0 版本上使用可以旋转和缩放一起用，但是 2.0 版本以后，缩放和旋转不可以同时操作.

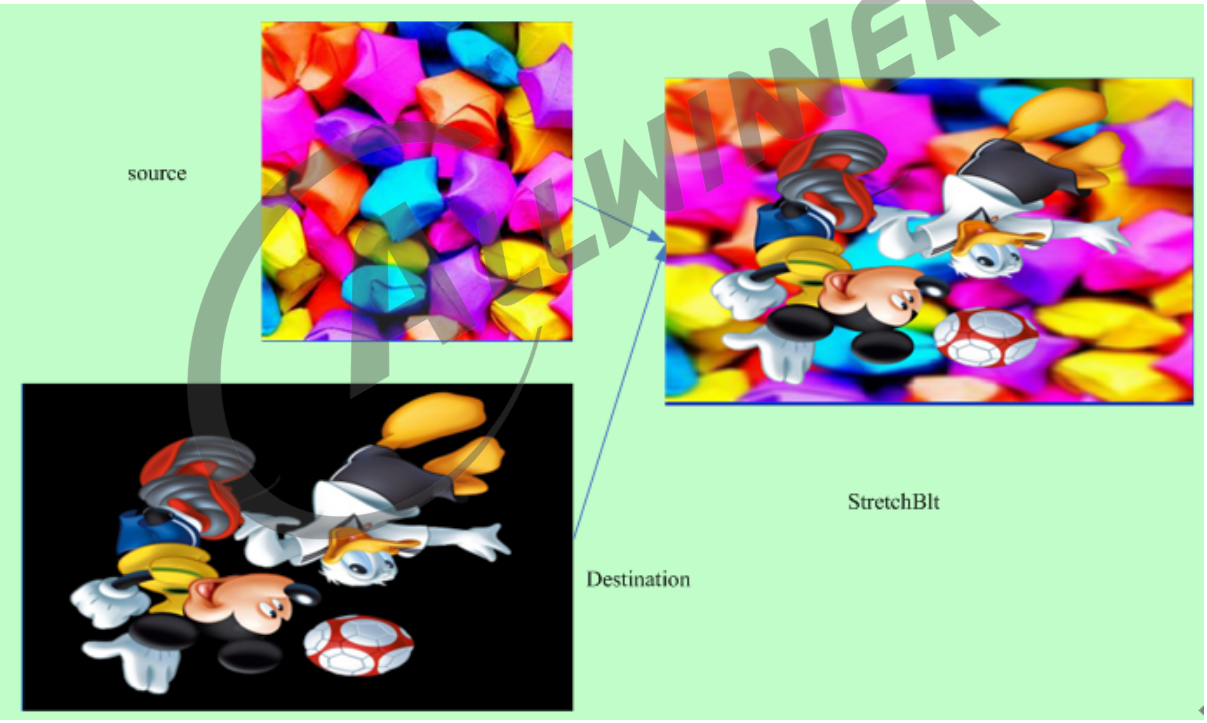

图 2-7: scale and alpha blending 示意图

2.8 二元光栅操作 (rop2)

我们在画线和填充区域的时候将画笔和目标像素组合得到新的目标像素.

2.9 三元光栅操作 (maskblt rop3)

对于图像有同样光栅操作用于生成各种特殊效果, 我们要处理的有三种像素: 源图像像素, 目标图像像素, 画刷像素 (模板图像像素). 如下图所示, 从左上到右下分别是 src ptn mask dst.

图 2-8: mask 示意图

3 模块配置介绍

G2D 模块需要配置的参数主要包含 G2D 寄存器起始地址、中断号。上述配置参数可在 rtos-hal/hal/source/g2d_rcq/g2d.c 中相应的配置，如下所示：

```
#define SUNXI_GIC_START 32
#define SUNXI_IRQ_G2D (SUNXI_GIC_START + 21)
#define SUNXI_G2D_START 0x01480000
```

由于当前尚不支持除 V833 平台以外的其他平台，故上述宏未对平台做限制，后续支持新 SOC 平台可增加相应的平台相关的宏进行配置。

4 数据结构

4.1 g2d_blt_flags

- DESCRIPTION g2d_blt_flags 用于描述一个 bitblt 和 stretchblt 的 flag 属性信息
- PROTOTYPE

```
typedef enum {
    G2D_BLT_NONE                = 0x00000000,
    G2D_BLT_PIXEL_ALPHA     = 0x00000001,
    G2D_BLT_PLANE_ALPHA     = 0x00000002,
    G2D_BLT_MULTI_ALPHA     = 0x00000004,
    G2D_BLT_SRC_COLORKEY    = 0x00000008,
    G2D_BLT_DST_COLORKEY    = 0x00000010,
    G2D_BLT_FLIP_HORIZONTAL = 0x00000020,
    G2D_BLT_FLIP_VERTICAL   = 0x00000040,
    G2D_BLT_ROTATE90            = 0x00000080,
    G2D_BLT_ROTATE180       = 0x00000100,
    G2D_BLT_ROTATE270       = 0x00000200,
    G2D_BLT_MIRROR45            = 0x00000400,
    G2D_BLT_MIRROR135       = 0x00000800,
}g2d_blt_flags;
```

- MEMBERS

```
G2D_BLT_NONE          - 纯拷贝
G2D_BLT_PIXEL_ALPHA   - 点alpha标志
G2D_BLT_PLANE_ALPHA     - 面alpha标志
G2D_BLT_MULTI_ALPHA     - 混合alpha标志
G2D_BLT_SRC_COLORKEY    - 源colorkey标志
G2D_BLT_DST_COLORKEY    - 目标colorkey标志
G2D_BLT_FLIP_HORIZONTAL - 水平翻转
G2D_BLT_FLIP_VERTICAL    - 垂直翻转
G2D_BLT_ROTATE90              - 逆时针旋转90度
G2D_BLT_ROTATE180           - 逆时针旋转180度
G2D_BLT_ROTATE270           - 逆时针旋转270度
G2D_BLT_MIRROR45              - 镜像45度
G2D_BLT_MIRROR135           - 镜像135度
```

4.2 g2d_fillrect_flags

- DESCRIPTION g2d_fillrect_flags 用于描述一个 fillrect 属性信息

- PROTOTYPE

```
typedef enum {
    G2D_FIL_NONE               = 0x00000000,
    G2D_FIL_PIXEL_ALPHA     = 0x00000001,
    G2D_FIL_PLANE_ALPHA     = 0x00000002,
    G2D_FIL_MULTI_ALPHA     = 0x00000004,
}g2d_fillrect_flags;
```

- MEMBERS

```
G2D_FIL_NONE         - 纯填充
G2D_FIL_PIXEL_ALPHA  - 填充区域和目标做点alpha
G2D_FIL_PLANE_ALPHA  - 填充区域和目标做面alpha
G2D_FIL_MULTI_ALPHA  - 填充区域的alpha值*面alpha值后再和目标做alpha
```

4.3 g2d_data_fmt(version 1.0)

- DESCRIPTION
  g2d_data_fmt 用于描述像素格式
- PROTOTYPE
  1.0 版本支持的图像格式

```
typedef enum {
  G2D_FMT_ARGB_AYUV8888 = (0x0),
    G2D_FMT_BGRA_VUYA8888   = (0x1),
    G2D_FMT_ABGR_AVUY8888   = (0x2),
    G2D_FMT_RGBA_YUVA8888   = (0x3),
    G2D_FMT_XRGB8888          = (0x4),
    G2D_FMT_BGRX8888          = (0x5),
    G2D_FMT_XBGR8888          = (0x6),
    G2D_FMT_RGBX8888          = (0x7),
    G2D_FMT_ARGB4444          = (0x8),
    G2D_FMT_ABGR4444          = (0x9),
    G2D_FMT_RGBA4444          = (0xA),
    G2D_FMT_BGRA4444          = (0xB),
    G2D_FMT_ARGB1555          = (0xC),
    G2D_FMT_ABGR1555          = (0xD),
    G2D_FMT_RGBA5551          = (0xE),
    G2D_FMT_BGRA5551          = (0xF),
    G2D_FMT_RGB565          = (0x10),
    G2D_FMT_BGR565          = (0x11),
    G2D_FMT_IYUV422         = (0x12),
    G2D_FMT_8BPP_MONO       = (0x13),
    G2D_FMT_4BPP_MONO       = (0x14),
    G2D_FMT_2BPP_MONO       = (0x15),
    G2D_FMT_1BPP_MONO       = (0x16),
    G2D_FMT_PYUV422UVC      = (0x17),
    G2D_FMT_PYUV420UVC      = (0x18),
    G2D_FMT_PYUV411UVC      = (0x19),
```

```
只有输出才有的格式:
    G2D_FMT_PYUV422         = (0x1A),
    G2D_FMT_PYUV420         = (0x1B),
    G2D_FMT_PYUV411         = (0x1C),

只有输入才支持的格式:
    G2D_FMT_8BPP_PALETTE    = (0x1D),
    G2D_FMT_4BPP_PALETTE    = (0x1E),
    G2D_FMT_2BPP_PALETTE    = (0x1F),
    G2D_FMT_1BPP_PALETTE    = (0x20),
    G2D_FMT_PYUV422UVC_MB16 = (0x21),
    G2D_FMT_PYUV420UVC_MB16 = (0x22),
    G2D_FMT_PYUV411UVC_MB16 = (0x23),
    G2D_FMT_PYUV422UVC_MB32 = (0x24),
    G2D_FMT_PYUV420UVC_MB32 = (0x25),
    G2D_FMT_PYUV411UVC_MB32 = (0x26),
    G2D_FMT_PYUV422UVC_MB64 = (0x27),
    G2D_FMT_PYUV420UVC_MB64 = (0x28),
    G2D_FMT_PYUV411UVC_MB64 = (0x29),
    G2D_FMT_PYUV422UVC_MB128= (0x2A),
    G2D_FMT_PYUV420UVC_MB128= (0x2B),
    G2D_FMT_PYUV411UVC_MB128= (0x2C),
}g2d_data_fmt;
```

