AX FB 开发指南

文档版本：V1.1

发布日期：2024/10/31

目 录

权利声明

注意

您购买的产品、服务或特性等应受商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非商业合同另有约定，本公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

前 言

本文档主要介绍如何在本公司平台上进行 FB 相关的开发。

适用产品

AX620E 系列产品（AX630C、AX620Q）

适读人群

- 终端用户
- 售前
- 售后
- 技术人员

符号与格式定义

符号/格式	说明
`Xxx`	表示您可以执行的命令行。
☞ 说明/备注：	表示您在使用产品的过程中，我们向您说明的事项。
！ 注意：	表示您在使用产品的过程中，需要您特别注意的事项。

修订历史

文档版本	发布时间	修订说明
V1.0	2023/11/30	文档初版
V1.1	2024/10/31	增加 FB 相关格式支持的说明

1 概述

1.1 FB 简介

Framebuffer（以下简称 FB）用于管理叠加图形层的模块。它基于 Linux Framebuffer 框架实现的一个虚拟 FB，提供了 Linux Framebuffer 的基本功能，还扩展了一些控制功能，如图形层的 colorkey、鼠标层。

FB 支持以下的 Linux Framebuffer 标准功能：

- 将物理显存映射（或解除映射）到虚拟内存空间。
- 像操作普通文件一样操作物理显存。
- 设置分辨率和像素格式，每个叠加图形层的支持的最大分辨率和像素格式可以通过支持能力接口获取。
- 从物理显存的任何位置进行读、写、显示等操作。

1.2 体系结构

应用程序基于 Linux 文件系统使用 FB。FB 的体系结构如图 1-1 所示。

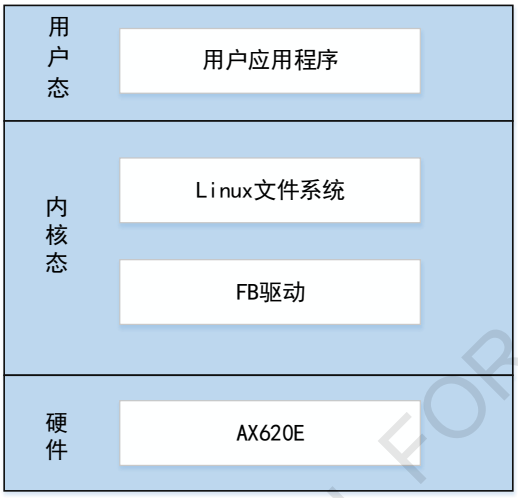

图1-1 FB 体系结构

1.3 格式支持

本平台 FB 支持的图像格式：

ARGB1555/RGBA5551，ARGB4444/RGBA4444，ARGB8565/RGBA5658，ARGB8888/RGBA8888，RGB565/BGR565，RGB888/BGR888

1.4 应用场景

FB 可应用于以下场景：

- MiniGUI 窗口系统，MiniGUI 支持 Linux Framebuffer，对 MiniGUI 做少量改动即可移植到本平台上，实现快速移植。
- 其他的基于 Linux Frambuffer 的应用程序，对基于 Linux Framebuffer 的应用程序做少量改动即可移植到本平台上，实现快速移植。

2 FB 使用

2.1 FB 使能及参数设置

FB 的使能情况、显存大小及显存 buffer 数量是通过 dts 来配置的，如图 2-1 所示。

```
vfb2: vfb@2 {
    compatible = "axera,vfb";
    id = <2>;
    width = <1920>;
    height = <1080>;
    bpp = <32>;
    buf-num = <2>;
    status = "okay";
};

vfb3: vfb@3 {
    compatible = "axera,vfb";
    id = <3>;
    width = <1920>;
    height = <1080>;
    bpp = <32>;
    buf-num = <1>;
    cursor;
    status = "okay";
};
```

图2-1 FB 的参数配置

图 2-1 中 FB 的配置，关键属性的含义说明见下表：

属性	描述	备注
compatible	必须为 axera,vfb	
id	FB 的编号	
width	支持的最大分辨率宽	

height	支持的最大分辨率高	
bpp	Bits per pixel 的最大取值	
buf-num	最大分辨率情况下的帧缓冲数量	
cursor	表示些 fb 用做鼠标层	cursor 属性没有值

FB 内存使用情况计算：width * height * (bpp / 8) * buf-num。

如上图中 vfb2 的配置：1920*1080*(32/8)*2 = 0xFD2000 bytes。运行期间如更改分辨率、颜色深度等显示属性需确保所使用的内存不能超过此配置的最大内存。

2.2 FB 支持的相关操作

FB 主要支持如下几类操作：

- 文件操作类，该类操作用于提供操作 FB 接口，通过调用这些接口可以像操作文件一样操作图形叠加层，这些接口是 Linux 本身提供的标准接口，主要有 open、close、write、read、lseek 等。本文档不对这些标准接口进行描述。
- 显存映射类，该类操作用于提供将物理显存映射到用户虚拟内存空间的接口。这些接口是 Linux 本身提供的标准接口，主要有 mmap、munmap 等。本文档不对这些标准接口进行描述。
- 显存控制和状态查询类，该类操作允许设置像素格式和颜色深度等属性的接口。这些接口是 Linux 本身提供的标准接口，经常使用。本文档将对其进行简要描述。

