HiFB

API Reference

Issue	**06**
Date	**2016-11-17**

Trademarks and Permissions

, HISILICON, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

HiSilicon Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://www.hisilicon.com

Email: support@hisilicon.com

About This Document

Purpose

This document describes the application programming interfaces (APIs), data types, and proc debugging information about the HiSilicon frame buffer (HiFB).

NOTE

This document uses the Hi3516A as an example. Unless otherwise specified, the contents of the Hi3516A are consistent with those of the Hi3516D, Hi3518E V200/Hi3518E V201, and Hi3516C V200.

Unless otherwise specified, the contents of Hi3516C V200 are consistent with those of Hi3518E V200/Hi3518E V201.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3516A	V100
Hi3516D	V100
Hi3518E	V200
Hi3518E	V201
Hi3516C	V200
Hi3519	V100
Hi3519	V101

Intended Audience

This document is intended for:

- Technical support engineers
- Software development engineers

Change History

Changes between document issues are cumulative. Therefore, the latest document issue contains all changes made in previous issues.

Issue 06 (2016-11-17)

This issue is the sixth official release, which incorporates the following changes:

In section 3.2, HIFB_LAYER_INFO_S is modified.

Issue 05 (2016-05-15)

This issue is the fifth official release, which incorporates the following changes:

The contents related to Hi3519 V101 are added.

Issue 04 (2015-12-18)

This issue is the fourth official release, which incorporates the following changes:

Section 5.1 and section 5.2 are modified.

Issue 03 (2015-08-20)

This issue is the third official release, which incorporates the following changes:

The contents related to Hi3519 V100 are added.

Issue 02 (2015-06-23)

This issue is the second official release, which incorporates the following changes:

The contents related to Hi3518E V200, Hi3518E V201 and Hi3516C V200 are added.

In section 2.4.1, FBIOGET_MIRROR_MODE and FBIOPUT_MIRROR_MODE are added.

Section 2.4.2 is modified.

In section 3.2, HIFB_MIRROR_MODE_E is added.

Section 5.2 is modified.

Issue 01 (2014-12-20)

This issue is the first official release, which incorporates the following changes:

The contents related to the Hi3516D are added.

Issue 00B01 (2014-09-14)

This issue is the first draft release.

Contents

Figures

Tables

1 Overview

1.1 Description

As a module of the HiSilicon digital media processing platform (MPP), the HiSilicon frame buffer (HiFB) is used to manage the graphics layers. The HiFB is developed based on the Linux frame buffer. Besides the basic functions provided by the Linux frame buffer, the HiFB also provides extended functions for controlling graphics layers such as the interlayer alpha, origin setting, and extended FB mode.

1.2 Reference Fields

1.2.1 API Reference Fields

The API reference information is described in the following nine fields as shown in Table 1-1.

Table 1-1 API Reference Fields

Reference Field	Description
Purpose	Describes the major function of an API.
Syntax	Lists the header files that must be included when an API is called and the API prototype.
Parameter	Lists the parameters of an API and the related information.
Description	Describes the working process of an API.
Return Value	Lists the return values of an API and the related information.
Request	Lists the required header files and library files when an API is called.
Note	Lists the precautions when an API is called.
Example	Lists the examples of calling an API.
See Also	Lists the related APIs.

1.2.2 Data Type Reference Fields

Data types are described in the following five reference fields as shown in Table 1-2.

Table 1-2 Data type reference fields

Reference Field	Description
Description	Describes the major function of a data type.
Definition	Provides the definition of a data type.
Member	Lists the members of a data type and the related information.
Note	Lists the matters that you need to pay attention to when using a data type.
See Also	Lists the related data types and APIs.

2 API Reference

2.1 API Types

The HiFB APIs are classified into the following four types:

- File operation APIs

  The file operation APIs provide the HiFB operation interfaces. By calling the APIs, you can regard overlay layers as files. The APIs are standard interfaces provided by the Linux, including open, close, write, read, and seek. The standard APIs are not described in the document.

- Display buffer mapping APIs

  The display buffer mapping APIs provide interfaces used to map the display buffer to the user virtual memory. The APIs are standard interfaces provided by the Linux, such as mmap and munmap. The standard APIs are not described in the document.

- Display buffer control and state querying APIs

  The display buffer control and state querying APIs provide interfaces used to configure attributes such as the pixel format and the color depth. The APIs are standard interfaces provided by the Linux and are frequently used. These APIs are briefly described in this document.

