HiFB

开发指南

文档版本 06

发布日期 2014-05-09

版权所有 © 深圳市海思半导体有限公司 2010-2014。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明

、HISILICON、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司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。除非合同另有约定，海思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。

深圳市海思半导体有限公司

地址：深圳市龙岗区坂田华为基地华为电气生产中心　　邮编：518129

网址：http://www.hisilicon.com

客户服务电话：+86-755-28788858

客户服务传真：+86-755-28357515

客户服务邮箱：support@hisilicon.com

前 言

概述

Hisilicon Framebuffer（以下简称 HiFB）是海思数字媒体处理平台提供的管理图像叠加层的模块，它基于 Linux Framebuffer 实现，在提供 Linux Framebuffer 基本功能的基础上，还扩展了一些图形层控制功能，如层间 Alpha、设置原点等。本文档主要介绍 HiFB 模块加载和第一次如何开发应用。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3531 芯片	V100
Hi3532 芯片	V100
Hi3521 芯片	V100
Hi3520A 芯片	V100
Hi3520D 芯片	V100
Hi3515A 芯片	V100
Hi3515C 芯片	V100
Hi3518 芯片	V100

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
危险	表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。
警告	表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
注意	表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
窍门	表示能帮助您解决某个问题或节省您的时间。
说明	表示是正文的附加信息，是对正文的强调和补充。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 06 (2014-05-09)

增加 1.4 注意事项。

文档版本 05 (2013-06-21)

增加 Hi3515C 的相关描述。

文档版本 04 (2013-04-03)

2.2 参数设置中，在参数 softcursor 的描述里，增加“优先考虑硬件鼠标”的描述。

文档版本 03 (2013-02-05)

增加 Hi3520D/Hi3515A 的相关描述。

文档版本 02 (2012-09-20)

图 1-4 下，增加 Hi3518 不支持绘制 buffer 到显示 buffer 过程的缩放。

文档版本 01 (2012-08-30)

将 Hi3520A 的相关描述与 Hi3521 合并。

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

文档版本 00B50 (2012-08-09)

1.2 HiFB 与 Linux Framebuffer 对比

Hi3520A 管理叠加图形层 1 改为 2，支持的标清设备由 SD0 改为 SD1。

增加 Hi3518 芯片的相关描述。

芯片差异中增加 Hi3518 的相关描述

2.2 参数设置

增加 Hi3518 的相关描述

文档版本 00B40 (2012-07-13)

1.2 HiFB 与 Linux Framebuffer 对比

Hi3520A 管理叠加图形层 1 改为 2，支持的标清设备由 SD0 改为 SD1，以及其他涉及的描述都做相同修改。

2.2 参数设置

增加参数 u32VcmpBufNum。

文档版本 00B30 (2012-06-08)

增加关于 Hi3520A 芯片的相关描述。

文档版本 00B20 (2012-04-20)

增加关于 Hi3521 芯片的相关描述。

文档版本 00B10 (2012-01-15)

修改图 1-4 下面的说明。

文档版本 00B01 (2011-11-10)

第 1 次临时版本发布。

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

目 录

前 言 i

1 概述 1

1.1 HiFB 简介 1

1.1.1 体系结构 1

1.1.2 应用场景 1

1.2 HiFB 与 Linux Framebuffer 对比 2

1.3 相关文档 6

1.4 注意事项 7

2 模块加载 9

2.1 原理介绍 9

2.2 参数设置 9

2.3 配置举例 11

2.4 异常情况 11

3 第一次开发应用 12

3.1 开发流程 12

3.2 实例介绍 14

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

插图目录

图 1-1 HiFB 体系结构 1
图 1-2 0 buffer 示意图 6
图 1-3 1 buffer 示意图 6
图 1-4 2 buffer 示意图 6
图 3-1 HiFB 的开发流程 13

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

表格目录

表 1-1 Hi3531 FB 设备文件、图形层以及输出设备的对应关系 2
表 1-2 Hi3521/Hi3520A/Hi3520D/Hi3515A/Hi3515C FB 设备文件、图形层以及输出设备的对应关系 3
表 1-3 Hi3518 FB 设备文件、图形层以及输出设备的对应关系 3
表 3-1 HiFB 的开发阶段任务表 14