2.2.1 ioctl 函数

FB 的用户态接口以 ioctl 形式体现，其形式如下：

int ioctl(int fd, unsigned long cmd, ……);

该函数是 Linux 标准接口，具备可变参数特性。但在本平台中，实际只需要 3 个参数。因此，其语法形式等同于：

int ioctl (int fd, unsigned long cmd, void *cmddata);

其中，cmddata 的具体类型随参数 cmd 的变化而变化。这 3 个参数的详细描述如表 2-1 所示：

表 2-1 ioctl 函数的 3 个参数

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符，是调用 open 函数打开 Framebuffer 设备之后的返回值。	输入
cmd	主要的 cmd（命令控制字）如下： • FBIOGET_VSCREENINFO：获取屏幕可变信息 • FBIOPUT_VSCREENINFO：设置屏幕可变信息 • FBIOGET_FSCREENINFO：获取屏幕固定信息 • FBIOPAN_DISPLAY：设置 PAN 显示	输入

cmddata	各 cmd 对应的数据类型分别是： • 获取或设置屏幕可变信息：struct fb_var_screeninfo *类型 • 获取屏幕固定信息：struct fb_fix_screeninfo *类型 • 设置 PAN 显示：struct fb_var_screeninfo *类型	输入、输出

2.2.2 标准功能

FBIOGET_VSCREENINFO

【目的】

获取屏幕的可变信息。

【语法】

int ioctl (int fd, FBIOGET_VSCREENINFO, struct fb_var_screeninfo *var);

【描述】

使用此接口获取屏幕的可变信息，主要包括分辨率和像素格式。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符	输入
FBIOGET_VSCREENINFO	ioctl 号	输入
var	可变信息结构体指针	输出

【返回值】

返回值	描述
0	成功
－1	失败

【需求】

头文件：fb.h

【注意】

无

【举例】

```
struct fb_var_screeninfo vinfo;
if (ioctl(fd, FBIOGET_VSCREENINFO, &vinfo) < 0) {
    return -1;
}
```

FBIOPUT_VSCREENINFO

【目的】

设置 Framebuffer 的屏幕分辨率和像素格式等。

【语法】

int ioctl (int fd, FBIOPUT_VSCREENINFO, struct fb_var_screeninfo *var);

【描述】

使用此接口设置屏幕的分辨率和像素格式。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符	输入
FBIOPUT_VSCREENINFO	ioctl 号	输入
var	可变信息结构体指针	输入

【返回值】

返回值	描述
0	成功
–1	失败

【需求】

头文件：fb.h

【注意】

必须保证实际分辨率与偏移的和在虚拟分辨率范围内，否则系统会自动调整实际分辨率的大小让其在虚拟分辨率范围内。

【举例】

设置实际分辨率为 720x576，虚拟分辨率为 720x576，偏移为（0，0），像素格式为

RGBA8888 的示例代码如下：

```
struct fb_bitfield r32 = {24, 8, 0};
struct fb_bitfield g32 = {16, 8, 0};
struct fb_bitfield b32 = {8, 8, 0};
struct fb_bitfield a32 = {0, 8, 0};
struct fb_var_screeninfo vinfo;
if (ioctl(fd, FBIOGET_VSCREENINFO, &vinfo) < 0) {
     return -1;
}
vinfo.xres_virtual = 720;
vinfo.yres_virtual = 576;
vinfo.xres = 720;
vinfo.yres = 576;
vinfo.activate = FB_ACTIVATE_NOW;
vinfo.bits_per_pixel = 32;
vinfo.xoffset = 0;
vinfo.yoffset = 0;
vinfo.red = r32;
vinfo.green = g32;
vinfo.blue = b32;
vinfo.transp= a32;
if (ioctl(fd, FBIOPUT_VSCREENINFO, &vinfo) < 0) {
   return -1;
}
```

FBIOGET_FSCREENINFO

【目的】

获取 Framebuffer 的固定信息。

【语法】

int ioctl (int fd, FBIOGET_FSCREENINFO, struct fb_fix_screeninfo *fix);

【描述】

使用此接口获取 Framebuffer 固定信息，包括显存起始物理地址、显存大小和行间距等。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符	输入
FBIOGET_FSCREENINFO	ioctl 号	输入
fix	可变信息结构体指针	输出

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

头文件：fb.h

【注意】

无

FBIOPAN_DISPLAY

【目的】

设置从虚拟分辨率中不同的偏移处开始显示。

【语法】

int ioctl (int fd, FBIOPAN_DISPLAY, struct fb_var_screeninfo *var);

【描述】

使用此接口设置从虚拟分辨率中的不同偏移处开始显示，实际的分辨率不变。如图 2-1 所示：(xres_virtual, yres_virtual)是虚拟分辨率，(xres, yres)是实际显示的分辨率，(xoffset, yoffset)是显示的偏移。