- MEMBERS

```
G2D_FMT_ARGB8888        : alpha(8bit)R(8bit)G(8bit)B(8bit)
G2D_FMT_BGRA8888        : B(8bit)G(8bit)R(8bit)alpha(8bit)
G2D_FMT_ABGR8888        : alpha(8bit)B(8bit)G(8bit)R(8bit)
G2D_FMT_RGBA8888        : R(8bit)G(8bit)B(8bit)alpha(8bit)

G2D_FMT_XRGB8888        : 24bit,RGB各8bit,alpha为高位自动填充为0xFF
G2D_FMT_BGRX8888        : 24bit,BGR各8bit,alpha为低位自动填充为0xFF
G2D_FMT_XBGR8888        : 24bit,BGR各8bit,alpha为高位自动填充为0xFF
G2D_FMT_RGBX8888        : 24bit,RGB各8bit,alpha为低位自动填充为0xFF

G2D_FMT_ARGB4444        : alpha(4bit)R(4bit)G(4bit)B(4bit)
G2D_FMT_BGRA4444        : B(4bit)G(4bit)R(4bit)alpha(4bit)
G2D_FMT_ABGR4444        : alpha(4bit)B(4bit)G(4bit)R(4bit)
G2D_FMT_RGBA4444        : R(4bit)G(4bit)B(4bit)alpha(4bit)
G2D_FMT_ARGB1555        : alpha(1bit)R(5bit)G(5bit)B(5bit)
G2D_FMT_BGRA1555        : B(5bit)G(5bit)R(5bit)alpha(1bit)
G2D_FMT_ABGR1555        : alpha(1bit)B(5bit)G(5bit)R(5bit)
G2D_FMT_RGBA1555        : R(5bit)G(5bit)B(5bit)alpha(1bit)

G2D_FMT_RGB565      : R(5bit)G(6bit)B(5bit)
G2D_FMT_BGR565      : B(5bit)G(6bit)R(5bit)

G2D_FMT_IYUV422     : Interleaved YUV422

G2D_FMT_8BPP_MONO   : 8bit per pixel mono
G2D_FMT_4BPP_MONO   : 4bit per pixel mono
G2D_FMT_2BPP_MONO   : 2bit per pixel mono
G2D_FMT_1BPP_MONO   : 1bit per pixel mono

G2D_FMT_PYUV422UVC  : Planar UV combined only
G2D_FMT_PYUV420UVC  : Planar UV combined only
```

```
G2D_FMT_PYUV411UVC  : Planar UV combined only

G2D_FMT_PYUV422     : Planar YUV422
G2D_FMT_PYUV420     : Planar YUV420
G2D_FMT_PYUV411     : Planar YUV411

G2D_FMT_8BPP_PALETTE: 8bit per pixel palette only for input
G2D_FMT_4BPP_PALETTE: 4bit per pixel palette only for input
G2D_FMT_2BPP_PALETTE: 2bit per pixel palette only for input
G2D_FMT_1BPP_PALETTE: 1bit per pixel palette only for input

G2D_FMT_PYUV422UVC_MB16: 16x16 tile base planar uv combined only for input
G2D_FMT_PYUV420UVC_MB16: 16x16 tile base planar uv combined only for input
G2D_FMT_PYUV411UVC_MB16: 16x16 tile base planar uv combined only for input
G2D_FMT_PYUV422UVC_MB32: 16x16 tile base planar uv combined only for input
G2D_FMT_PYUV420UVC_MB32: 16x16 tile base planar uv combined only for input
G2D_FMT_PYUV411UVC_MB32: 16x16 tile base planar uv combined only for input
G2D_FMT_PYUV422UVC_MB64: 16x16 tile base planar uv combined only for input
G2D_FMT_PYUV420UVC_MB64: 16x16 tile base planar uv combined only for input
G2D_FMT_PYUV411UVC_MB64: 16x16 tile base planar uv combined only for input
G2D_FMT_PYUV422UVC_MB128: 16x16 tile base planar uv combined only for input
G2D_FMT_PYUV420UVC_MB128: 16x16 tile base planar uv combined only for input
G2D_FMT_PYUV411UVC_MB128: 16x16 tile base planar uv combined only for input
```

4.4 g2d_pixel_seq(version 1.0)

- DESCRIPTION g2d_pixel_seq 用于描述像素序列
- PROTOTYPE

```
typedef enum {
    G2D_SEQ_NORMAL                  = 0x0,
    G2D_SEQ_VYUY                    = 0x1,
    G2D_SEQ_YVYU                    = 0x2,
    G2D_SEQ_VUVU                    = 0x3,
    G2D_SEQ_P10                     = 0x4,
    G2D_SEQ_P01                     = 0x5,
    G2D_SEQ_P3210                   = 0x6,
    G2D_SEQ_P0123                   = 0x7,
    G2D_SEQ_P76543210           = 0x8,
    G2D_SEQ_P67452301           = 0x9,
    G2D_SEQ_P10325476           = 0xA,
    G2D_SEQ_P01234567           = 0xB,
    G2D_SEQ_2BPP_BIG_BIG        = 0xC,
    G2D_SEQ_2BPP_BIG_LITTER      = 0xD,
    G2D_SEQ_2BPP_LITTER_BIG        = 0xE,
    G2D_SEQ_2BPP_LITTER_LITTER   = 0xF,
    G2D_SEQ_1BPP_BIG_BIG             = 0x10,
    G2D_SEQ_1BPP_BIG_LITTER        = 0x11,
    G2D_SEQ_1BPP_LITTER_BIG        = 0x12,
    G2D_SEQ_1BPP_LITTER_LITTER   = 0x13,

  } g2d_pixel_seq;
```

- MEMBERS

```
G2D_SEQ_NORMAL          : Normal sequence

//for interleaved yuv422
G2D_SEQ_VYUY                : pixel 0在低16位
G2D_SEQ_YVYU                : pixel 1在低16位

// for uv_combined yuv420
G2D_SEQ_VUVU                : Planar VU combined only

// for 16bpp rgb
G2D_SEQ_P10             : pixel 0在低16位
G2D_SEQ_P01             : pixel 1在低16位

// planar format or 8bpp rgb
G2D_SEQ_P3210               : pixel 0在低8位
G2D_SEQ_P0123               : pixel 3在低8位

// for 4bpp rgb
G2D_SEQ_P76543210       :   7,6,5,4,3,2,1,0
G2D_SEQ_P67452301       :   6,7,4,5,2,3,0,1
G2D_SEQ_P10325476       :   1,0,3,2,5,4,7,6
G2D_SEQ_P01234567       :   0,1,2,3,4,5,6,7

// for 2bpp rgb
G2D_SEQ_2BPP_BIG_BIG        :
15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0

G2D_SEQ_2BPP_BIG_LITTER     :
12,13,14,15,8,9,10,11,4,5,6,7,0,1,2,3

G2D_SEQ_2BPP_LITTER_BIG     :
3,2,1,0,7,6,5,4,11,10,9,8,15,14,13,12

G2D_SEQ_2BPP_LITTER_LITTER  :
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

// for 1bpp rgb
G2D_SEQ_1BPP_BIG_BIG        :
31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0

G2D_SEQ_1BPP_BIG_LITTER     :
24,25,26,27,28,29,30,31,16,17,18,19,20,21,22,23,8,9,10,11,12,13,14,15,0,1,2,3,4,5,6,7

G2D_SEQ_1BPP_LITTER_BIG     :
7,6,5,4,3,2,1,0,15,14,13,12,11,10,9,8,23,22,21,20,19,18,17,16,31,30,29,28,27,26,25,24

G2D_SEQ_1BPP_LITTER_LITTER      :
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
```