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

1 概述

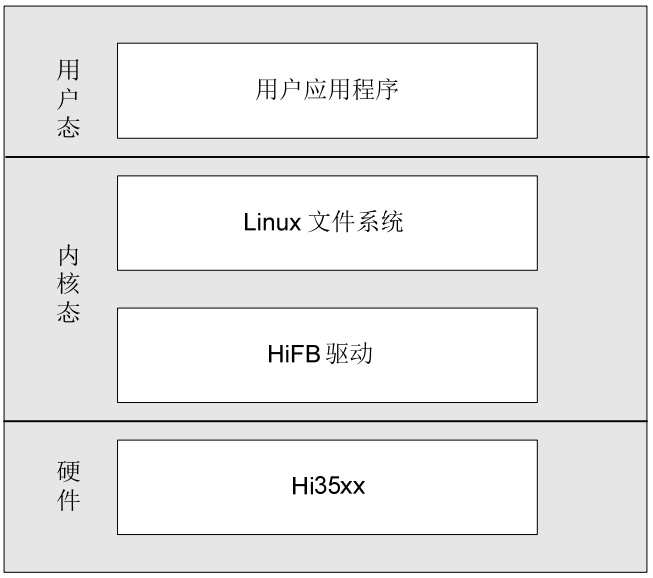
1.1 HiFB 简介

Hisilicon Framebuffer（以下简称 HiFB）是海思数字媒体处理平台提供的用于管理叠加图形层的模块，它不仅提供 Linux Framebuffer 的基本功能，还在 Linux Framebuffer 的基础上增加层间 colorkey、层间 colorkey mask、层间 Alpha、原点偏移等扩展功能。

1.1.1 体系结构

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

图1-1 HiFB 体系结构



1.1.2 应用场景

HiFB 可应用于以下场景：

- MiniGUI 窗口系统

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

MiniGUI 支持 Linux Framebuffer。对 MiniGUI 做少量改动即可移植到 Hi35xx，实现快速移植。

- 其他的基于 Linux Framebuffer 的应用程序

  对基于 Linux Framebuffer 的应用程序不做或做少量改动即可移植到 Hi35xx，实现快速移植。

1.2 HiFB 与 Linux Framebuffer 对比

叠加图形层管理

Linux Framebuffer 是一个子设备号对应一个显卡，HiFB 则是一个子设备号对应一个叠加图形层，HiFB 可以管理多个叠加图形层，具体个数和芯片相关。

说明

- 对于 Hi3531 芯片，HiFB 最多可以管理 7 个叠加图形层：图形层 0～图形层 6（最后两个层为鼠标层），对应的设备文件依次为/dev/fb0~/dev/fb6。Hi3531 芯片支持 4 个输出设备上可以叠加图形层：高清输出设备 0（简称 HD0）、高清输出设备 1（简称 HD1）、标清输出设备 0（简称 SD0）与标清输出设备 1（简称 SD1）。7 个图形层与这 4 个输出设备的关系如表 1-1 所示。
- 对于 Hi3532 芯片，HiFB 最多可以管理 2 个叠加图形层：叠加图形层 0 与叠加图形层 5（G5 为鼠标层），对应的设备文件分别为/dev/fb0、/dev/fb5。Hi3532 芯片只支持 1 个输出设备——高清输出设备 0（简称 HD0），这两个图形层能同时叠加在 HD0 上，无需进行绑定。除此之外，这两个图形层的其他功能都与 Hi3531 描述的一致。
- 对于 Hi3521/Hi3520A/Hi3520D/Hi3515A/Hi3515C 芯片，HiFB 最多可以管理 4 个叠加图形层：叠加图形层 0～叠加图形层 3（G3 为鼠标层），对应的设备文件分别为/dev/fb0～/dev/fb3。Hi3521/Hi3520A/Hi3520D/Hi3515A/Hi3515C 芯片支持 3 个输出设备——高清输出设备 0（简称 HD0），标清输出设备 0（简称 SD0）与标清输出设备 1（简称 SD1）。4 个图形层与这 3 个输出设备的关系如表 1-2 所示。
- 对于 Hi3518 芯片，HiFB 最多可以管理 1 个叠加图形层，对应的设备文件为/dev/fb0。Hi3518 芯片支持 1 个输出设备——标清输出设备 1（简称 SD1），图形层与输出设备的关系如表 1-4 所示。

表1-1 Hi3531 FB 设备文件、图形层以及输出设备的对应关系

FB 设备文件	图形层	对应显示设备
/dev/fb0	G0	只能在 HD0 设备上显示。
/dev/fb1	G1	只能在 HD1 设备上显示。
/dev/fb2	G2	只能在 SD0 设备上显示。
/dev/fb3	G3	只能在 SD1 设备上显示。