- Inter-layer effect control and state querying APIs

  The HiFB can manage multiple graphics layers. The alpha and origin of each layer can be configured. The APIs are newly added based on those provided by the Linux frame buffer. The document describes the APIs in detail.

2.2 ioctl Function

The HiFB user state interface is presented in ioctl format as follows:

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

The function is the Linux standard interface with the attribute of variable parameters. For the HiFB, only three parameters are needed. Therefore, the syntax format is:

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

The change of the parameter cmd leads to the change of CMD_DATA_TYPE. Table 2-1 describes the three parameters.

Table 2-1 Three parameters of the ioctl function

Parameter	Description	Input/Output
fd	File descriptor (FD) of a frame buffer (FB). The return value of the function used to open the frame buffer device.	Input
cmd	Major commands are as follows: • FBIOGET_VSCREENINFO: Obtains the screen variable information. • FBIOPUT_VSCREENINFO: Sets the screen variable information. • FBIOGET_FSCREENINFO: Obtains the screen fixed information. • FBIOPAN_DISPLAY: Sets the PAN display. • FBIOGET_CAPABILITY_HIFB: Obtains the capability of an overlay layer. • FBIOGET_SCREEN_ORIGIN_HIFB: Obtains the origin of an overlay layer. • FBIOPUT_SCREEN_ORIGIN_HIFB: Sets the origin of an overlay layer. • FBIOGET_SHOW_HIFB: Obtains the display state of an overlay layer. • FBIOPUT_SHOW_HIFB: Sets the display state of an overlay layer. • FBIOGET_ALPHA_HIFB: Obtains the alpha of an overlay layer. • FBIOPUT_ALPHA_HIFB: Sets the alpha of an overlay layer. • FBIOGET_COLORKEY_HIFB: Obtains the colorkey attribute of an overlay layer. • FBIOPUT_COLORKEY_HIFB: Sets the colorkey attribute of an overlay layer. • FBIOGET_MDDRDETECT_HIFB: Obtains DDR detection attributes. • FBIOPUT_MDDRDETECT_HIFB: Sets DDRdetection attributes. • Operations related to the software cursor	Input

Parameter	Description	Input/Output
cmddata	The data types corresponding to different commands are as follows: • Obtains or sets the screen variable information: struct fb_var_screeninfo * type. • Obtains the screen fixed information: struct fb_fix_screeninfo * type. • Set the PAN display: struct fb_var_screeninfo * type. • Obtains the capability of an overlay layer: HIFB_CAPABILITY_S * type. • Obtains or sets the origin of a screen overlay layer: HIFB_POINT_S * type. • Obtains or sets the display state of an overlay layer: HI_BOOL * type. • Obtains or sets the alpha value of an overlay layer: • HIFB_ALPHA_S * type. • Obtains or sets the DDR detection attributes: HIFB_DDRZONE_S * type.	Input or output

2.3 Standard APIs

FBIOGET_VSCREENINFO

[Purpose]

To obtain the screen variable information.

[Syntax]

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

[Description]

This API is used to obtain the screen variable information, such as the resolution and the pixel format. For details, see section 3.1 "struct fb_var_screeninfo."

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOGET_VSCREENINFO	ioctl serial number	Input
var	Pointer to the variable information structure	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: fb.h.

[Note]

For the HD device, the default resolution of the graphics layer is 1280 x 720, and the default resolution of the cursor layer is 128 x 128. For the SD device, the default resolution of the graphics layer is 720 x 576, and the default pixel format is ARGB1555.

[Example]

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

[See Also]

FBIOPUT_VSCREENINFO.

FBIOPUT_VSCREENINFO

[Purpose]

To set the screen resolution and the pixel format of the frame buffer.

[Syntax]

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

[Description]

This API is used to set the screen resolution and the pixel format.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOPUT_VSCREENINFO	ioctl serial number	Input

Parameter	Description	Input/Output
var	Pointer to the variable information structure	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Difference]

Chip	Description
Hi3516A/Hi3519 V100/Hi3519 V101	Only the ARGB1555, ARGB4444 and ARGB8888 pixel formats are supported.
Hi3518E V200	Only the ARGB1555 and ARGB4444 pixel formats are supported.

[Requirement]

Header file: fb.h.