4.5 g2d_blt_flags_h

- DESCRIPTION
  g2d_blt_flags_h 定义二元光栅操作码
- PROTOTYPE

```
typedef enum {
    G2D_BLT_NONE_0 = 0x0,
    G2D_BLT_BLACKNESS,
    G2D_BLT_NOTMERGEPEN,
    G2D_BLT_MASKNOTPEN,
    G2D_BLT_NOTCOPYPEN,
    G2D_BLT_MASKPENNOT,
    G2D_BLT_NOT,
    G2D_BLT_XORPEN,
    G2D_BLT_NOTMASKPEN,
    G2D_BLT_MASKPEN,
    G2D_BLT_NOTXORPEN,
    G2D_BLT_NOP,
    G2D_BLT_MERGENOTPEN,
    G2D_BLT_COPYPEN,
    G2D_BLT_MERGEPENNOT,
    G2D_BLT_MERGEPEN,
    G2D_BLT_WHITENESS = 0x000000ff,

    G2D_ROT_90   =    0x00000100,
    G2D_ROT_180  =    0x00000200,
    G2D_ROT_270  =    0x00000300,
    G2D_ROT_H    =    0x00001000,
    G2D_ROT_V    =    0x00002000,

//G2D_SM_TDLR_1  =    0x10000000,
    G2D_SM_DTLR_1  =    0x10000000,
//G2D_SM_TDRL_1  =    0x20000000,
//G2D_SM_DTRL_1  =    0x30000000,
} g2d_blt_flags_h;
```

- MEMBERS

```
MEMBER  DESCRIPTION
G2D_BLT_NONE    单个源操作
G2D_BLT_BLACK   BLACKNESS                :使用与物理调色板的索引0相关的色彩来填充目标矩形区域,(对缺省
    的物理调色板,该颜色为黑色)
G2D_BLT_NOTMERGEPEN dst = ~(dst+src)  :
G2D_BLT_MASKNOTPEN  dst =~src&dst
G2D_BLT_NOTCOPYPEN  dst =~src
G2D_BLT_MASKPENNOT  dst =src&~dst
G2D_BLT_NOT dst =~dst                    :使目标矩形区域颜色取反
G2D_BLT_XORPEN  dst =src^dst
G2D_BLT_NOTMASKPEN  dst =~(src&dst)
G2D_BLT_MASKPEN dst =src&dst
G2D_BLT_NOTXORPEN   dst =~(src^dst)
G2D_BLT_NOP dst =dst
G2D_BLT_MERGENOTPEN dst =~src+dst
G2D_BLT_COPEPEN dst =src
G2D_BLT_MERGEPENNOT dst =src+~dst
G2D_BLT_MERGEPEN    dst =src+dst
G2D_BLT_WHITE   WHITENESS                :使用与物理调色板中索引1有关的颜色填充目标矩形区域(对于缺省物
    理调色板来说,这个颜色为白色)
```

4.6 g2d_image(version 1.0)

- DESCRIPTION
  g2d_image 用于描述 image 属性信息
- PROTOTYPE

```
typedef struct {
  __u32          addr[3];
  __u32          w;
  __u32          h;
  g2d_data_fmt   format;
  g2d_pixel_seq  pixel_seq;
}g2d_image;
```

- MEMBERS

```
addr[3]:    图像帧的基地址，对于UV combined，addr[0,1]有效，planar类型addr[0,1，2]有效，其他addr[0]
    有效
w:          图像帧的宽
h:          图像帧的高
format:     图像帧buffer的像素格式，详见g2d_data_fmt
pixel_seq:  图像帧buffer的像素序列，详见g2d_pixel_seq
```

4.7 g2d_image_enh

- DESCRIPTION
  g2d_image_enh 主要描述图片的宽高、存放地址、是否做 Clip 处理，是否为预乘等。
- PROTOTYPE

```
typedef struct {
  int           bbuff;
    __u32       color;
    g2d_fmt_enh format;
    __u32           laddr[3];
    __u32           haddr[3];
    __u32           width;
    __u32           height;
    __u32       align[3];
    g2d_rect    clip_rect;
    __u32         gamut;
    int             bpremul;
    __u8            alpha;
    g2d_alpha_mode_enh      mode;
    int fd;
    u32 use_phy_addr;
} g2d_image_enh;
```

- MEMBERS

```
MEMBER  DESCRIPTION
format:          图格式
laddr   Buffer: 起始低位地址
haddr   Buffer: 起始高位地址
width          : 图宽度（in pixel）
height       :   图高度（in pixel）
pitch        : Buffer的pitch
clip_rect    :   ROI矩形
gamut        : 图的色域
bpremul      : 是否为预乘
alpha        : 面alpha值
mode         : alpha模式设置
use_phy_addr: 是否使用物理地址的标志。1表示使用，0表示使用fd
```

注意: 当 use_phy_addr 为 1 的时候，你必须自己设置好 laddr 和 haddr，并自行做好偏移。使用 fd 则不需要设置这两个数组。

4.8 g2d_fmt_enh

- DESCRIPTION
  g2d_fmt_enh 用于描述 G2D 模块支持的格式
- PROTOTYPE

```
typedef enum{
    G2D_FORMAT_ARGB8888,
    G2D_FORMAT_ABGR8888,
    G2D_FORMAT_RGBA8888,
    G2D_FORMAT_BGRA8888,
    G2D_FORMAT_XRGB8888,
    G2D_FORMAT_XBGR8888,
    G2D_FORMAT_RGBX8888,
    G2D_FORMAT_BGRX8888,
    G2D_FORMAT_RGB888,
    G2D_FORMAT_BGR888,
    G2D_FORMAT_RGB565,
    G2D_FORMAT_BGR565,
    G2D_FORMAT_ARGB4444,
    G2D_FORMAT_ABGR4444,
    G2D_FORMAT_RGBA4444,
    G2D_FORMAT_BGRA4444,
    G2D_FORMAT_ARGB1555,
    G2D_FORMAT_ABGR1555,
    G2D_FORMAT_RGBA5551,
    G2D_FORMAT_BGRA5551,
    G2D_FORMAT_ARGB2101010,
    G2D_FORMAT_ABGR2101010,
    G2D_FORMAT_RGBA1010102,
    G2D_FORMAT_BGRA1010102,

    /* invailed for UI channel */
    G2D_FORMAT_IYUV422_V0Y1U0Y0 = 0x20,
```

```
    G2D_FORMAT_IYUV422_Y1V0Y0U0,
    G2D_FORMAT_IYUV422_U0Y1V0Y0,
    G2D_FORMAT_IYUV422_Y1U0Y0V0,

    G2D_FORMAT_YUV422UVC_V1U1V0U0,
    G2D_FORMAT_YUV422UVC_U1V1U0V0,
    G2D_FORMAT_YUV422_PLANAR,

    G2D_FORMAT_YUV420UVC_V1U1V0U0 = 0x28,
    G2D_FORMAT_YUV420UVC_U1V1U0V0,
    G2D_FORMAT_YUV420_PLANAR,

    G2D_FORMAT_YUV411UVC_V1U1V0U0 = 0x2c,
    G2D_FORMAT_YUV411UVC_U1V1U0V0,
    G2D_FORMAT_YUV411_PLANAR,

    G2D_FORMAT_Y8 = 0x30,

    /* YUV 10bit format */
    G2D_FORMAT_YVU10_P010 = 0x34,

    G2D_FORMAT_YVU10_P210 = 0x36,

    G2D_FORMAT_YVU10_444 = 0x38,
    G2D_FORMAT_YUV10_444 = 0x39,
}g2d_fmt_enh;
```

4.9 g2d_rop3_cmd_flag

- DESCRIPTION
  g2d_rop3_cmd_flag 用于定义三元光栅操作码
- PROTOTYPE

```
typedef enum {
    G2D_ROP3_BLACKNESS   = 0x00,
    G2D_ROP3_NOTSRCERASE = 0x11,
    G2D_ROP3_NOTSRCCOPY  = 0x33,
    G2D_ROP3_SRCERASE    = 0x44,
    G2D_ROP3_DSTINVERT   = 0x55,
    G2D_ROP3_PATINVERT   = 0x5A,
    G2D_ROP3_SRCINVERT   = 0x66,
    G2D_ROP3_SRCAND      = 0x88,
    G2D_ROP3_MERGEPAINT  = 0xBB,
    G2D_ROP3_MERGECOPY   = 0xC0,
    G2D_ROP3_SRCCOPY     = 0xCC,
    G2D_ROP3_SRCPAINT    = 0xEE,
    G2D_ROP3_PATCOPY     = 0xF0,
    G2D_ROP3_PATPAINT    = 0xFB,
    G2D_ROP3_WHITENESS   = 0xFF,
}g2d_rop3_cmd_flag;
```