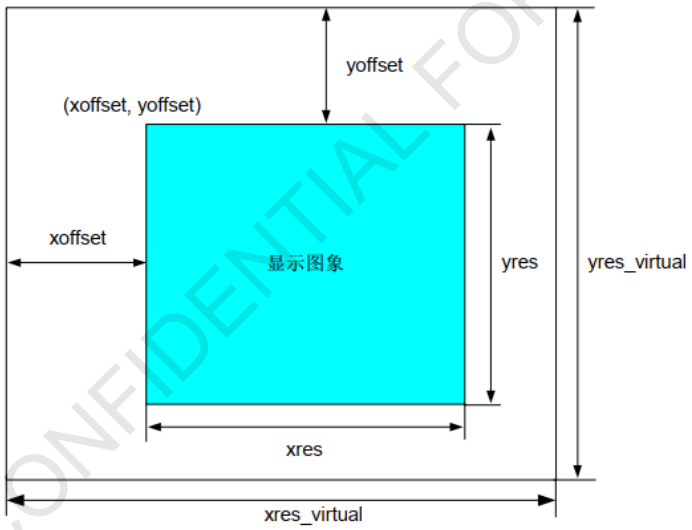

图2-2 FB 从虚拟分辨率中的不同偏移处开始显示

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符	输入
FBIOPAN_DISPLAY	ioctl 号	输入
var	可变信息结构体指针	输入

【返回值】

返回值	描述
0	成功
–1	失败

【需求】

头文件：fb.h

【注意】

必须保证实际分辨率与偏移的和在虚拟分辨率范围内，否则设置不成功。

【举例】

设置像素格式为 RGBA8888，实际分辨率为 300×300，虚拟分辨率为 720×576，起始偏移为（50，50），然后偏移到（300，0）处开始显示的 PAN 设置代码如下：

```
struct fb_bitfield r32 = {24, 8, 0};
struct fb_bitfield g32 = {16, 8, 0};
struct fb_bitfield b32 = {8, 8, 0};
struct fb_bitfield a32 = {0, 8, 0};
struct fb_var_screeninfo vinfo;
if (ioctl(fd, FBIOGET_VSCREENINFO, &vinfo) < 0) {
    return -1;
}
vinfo.xres_virtual = 720;
vinfo.yres_virtual = 576;
vinfo.xres = 300;
vinfo.yres = 300;
vinfo.activate = FB_ACTIVATE_NOW;
vinfo.bits_per_pixel = 32;
vinfo.xoffset = 50;
vinfo.yoffset = 50;
vinfo.red = r32;
vinfo.green = g32;
vinfo.blue = b32;
vinfo.transp= a32;
if (ioctl(fd, FBIOPUT_VSCREENINFO, &vinfo) < 0) {
```

```
            return -1;
        }
        vinfo.xoffset = 300;
        vinfo.yoffset = 0;
        if (ioctl(fd, FBIOPAN_DISPLAY, &vinfo) < 0) {
            return -1;

}
```

2.2.3 扩展功能

关键色功能

AX_FBIOGET_COLORKEY

【目的】

获取图形层的关键色信息。

【语法】

int ioctl (int fd, AX_FBIOGET_COLORKEY, AX_FB_COLORKEY_T *var);

【描述】

使用此接口获取图形层的关键色相关信息。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符	输入
AX_FBIOGET_COLORKEY	ioctl 号	输入
var	关键色信息结构体指针	输出

【返回值】

返回值	描述
0	成功
–1	失败

【需求】

头文件：ax_vo_api.h

【注意】

无

【举例】

```
AX_FB_COLORKEY_T colorkey
if (ioctl(fd, AX_FBIOGET_COLORKEY, &colorkey) < 0) {
    return -1;
}
```

AX_FBIOPUT_COLORKEY

【目的】

设置图形层的关键色信息。

【语法】

int ioctl (int fd, AX_FBIOPUT_COLORKEY, AX_FB_COLORKEY_T *var);

【描述】

使用此接口设置图形层的关键色相关信息。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符	输入
AX_FBIOPUT_COLORKEY	ioctl 号	输入
var	关键色信息结构体指针	输入

【返回值】

返回值	描述
0	成功
–1	失败

【需求】

头文件：ax_vo_api.h

【注意】

无

【举例】

假设要过滤掉 RGB 像素格式中红色分量为 0xc0、绿色分量为 0xc0、蓝色分量为 0x0 的颜色值，具体设置如下:

```
AX_FB_COLORKEY_T colorkey
colorkey.u16Enable = 0;
colorkey.u16Inv = 0;
colorkey.u32KeyLow.b = 0;
colorkey.u32KeyLow.g = 0xc0;
colorkey.u32KeyLow.r = 0xc0;
colorkey.u32KeyUp.b = 0;
colorkey.u32KeyUp.g = 0xc0;
colorkey.u32KeyUp.r = 0xc0;

if (ioctl(s32Fd, AX_FBIOPUT_COLORKEY, &colorkey) < 0){
        printf("set colorkey fb%d failed!\n", u32FbIndex);
        return -1;
}
```