[Note]

- The resolution value must be within the range supported by the overlay layer. The maximum and the minimum resolutions supported by each overlay layer can be obtained through FBIOGET_CAPABILITY_HIFB.
- Ensure that the sum of the actual resolution and the offset is within the range of the virtual resolution; otherwise, the system automatically adjusts the actual resolution to a value that is within the range of the virtual resolution.
- For the interlaced display device, the height in the resolution must be an even number.
- When the compression function is enabled, you must disable before changing the actual resolution.

[Example]

In the following example, the actual resolution is 720x576, the virtual resolution is 720x576, the offset is (0, 0), and the pixel format is ARGB1555.

```
struct fb_bitfield r16 = {10, 5, 0};
struct fb_bitfield g16 = {5, 5, 0};
struct fb_bitfield b16 = {0, 5, 0};
struct fb_bitfield a16 = {15, 1, 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 = 16;
vinfo.xoffset = 0;
vinfo.yoffset = 0;
vinfo.red = r16;
vinfo.green = g16;
vinfo.blue = b16;
vinfo.transp= a16;
if (ioctl(fd, FBIOPUT_VSCREENINFO, &vinfo) < 0)
{
    return -1;
}
```

[See Also]

FBIOGET_VSCREENINFO.

FBIOGET_FSCREENINFO

[Purpose]

To obtain the fixed information of the frame buffer.

[Syntax]

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

[Description]

This API is used to obtain the frame buffer fixed information, such as the start position, size and stride of the display buffer. For details, see section 3.1 "struct fb_fix_screeninfo."

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOGET_FSCREENINFO	ioctl serial number	Input
fix	Pointer to the fixed information structure	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: fb.h.

[Note]

None

[Example]

None

[See Also]

None

FBIOPAN_DISPLAY

[Purpose]

To display an image from a different offset position of the virtual resolution.

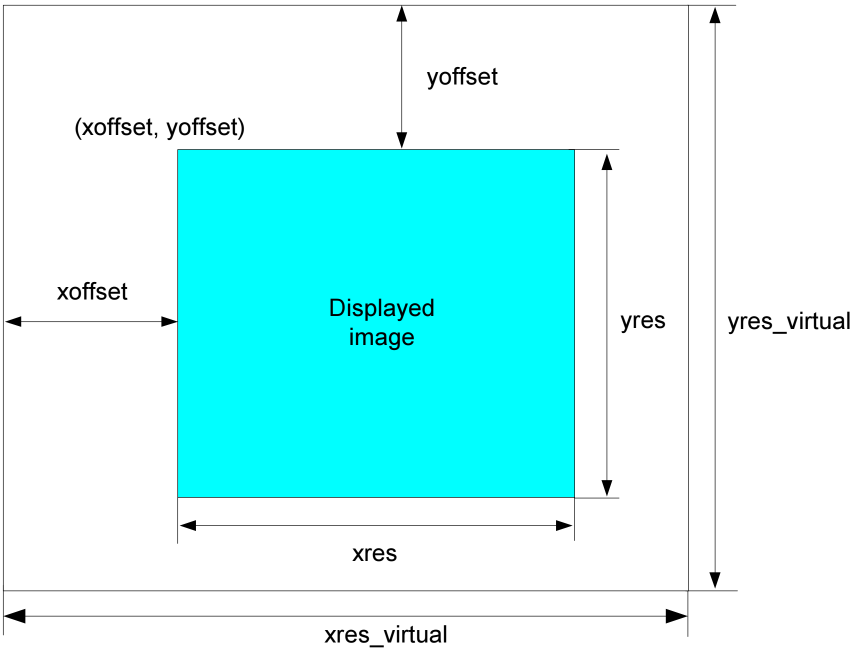
[Syntax]

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

[Description]

This API is used to display an image from a different offset position of the virtual resolution. The actual resolution is not changed. As shown in Figure 2-1, (xres_virtual, yres_virtual) is the virtual resolution; (xres, yres) is the actual resolution; (xoffset, yoffset) is the offset.

Figure 2-1 Display image from a different offset position of the virtual resolution



[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOPAN_DISPLAY	ioctl serial number	Input
var	Pointer to the variable information structure	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: fb.h.