- MEMBERS

```
MEMBER  DESCRIPTION
G2D_ROP3_BLACKNESS    dst = BLACK
G2D_ROP3_NOTSRCERASE    dst = (NOT src) AND (NOT dst)
G2D_ROP3_NOTSRCCOPY     dst = (NOT src)         :将源矩形区域颜色取反,拷贝到目标矩形区域
G2D_ROP3_SRCERASE       dst = src AND (NOT dst )
G2D_ROP3_DSTINVERT    dst = (NOT dst)
G2D_ROP3_PATINVERT    dst = pattern XOR dst     :通过使用布尔型的异或(XOR)操作符将特定模式和目标矩形
    区域颜色合并
G2D_ROP3_SRCINVERT    dst = src XOR dst         :通过使用布尔型的异或(XOR)操作符将源和目标矩形区域颜
    色合并
G2D_ROP3_SRCAND       dst = srcAND dst          :通过使用与操作符将源和目标矩形区域颜色值合并
G2D_ROP3_MERGEPAINT   dst = (NOT src) OR dst    :通过使用布尔型的或(OR)操作符将反向的源矩形区域的颜
    色与目标矩形区域颜色合并
G2D_ROP3_MERGECOPY    dst = (src AND pattern)
G2D_ROP3_SRCCOPY        dst = src                 :将源矩形区域直接拷贝到目标矩形区域
G2D_ROP3_SRCPAINT       dst = src OR dst          :通过使用布尔型的或(OR)操作符将源和目标矩形区域颜
    色合并
G2D_ROP3_PATCOPY        dst = pattern
G2D_ROP3_PATPAINT       dst = DPSnoo              :通过使用布尔型的或(OR)操作符将源矩形区域取反后的
    颜色值与特定模式的颜色合并,然后使用OR操作符与该操作的结果与目标矩形区域内的颜色合并.
G2D_ROP3_WHITENESS    dst = WHITE
```

4.10 g2d_bld_cmd_flag

- DESCRIPTION
  g2d_bld_cmd_flag 定义 BLD 操作命令
- PROTOTYPE

```
typedef enum {
    G2D_BLD_CLEAR     = 0x00000001,
    G2D_BLD_COPY      = 0x00000002,
    G2D_BLD_DST       = 0x00000003,
    G2D_BLD_SRCOVER   = 0x00000004,
    G2D_BLD_DSTOVER   = 0x00000005,
    G2D_BLD_SRCIN     = 0x00000006,
    G2D_BLD_DSTIN     = 0x00000007,
    G2D_BLD_SRCOUT    = 0x00000008,
    G2D_BLD_DSTOUT    = 0x00000009,
    G2D_BLD_SRCATOP   = 0x0000000a,
    G2D_BLD_DSTATOP   = 0x0000000b,
    G2D_BLD_XOR       = 0x0000000c,
    G2D_CK_SRC        = 0x00010000,
    G2D_CK_DST        = 0x00020000,
}g2d_bld_cmd_flag;
```

4.11 g2d_ck

- DESCRIPTION
  g2d_ck 定义了 colorkey 操作的参数

- PROTOTYPE

```
typedef struct {
    int match_rule;
    __u32 max_color;
    __u32 min_color;
}g2d_ck;
```

- MEMBERS

```
MEMBER  DESCRIPTION
match_rule  当match_rule为假时，Color Min=<Color<=Color Max表示满足匹配条件
当match_rule为真时，Color>Color Max or Color <Color Min表示满足匹配条件
ck_max_color    Color Max
ck_min_color    Color Min
```

4.12 g2d_alpha_mode_enh

- DESCRIPTION
  g2d_alpha_mode_enh 定义进行 alpha blend 操作时，选择的 alpha mode
- PROTOTYPE

```
typedef enum{
    G2D_PIXEL_ALPHA,
    G2D_GLOBAL_ALPHA,
    G2D_MIXER_ALPHA,
}g2d_alpha_mode_enh;
```

- MEMBERS

```
MEMBER  DESCRIPTION
G2D_PIXEL_ALPHA 点alpha
G2D_GLOBAL_ALPHA    面alpha
G2D_MIXER_ALPHA 混合alpha
```

4.13 g2d_color_gmt

- DESCRIPTION
  g2d_color_gmt 定义进行位操作时，选择的颜色空间
- PROTOTYPE

```
typedef enum{
    G2D_BT601,
    G2D_BT709,
    G2D_BT2020,
}g2d_color_gmt;
```

4.14 g2d_scan_order(version 1.0)

- DESCRIPTION
  g2d_scan_order 定义进行 alpha blend 操作时，选择的图像扫行模式
- PROTOTYPE

```
enum g2d_scan_order {
    G2D_SM_TDLR = 0x00000000,
    G2D_SM_TDRL = 0x00000001,
    G2D_SM_DTLR = 0x00000002,
    G2D_SM_DTRL = 0x00000003,
};
```

- MEMBERS

```
MEMBER  DESCRIPTION
G2D_SM_TDLR Top to down, Left to right
G2D_SM_DTLR Down to top, Left to right
G2D_SM_TDRL Top to down, Right to left
G2D_SM_DTRL Down to top, Left to right
```

4.15 g2d_blt(version 1.0)

- DESCRIPTION
  g2d_blt 用于一个源和目标做 blt 的信息
- PROTOTYPE

```
typedef struct {
    g2d_blt_flags       flag;
    g2d_image           src_image;
    g2d_rect            src_rect;
    g2d_image           dst_image;
    __s32               dst_x;
    __s32               dst_y;
    __u32               color;
    __u32               alpha;
}g2d_blt;
```

- MEMBERS

```
flag        : block transfer标志，详见g2d_blt_flags
src_image   : 源图像信息，详见g2d_image
dst_image   : 目标图像信息，详见g2d_image
dst_x       : 目标矩形左上角x
dst_y       : 目标矩形左上角y
color       : colorkey颜色
alpha       : 面alpha值
```

4.16 g2d_fillrect(version 1.0)

- DESCRIPTION
  g2d_fillrect 用于描述一个 fill rectangle 参数信息
- PROTOTYPE

```
typedef struct {
    g2d_fillrect_flags   flag;
    g2d_image             dst_image;
    g2d_rect               dst_rect;
    __u32                  color;
    __u32                  alpha;
}g2d_fillrect;
```

- MEMBERS

```
flag        : 填充矩形标志，详见g2d_fillrect_flags
dst_image   : 目标图像信息，详见g2d_image
dst_rect    : 目标矩形信息，x/y/w/h-左上角x/左上角y/宽/高
color       : 填充颜色
alpha       : 面alpha值
```

4.17 g2d_stretchblt(version 1.0)

- DESCRIPTION g2d_stretchblt 用于描述一个 stretchblt 参数信息
- PROTOTYPE

```
typedef struct {
    g2d_blt_flags        flag;
    g2d_image            src_image;
    g2d_rect             src_rect;
    g2d_image            dst_image;
    g2d_rect             dst_rect;
    __u32                color;
```

```
    __u32                alpha;
} g2d_stretchblt;
```

- MEMBERS

```
flag        : block transfer标志，详见g2d_blt_flags
src_image   : 源图像信息，详见g2d_image
src_rect    : 源矩形信息，x/y/w/h-左上角x/左上角y/宽/高
dst_image   : 目标图像信息，详见g2d_image
dst_rect    : 目标矩形信息，x/y/w/h-左上角x/左上角y/宽/高
color       : colorkey颜色
alpha       : 面alpha值
```

4.18 g2d_blt_h

- DESCRIPTION
  g2d_blt_h 实现对 foreground 带缩放的 ROP2 处理。
- PROTOTYPE

```
typedef struct {
    g2d_blt_flags_h      flag_h;
    g2d_image_enh        src_image_h;
    g2d_image_enh        dst_image_h;
    __u32                   color;
    __u32                   alpha;
}g2d_blt_h;
```

- MEMBERS

```
flag_h          : blt操作flag标志，增强版标志
src_image_h   : 源图像信息,增强版的图像参数,详见g2d_image_enh
dst_image_h   : 目标图像信息，增强版的图像参数
color             : colorkey颜色
alpha             : 面alpha值
```

4.19 g2d_bld(version 1.0)

- DESCRIPTION
  g2d_bld 实现两幅图的 BLD 和 colorkey 操作。
- PROTOTYPE

```
typedef struct {
    g2d_bld_cmd_flag     bld_cmd;
    g2d_image_enh        dst_image_h;
    g2d_image_enh        src_image_h;
    g2d_ck               ck_para;
}g2d_bld;/* blending enhance */
```

- MEMBERS

```
bld_cmd     : blending的操作flag标志，增强版标志
src_image_h : 源图像信息,增强版的图像参数
dst_image_h : 目标图像信息，增强版的图像参数
ck_para     : colorkey参数
```

5 函数接口

G2D 驱动向 OS 或其他 driver hal 暴露的接口如下表，模块使用主要通过 ioctl 实现，即 sunxi_g2d_control，后续将主要介绍该接口的用法。

表 5-1: API 说明

API	解释说明
g2d_probe	初始化 g2d 驱动
sunxi_g2d_control	g2d 驱动 ioctl 接口
sunxi_g2d_close	关闭 g2d 驱动
sunxi_g2d_open	打开 g2d 驱动

