



HiFB

开发指南

文档版本	05
发布日期	2009-12-23
BOM编码	N/A

深圳市海思半导体有限公司为客户提供全方位的技术支持，用户可与就近的海思办事处联系，也可直接与公司总部联系。

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

版权所有 © 深圳市海思半导体有限公司 2007-2008。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

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。

秘密

版权所有 © 深圳市海思半导体有限公司



目 录

前 言.....	1
1 概述.....	1-1
1.1 HiFB简介	1-2
1.1.1 体系结构.....	1-2
1.1.2 应用场景.....	1-2
1.2 HiFB与Linux Framebuffer对比.....	1-3
1.3 相关文档.....	1-7
2 模块加载.....	2-1
2.1 原理介绍	2-2
2.2 参数设置.....	2-2
2.3 配置举例	2-3
2.4 异常情况	2-3
3 第一次开发应用.....	3-1
3.1 开发流程	3-2
3.2 实例介绍	3-3



插图目录

图 1-1 HiFB体系结构.....	1-2
图 1-2 Hi3515 图像层与输出设备的对应关系	1-4
图 3-1 HiFB的开发流程.....	3-2



表格目录

表 1-1 Hi3520 图像层与输出设备的对应关系	1-3
表 1-2 HiFB在Hi3510 上支持的像素格式	1-5
表 1-3 HiFB在Hi3511/Hi3512 上支持的像素格式	1-5
表 1-4 HiFB在Hi3520 上支持的像素格式	1-5
表 1-5 HiFB在Hi3515 上支持的像素格式	1-6
表 2-1 vramn_size和叠加图像层对应关系.....	2-2
表 3-1 HiFB的开发阶段任务表.....	3-3



前言

概述

本节介绍本文档的内容、对应的产品版本、适用的读者对象、行文表达约定、历史修订记录等。

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
Hi3510 通信媒体处理器	V100
Hi3511 H.264 编解码处理器	V100
Hi3512 H.264 编解码处理器	V100
Hi3520 H.264 编解码处理器	V100
Hi3515 H.264 编解码处理器	V100

读者对象

本文档（本指南）主要适用于以下工程师：





- 电子产品设计维护人员
- 电子产品元器件市场销售人员

约定

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	以本标志开始的文本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	以本标志开始的文本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	以本标志开始的文本表示有潜在风险，如果忽视这些文本，可能导致设备或器件损坏、数据丢失、设备性能降低或不可预知的结果。
 窍门	以本标志开始的文本能帮助您解决某个问题或节省您的时间。
 说明	以本标志开始的文本是正文的附加信息，是对正文的强调和补充。

通用格式约定

格式	说明
宋体	正文采用宋体表示。
黑体	一级、二级、三级标题采用黑体。
楷体	警告、提示等内容一律用楷体，并且在内容前后增加线条与正文隔离。
“Terminal Display” 格式	“Terminal Display” 格式表示屏幕输出信息。此外，屏幕输出信息中夹杂的用户从终端输入的信息采用加粗字体表示。

命令行格式约定

格式	意义
粗体	命令行关键字（命令中保持不变、必须照输的部分）采用加粗字体表示。
<i>斜体</i>	命令行参数（命令中必须由实际值进行替代的部分）采用斜体表示。
[]	表示用 “[]” 括起来的部分在命令配置时是可选的。
{ x y ... }	表示从两个或多个选项选取一个。
[x y ...]	表示从两个或多个选项选取一个或者不选。



格式	意义
{ x y ... } *	表示从两个或多个选项中选择多个，最少选取一个，最多选取所有选项。
[x y ...] *	表示从两个或多个选项中选择多个或者不选。