[Note]

- Use this API only in standard FB mode. This API can be used to change the FB mode from extended mode to standard mode.
- The sum of the actual resolution and offset value must be within the range of the virtual resolution. Otherwise, the setting is invalid. In addition, ensure that the offset address defined by xoffset and yoffset is 16-byte aligned. Otherwise, the value of xoffset is

decreased until the offset address is 16-byte aligned.

- For the interlaced display device, the height in the resolution must be an even number.

[Example]

In the following example, the actual resolution is 300x300; the virtual resolution is 720x576; the initial offset is (50, 50); the image is displayed from offset position (300, 0).

```
struct fb_bitfield r32 = {16, 8, 0};
struct fb_bitfield g32 = {8, 8, 0};
struct fb_bitfield b32 = {0, 8, 0};
struct fb_bitfield a32 = {24, 8, 0};
struct fb_var_screeninfo vinfo;

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;
}
```

[See Also]

None

2.4 Extended APIs

2.4.1 Common APIs

FBIOGET_CAPABILITY_HIFB

[Purpose]

To obtain the capability of an overlay layer.

[Syntax]

```
int ioctl (int fd,
           FBIOGET_CAPABILITY_HIFB,
           HIFB_CAPABILITY_S *pstCap);
```

[Description]

Before using an API, you can query whether the API is supported by an overlay layer by calling FBIOGET_CAPABILITY_HIFB.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOGET_CAPABILITY_HIFB	ioctl serial number	Input
pstCap	Pointer to the capability structure	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h.

[Note]

None

[Example]

None

[See Also]

None

FBIOGET_SCREEN_ORIGIN_HIFB

[Purpose]

To obtain the origin of an overlay layer on the screen.

[Syntax]

```
int ioctl (int fd,
           FBIOGET_SCREEN_ORIGIN_HIFB,
           HIFB_POINT_S *pstPoint);
```

[Description]

This API is used to obtain the origin of an overlay layer on the screen.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOGET_SCREEN_ORIGIN_HIFB	ioctl serial number	Input
pstPoint	Pointer to the origin structure	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h.

[Note]

None

[Example]

This API is not applicable to the software cursor.

[See Also]

FBIOPUT_SCREEN_ORIGIN_HIFB

FBIOPUT_SCREEN_ORIGIN_HIFB

[Purpose]

To set the origin of an overlay layer on the screen.

[Syntax]

```
int ioctl (int fd,
           FBIOPUT_SCREEN_ORIGIN_HIFB,
           HIFB_POINT_S *pstPoint);
```

[Description]

This API is used to set the origin of an overlay layer on the screen. The coordinates of the origin range from (0, 0) to the supported maximum resolution.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOPUT_SCREEN_ORIGIN_HIFB	ioctl serial number	Input
pstPoint	Pointer to the origin structure	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h.

[Note]

- If the origin of an overlay layer is not within the specified range, the system configures the origin to (u32MaxWidth, u32MaxHeight) by default.
- For the interlaced device, the vertical coordinate of the origin must be an even.

[Example]

None

[See Also]

FBIOGET_SCREEN_ORIGIN_HIFB

FBIOGET_SHOW_HIFB

[Purpose]

To obtain the display state of an overlay layer.

[Syntax]

```
int ioctl (int fd,
           FBIOGET_SHOW_HIFB,
           HI_BOOL *bShow);
```

[Description]

This API is used to obtain the display state of an overlay layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOGET_SHOW_HIFB	ioctl serial number	Input
bShow	Point to the state of the current overlay layer: • *bShow = HI_TRUE: The current overlay layer is displayed. • *bShow = HI_FALSE: The current overlay layer is hidden.	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h.

[Note]

This API is not applicable to the software cursor.

[Example]

None

[See Also]

FBIOPUT_SHOW_HIFB

FBIOPUT_SHOW_HIFB

[Purpose]

To display or hide an overlay layer.

[Syntax]

```
int ioctl (int fd,
           FBIOPUT_SHOW_HIFB,
           HI_BOOL *bShow);
```

[Description]

This API is used to set the state of an overlay layer, namely, displayed or hidden.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOPUT_SHOW_HIFB	ioctl serial number	Input
bShow	Display state of an overlay layer ● *bShow = HI_TRUE: The current overlay layer is displayed. ● *bShow = HI_FALSE: The current overlay layer is hidden.	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h.