鼠标功能

鼠标功能泛指软鼠标。使用鼠标功能需要加载 ax_fb.ko 驱动，并在 dts 中配置相应的鼠标层内存信息，参考配置如下：

```
vfb3: vfb@3 {
        compatible = "axera,vfb";
        id = <3>;
        width = <1920>;
        height = <1080>;
        bpp = <32>;
        buf-num = <1>;
        cursor;
        status = "okay";
};
```

AX_FBIOPUT_CURSOR_POS

【目的】

设置鼠标显示的起始坐标。

【语法】

int ioctl (int fd, AX_FBIOPUT_CURSOR_POS, AX_FB_CURSOR_POS_T *var);

【描述】

使用此接口设置鼠标层显示的起始位置。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符	输入
AX_FBIOPUT_CURSOR_POS	ioctl 号	输入
var	起始坐标结构体指针	输入

【返回值】

返回值	描述
0	成功
–1	失败

【需求】

头文件：ax_vo_api.h

【注意】

无

【举例】

```
AX_FB_CURSOR_POS_T stPos;
stPos.u16X = u32StartX;
stPos.u16Y = u32StartY;
if (ioctl(fd, AX_FBIOPUT_CURSOR_POS, &stPos) < 0) {
        return -1；
}
```

AX_FBIOPUT_CURSOR_RES

【目的】

设置鼠标显示的宽高。

【语法】

int ioctl (int fd, AX_FBIOPUT_CURSOR_RES, AX_FB_CURSOR_RES_T *var);

【描述】

使用此接口设置鼠标显示的宽度和高度。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符	输入
AX_FBIOPUT_CURSOR_RES	ioctl 号	输入
var	鼠标宽高结构体指针	输入

【返回值】

返回值	描述
0	成功
–1	失败

【需求】

头文件：ax_vo_api.h

【注意】

无

【举例】

```
AX_FB_CURSOR_RES_T stRes;
stRes.u32Width = 50;
stRes.u32Height = 50;
if (ioctl(s32Fd, AX_FBIOPUT_CURSOR_RES, &stRes) < 0) {
    return -1;
}
```

AX_FBIOPUT_CURSOR_SHOW

【目的】

设置鼠标显示状态。

【语法】

int ioctl (int fd, AX_FBIOPUT_CURSOR_SHOW, AX_U16 *var);

【描述】

使用此接口设置鼠标显示或隐藏。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符	输入
AX_FBIOPUT_CURSOR_SHOW	ioctl 号	输入
var	显示标识指针	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

头文件：ax_vo_api.h

【注意】

默认处于隐藏状态，要想显示需调用此接口设置其为显示状态。

【举例】

无。

AX_FBIOGET_CURSORINFO

【目的】

获取鼠标相关信息。

【语法】

int ioctl (int fd, AX_FBIOGET_CURSORINFO, AX_FB_CURSOR_INFO_T *var);

【描述】

使用此接口获取鼠标相关信息，包括画布起始位置、大小、显示状态。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符	输入
AX_FBIOGET_CURSORINFO	ioctl 号	输入
var	鼠标信息结构体指针	输出

【返回值】

返回值	描述
0	成功
–1	失败

【需求】

头文件：ax_vo_api.h

【注意】

无。

2.2.4 数据类型

struct fb_bitfield

【说明】

位域信息，用于设置像素格式。

【定义】

```
struct fb_bitfield
{
        __u32 offset;     /* beginning of bitfield */
        __u32 length;     /* length of bitfield */
        __u32 msb_right;     /* !=0: Most significant bit is right */
};
```

【成员】

成员名称	描述	支持情况
offset	颜色分量起始比特位。	支持。
length	颜色分量所占比特长度。	支持。
msb_right	右边的比特是否为最高有效位。	只支持该位为 0，即最左边的 bit 为最高有效位。

【注意】

例如 ARGB1555 格式，其位域信息的赋值如下：

```
struct fb_bitfield a16 = {15, 1, 0};
struct fb_bitfield r16 = {10, 5, 0};
struct fb_bitfield g16 = {5, 5, 0};
struct fb_bitfield b16 = {0, 5, 0};
```

struct fb_var_screeninfo

【说明】

可变的屏幕信息。

【定义】

```
struct fb_var_screeninfo
{
    __u32 xres; /* visible resolution */
    __u32 yres;
    __u32 xres_virtual; /* virtual resolution */
    __u32 yres_virtual;
    __u32 xoffset; /* offset from virtual to visible */
    __u32 yoffset; /* resolution */
    __u32 bits_per_pixel; /* guess what */
    __u32 grayscale; /* != 0 Graylevels instead of colors */
    struct fb_bitfield red; /* bitfield in fb mem if true color, */
    struct fb_bitfield green; /* else only length is significant */
    struct fb_bitfield blue;
    struct fb_bitfield transp; /* transparency */
    __u32 nonstd; /* != 0 Non standard pixel format */
    __u32 activate; /* see FB_ACTIVATE_* */
    __u32 height; /* height of picture in mm */
    __u32 width; /* width of picture in mm */
    __u32 accel_flags; /* (OBSOLETE) see fb_info.flags */
    /* Timing: All values in pixclocks, except pixclock (of course) */
    __u32 pixclock; /* pixel clock in ps (pico seconds) */
    __u32 left_margin; /* time from sync to picture */
    __u32 right_margin; /* time from picture to sync */
    __u32 upper_margin; /* time from sync to picture */
    __u32 lower_margin;
    __u32 hsync_len; /* length of horizontal sync */
    __u32 vsync_len; /* length of vertical sync */
    __u32 sync; /* see FB_SYNC_* */
    __u32 vmode; /* see FB_VMODE_* */
    __u32 rotate; /* angle we rotate counter clockwise */
    __u32 reserved[5]; /* Reserved for future compatibility */
};
```