5.1 1.0 版本接口

5.1.1 G2D_CMD_BITBLT

- PROTOTYPE

```
int sunxi_g2d_control(int cmd, void *arg)
```

- ARGUMENTS

```
cmd         G2D_CMD_BITBLT
arg         arg为g2d_blt结构体指针
```

- RETURNS
  成功：0，失败：失败号
- DESCRIPTION
  BITBLT 函数实现的是两个图层的运算，比如源拷贝到目标；源旋转放入目标；源和目标做 alpha blending/colorkey 后拷贝到目标
- DEMO

```
/* 输入/输出image buffer */
g2d_image image_front,scn;
g2d_rect src_rect;
g2d_blt blit;
__s32 dst_x, dst_y;

image_front.addr[0]     = mem_in;
image_front.w           = 800;
image_front.h           = 480;
image_front.format      = G2D_FMT_ARGB8888;
image_front.pixel_seq   = G2D_SEQ_NORMAL;

scn.addr[0]             = mem_out;
scn.w                   = 800;
scn.h                   = 480;
scn.format              = G2D_FMT_RGBA8888;
scn.pixel_seq           = G2D_SEQ_NORMAL;
src_rect.x              = 0;
src_rect.y              = 0;
src_rect.w              = 480;
src_rect.h              = 272;

dst_x                   = 0;
dst_y                   = 0;

/* 设置BITBLT flag标志：做点alpha和水平翻转 */
blit.flag = G2D_BLT_PIXEL_ALPHA| G2D_BLT_FLIP_HORIZONTAL;
blit.color = 0xee8899;
blit.alpha = 0x73;

/* 设置源imgae和源rect */
blit.src_image.addr[0]  = image_front.addr[0];
blit.src_image.w        = image_front.w;
blit.src_image.h        = image_front.h;
blit.src_image.format   = image_front.format;
blit.src_image.pixel_seq= image_front.pixel_seq;
blit.src_rect.x         = src_rect.x;
blit.src_rect.y         = src_rect.y;
blit.src_rect.w         = src_rect.w;
blit.src_rect.h         = src_rect.h;

/* 设置目标imgae和目标rect */
blit.dst_image.addr[0]  = scn.addr[0];
blit.dst_image.w        = scn.w;
blit.dst_image.h        = scn.h;
blit.dst_image.format   = scn.format;
blit.dst_image.pixel_seq= scn.pixel_seq;
blit.dst_x              = dst_x;
blit.dst_y              = dst_y;

if(sunxi_g2d_control(G2D_CMD_BITBLT, &blit)<0)
{
    printf("G2D_CMD_BITBLT failed!\n");
}
```

5.1.2 G2D_CMD_FILLRECT

- PROTOTYPE

```
int sunxi_g2d_control(int cmd, void *arg)
```

- ARGUMENTS

```
cmd         G2D_CMD_FILLRECT
arg         arg为g2d_fillrect结构体指针
```

- RETURNS
  成功：0，失败：失败号
- DESCRIPTION
  用一种颜色的画点画直线及矩形填充，同时也能实现填充颜色和目标做 alpha blending
- DEMO

```
/* 输出image buffer */
g2d_image scn;
g2d_rect dst_rect;
g2d_fillrect fillrect;

/* 设置FILLRECT标志：做面alpha */
fillrect.flag               = G2D_FIL_PLANE_ALPHA;
fillrect.color              = 0xFF345678;
fillrect.alpha              = 0x40;

/* 设置目标image和目标rect */
fillrect.dst_image.addr[0]  = scn.addr[0];
fillrect.dst_image.w        = scn.w;
fillrect.dst_image.h        = scn.h;
fillrect.dst_image.format   = scn.format;
fillrect.dst_image.pixel_seq= scn.pixel_seq;
fillrect.dst_rect.x         = dst_rect.x;
fillrect.dst_rect.y         = dst_rect.y;
fillrect.dst_rect.w         = dst_rect.w;
fillrect.dst_rect.h         = dst_rect.h;

if(sunxi_g2d_control(G2D_CMD_FILLRECT, &fillrect)<0)
{
        printf("G2D_CMD_FILLRECT failed!\n");
}
```

5.1.3 G2D_CMD_STRETCHBLT

- PROTOTYPE

```
int sunxi_g2d_control(int cmd, void *arg)
```

- ARGUMENTS

```
cmd         G2D_CMD_STRETCHBLT
arg         arg为g2d_stretchblt结构体指针
```

- RETURNS
  成功：0，失败：失败号
- DESCRIPTION
  STRETCHBLT 函数实现的是两个图层的运算，比如源缩放到目标大小后拷贝到目标；源缩放到目标大小旋转放入目标；源缩放到目标大小后和目标做 alpha blending/colorkey 拷贝到目标
- DEMO

```
/* 输出image buffer */
g2d_image image_front,scn;
g2d_rect src_rect,dst_rect;
g2d_stretchblt str;

image_front.addr[0]     = mem_in;
image_front.w           = 800;
image_front.h           = 480;
image_front.format      = G2D_FMT_PYUV420UVC;
image_front.pixel_seq   = G2D_SEQ_NORMAL;
image_front.addr[1]     = mem_in+ image_front.w*image_front.h;

scn.addr[0]             = mem_out;
scn.w                   = 800;
scn.h                   = 480;
scn.format              = G2D_FMT_ARGB8888;
scn.pixel_seq           = G2D_SEQ_NORMAL;
src_rect.x              = 0;
src_rect.y              = 0;
src_rect.w              = 480;
src_rect.h              = 272;
dst_rect.x              = 17;
dst_rect.y              = 100;
dst_rect.w              = 480;
dst_rect.h              = 272;

/* 设置STRETCHBLT标志:做点alpha和旋转90度 */
str.flag = G2D_BLT_PIXEL_ALPHA|G2D_BLT_ROTATE90;
str.color               = 0xee8899;
str.alpha               = 0x73;

/* 设置源image和源rect */
str.src_image.addr[0]   = image_front.addr[0];
str.src_image.addr[1]   = image_front.addr[1];
str.src_image.w         = image_front.w;
str.src_image.h         = image_front.h;
str.src_image.format    = image_front.format;
```

```
38 str.src_image.pixel_seq = image_front.pixel_seq;
39 str.src_rect.x          = src_rect.x;
40 str.src_rect.y          = src_rect.y;
41 str.src_rect.w          = src_rect.w;
42 str.src_rect.h          = src_rect.h;
43
44 /* 设置目标image和目标rect */
45 str.dst_image.addr[0]   = scn.addr[0];
46 str.dst_image.w         = scn.w;
47 str.dst_image.h         = scn.h;
48 str.dst_image.format    = scn.format;
49 str.dst_image.pixel_seq = scn.pixel_seq;
50 str.dst_rect.x          = dst_rect.x;
51 str.dst_rect.y          = dst_rect.y;
52 str.dst_rect.w          = dst_rect.w;
53 str.dst_rect.h          = dst_rect.h;
54
55 if(sunxi_g2d_control(G2D_CMD_STRETCHBLT, &str) < 0)
56 {
57     printf("G2D_CMD_STRETCHBLT failed!\n");
58 }
```

5.1.4 G2D_CMD_PALETTE_TBL

- PROTOTYPE

```
int sunxi_g2d_control(int cmd, void *arg)
```

- ARGUMENTS

```
cmd         G2D_CMD_PALETTE_TBL
arg         arg为g2d_palette结构体指针
```

- RETURNS
  成功：0，失败：失败号
- DESCRIPTION
  PALETTE_TAL 函数实现的是把查找表写入硬件 SDRAM，也只有在前面接口的源数据 format 设置为 palette 模式时才需要先使用这条命令
- DEMO

```
1 unsigned long length;
2 /* 查找表数组 */
3 unsigned long palette[0x100];
4 g2d_palette pal;
5
6 pal->pbuffer = &palette;
```

```
pal.size = length;

if(sunxi_g2d_control(G2D_CMD_PALETTE_TBL, &pal)<0)
{
    printf("G2D_CMD_PALETTE_TBL failed!\n");
}
```