[Note]

- Before displaying pictures, you must set bShow to HI_TRUE by calling ioctl(fd, FBIOPUT_SHOW_HIFB, &bShow) to enable the corresponding graphics layer. This ensures that pictures are displayed properly.
- The resolution of the graphics layer cannot be greater than the resolution of the display device.
- Ensure that the display device supports the resolution of the picture to be displayed.

[Example]

None

[See Also]

FBIOGET_SHOW_HIFB

FBIOGET_MIRROR_MODE

[Purpose]

To obtain the mirror mode of the current overlay layer.

[Syntax]

```
int ioctl (int fd,
           FBIOGET_MIRROR_MODE,
            HIFB_MIRROR_MODE_E *eMirrorMode);
```

[Description]

This API is used to obtain the mirror mode of the current overlay layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOGET_MIRROR_MODE	ioctl serial number	Input
eMirrorMode	Mirror mode of the current overlay layer	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

This API applies only to the extended mode and does not apply to the software cursor.

[Example]

None

[See Also]

FBIOPUT_MIRROR_MODE

FBIOPUT_MIRROR_MODE

[Purpose]

To set the mirror mode of the current overlay layer.

[Syntax]

```
int ioctl (int fd,
           FBIOPUT_MIRROR_MODE,
           HIFB_MIRROR_MODE_E *eMirrorMode);
```

[Description]

This API is used to set the mirror mode of the current overlay layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOGET_MIRROR_MODE	ioctl serial number	Input
eMirrorMode	Mirror mode of the current overlay layer	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

This API applies only to the extended mode and does not apply to the software cursor.

[Example]

None

[See Also]

FBIOGET_MIRROR_MODE

FBIOGET_ALPHA_HIFB

[Purpose]

To obtain the alpha of an overlay layer.

[Syntax]

```
int ioctl (int fd,
           FBIOGET_ALPHA_HIFB,
           HIFB_ALPHA_S *pstAlpha);
```

[Description]

This API is used to obtain the alpha of the current overlay layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOGET_ALPHA_HIFB	ioctl serial number	Input

Parameter	Description	Input/Output
pstAlpha	Pointer to the alpha structure	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h.

[Note]

For details, see the description of

HIFB_ALPHA_S.

[Example]

None

[See Also]

FBIOPUT_ALPHA_HIFB

FBIOPUT_ALPHA_HIFB

[Purpose]

To set the alpha of an overlay layer.

[Syntax]

```
int ioctl (int fd,
           FBIOPUT_ALPHA_HIFB,
           HIFB_ALPHA_S *pstAlpha);
```

[Description]

This API is used to set the alpha of an overlay layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB	Input
FBIOPUT_ALPHA_HIFB	ioctl serial number	Input
pstAlpha	Pointer to the alpha structure	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h.

[Note]

For details, see the description of

HIFB_ALPHA_S.

[Example]

None

[See Also]

FBIOGET_ALPHA_HIFB

FBIOGET_COLORKEY_HIFB

[Purpose]

To obtain the colorkey of an overlay layer.

[Syntax]

```
int ioctl (int fd,
           FBIOGET_COLORKEY_HIFB,
           HIFB_COLORKEY_S *pstColorKey);
```

[Description]

This API is used to obtain the colorkey of an overlay layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOGET_COLORKEY_HIFB	ioctl number	Input
pstColorKey	Pointer to the colorkey data type	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100/Hi3519 V101	Only one graphics layer that supports colorkey is provided.

[Requirement]

Header file: hifb.h

[Note]

None

[Example]

None

[See Also]

FBIOPUT_COLORKEY_HIFB

FBIOPUT_COLORKEY_HIFB

[Purpose]

To set the colorkey of an overlay layer.

[Syntax]

```
int ioctl (int fd,
           FBIOPUT_COLORKEY_HIFB,
           HIFB_COLORKEY_S *pstColorKey);
```

[Description]

This API is used to set the colorkey of the current overlay layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOPUT_COLORKEY_HIFB	ioctl number	Input
pstColorKey	Pointer to the colorkey data type	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100/Hi3519 V101	Only one graphics layer that supports colorkey is provided.