【成员】

成员名称	描述	支持情况
xres	可见屏幕宽度（像素数）。	支持，fb0 默认值为 640。
yres	可见屏幕高度（像素数）。	支持。fb0 默认为 480。
xres_virtual	虚拟屏幕宽度（显存中图像宽度），当该值小于 xres 时会修改 xres，使 xres 值与该值相等。	支持，fb0 默认值为 640。
yres_virtual	虚拟屏幕高度（显存中图像高度），当该值小于 yres 时会修改 yres，使 yres 值与该值相等。结合 xres_virtual，可以用来快速水平或垂直平移图像。	支持，fb0 默认为 640。
xoffset	在 x 方向上的偏移像素数。	支持，默认为 0。
yoffset	在 y 方向上的偏移像素数。	支持，默认为 0。
bits_per_pixel	每个像素所占的比特数。	支持，默认为 32。
grayscale	灰度级。	不支持，缺省值为 0，表示彩色。
red	颜色分量中红色的位域信息。	支持，默认为（24，8，0）。
green	颜色分量中绿色的位域信息。	支持，默认为（16，8，0）。
blue	颜色分量中蓝色的位域信息。	支持，默认为（8，8，0）。

成员名称	描述	支持情况
transp	颜色分量中 alpha 分量的位域信息。	支持，默认为（0，8，0）。
nonstd	是否为标准像素格式。	不支持，缺省值为 0，表示支持标准像素格式。
activate	设置生效的时刻。	不支持，缺省值为 FB_ACTIVATE_NOW，表示设置立刻生效。
height	屏幕高，单位为 mm。	不支持，缺省值为–1。
width	屏幕宽，单位为 mm。	不支持，缺省值为–1。
accel_flags	加速标志。	不支持，缺省值为–1。
pixclock	显示一个点需要的时间，单位为 ns。	不支持，缺省值为–1。
left_margin	左消隐信号，单位为点时钟。	不支持，缺省值为–1。
right_margin	右消隐信号，单位为点时钟。	不支持，缺省值为–1。
hsync_len	水平同步时长，单位为点时钟。	不支持，缺省值为–1。
upper_margin	上消隐信号，单位为点时钟。	不支持，缺省值为–1。
lower_margin	下消隐信号，单位为点时钟。	不支持，缺省值为–1。
vsync_len	垂直同步时长，单位为点时钟。	不支持，缺省值为–1。
sync	同步信号方式。	不支持，缺省值为–1。

成员名称	描述	支持情况
vmode	扫描模式。	不支持，缺省值为–1。
rotate	顺时针旋转的角度。	不支持，缺省值为 0，表示无旋转。

【注意】

无

struct fb_fix_screeninfo

【说明】

固定的屏幕信息。

【定义】

```
struct fb_var_screeninfo
{
        __u32 xres; /* visible resolution */
        __u32 yres;
        __u32 xres_virtual; /* virtual resolution */
        __u32 yres_virtual;
        __u32 xoffset; /* offset from virtual to visible */
        __u32 yoffset; /* resolution */
        __u32 bits_per_pixel; /* guess what */
        __u32 grayscale; /* != 0 Graylevels instead of colors */
        struct fb_bitfield red; /* bitfield in fb mem if true color, */
        struct fb_bitfield green; /* else only length is significant */
        struct fb_bitfield blue;
        struct fb_bitfield transp; /* transparency */
        __u32 nonstd; /* != 0 Non standard pixel format */
        __u32 activate; /* see FB_ACTIVATE_* */
        __u32 height; /* height of picture in mm */
        __u32 width; /* width of picture in mm */
        __u32 accel_flags; /* (OBSOLETE) see fb_info.flags */
        /* Timing: All values in pixclocks, except pixclock (of course) */
        __u32 pixclock; /* pixel clock in ps (pico seconds) */
        __u32 left_margin; /* time from sync to picture */
        __u32 right_margin; /* time from picture to sync */
        __u32 upper_margin; /* time from sync to picture */
        __u32 lower_margin;
        __u32 hsync_len; /* length of horizontal sync */
        __u32 vsync_len; /* length of vertical sync */
        __u32 sync; /* see FB_SYNC_* */
        __u32 vmode; /* see FB_VMODE_* */
        __u32 rotate; /* angle we rotate counter clockwise */
        __u32 reserved[5]; /* Reserved for future compatibility */
};
```