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

FB 设备文件	图形层	对应显示设备
/dev/fb4、 /dev/fb5、 /dev/fb6	G4、 鼠标层 0(G5）、 鼠标层 1(G6）	G4~G6 可在 HD0、HD1、SD0 或 SD1 设备上显示，调用者可通过绑定接口指定在哪个显示设备上进行输出。 G5 与 G6 为鼠标层，它们总是处在显示设备叠加层的最高层，G4 则位于其次。如 HD0 上有视频层、G0、G4 和 G5，则叠加顺序从下到上依次为：视频层，G0，G4，G5。 G5 与 G6 可作为硬件鼠标层，也可作为软件鼠标层，由模块加载参数 softcursor 来决定。作为硬件鼠标层，它们的操作方法与其他图形层一样；而作为软件鼠标层，则应使用 HiFB 中实现的软件鼠标专用接口进行操作。

表1-2 Hi3521/Hi3520A/Hi3520D/Hi3515A/Hi3515C FB 设备文件、图形层以及输出设备的对应关系

FB 设备文件	图形层	对应显示设备
/dev/fb0	G0	只能在 HD0 设备上显示。
/dev/fb1	G1	只能在 SD0 设备上显示。
/dev/fb2	G2	只能在 SD1 设备上显示。
/dev/fb3	鼠标层(G3）	G3 可在 HD0、SD0 或 SD1 设备上显示，调用者可通过绑定接口指定在哪个显示设备上进行输出。 G3 为鼠标层，它总是处在显示设备叠加层的最高层。如 HD0 上有视频层、G0 和 G3，则叠加顺序从下到上依次为：视频层，G0，G3。 G3 可作为硬件鼠标层，也可作为软件鼠标层，由模块加载参数 softcursor 来决定，详见 2.2.2 小节。作为硬件鼠标层，它们的操作方法与其他图形层一样；而作为软件鼠标层，则应使用 HiFB 中实现的软件鼠标专用接口进行操作。

表1-3 Hi3518 FB 设备文件、图形层以及输出设备的对应关系

FB 设备文件	图形层	对应显示设备
/dev/fb0	G3	只能在 SD1 设备上显示。

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

通过模块加载参数，可以控制 HiFB 管理其中的一个或多个叠加图形层，并像操作普通文件一样操作叠加图形层。

芯片差异

芯片	支持的图形层	是否支持压缩	是否支持 colorkey	绑定关系
Hi3531	G0～G6	G0~G4 支持压缩，其它层不支持	鼠标层 G5,G6 支持，其他层不支持	G0~G3 固定绑定到 HD0，HD1，SD0，SD1 上 G4、G5 和 G6 可动态绑定 G5 和 G6 不可同时绑定到一个设备上
Hi3532	G0，G5	不支持压缩	鼠标层 G5 支持，其他层不支持	G0 和 G5 固定绑定在同一个设备上
Hi3521/Hi3520A/Hi3520D/Hi3515A/Hi3515C	G0～G3	G0 支持压缩，其它层不支持	所有层均支持	G0 固定绑定到 HD0 上，G1 固定绑定在 SD0，G2 固定绑定在 SD1 上 G3 可动态绑定
Hi3518	G3	不支持压缩	支持	固定绑定到 SD1

时序控制

Linux Framebuffer 提供同步时序、扫描方式、同步信号组织等控制方式（需要硬件支持），将物理显存的内容显示在不同的输出设备（如 PC 显示器、TV、LCD 等）上。目前 HiFB 不支持同步时序、扫描方式、同步信号组织等控制方式。

标准功能与扩展功能

HiFB 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。
- 设置硬件显示分辨率和象素格式，每个叠加图形层的支持的最大分辨率和象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。
- 在叠加图形层支持索引格式的情况下，支持设置和获取 256 色的调色板。

说明

对于除了 Hi3518 以外的其它 Hi35xx 芯片，每一个图形层都是只支持 ARGB1555 与 ARGB8888 这两种格式，而 Hi3518 芯片只支持 ARGB1555 和 ARGB4444。