表格内容约定

内容	说明
-	表格中的无内容单元。
*	表格中的内容用户可根据需要进行配置。

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。

修订日期	版本	修订说明
2009-12-01	05	第 1 章 概述 增加 Hi3515 的信息，例如修改图 1-1、增加表 1-2 和表 1-5 等。
2009-09-03	04	<ul style="list-style-type: none">图 1-1 中增加 Hi3520。1.2 HiFB与Linux Framebuffer对比中叠加图像层管理增加 Hi3520 的说明和表 1-1。增加表 1-4 HiFB在Hi3520上支持的象素格式。2.2 参数设置中增加分配多个图形层时，FB 系统设备号与图形层之间的对应关系。
2008-08-22	03	<ul style="list-style-type: none">增加关于 Hi3512 芯片的说明。
2008-04-08	02	<ul style="list-style-type: none">修改 2.3，由原来的 720×2×576=829440 改为 720×576×2=829440。删除1.2 HiFB增加功能“支持水平和垂直二阶抗闪烁（Hi3511）”。修改2.4 异常情况中的描述。修改3.2中的存放路径。
2007-11-30	01	第一次发布



1 概述

关于本章

本章描述内容如下表所示。

标题	内容
1.1 HiFB 简介	介绍 HiFB 的体系结构、应用场景。
1.2 HiFB 与 Linux Framebuffer 对比	介绍 HiFB 与 Linux Framebuffer 的功能对比。
1.3 相关文档	介绍与本指南相关的文档。



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像层的模块，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
MiniGUI 支持 Linux Framebuffer。对 MiniGUI 做少量改动即可移植到 Hi3510、Hi3511/Hi3512、Hi3520 或 Hi3515，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 Hi3510、Hi3511/Hi3512、Hi3520 或 Hi3515，实现快速移植。



1.2 HiFB 与 Linux Framebuffer 对比

叠加图像层管理

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像层，HiFB 可以管理多个叠加图像层，具体个数和芯片相关。

说明

- 对于 Hi3510 芯片，HiFB 最多可以管理 2 个叠加图像层：叠加图像层 0 和叠加图像层 1。
- 对于 Hi3511/Hi3512 芯片，HiFB 最多可以管理 2 个叠加图像层：叠加图像层 0 和叠加图像层 1。
- 对于 Hi3520 芯片，HiFB 最多可以管理 5 个叠加图像层：叠加图像层 0~叠加图像层 4。
Hi3520 芯片支持 3 个输出设备：高清输出设备（简称 HD）、辅助显示设备（简称 AD）、单画面轮循显示设备（简称 SPOT 设备或 SD）。5 个图像层与对应输出设备的关系如表 1-1 所示。
- 对于 Hi3515 芯片，HiFB 最多可以管理 4 个叠加图像层：叠加图像层 0、1、2 和 4。Hi3515 芯片支持 2 个输出设备：高清输出设备（简称 HD）、单画面轮循显示设备（简称 SPOT 设备或 SD）。4 个图像层与对应输出设备的关系如图 1-2 所示。

表1-1 Hi3520 图像层与输出设备的对应关系

图形层	对应显示设备	说明
叠加图像层 0 (G0)	HD	G0 只能在 HD 设备上显示。
叠加图像层 1 (G1)	HD 或 AD	G1 可在 HD 或 AD 设备上显示，调用者可通过绑定接口指定显示设备。 G1 切换时需要关闭 HD 和 AD 设备，再进行绑定。
叠加图像层 2 (G2)	AD	G2 只能在 AD 设备上显示。
叠加图像层 3 (G3)	SD	G3 只能在 SD 设备上显示。
叠加图像层 4 (G4)	HD 或 AD	G4 可在 HD 或 AD 设备上显示，调用者可通过绑定接口动态指定显示设备。 G4 支持动态在 HD 或 AD 设备间切换，不需要关闭 HD 或 AD 设备。 G4 总是处在显示设备叠加层的最高层。如 HD 上有视频层、G0 和 G4，则叠加顺序为 G0 在视频层之上、G4 在 G0 之上。