【成员】

成员名称	描述	支持情况
id	设备驱动名称	支持
smem_start	显存起始物理地址	支持
smem_len	显存大小	支持
type	显卡类型	固定为 FB_TYPE_PACKED_PIXELS，表示像素值紧密排列
type_aux	附加类型	不支持，在 FB_TYPE_PACKED_PIXELS 显卡类型下无含义
visual	色彩模式	不支持，默认为 FB_VISUAL_TRUECOLOR，真彩色
xpanstep	支持水平方向上的 PAN 显示：0：不支持。非 0：支持，此时该值用于表示在水平方向上每步进的像素值	固定为 1
ypanstep	支持垂直方向上的 PAN 显示：0：不支持。非 0：支持，此时该值用于表示在垂直方向上每步进的像素值	固定为 1
ywrapstep	该方式类似于 ypanstep，不同之处在于：当其显示到底部时，能回到显存的开始处进行显示	不支持，默认为 0
line_length	每行字节数	支持

成员名称	描述	支持情况
mmio_start	显存映射 I/O 首地址	不支持，默认为 0
mmio_len	显存映射 I/O 长度	不支持，默认为 0
accel	显示所支持的硬件加速设备	不支持，默认为 FB_ACCEL_NONE，无加速设备
reserved	保留	不支持，缺省值为 0

【注意】

无

AX_FB_COLORKEY_T

【说明】

扩展属性 colorkey 信息。

【定义】

```
typedef struct axFB_COLORKEY_T {
    AX_U16 u16Enable;
    AX_U16 u16Inv;
        AX_U32 u32KeyLow;
        AX_U32 u32KeyHigh;
} AX_FB_COLORKEY_T;
```

【成员】

成员名称	描述
u16Enable	使能 Colorkey
u16Inv	反色
u32KeyLow	Colorkey 值下限，格式：RGB888
u32KeyHigh	Colorkey 值下限，格式：RGB888

【注意】

关键色的设置是一个范围，其逻辑为：

(R_low <= R <= R_high) && (G_low <= G <= G_high) && (B_low <= B <= B_high)

当 u16Inv 为 0 时，满足上述条件的像素全部透明，当 u16Inv 为 1 时，不满足上述条件的像素全部透明。通常用户在使用关键色时基本上是一个固定值，并不是一个范围，此时可以将 u32KeyLow 和 u32KeyHigh 设置成相同的即可

AX_FB_CURSOR_POS_T

【说明】

扩展属性鼠标起始坐标描述。

【定义】

```
typedef struct axFB_CURSOR_POS_T {
    AX_U16 u16X;
    AX_U16 u16Y;
} AX_FB_CURSOR_POS_T;
```

【成员】

成员名称	描述
u16X	鼠标起始 x 坐标
u16Y	鼠标起始 y 坐标

【注意】

无。

AX_FB_CURSOR_RES_T

【说明】

扩展属性鼠标宽高信息描述。

【定义】

```
typedef struct axFB_CURSOR_RES_T {
    AX_U32 u32Width;
    AX_U32 u32Height;
} AX_FB_CURSOR_RES_T;
```

【成员】

成员名称	描述
u32Width	鼠标显示宽度
u32Height	鼠标显示高度

【注意】

无。

AX_FB_CURSOR_INFO_T

【说明】

扩展属性鼠标相关信息描述。

【定义】

```
typedef struct axFB_CURSOR_INFO_T {
    AX_U16 u16Enable;
    AX_FB_CURSOR_POS_T stHot;
    AX_FB_CURSOR_RES_T stRes;
} AX_FB_CURSOR_INFO_T;
```