HiFB 增加以下的扩展功能：

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

- 设置和获取叠加图形层的 Alpha 值
- 设置和获取叠加图形层的 colorkey 值
- 设置当前叠加图形层的起始位置（相对于屏幕原点的偏移）
- 设置和获取当前叠加图形层的显示状态（显示/隐藏）
- 通过模块加载参数配置 HiFB 的物理显存大小和管理叠加图形层的数目
- 设置和获取抗闪烁功能的状态
- 设置和获取预乘模式
- 设置和获取压缩模式的状态
- 设置/获取图形层刷新类型（0 buffer、1 buffer 与 2 buffer）
- 支持软鼠标的一系列操作

HiFB 不支持以下的 Linux Framebuffer 标准功能：

- 设置和获取控制台对应的 Linux Framebuffer
- 获取硬件扫描的实时信息
- 获取硬件相关信息
- 设置硬件同步时序
- 设置硬件同步信号机制

图形层刷新类型——FB 扩展模式

HiFB 为上层用户提供了一套完整的刷新方案，称为 FB 的扩展模式。在对系统性能、内存、图形显示效果各方面综合衡量的基础上，可根据需要选择合适的刷新方案。目前提供的刷新类型有：

- 0 buffer（即 HIFB_LAYER_BUF_NONE）

  上层用户的绘制 buffer 即为显示 buffer。该刷新类型可以节省内存消耗，速度也最快，但是用户会看到图形的绘制过程。示意如图 1-2 所示。

- 1 buffer（即 HIFB_LAYER_BUF_ONE）

  显示 buffer 由 HIFB 提供，因此需要一定的内存。该刷新类型是对显示效果和内存需求的一种折中考虑。但是会有锯齿。示意如图 1-3 所示。

- 2 buffer

  显示 buffer 由 HIFB 提供。和前面的刷新类型相比，其要求内存最多，但图形显示效果最好。示意如图 1-4 所示。包含以下：

  - HIFB_LAYER_BUF_DOUBLE
  - HIFB_LAYER_BUF_DOUBLE_IMMEDIATE

  两者的区别在于后面的每次刷新操作都要等待到绘制的内容真正显示后才返回）。

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

图1-2 0 buffer 示意图

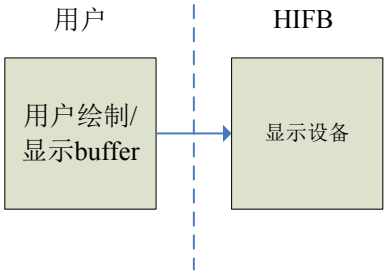


图1-3 1 buffer 示意图

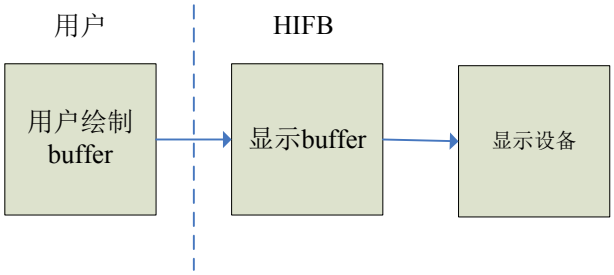
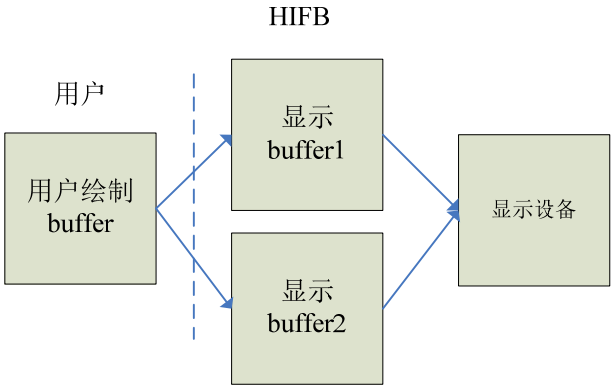


图1-4 2 buffer 示意图



说明

上文提到的三个分辨率：画布分辨率（即用户绘制 buffer 的分辨率）、显存分辨率、屏幕显示分辨率。绘图内容从用户绘制 buffer 到显示 buffer 的过程支持缩放（Hi3518 该过程不支持缩放），也支持抗闪烁；而从显示 buffer 到显示设备的过程不支持缩放，也不支持抗闪，所以显存分辨率与屏幕显示分辨率总是相同。

1.3 相关文档

与本指南相关的文档有：

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

- 《HiFB API 参考》

1.4 注意事项

对 HiFB 的操作需要保证 VO 模块处于正常的打开状态，如果 HiFB 和 VO 在不同的进程进行操作，在进行 VO 的初始化和去初始化时，需要 HiFB 同步进行初始化和去初始化操作，HiFB 需要重新关闭和打开。