[Requirement]

Header file: hifb.h

[Note]

None

[Example]

Assume that the pixel format is ARGB8888. If you want to filter the color values whose R component is 0x1F, G component is 0x2F, and B component is 0x3F, use the following settings:

```
HIFB_COLORKEY_S stColorKey;

stColorKey.bKeyEnable = HI_TRUE;
stColorKey.u32Key = 0x1F2F3F;
if (ioctl(fd, FBIOPUT_COLORKEY_HIFB, &stColorKey) < 0)
{
   return -1;
}
```

[See Also]

FBIOGET_COLORKEY_HIFB

FBIOGET_DEFLICKER_HIFB

[Purpose]

To obtain the anti-flicker setting of an overlay layer.

[Syntax]

```
int ioctl (int fd,
         FBIOGET_DEFLICKER_HIFB,
         HIFB_DEFLICKER_S *pstDeflicker);
```

[Description]

This API is used to obtain the anti-flicker setting of the current overlay layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOGET_DEFLICKER_HIFB	ioctl number	Input
pstDeflicker	Pointer to the anti-flicker data type	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: hifb.h

[Note]

The Hi35xx does not support anti-flicker. If you call this API on the Hi35*xx*, an error code indicating failure is returned.

[Example]

None

[See Also]

FBIOPUT_DEFLICKER_HIFB

FBIOPUT_DEFLICKER_HIFB

[Purpose]

To set the anti-flicker functions of an overlay layer.

[Syntax]

```
int ioctl (int fd,
         FBIOPUT_DEFLICKER_HIFB,
         HIFB_DEFLICKER_S *pstDeflicker);
```

[Description]

This API is used to set the anti-flicker of the current overlay layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOPUT_DEFLICKER_HIFB	ioctl number	Input
pstDeflicker	Pointer to the anti-flicker data type	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

The Hi35xx does not support anti-flicker. If you call this API on the Hi35xx, an error code indicating failure is returned.

[Example]

None

[See Also]

FBIOGET_DEFLICKER_HIFB

FBIOGET_VBLANK_HIFB

[Purpose]

To wait for the vertical blanking region of an overlay layer. To operate the display buffer without tearing, you are advised to operate it in the vertical blanking region.

[Syntax]

```
int ioctl (int fd, FBIOGET_VBLANK_HIFB);
```

[Description]

This API is used to obtain the blanking region of the current overlay layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input

Parameter	Description	Input/Output
FBIOGET_VBLANK_HIFB	ioctl number	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

In general, the vertical blanking interval is about dozens of milliseconds. The operation time must be short to ensure that the operation is complete before the end of the vertical blanking region.

[Example]

None

[See Also]

None

FBIOFLIP_SURFACE

[Purpose]

To display multiple surfaces in turn and set the alpha and colorkey attributes.

[Syntax]

```
int ioctl (int fd,
           FBIOFLIP_SURFACE,
           HIFB_SURFACE_S *pstSurface);
```

[Description]

The API is the extended interface of FBIOPAN_DISPLAY and is used to display multiple surfaces and set the alpha and colorkey at the same time.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOFLIP_SURFACE	ioctl number	Input
pstSurface	Pointer to the surface structure	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100/Hi3519 V101	Only one graphics layer that supports colorkey is provided.

[Requirement]

Header file: hifb.h

[Note]

- Use this API only in standard FB mode. This API can be used to change the FB mode from extended mode to standard mode.
- The surface physical address must be within the address range of the display buffer configured at the overlay layer. In addition, the surface physical address must be 16-byte aligned. Otherwise, there is offset between the actual display position and the configured display position.

[Example]

None

[See Also]

FBIOPAN_DISPLAY

FBIOPUT_COMPRESSION_HIFB

[Purpose]

To enable the compression function for an overlay layer.

[Syntax]

```
int ioctl (int fd,
           FBIOPUT_COMPRESSION_HIFB,
           HI_BOOL *pbCompress);
```

[Description]

This API is used to enable the compression function for an overlay layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOPUT_COMPRESSION_HIFB	ioctl number	Input
pbCompress	Pointer to the compression enable identifier	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100/Hi3519 V101	Compression is not supported.

[Requirement]

Header file: hifb.h

[Note]

- If no DDR detection zone is set when the compression function is enabled, the drawn contents are displayed only after corresponding refresh operations are performed. In standard FB mode, the refresh operation APIs include FBIOPAN_DISPLAY and FBIOFLIP_SURFACE; in extended FB mode, the refresh operation API is FBIO_REFRESH. When the origin coordinates are changed by calling FBIOPUT_SCREEN_ORIGIN_HIFB, a refresh operation is performed.
- When DDR detection zones are set, you do not need to invoke refresh operations because the refresh operations are triggered by the DDR detection function. Note that DDR detection takes effect only in non-buffer mode or standard mode.
- Only the ARGB8888 picture can be compressed.
- This API is not applicable to the cursor layer. If the compression function is enabled, the software cursor is not recommended.
- The compression function is disabled by default.