图1-2 Hi3515 图像层与输出设备的对应关系

图形层	对应显示设备	说明
叠加图形层 0 (G0)	HD	G0 默认并固定在 HD 设备上显示。
叠加图形层 1 (G1)	HD 或 SD	G1 可在 HD 或 SD 设备上显示，调用者可通过绑定接口指定显示设备。 G1 切换为静态切换，即 HD 和 SD 设备会自动关闭再开启。G1 切换的用户操作如下：先关闭 G1→设置新的绑定关系→配置并开启 G1。
叠加图形层 2 (G2)	SD	G2 默认并固定在 SD 设备上显示。
叠加图形层 4 (G4)	HD 或 SD	G4 常用作鼠标显示。 G4 可在 HD 或 SD 设备上显示，调用者可通过绑定接口指定显示设备。 G4 切换的用户操作和 G1 切换时一致，即先关闭 G4→设置新的绑定关系→配置并开启 G4。 G4 切换与 G1 切换的不同点是：G4 切换时 HD 和 SD 设备状态保持不变，不会关闭再开启，故不会出现短暂黑屏现象；而 G1 在用户设置绑定关系时，会自动关闭再开启 HD 和 SD 设备，故会出现短暂黑屏。 G4 总是处在显示设备叠加层的最高层。如 HD 上有视频层、G0 和 G4，则叠加顺序为 G0 在视频层之上、G4 在 G0 之上。

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像层，并像操作普通文件一样操作叠加图像层。

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，每个叠加图像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，支持的象素格式参见表 1-2、表 1-3、表 1-4 和表 1-5。
- 从物理显存的任何位置进行读、写、显示等操作。
- 在叠加图像层支持索引格式的情况下，支持设置和获取 256 色的调色板。

表1-2 HiFB 在 Hi3510 上支持的象素格式

叠加图像层	图像类型	象素格式
叠加图像层 0	索引格式	不支持
	16 比特格式	ARGB1555
	32 比特格式	ARGB8888
叠加图像层 1	索引格式	不支持
	16 比特格式	ARGB1555
	32 比特格式	ARGB8888

表1-3 HiFB 在 Hi3511/Hi3512 上支持的象素格式

叠加图像层	图像类型	象素格式
叠加图像层 0	索引格式	不支持
	16 比特格式	ARGB1555
	32 比特格式	ARGB8888
叠加图像层 1	索引格式	不支持
	16 比特格式	ARGB1555
	32 比特格式	ARGB8888

表1-4 HiFB 在 Hi3520 上支持的象素格式

叠加图像层	图像类型	象素格式
叠加图像层 0	索引格式	不支持
	16 比特格式	ARGB1555
	32 比特格式	不支持
叠加图像层 1	索引格式	不支持
	16 比特格式	ARGB1555

叠加图像层	图像类型	象素格式
叠加图像层 2	32 比特格式	不支持
	索引格式	不支持
	16 比特格式	ARGB1555
叠加图像层 3	32 比特格式	不支持
	索引格式	不支持
	16 比特格式	ARGB1555
叠加图像层 4	32 比特格式	不支持
	索引格式	不支持
	16 比特格式	ARGB1555

表1-5 HiFB 在 Hi3515 上支持的象素格式

叠加图像层	图像类型	象素格式
叠加图像层 0	32 比特格式	不支持
	索引格式	不支持
	16 比特格式	ARGB1555
叠加图像层 1	32 比特格式	ARGB8888
	索引格式	不支持
	16 比特格式	ARGB1555
叠加图像层 2	32 比特格式	ARGB8888
	索引格式	不支持
	16 比特格式	ARGB1555
叠加图像层 4	32 比特格式	ARGB8888
	索引格式	不支持
	16 比特格式	ARGB1555

HiFB 增加以下的扩展功能：

- 设置和获取叠加图像层的 Alpha 值



- 设置和获取叠加图像层的 colorkey 值和 colorkey mask 值
- 设置当前叠加图像层的起始位置（相对于屏幕原点的偏移）
- 设置和获取当前叠加图像层的显示状态（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像层的数目

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

1.3 相关文档

与本指南相关的文档有：

- 《HiFB API 参考》



2 模块加载

关于本章

本章描述内容如下表所示。

标题	内容
2.1 原理介绍	介绍 HiFB 加载的原理。
2.2 参数设置	介绍 HiFB 管理叠加图像层的参数配置过程。
2.3 配置举例	介绍 HiFB 管理一个或多个叠加图像层的配置示例。
2.4 异常情况	介绍配置 HiFB 时可能出现的异常情况。

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像层物理显存的大小。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，物理显存大小一经设置就不会改变。