【成员】

成员名称	描述
u16Enable	鼠标使能
stHot	鼠标显示起始坐标
stRes	鼠标显示宽高信息

【注意】

无。

3 首次应用开发

3.1 开发流程

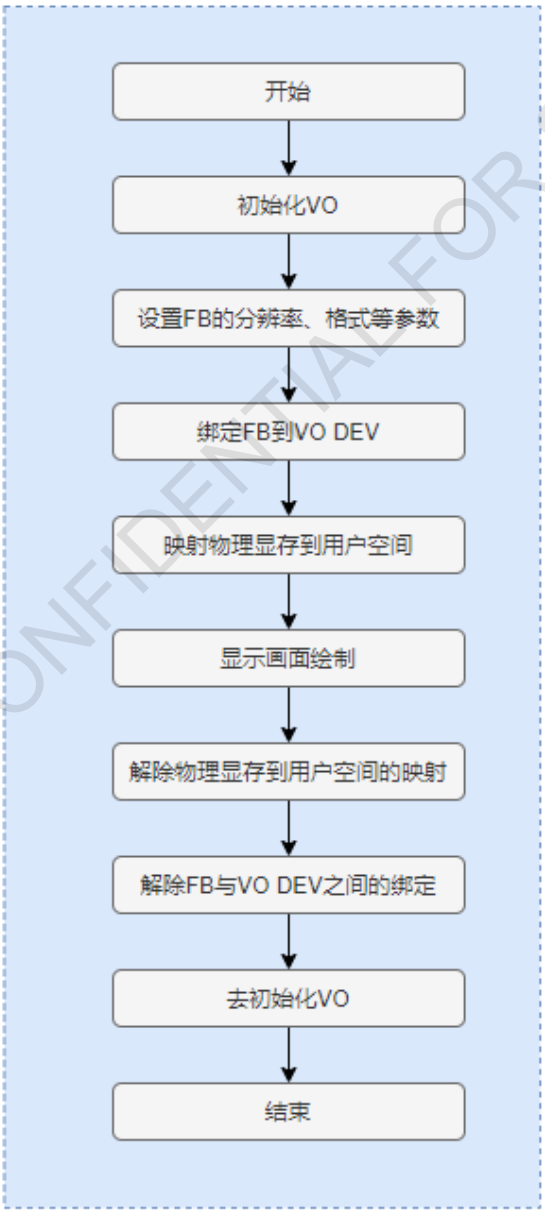

图3-1 FB 开发流程

3.2 实例介绍

本实例利用 PAN_DISPLAY 连续显示 200 帧 1920x1080 的单色图。参考代码如下：

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <linux/fb.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <pthread.h>
#include <signal.h>

#include "ax_sys_api.h"
#include "ax_base_type.h"
#include "ax_vo_api.h"

#ifndef SAMPLE_PRT
#define SAMPLE_PRT(fmt,...)     \
{ \
    printf("[SAMPLE-VO][%s-%d] "fmt, __FUNCTION__, __LINE__, ##__VA_ARGS__); \
}
#endif
```

```
static AX_S32 SAMPLE_VO_FB_INIT(AX_U32 u32Width, AX_U32 u32Height, AX_U32
u32FbIndex)
{
    AX_S32 s32Fd, s32Ret = 0;
    AX_CHAR fbPath[32];
    struct fb_var_screeninfo stVar;
    struct fb_bitfield stR32 = {24, 8, 0};
    struct fb_bitfield stG32 = {16, 8, 0};
    struct fb_bitfield stB32 = {8, 8, 0};
    struct fb_bitfield stA32 = {0, 8, 0};

    /* 1.Open framebuffer device */
    snprintf(fbPath, sizeof(fbPath), "/dev/fb%d", u32FbIndex);
    s32Fd = open(fbPath, O_RDWR);
    if (s32Fd < 0) {
        SAMPLE_PRT("open %s failed, err:%s\n", fbPath, strerror(errno));
        return s32Fd;
    }

    /* 2.Get the variable screen info */
    s32Ret = ioctl(s32Fd, FBIOGET_VSCREENINFO, &stVar);
    if (s32Ret < 0) {
        SAMPLE_PRT("get variable screen info from fb%d failed\n", u32FbIndex);
        goto exit;
    }
```

```
    /* 3.Modify the variable screen info, the screen size: u32Width*u32Height, the
     * virtual screen size: u32Width*(u32Height*2), the pixel format: ARGB8888
     */
    stVar.xres = stVar.xres_virtual = u32Width;
    stVar.yres = u32Height;
    stVar.yres_virtual = u32Height * 2;
    stVar.transp = stA32;
    stVar.red = stR32;
    stVar.green = stG32;
    stVar.blue = stB32;
    stVar.bits_per_pixel = 32;

    /* 4.Set the variable screeninfo */
    s32Ret = ioctl(s32Fd, FBIOPUT_VSCREENINFO, &stVar);
    if (s32Ret < 0) {
        SAMPLE_PRT("put variable screen info to fb%d failed\n", u32FbIndex);
        goto exit;
    }

    SAMPLE_PRT("init fb%d done\n", u32FbIndex);

exit:
    close(s32Fd);

    return s32Ret;
}
```

```
static AX_VOID SAMPLE_VO_FB_FILL(AX_U32 u32Width, AX_U32 u32Height, AX_U32
u32Color, AX_U8 *pShowScreen)
{
    AX_S32 i, j;
    AX_U32 *u32Pixel;

    for (i = 0; i < u32Height; i++) {
        u32Pixel = (AX_U32 *)(pShowScreen + i * u32Width * 4);
        for (j = 0; j < u32Width; j++) {
            u32Pixel[j] = u32Color;
        }
    }
}

static AX_S32 SAMPLE_VO_FB_DRAW(AX_U32 u32FbIndex)
{
    AX_S32 i, s32Fd, s32Ret = 0;
    AX_U32 u32Offs, u32Color;
    AX_CHAR fbPath[32];
    AX_U8 *pShowScreen;
    struct fb_var_screeninfo stVar;
    struct fb_fix_screeninfo stFix;

    /* 1.Open framebuffer device */
    snprintf(fbPath, sizeof(fbPath), "/dev/fb%d", u32FbIndex);
    s32Fd = open(fbPath, O_RDWR);
```

```
    if (s32Fd < 0) {
        SAMPLE_PRT("open %s failed, err:%s\n", fbPath, strerror(errno));
        return s32Fd;
    }