[Example]

None

[See Also]

FBIOGET_COMPRESSION_HIFB

FBIOGET_COMPRESSION_HIFB

[Purpose]

To obtain the compression function status of an overlay layer.

[Syntax]

```
int ioctl (int fd,
          FBIOGET_COMPRESSION_HIFB,
          HI_BOOL *pbCompress);
```

[Description]

This API is used to obtain the compression function status of an overlay layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOGET_COMPRESSION_HIFB	ioctl number	Input
pbCompress	Pointer to the obtained compression status	None

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100/Hi3519 V101	Compression is not supported.

[Requirement]

Header file: hifb.h

[Note]

None

[Example]

None

[See Also]

FBIOPUT_COMPRESSION_HIFB

FBIOPUT_MDDRDETECT_HIFB

[Purpose]

To set the DDR detection attributes of a graphics layer.

[Syntax]

```
int ioctl (int fd,
           FBIOPUT_MDDRDETECT_HIFB,
           HIFB_DDRZONE_S *stDdrZone);
```

[Description]

This API is used to set the DDR detection attributes of a graphics layer.

[Parameter]

Parameter	Description	Input/Output
Fd	FB FD	Input
FBIOPUT_MDDRDETECT_HIFB	ioctl number	Input
stDdrZone	Pointer to DDR detection attributes	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100/Hi3519 V101	DDR detection is not supported.

[Requirement]

Header file: hifb.h

[Note]

- DDR detection takes effect only when the mode is non-buffer mode or standard mode and the compression function is enabled.

- When the compression function is enabled, DDR detection is enabled by default. A maximum of 32 DDR detection zones are supported. G0 occupies zones 0–15, and G1 occupies zones 16–31 by default.
- Based on the number of DDR detection zones, the display buffer is divided by pixel for DDR detection.
- If the number of DDR detection zones is set to **0**, DDR detection is disabled.

[Example]

None

[See Also]

FBIOGET_COMPRESSION_HIFB

FBIOGET_MDDRDETECT_HIFB

[Purpose]

To obtain the DDR detection status of a graphics layer.

[Syntax]

```
int ioctl (int fd,
            FBIOGET_MDDRDETECT_HIFB,
            HIFB_DDRZONE_S *stDdrZone);
```

[Description]

This API is used to obtain the DDR detection attributes of a graphics layer.

[Parameter]

Parameter	Description	Input/Output
fd	FB FD	Input
FBIOGET_MDDRDETECT_HIFB	ioctl number	Input
stDdrZone	Pointer to the obtained DDR detection status	None

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Difference]

Chip	Description
Hi3516A/Hi3518E	DDR detection is not supported.

Chip	Description
V200/Hi3519 V100/Hi3519 V101	

[Requirement]

Header file: hifb.h.

[Note]

None

[Example]

None

[See Also]

FBIOGET_MDDRDETECT_HIFB

FBIOPUT_LAYER_INFO

[Purpose]

To set the layer information. This API is used to switch the mode between the standard FB mode and extended FB mode and set the refresh information in extended mode.

[Syntax]

```
int ioctl (int fd,
            FBIOPUT_LAYER_INFO,
            HIFB_LAYER_INFO_S* pstLayerInfo);
```

[Description]

This API is used to set the layer information, including the refresh mode, anti-flicker level, position of the start point of the screen, canvas resolution, display buffer resolution, screen display resolution, and pre-multiply enable. For details, see the descriptions of HIFB_LAYER_INFO_S and HIFB_LAYER_BUF_E.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOPUT_LAYER_INFO	ioctl number	Input
pstLayerInfo	Pointer to the data type of the layer information	Input

[Return Value]

Return Value	Description

Return Value	Description
0	Success
–1	Failure

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100/Hi3519 V101	• The TDE supports picture scaling. In extended mode, the canvas resolution can differ from the display buffer resolution. • All graphics layers do not support premultiplication.