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

2 模块加载

2.1 原理介绍

某些 Linux Framebuffer 驱动（如 versa）不支持在运行期间更改分辨率、颜色深度、时序等显示属性。对此，Linux 系统提供一种机制，允许在内核启动或模块加载时，通过参数将相应选项传递给 Linux Framebuffer。可以在内核加载器中配置内核启动参数。HiFB 驱动在加载时只能设置物理显存的大小，不允许设置其它选项。

加载 HiFB 驱动 hifb.ko 时必须保证内核中已经加载了标准的 Framebuffer 驱动 fb.ko。如果没有加载，可以先用“modprobe fb”加载 fb.ko，然后再加载 hifb.ko。

2.2 参数设置

HiFB 可配置其管理的叠加图形层物理显存的大小。物理显存大小决定了 HiFB 可使用的最大物理显存和系统的可设置虚拟分辨率。在加载 HiFB 驱动时通过参数传递设置物理显存大小，物理显存的大小一经设置就不会改变。

参数 video

```
video="hifb:vram0_size:xxx, vram1_size:xxx,…"
```

说明

- 选项之间用逗号“,”隔开。
- 选项和选项值之间用冒号“:”隔开。
- 如果某个图层不配置物理显存大小，则系统默认分配为 0。
- vram0_size ~ vram6_size 分别对应于叠加图形层 0 ~ 叠加图形层 6。

其中，vramn_size:xxx 表示对叠加图形层 n 配置 xxx K 字节的物理显存。

（1）对于 FB 标准模式，vramn_size 和虚拟分辨率的关系如下：

```
Vramn_size * 1024 >= xres_virtual * yres_virtual * bpp;
```

其中：xres_virtual * yres_virtual 是虚拟分辨率，bpp 是每个像素所占字节数。

（2）对于 FB 扩展模式，各个图形层需要的内存大小取决于 displaysize 的大小、图层像素格式以及刷新模式，具体关系如下：

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

```
vramn_size * 1024 >= displaywidth * displayHeight * bpp * BufferMode;
```

如：图形层 0 在 1280*720 分辨率、ARGB8888 格式的 2 buffer 模式下需要的内存 vram0_size = 1280*720*4*2 = 7200 K。

说明

vramn_size 必须是 PAGE_SIZE（4K byte）的倍数，否则 HiFB 驱动强制将其设为 PAGE_SIZE 的倍数，向上取整。

参数 softcursor

该参数决定是否启用软鼠标功能。当它的值为“off”时，则软鼠标功能为禁止状态（即硬件鼠标功能可用），否则软鼠标功能启用。模块一旦加载，是否启用软鼠标功能便确定。

说明

建议优先考虑使用硬件鼠标。

参数 apszLayerMmzNames

该参数决定各个图形层要用到的内存将从哪个 mmz 上进行分配。该参数为一个字符串数组，最多允许设置 7 个值，分别依次对应 fb0~fb6。模块一旦加载，各个图形层所要用到的内存在哪个 mmz 上分配便确定。如果不指定值，则系统默认相应层要用到的内存将在无名的 mmz 上分配。

参数 u32VcmpBufNum

该参数决定各个图形层压缩模式时用到的 VCMP Buf 数。数目越多时占用内存也越多，但是当数目小于 3 时，可能会在整个图形层分辨率的底部的数行出现花屏现象。该值最大为 3，默认情况下为 2。

参数默认值

如果加载 HiFB 驱动时不带任何参数，则系统默认配置的参数值见下。

（1） Hi3531

video="hifb:vram0_size:7200,vram1_size:7200,vram2_size:3240,vram3_size:3240,vram4_size:7200,vram5_size:128,vram6_size:128" softcursor="off"

（2） Hi3532

video = "hifb:vram0_size:7200,vram5_size:128" softcursor="off"

（3） Hi3521/Hi3520A/Hi3520D/Hi3515A/Hi3515C

video="hifb:vram0_size:8100, vram1_size:1620,vram2_size:1620,vram3_size:32" softcursor="off"

（4） Hi3518

video = "hifb:vram0_size:1620"