5.2 2.0 版本接口

5.2.1 G2D_CMD_BITBLT_H

- PROTOTYPE

```
int sunxi_g2d_control(int cmd, void *arg)
```

- ARGUMENTS

```
cmd         G2D_CMD_BITBLT_H
arg         arg为g2d_blt_h结构体指针
```

- RETURNS
  成功：0，失败：失败号
- DESCRIPTION
  实现单幅图的缩放、格式转换等。实现对 foreground 带缩放的 ROP2 处理。
- DEMO

```
/* 旋转功能 */
blit.flag_h = G2D_ROT_90;
blit.src_image_h.addr[0] = saddr[0];
blit.src_image_h.format = G2D_FORMAT_ARGB8888;
blit.src_image_h.mode = G2D_GLOBAL_ALPHA;
blit.src_image_h.clip_rect.x = 0;
blit.src_image_h.clip_rect.y = 0;
blit.src_image_h.clip_rect.w = 1920;
blit.src_image_h.clip_rect.h = 1080;
blit.src_image_h.width = 1920;
blit.src_image_h.height = 1080;
blit.src_image_h.alpha = 0xff;
blit.dst_image_h.addr[0] = daddr[0];
blit.dst_image_h.format = G2D_FORMAT_ARGB8888;
blit.dst_image_h.mode = G2D_GLOBAL_ALPHA;
blit.dst_image_h.clip_rect.x = 0;
blit.dst_image_h.clip_rect.y = 0;
blit.dst_image_h.clip_rect.w = 1920;
blit.dst_image_h.clip_rect.h = 1080;
```

```
blit.dst_image_h.alpha = 0xff;
blit.dst_image_h.width = 1920;
blit.dst_image_h.height = 1080;

if(sunxi_g2d_control(G2D_CMD_BITBLT_H ,(unsigned long)(&blit)) < 0)
{
    printf("[%d][%s][%s]G2D_CMD_BITBLT_H failure!\n",
__LINE__, __FILE__,__FUNCTION__);
            return -1;
}

/* 缩放功能 */
blit.flag_h = G2D_BLT_NONE_0;
blit.src_image_h.addr[0] = saddr[0];
blit.src_image_h.format = G2D_FORMAT_ARGB8888;
blit.src_image_h.mode = G2D_GLOBAL_ALPHA;
blit.src_image_h.clip_rect.x = 0;
blit.src_image_h.clip_rect.y = 0;
blit.src_image_h.clip_rect.w = 1280;
blit.src_image_h.clip_rect.h = 800;
blit.src_image_h.width = 1280;
blit.src_image_h.height = 800;
blit.src_image_h.alpha = 0xff;
blit.dst_image_h.addr[0] = daddr[0];
blit.dst_image_h.format = G2D_FORMAT_ARGB8888;
blit.dst_image_h.mode = G2D_GLOBAL_ALPHA;
blit.dst_image_h.clip_rect.x = 0;
blit.dst_image_h.clip_rect.y = 0;
blit.dst_image_h.clip_rect.w = 1920;
blit.dst_image_h.clip_rect.h = 1080;
blit.dst_image_h.alpha = 0xff;
blit.dst_image_h.width = 1920;
blit.dst_image_h.height = 1080;

if(sunxi_g2d_control(G2D_CMD_BITBLT_H ,(unsigned long)(&blit)) < 0)
{
    printf("[%d][%s][%s]G2D_CMD_BITBLT_H failure!\n",
__LINE__, __FILE__,__FUNCTION__);
            return -1;
}

/* 格式转换 */
blit.flag_h = G2D_BLT_NONE_0;
blit.src_image_h.addr[0] = saddr[0];
blit.src_image_h.format = G2D_FORMAT_ARGB8888;
blit.src_image_h.mode = G2D_GLOBAL_ALPHA;
blit.src_image_h.clip_rect.x = 0;
blit.src_image_h.clip_rect.y = 0;
blit.src_image_h.clip_rect.w = 1280;
blit.src_image_h.clip_rect.h = 800;
blit.src_image_h.width = 1280;
blit.src_image_h.height = 800;
blit.src_image_h.alpha = 0xff;
blit.dst_image_h.addr[0] = daddr[0];
blit.dst_image_h.format = G2D_FORMAT_YUV420UVC_V1U1V0U0;
blit.dst_image_h.mode = G2D_GLOBAL_ALPHA;
blit.dst_image_h.clip_rect.x = 0;
blit.dst_image_h.clip_rect.y = 0;
blit.dst_image_h.clip_rect.w = 1280;
blit.dst_image_h.clip_rect.h = 800;
```

```
blit.dst_image_h.alpha = 0xff;
blit.dst_image_h.width = 1280;
blit.dst_image_h.height = 800;

if(sunxi_g2d_control(G2D_CMD_BITBLT_H ,(unsigned long)(&blit)) < 0)
{
    printf("[%d][%s][%s]G2D_CMD_BITBLT_H failure!\n",
__LINE__, __FILE__,__FUNCTION__);
            return -1;
}
```

5.2.2 G2D_CMD_BLD_H

- PROTOTYPE

```
int sunxi_g2d_control(int cmd, void *arg)
```

- ARGUMENTS

```
cmd         G2D_CMD_BLD_H
arg         arg为g2d_bld结构体指针
```

- RETURNS
  成功：0，失败：失败号
- DESCRIPTION
  实现两幅图的 BLD(porter-duff) 操作
- DEMO

```
blend.bld_cmd = G2D_BLD_COPY;
blend.src_image_h.mode = G2D_GLOBAL_ALPHA;
blend.src_image_h.format = G2D_FORMAT_ARGB8888;
blend.src_image_h.alpha = 128;
blend.src_image_h.clip_rect.x = 0;
blend.src_image_h.clip_rect.y = 0;
blend.src_image_h.clip_rect.w = 1280;
blend.src_image_h.clip_rect.h = 800;
blend.src_image_h.width = 1280;
blend.src_image_h.height = 800;
blend.dst_image_h.mode = G2D_GLOBAL_ALPHA;
blend.dst_image_h.format = G2D_FORMAT_ARGB8888;
blend.dst_image_h.alpha = 128;
blend.dst_image_h.clip_rect.x = 0;
blend.dst_image_h.clip_rect.y = 0;
blend.dst_image_h.clip_rect.w = 1280;
blend.dst_image_h.clip_rect.h = 800;
blend.dst_image_h.width = 1280;
blend.dst_image_h.height = 800;
```

```
20
21 if(sunxi_g2d_control(G2D_CMD_BLD_H ,(unsigned long)(&blend)) < 0)
22 {
23 printf("[%d][%s][%s]G2D_CMD_BLD_H failure!\n",
24         __LINE__, __FILE__,__FUNCTION__);
25         return -1;
26 }
```

5.2.3 G2D_CMD_MASK_H

- PROTOTYPE

```
int sunxi_g2d_control(int cmd, void *arg)
```

- ARGUMENTS

```
cmd          G2D_CMD_MASK_H
arg          arg为g2d_maskblt结构体指针
```

- RETURNS
  成功：0，失败：失败号
- DESCRIPTION
  根据掩膜图和光栅操作码对 src、pattern 和 dst 进行操作，并将结果保存到 dst 中.
- DEMO

```
1  mask.back_flag = G2D_ROP3_NOTSRCCOPY;
2  mask.fore_flag = G2D_ROP3_SRCINVERT;
3  mask.src_image_h.clip_rect.x = 0;
4  mask.src_image_h.clip_rect.y = 0;
5  mask.src_image_h.clip_rect.w = 1280;
6  mask.src_image_h.clip_rect.h = 800;
7  mask.src_image_h.width = 1280;
8  mask.src_image_h.height = 800;
9  mask.src_image_h.mode = G2D_GLOBAL_ALPHA;
10 mask.dst_image_h.clip_rect.x = 0;
11 mask.dst_image_h.clip_rect.y = 0;
12 mask.dst_image_h.clip_rect.w = 1280;
13 mask.dst_image_h.clip_rect.h = 800;
14 mask.dst_image_h.width = 1280;
15 mask.dst_image_h.height = 800;
16 mask.dst_image_h.mode = G2D_GLOBAL_ALPHA;
17 mask.mask_image_h.clip_rect.x = 0;
18 mask.mask_image_h.clip_rect.y = 0;
19 mask.mask_image_h.clip_rect.w = 1280;
20 mask.mask_image_h.clip_rect.h = 800;
21 mask.mask_image_h.width = 1280;
22 mask.mask_image_h.height = 800;
23 mask.mask_image_h.mode = G2D_GLOBAL_ALPHA;
```

```
mask.ptn_image_h.clip_rect.x = 0;
mask.ptn_image_h.clip_rect.y = 0;
mask.ptn_image_h.clip_rect.w = 1280;
mask.ptn_image_h.clip_rect.h = 800;
mask.ptn_image_h.width = 1280;
mask.ptn_image_h.height = 800;
mask.ptn_image_h.mode = G2D_GLOBAL_ALPHA;
mask.src_image_h.alpha = 0xff;
mask.mask_image_h.alpha = 0xff;
mask.ptn_image_h.alpha = 0xff;
mask.dst_image_h.alpha = 0xff;
mask.src_image_h.format = G2D_FORMAT_ARGB8888;
mask.mask_image_h.format = G2D_FORMAT_ARGB8888;
mask.ptn_image_h.format = G2D_FORMAT_ARGB8888;
mask.dst_image_h.format = G2D_FORMAT_ARGB8888;

if(sunxi_g2d_control(G2D_CMD_MASK_H ,(unsigned long)(&mask)) < 0)
{
printf("[%d][%s][%s]G2D_CMD_MASK_H failure!\n",__LINE__, __FILE__,__FUNCTION__);
            return -1;
}
```