HiFB 模块参数的语法如下：

```
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。

其中，vramn_size:xxx 表示对叠加图形层 n 配置 xxx 字节的物理显存，vramn_size 和虚拟分辨率的关系如下：

$$vramn_size \geq xres_virtual \times yres_virtual \times bpp$$

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

vramn_size 和叠加图像层对应关系如表 2-1 所示。

表2-1 vramn_size 和叠加图像层对应关系

叠加图像层	字符串	含义
叠加图像层 0	vram0_size	叠加图像层 0 的物理显存大小，单位为字节。
叠加图像层 1	vram1_size	叠加图像层 1 的物理显存大小，单位为字节。
硬件鼠标层	vram2_size	硬件鼠标层的物理显存大小，单位为字节。

如果加载 HiFB 驱动时不带任何参数，则系统默认管理一个叠加图像层：叠加图像层 0，并为其分配 829440 Byte 的显存。

用户需要从全局的角度出发配置 HiFB 需要管理的叠加图像层，并为每个叠加图像层分配适当的显存。



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。

当分配多个图形层时, 需要注意 FB 系统设备号与图形层之间的对应关系。由于 HiFB 是一个子设备号对应一个叠加图像层, 即一个 FB 图形层对应一个系统设备号。由于系统设备号由 Linux 系统分配, 且是递增的, 故图形层对应的设备号是与 FB 加载时参数配置相关的。如 FB 的加载参数如下:

```
video="hifb:vram0_size:xxx, vram2_size:xxx, vram3_size:xxx"
```

则图形层与设备号的对应关系如下:

- 图像层 0 对应系统设备号 0, 操作图形层 0 需要打开设备 fb0。
- 图像层 2 对应系统设备号 1, 操作图形层 2 需要打开设备 fb1。
- 图像层 3 对应系统设备号 2, 操作图形层 3 需要打开设备 fb2。

2.3 配置举例

配置 HiFB 管理叠加图像层的示例如下:

说明

HiFB 驱动模块文件为 hifb.ko。

- 配置 HiFB 管理一个叠加图像层。

如果只需要 HiFB 管理叠加图像层 0, 且最大虚拟分辨率为 720×576 , 用到的像素格式为 ARGB1555, 则叠加图像层 0 需要的最小显存为 $720 \times 576 \times 2 = 829440$, 配置参数如下:

```
insmod hifb.ko video="hifb:vram0_size:829440"
```

- 配置 HiFB 管理多个叠加图像层。

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

```
insmod hifb.ko video="hifb:vram0_size:829440, vram1_size: 829440"
```

2.4 异常情况

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



3 第一次开发应用

关于本章

本章描述内容如下表所示。

标题	内容
3.1 开发流程	介绍 HiFB 的开发流程。
3.2 实例介绍	介绍利用 PAN_DISPLAY 动态显示图片的实例。

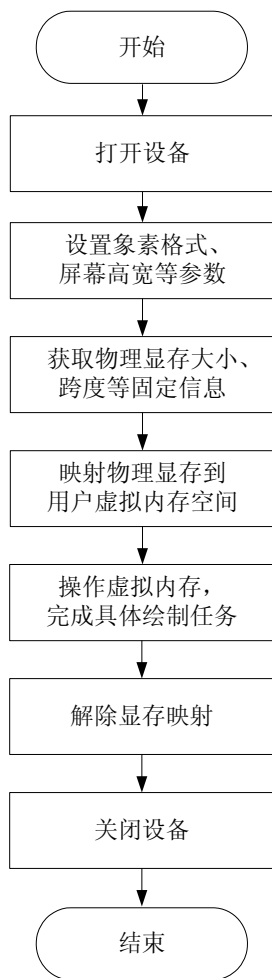


3.1 开发流程

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。

图3-1 HiFB 的开发流程



HiFB 的开发步骤如下：

- 步骤 1 调用 open 函数打开指定的 HiFB 设备。
- 步骤 2 调用 ioctl 函数设置 HiFB 的像素格式以及屏幕高宽等参数（详细内容请参见《HiFB API 参考》）。
- 步骤 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。
- 步骤 4 调用 mmap 函数将物理显存映射到虚拟内存空间。



步骤 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。

步骤 6 调用 munmap 解除显存映射。

步骤 7 调用 close 函数关闭设备。

----结束



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像素格式为 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};

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;
    char image_name[128];

    /*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)
        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);
        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 Framebuffer device*/
close(fd);

return 0;
}
```