    /* 2.Get the variable screen info */
    s32Ret = ioctl(s32Fd, FBIOGET_VSCREENINFO, &stVar);
    if (s32Ret < 0) {
        SAMPLE_PRT("get variable screen info from fb%d failed\n", u32FbIndex);
        goto exit;
    }

    /* 3.Get the fix screen info */
    s32Ret = ioctl(s32Fd, FBIOGET_FSCREENINFO, &stFix);
    if (s32Ret < 0) {
        SAMPLE_PRT("get fix screen info from fb%d failed\n", u32FbIndex);
        goto exit;
    }

    /* 4.Map the physical video memory for user use */
    pShowScreen = mmap(NULL, stFix.smem_len, PROT_READ | PROT_WRITE,
MAP_SHARED, s32Fd, 0);
    if (pShowScreen == (AX_U8 *) - 1) {
        SAMPLE_PRT("map fb%d failed\n", u32FbIndex);
        goto exit;
    }
```

```
    for (i = 0; i < 200; i++) {
        if (i % 2) {
            stVar.yoffset = 0;
            u32Color = 0xFF0000FF;
            u32Offs = 0;
        } else {
            stVar.yoffset = stVar.yres;
            u32Color = 0xFFFF0000;
            u32Offs = stVar.xres * stVar.yres * 4;
        }

        SAMPLE_VO_FB_FILL(stVar.xres, stVar.yres, u32Color, pShowScreen + u32Offs);

        s32Ret = ioctl(s32Fd, FBIOPAN_DISPLAY, &stVar);
        if (s32Ret) {
            SAMPLE_PRT("pan fb%d failed, i = %d\n", u32FbIndex, i);
            break;
        }

        usleep(50000);
    }

    munmap(pShowScreen, stFix.smem_len);

exit:
    close(s32Fd);
```

```
    return s32Ret;
}

AX_S32 main(AX_S32 argc, AX_CHAR *argv[])
{
    AX_S32 s32Ret = 0;
    AX_U32 u32Width, u32Height, u32FbIndex = 0;
    VO_DEV VoDev = 0;
    GRAPHIC_LAYER GraphicLayer = 0;
    VO_PUB_ATTR_T stVoPubAttr;

    memset(&stVoPubAttr, 0, sizeof(stVoPubAttr));
    stVoPubAttr.enIntfType = VO_INTF_HDMI;
    stVoPubAttr.enIntfSync = VO_OUTPUT_1080P60;
    u32Width = 1920;
    u32Height = 1080;

    s32Ret = AX_VO_Init();
    if (s32Ret) {
        SAMPLE_PRT("failed with %#x!\n", s32Ret);
        goto exit0;
    }

    s32Ret = AX_VO_SetPubAttr(VoDev, &stVoPubAttr);
    if (s32Ret) {
```

```
        SAMPLE_PRT("failed with %#x!\n", s32Ret);
        goto exit1;
    }

    s32Ret = SAMPLE_VO_FB_INIT(u32Width, u32Height, u32FbIndex);
    if (s32Ret) {
        SAMPLE_PRT("SAMPLE_VO_FB_INIT failed, s32Ret = %d\n", s32Ret);
        goto exit1;
    }

    s32Ret = AX_VO_Enable(VoDev);
    if (s32Ret) {
        SAMPLE_PRT("failed with %#x!\n", s32Ret);
        goto exit1;
    }

    s32Ret = AX_VO_BindGraphicLayer(GraphicLayer, VoDev);
    if (s32Ret) {
        SAMPLE_PRT("AX_VO_BindGraphicLayer failed, s32Ret = 0x%x\n", s32Ret);
        goto exit2;
    }

    SAMPLE_PRT("sample fb start show!!\n");

    SAMPLE_VO_FB_DRAW(u32FbIndex);
```

```
    s32Ret = AX_VO_UnBindGraphicLayer(GraphicLayer, VoDev);
    if (s32Ret) {
        SAMPLE_PRT("AX_VO_UnBindGraphicLayer failed, s32Ret = 0x%x\n", s32Ret);
    }

exit2:
    s32Ret = AX_VO_Disable(VoDev);
    if (s32Ret) {
        SAMPLE_PRT("failed with %#x!\n", s32Ret);
    }

exit1:
    s32Ret = AX_VO_Deinit();
    if (s32Ret) {
        SAMPLE_PRT("failed with %#x!\n", s32Ret);
    }

exit0:
    SAMPLE_PRT("sample fb exit!!\n");

    return 0;
}
```

3.3 FB 的宽度与 Stride 不相同的相关支持

显示硬件要求 stride 必须 8 字节对齐，对于有些分辨率如果 stride 与宽度一致则不满足 8 对齐的要求。

以分辨率：1366×768，格式：ARGB8888 为例。如果 stride 与宽度保持一致，则 stride 为 1366*4 = 5464，它不是 8 对齐，遇到到这种情况的解决办法：

在调用 AX_VO_BindGraphicLayer 之前配置 FB 信息时，将 stVar.xres 配置成 1366，stVar.xres_virtual 配置成((1366 * 4 + 0x7) & (~0x7)) / 4，即：1368