5.3 批处理接口

```
struct mixer_para {
    g2d_operation_flag op_flag;
    g2d_blt_flags_h flag_h;
    g2d_rop3_cmd_flag back_flag;
    g2d_rop3_cmd_flag fore_flag;
    g2d_bld_cmd_flag     bld_cmd;
    g2d_image_enh src_image_h;
    g2d_image_enh dst_image_h;
    g2d_image_enh ptn_image_h;
    g2d_image_enh mask_image_h;
    g2d_ck ck_para;
};

typedef enum {
    OP_FILLRECT = 0x1,
    OP_BITBLT = 0x2,
    OP_BLEND = 0x4,
    OP_MASK = 0x8,
    OP_SPLIT_MEM = 0x10,
} g2d_operation_flag;
```

struct mixer_para 是 RCQ 批处理的核心结构体，可以看到除了第一个成员，其它成员的类型都是旧驱动里面有的，struct mixer_para 是之前驱动接口结构体的一个合集，如图 2 所示：

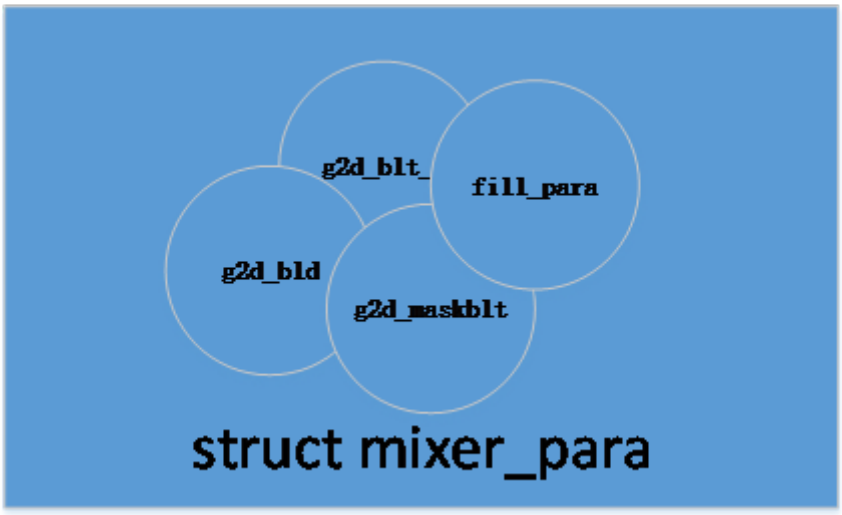

图 5-1: mixerpara

所以你可以用批处理接口完成上面其它接口的功能，只要你设置好对应的成员和 g2d_operation_flag 即可.

5.3.1 G2D_CMD_MIXER_TASK

- PROTOTYPE

```
int sunxi_g2d_control(int cmd, void *arg)
```

- ARGUMENTS

```
cmd:              G2D_CMD_MIXER_TASK

arg[0]:           设备文件标识符arg指向mixer_para指针，批处理的话就是数组指针。

arg[1]:           指针需要处理的帧的数量，大于等于1
```

- RETURN

```
成功：0，失败：失败号
```

用户要做的事情，就是填充好 mixer_para 数组，申请好输入输出内存，将要处理的图像写入到输入内存里面，将处理好的图像在输出内存里面取出来。

下面是批处理缩放 16 帧示例，其中 4 帧是 rgb 格式的缩放，6 帧是 Y8 的是缩放，6 帧是 nv12 的缩放。

```
#define RGB_IMAGE_NAME "../../pic/c1080_good.rgb"
 #define Y8_IMAGE_NAME "../../pic/en_dmabuf_bike_1280x720_220_Y8.bin"
 #define NV12_IMAGE_NAME "../../pic/bike_1280x720_220.bin"

 #define FRAME_TO_BE_PROCESS 16
 /*4 rgb convert  6 Y8 convert  6 yuv420 convert*/
 unsigned int out_width[FRAME_TO_BE_PROCESS] = {
     192,  154,  108,  321,  447,  960, 241, 320,
     1920, 1439, 1280, 1920, 2048, 720, 800, 480};
 unsigned int out_height[FRAME_TO_BE_PROCESS] = {108,  87,  70,   217, 213, 640,
                                                 840,  240, 1080, 777, 800, 1080,
                                                 2048, 480, 480,  240};

struct test_info_t
{
         struct mixer_para info[FRAME_TO_BE_PROCESS];
          .....
};

Int main()
{
....
  test_info.info[0].flag_h = G2D_BLT_NONE_H;
         test_info.info[0].op_flag = OP_BITBLT;
         test_info.info[0].src_image_h.format = G2D_FORMAT_RGB888;
         test_info.info[0].src_image_h.width = 1920;
         test_info.info[0].src_image_h.height = 1080;
         test_info.info[0].src_image_h.clip_rect.x = 0;
         test_info.info[0].src_image_h.clip_rect.y = 0;
         test_info.info[0].src_image_h.clip_rect.w = 1920;
         test_info.info[0].src_image_h.clip_rect.h = 1080;
         test_info.info[0].src_image_h.color = 0xee8899;
         test_info.info[0].src_image_h.mode = G2D_PIXEL_ALPHA;
         test_info.info[0].src_image_h.alpha = 0xaa;
         test_info.info[0].src_image_h.align[0] = 0;
         test_info.info[0].src_image_h.align[1] = 0;
         test_info.info[0].src_image_h.align[2] = 0;

         test_info.info[0].dst_image_h.format = G2D_FORMAT_RGB888;
         test_info.info[0].dst_image_h.width = 800;
         test_info.info[0].dst_image_h.height = 480;
         test_info.info[0].dst_image_h.clip_rect.x = 0;
         test_info.info[0].dst_image_h.clip_rect.y = 0;
         test_info.info[0].dst_image_h.clip_rect.w = 1920;
         test_info.info[0].dst_image_h.clip_rect.h = 1080;
         test_info.info[0].dst_image_h.color = 0xee8899;
         test_info.info[0].dst_image_h.mode = G2D_PIXEL_ALPHA;
         test_info.info[0].dst_image_h.alpha = 255;
         test_info.info[0].dst_image_h.align[0] = 0;
         test_info.info[0].dst_image_h.align[1] = 0;
         test_info.info[0].dst_image_h.align[2] = 0;
for (i = 0; i < FRAME_TO_BE_PROCESS; ++i) {
                memcpy(&test_info.info[i], &test_info.info[0],
                       sizeof(struct mixer_para));
                test_info.info[i].dst_image_h.width = out_width[i];
                test_info.info[i].dst_image_h.height = out_height[i];
                test_info.info[i].dst_image_h.clip_rect.w = out_width[i];
                test_info.info[i].dst_image_h.clip_rect.h = out_height[i];
                if (i < 4) {
```

```
61                          test_info.out_size[i] = test_info.info[i].dst_image_h.width *
    test_info.info[i].dst_image_h.height * 3;
62                          test_info.info[i].src_image_h.format = G2D_FORMAT_BGR888;
63                          test_info.info[i].src_image_h.width = 1920;
64                          test_info.info[i].src_image_h.height = 1080;
65                          test_info.info[i].src_image_h.clip_rect.w = 1920;
66                          test_info.info[i].src_image_h.clip_rect.h = 1080;
67                          test_info.in_size[i] = 1920*1080*3;
68                          snprintf(test_info.src_image_name[i], 100,"%s",RGB_IMAGE_NAME);
69                  } else if (i < 10) {
70                          test_info.out_size[i] = test_info.info[i].dst_image_h.width *
    test_info.info[i].dst_image_h.height;
71                          test_info.info[i].src_image_h.format = G2D_FORMAT_Y8;
72                          test_info.info[i].src_image_h.width = 1280;
73                          test_info.info[i].src_image_h.height = 720;
74                          test_info.info[i].src_image_h.clip_rect.w = 1280;
75                          test_info.info[i].src_image_h.clip_rect.h = 720;
76                          test_info.in_size[i] = 1280*720;
77                          snprintf(test_info.src_image_name[i], 100,"%s",Y8_IMAGE_NAME);
78                  } else {
79                          test_info.out_size[i] = test_info.info[i].dst_image_h.width *
    test_info.info[i].dst_image_h.height * 2;
80                          test_info.info[i].src_image_h.format =
    G2D_FORMAT_YUV420UVC_U1V1U0V0;
81                          test_info.info[i].src_image_h.width = 1280;
82                          test_info.info[i].src_image_h.height = 720;
83                          test_info.info[i].src_image_h.clip_rect.w = 1280;
84                          test_info.info[i].src_image_h.clip_rect.h = 720;
85                          test_info.in_size[i] = 1280*720*2;
86                          snprintf(test_info.src_image_name[i], 100,"%s",NV12_IMAGE_NAME);
87                  }
88                  ret = ion_memory_request(&test_info.dst_ion[i], 1, NULL, test_info.
    out_size[i]);
89                  test_info.info[i].dst_image_h.fd = test_info.dst_ion[i].fd_data.fd;//rtos-
    hal中的驱动不支持使用fd，这里请修改为物理地址，并设置好偏移
90
91                  test_info.info[i].dst_image_h.format = test_info.info[i].src_image_h.
    format;
92                  ret = ion_memory_request(&test_info.src_ion[i], 0, test_info.
    src_image_name[i], test_info.in_size[i]);
93                  test_info.info[i].src_image_h.fd = test_info.src_ion[i].fd_data.fd;//rtos-
    hal中的驱动不支持使用fd，这里请修改为物理地址，并设置好偏移
94          }
95  arg[0] = (unsigned long)test_info.info;
96          arg[1] = FRAME_TO_BE_PROCESS;
97          if (sunxi_g2d_control(G2D_CMD_MIXER_TASK, (arg)) < 0) {
98                  printf("[%d][%s][%s]G2D_CMD_MIXER_TASK failure!\n", __LINE__,
99                         __FILE__, __FUNCTION__);
100                 goto FREE_SRC;
101         }
102         printf("[%d][%s][%s]G2D_CMD_MIXER_TASK SUCCESSFULL!\n", __LINE__,
103                __FILE__, __FUNCTION__);
104
105
106         printf("save result data to file\n");
107         char sufix[40] = {0};
108         for (i = 0; i < FRAME_TO_BE_PROCESS; ++i) {
109                 if (i < 4) {
110                         snprintf(sufix, 40, "rgb888");
111                 } else if (i < 10)
```

```
112                         snprintf(sufix, 40, "y8");
113                 else
114                         snprintf(sufix, 40, "nv12");
115
116                 snprintf(test_info.dst_image_name[i], 100,
117                          "../../result/frame%d_%dx%d_to_%dx%d.%s",i,
118                          test_info.info[i].src_image_h.width,
119                          test_info.info[i].src_image_h.height,
120                          test_info.info[i].dst_image_h.width,
121                          test_info.info[i].dst_image_h.height, sufix);
122                 if((test_info.dst_fp[i] = fopen(test_info.dst_image_name[i], "wb+")) ==
    NULL) {
123                         printf("open file %s fail.\n", test_info.dst_image_name[i]);
124                         break;
125                 } else {
126                         ret = fwrite(test_info.dst_ion[i].virt_addr,
127                                      test_info.out_size[i], 1, test_info.dst_fp[i]);
128                         fflush(test_info.src_fp);
129                         printf("Frame %d saved\n", i);
130                 }
131
132         }
133 ....
134 }
```