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

用户需要从全局的角度出发配置 HiFB 需要管理的叠加图形层以及相应的存储空间应从哪个 mmz 上分配，并且为每个叠加图形层分配适当的显存。

2.3 配置举例

配置 HiFB 管理叠加图形层的示例如下：

说明

HiFB 驱动的模块文件为 hifb.ko。

- 配置 HiFB 管理一个叠加图形层。

  如果只需要 HiFB 管理叠加图形层 0，且最大虚拟分辨率为 720 x 576，用到的象素格式为 ARGB1555，则叠加图形层 0 需要的最小显存为 720 x 576 x 2 = 829440 = 810K，配置参数如下：

  insmod hifb.ko video="hifb:vram0_size:810, vram2_size:0"。

  如果采用的是 double buffer 的方式，则需要乘以 2，即：

  insmod hifb.ko video="hifb:vram0_size:1620, vram2_size:0"。

- 配置 HiFB 管理多个叠加图形层。

  如果需要 HiFB 管理叠加图形层 0 和叠加图形层 1 两个叠加层，且最大虚拟分辨率为 720 x 576，用到的像素格式为 ARGB1555，则两个叠加层需要的最小显存都为 720 x 576 x 2 = 829440 = 810K，配置参数如下：

  insmod hifb.ko video="hifb:vram0_size:810, vram1_size: 810"

2.4 异常情况

配置 HiFB 时可能出现以下异常情况：如果配置叠加图形层物理显存数据错误，则 HiFB 将不管理相应的叠加图形层。

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

3 第一次开发应用

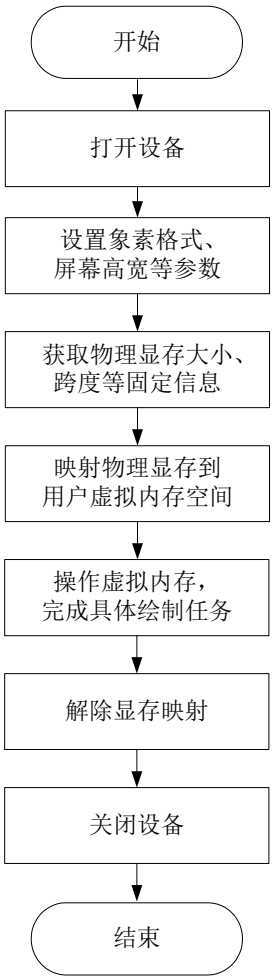
3.1 开发流程

HiFB 主要用于显示 2D 图形（以直接操作物理显存的方式）。

HiFB 的开发流程如图 3-1 所示。

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

图3-1 HiFB 的开发流程



HiFB 的开发步骤如下：

1. 调用 open 函数打开指定的 HiFB 设备。
2. 调用 ioctl 函数设置 HiFB 的象素格式以及屏幕高宽等参数（详细内容请参见《HiFB API 参考》）。
3. 调用 ioctl 函数获取 HiFB 所分配的物理显存大小、跨度等固定信息。调用 ioctl 函数也可以使用 HiFB 提供的层间 colorkey、层间 colorkey mask、层间 alpha、原点偏移等功能。
4. 调用 mmap 函数将物理显存映射到虚拟内存空间。
5. 操作虚拟内存，完成具体的绘制任务。在此步骤可以使用 HiFB 提供的双缓冲页翻转等功能实现一些绘制效果。
6. 调用 munmap 解除显存映射。
7. 调用 close 函数关闭设备。

----结束

海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

说明

由于修改虚拟分辨率将改变 HiFB 的固定信息 fb_fix_screeninfo::line_length（跨度），为保证绘制程序能够正确执行，推荐先设置 HiFB 的可变信息 fb_var_screeninfo，再获取 HiFB 的固定信息 fb_fix_screeninfo::line_length。

HiFB 各个开发各阶段完成的任务如表 3-1 所示。

表3-1 HiFB 的开发阶段任务表

阶段	任务
初始化阶段	完成显示属性的设置和物理显存的映射。
绘制阶段	完成具体的绘制工作。
终止阶段	完成资源清理工作。

3.2 实例介绍

本实例利用 PAN_DISPLAY 连续显示 15 幅分辨率为 640×352 的图片，以达到动态显示的效果。

15 幅图片数据存储在“SDK 的演示 sample 目录下的 hifb/res”，每个文件存储的都是象素格式为 ARGB1555 的纯数据（不包含附加信息的图像数据）。

本实例的代码文件存放在“SDK 的演示 sample 目录下的 hifb/api_sample_hifb.c”。

【参考代码】

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <linux/fb.h>
#include "hifb.h"