5.3.2 G2D_CMD_CREATE_TASK

- PROTOTYPE

```
int sunxi_g2d_control(int cmd, void *arg)
```

- ARGUMENTS

cmd	G2D_CMD_CREATE_TASK
arg[0]	arg指向mixer_para指针，批处理的话就是数组指针。
arg[1]	需要处理的帧的数量，大于等于1

- RETURN

```
成功：task id，大于等于1，其它情况则为失败

arg[0]对应的指针所指向的mixer_para内容会被更新。
```

该 ioctl 命令用于创建新的批处理实例，但不做硬件处理，只是准备好软件。

这个过程会构造对应帧数的 rcq 队列内存以及进行输入输出图像的 dma map 和 dma umap 操作，构造完毕之后会更新 mixer_para 回应用层。task_id 是唯一的，只要不销毁批处理实例，会一直占据这个 id，根据这个 id 用户可以进一步操作，比如设置，销毁，获取当前 mixer_para。

如下例子，会创建两个不同帧数和输入输出格式的批处理实例，最终得到两个不同的 task id，task0 和 task1。mixer_para 如何构造参考 G2D_CMD_MIXER_TASK 的例子。

```
arg[0] = (unsigned long)test_info.info;
    arg[1] = FRAME_TO_BE_PROCESS;
    task0 = sunxi_g2d_control(G2D_CMD_CREATE_TASK, (arg));
    if (task0 < 1) {
        printf("[%d][%s][%s]G2D_CMD_CREATE_TASK failure!\n", __LINE__,
               __FILE__, __FUNCTION__);
        goto FREE_SRC;
    }
    printf("[%d][%s][%s]G2D_CMD_CREATE_TASK SUCCESSFULL!\n", __LINE__,
           __FILE__, __FUNCTION__);


    arg[0] = (unsigned long)test_info2.info;
    arg[1] = FRAME_TO_BE_PROCESS2;
    task1 = sunxi_g2d_control(G2D_CMD_CREATE_TASK, (arg));
    if (task1 < 1) {
        printf("[%d][%s][%s]G2D_CMD_CREATE_TASK failure!\n", __LINE__,
               __FILE__, __FUNCTION__);
        goto FREE_SRC;
    }
    printf("[%d][%s][%s]G2D_CMD_CREATE_TASK SUCCESSFULL!\n", __LINE__,
           __FILE__, __FUNCTION__);
```

5.3.3 G2D_CMD_TASK_APPLY

- PROTOTYPE

```
int sunxi_g2d_control(int cmd, void *arg)
```

- ARGUMENTS

```
cmd             G2D_CMD_TASK_APPLY

arg[0]          task id(由G2D_CMD_CREATE_TASK命令获得)

arg[1]          arg指向mixer_para指针，批处理的话就是数组指针
```

- RETURN

```
成功：0，失败：失败号
```

该 ioctl 命令的作用是执行批处理的硬件操作。

值得注意 arg[1] 中的 mixer_para，必须是 G2D_CMD_CREATE_TASK 之后返回的 mixer_para 或者是通过另外一个 ioctl 命令 G2D_CMD_TASK_GET_PARA 才行，这里不需要制定帧数的原因是前面的 G2D_CMD_CREATE_TASK 已经指定好帧数，而 G2D_CMD_TASK_APPLY 是基于 task id 来执行的。

```
arg[0] = task0;
    arg[1] = (unsigned long)test_info.info;
    if(sunxi_g2d_control(G2D_CMD_TASK_APPLY, (arg)) < 0) {
        printf("[%d][%s][%s]G2D_CMD_TASK_APPLY failure!\n", __LINE__,
               __FILE__, __FUNCTION__);
        goto FREE_SRC;
    }
    printf("[%d][%s][%s]G2D_CMD_TASK_APPLY SUCCESSFULL!\n", __LINE__,
           __FILE__, __FUNCTION__);

    arg[0] = task1;
    arg[1] = (unsigned long)test_info2.info;
    if(sunxi_g2d_control(G2D_CMD_TASK_APPLY, (arg)) < 0) {
        printf("[%d][%s][%s]G2D_CMD_TASK_APPLY failure!\n", __LINE__,
               __FILE__, __FUNCTION__);
        goto FREE_SRC;
    }
    printf("[%d][%s][%s]G2D_CMD_TASK_APPLY SUCCESSFULL!\n", __LINE__,
           __FILE__, __FUNCTION__);
```

5.3.4 G2D_CMD_TASK_DESTROY

- PROTOTYPE

```
int sunxi_g2d_control(int cmd, void *arg)
```

- ARGUMENTS

```
cmd             G2D_CMD_TASK_DESTROY

arg[0]          task id
```

- RETURN

```
成功：0，失败：失败号
```

该 ioctl 命令的作用是销毁指定 task id 的批处理实例。

```
arg[0] = task0;;
   if(sunxi_g2d_control(G2D_CMD_TASK_DESTROY, (arg)) < 0) {
       printf("[%d][%s][%s]G2D_CMD_TASK_DESTROY failure!\n", __LINE__,
              __FILE__, __FUNCTION__);
       goto FREE_SRC;
   }
   printf("[%d][%s][%s]G2D_CMD_TASK_DESTROY SUCCESSFULL!\n", __LINE__,
          __FILE__, __FUNCTION__);
   arg[0] = task1;;
   if(sunxi_g2d_control(G2D_CMD_TASK_DESTROY, (arg)) < 0) {
       printf("[%d][%s][%s]G2D_CMD_TASK_DESTROY failure!\n", __LINE__,
              __FILE__, __FUNCTION__);
       goto FREE_SRC;
   }
   printf("[%d][%s][%s]G2D_CMD_TASK_DESTROY SUCCESSFULL!\n", __LINE__,
          __FILE__, __FUNCTION__);
```

5.3.5 G2D_CMD_TASK_GET_PARA

- PROTOTYPE

```
int sunxi_g2d_control(int cmd, void *arg)
```

- ARGUMENTS

```
cmd             G2D_CMD_TASK_DESTROY

arg[0]          task id

arg[1]          指向mixer_para指针，多帧的话就是数组指针
```

- RETURN

```
成功：0，失败：失败号
```

该 ioctl 命令的作用是获取指定 task id 的 mixer para。

用户必须自行保证传入的指针所指向的内存足够存放这么多帧的参数

6 FAQ

6.1 常见问题

6.1.1 输出宽度

G2D 硬件模块不支持输出宽度等于 1 pixel。

著作权声明

商标声明

免责声明