#define IMAGE_WIDTH     640
#define IMAGE_HEIGHT    352
#define IMAGE_SIZE      (640*352*2)
#define IMAGE_NUM       14
#define IMAGE_PATH      "./res/%d.bits"

static struct fb_bitfield g_r16 = {10, 5, 0};
static struct fb_bitfield g_g16 = {5, 5, 0};
static struct fb_bitfield g_b16 = {0, 5, 0};
static struct fb_bitfield g_a16 = {15, 1, 0};
```


海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

```
int main()
{
    int fd;
    int i;
    struct fb_fix_screeninfo fix;
    struct fb_var_screeninfo var;
    unsigned char *pShowScreen;
    unsigned char *pHideScreen;
    HIFB_POINT_S stPoint = {40, 112};
    FILE *fp;
    VO_PUB_ATTR_S stPubAttr = {0};
    char image_name[128];

    /*0. open VO device 0 */
    /* …… initialize the attributes for stPubAttr */
    HI_MPI_VO_SetPubAttr(0, &stPubAttr);
    HI_MPI_VO_Enable(0);

    /*1. open Framebuffer device overlay 0*/
    fd = open("/dev/fb/0", O_RDWR);
    if(fd < 0)
    {
        printf("open fb0 failed!\n");
        return -1;
    }

    /*2. set the screen original position*/
    if (ioctl(fd, FBIOPUT_SCREEN_ORIGIN_HIFB, &stPoint) < 0)
    {
        printf("set screen original show position failed!\n");
        return -1;
    }

    /*3. get the variable screen info*/
    if (ioctl(fd, FBIOGET_VSCREENINFO, &var) < 0)
    {
        printf("Get variable screen info failed!\n");
        close(fd);
        return -1;
    }

    /*4. modify the variable screen info
         the screen size: IMAGE_WIDTH*IMAGE_HEIGHT
         the virtual screen size: IMAGE_WIDTH*(IMAGE_HEIGHT*2)
```


海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

```
        the pixel format: ARGB1555
    */
    var.xres = var.xres_virtual = IMAGE_WIDTH;
    var.yres = IMAGE_HEIGHT;
    var.yres_virtual = IMAGE_HEIGHT*2;

    var.transp= g_a16;
    var.red = g_r16;
    var.green = g_g16;
    var.blue = g_b16;
    var.bits_per_pixel = 16;

    /*5. set the variable screeninfo*/
    if (ioctl(fd, FBIOPUT_VSCREENINFO, &var) < 0)
    {
        printf("Put variable screen info failed!\n");
        close(fd);
        return -1;
    }

    /*6. get the fix screen info*/
    if (ioctl(fd, FBIOGET_FSCREENINFO, &fix) < 0)
    {
        printf("Get fix screen info failed!\n");
        close(fd);
        return -1;
    }

    /*7. map the physical video memory for user use*/
    pShowScreen = mmap(NULL, fix.smem_len, PROT_READ|PROT_WRITE,
MAP_SHARED, fd, 0);
    pHideScreen = pShowScreen + IMAGE_SIZE;
    memset(pShowScreen, 0, IMAGE_SIZE);

    /*8. load the bitmaps from file to hide screen and set pan display the
hide screen*/
    for(i = 0; i < IMAGE_NUM; i++)
    {
        sprintf(image_name, IMAGE_PATH, i);
        fp = fopen(image_name, "rb");
        if(NULL == fp)
        {
            printf("Load %s failed!\n", image_name);
            close(fd);
```


海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司

```
            return -1;
        }

        fread(pHideScreen, 1, IMAGE_SIZE, fp);
        fclose(fp);
        usleep(10);
        if(i%2)
        {
            var.yoffset = 0;
            pHideScreen = pShowScreen + IMAGE_SIZE;
        }
        else
        {
            var.yoffset = IMAGE_HEIGHT;
            pHideScreen = pShowScreen;
        }

        if (ioctl(fd, FBIOPAN_DISPLAY, &var) < 0)
        {
            printf("FBIOPAN_DISPLAY failed!\n");
            close(fd);
            return -1;
        }
    }

    printf("Enter to quit!\n");
    getchar();

    /*9. close the devices*/
    close(fd);
    HI_MPI_VO_Disable(0);

    return 0;

}
```


海思专有和保密信息
版权所有 © 深圳市海思半导体有限公司