[Requirement]

Header file: hifb.h

[Note]

- After setting the attribute of an item, you must set the corresponding mask by setting the u32Mask parameter of pstLayerInfo. Otherwise, the setting does not take effect.
- The Hi3516A/Hi3518E V200/Hi3519 V100/Hi3519 V101 does not support layer scaling. When the display buffer resolution or the screen display resolution changes, the actual display resolution also changes. The display buffer resolution or the screen display resolution must be less than or equal to the device resolution.
- For the interlaced display device, the heights in the display buffer resolution and screen display resolution must be even numbers.

[Example]

```
HIFB_LAYER_INFO_S stLayerInfo = {0};
stLayerInfo.BufMode = HIFB_LAYER_BUF_ONE;
stLayerInfo.u32Mask = HIFB_LAYERMASK_BUFMODE;
stLayerInfo.u32DisplayWidth = 360;
stLayerInfo.u32DisplayHeight = 320;
stLayerInfo.s32XPos = 16;
stLayerInfo.s32YPos = 16;
stLayerInfo.u32Mask |= HIFB_LAYERMASK_DISPSIZE | HIFB_LAYERMASK_POS;
s32Ret = ioctl(s32Fd, FBIOPUT_LAYER_INFO, &stLayerInfo);
```

[See Also]

None

FBIOGET_LAYER_INFO

[Purpose]

To obtain the layer information.

[Syntax]

```
int ioctl (int fd,
           FBIOGET_LAYER_INFO
           HIFB_LAYER_INFO_S* pstLayerInfo);
```

[Description]

This API is used to obtain the layer information, including the refresh mode, anti-flicker level, position of the start point of the screen, canvas resolution, display buffer resolution, screen display resolution, and pre-multiply enable.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOGET_LAYER_INFO	ioctl number	Input
pstLayerInfo	Pointer to the data type of the layer information	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

None

[Example]

None

[See Also]

None

FBIOGET_CANVAS_BUFFER

[Purpose]

To obtain the canvas information.

[Syntax]

```
int ioctl (int fd,
        FBIOGET_CANVAS_BUFFER,
        HIFB_BUFFER_S *pstCanvasBuf)
```

[Description]

This API is used to obtain the canvas information.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOGET_CANVAS_BUFFER	ioctl number	Input
pstCanvasBuf	Pointer to the data type of the canvas information	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

This API is not applicable to the software cursor.

[Example]

None

[See Also]

None

FBIO_REFRESH

[Purpose]

To refresh the displayed contents in extended mode.

[Syntax]

```
int ioctl (int fd,
        FBIO_REFRESH,
        HIFB_BUFFER_S* pstBufInfo);
```

[Description]

This API is used to start a refresh operation in extended mode.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIO_REFRESH	ioctl number	Input
pstBufInfo	Pointer to the HIFB_BUFFER_S data type	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

This API applies only to the extended mode and does not apply to the software cursor.

[Example]

None

[See Also]

None

FBIO_WAITFOR_FREFRESH_DONE

[Purpose]

To wait for the completion of the started refresh operation, that is, to wait for the display of the refreshed contents in extended mode.

[Syntax]

```
int ioctl (int fd, FBIO_WAITFOR_FREFRESH_DONE)
```

[Description]

This API is used to wait for the completion of a refresh operation.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input

Parameter	Description	Input/Output
FBIO_WAITFOR_FREFRESH_DONE	ioctl number	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

This API applies only to the extended mode and does not apply to the software cursor.

[Example]

None

[See Also]

None

2.4.2 Software Cursor

The APIs described in this section are available only when the software cursor function is enabled. To enable the software cursor function, set **softcursor** to **on** when loading **hifb.ko**. After the software cursor function is enabled and the **/dev/fb0** file is opened by calling the open function, you can call the following APIs to perform the operations related to the software cursor. You are advised to call only the following APIs to use the software cursor.

FBIOPUT_CURSOR_INFO

[Purpose]

To set the information about the cursor layer.

[Syntax]

```
int ioctl (int fd, FBIOPUT_CURSOR_INFO, HIFB_CURSOR_S *pstCursor)
```

[Description]

This API is used to set the information about the cursor layer, including the start address, size, stride, and pixel format of the canvas.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input

Parameter	Description	Input/Output
FBIOPUT_CURSOR_INFO	ioctl number	Input
pstCursor	Information about the software cursor layer	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

- The width or height range of the software cursor is (0, 128].
- The horizontal and vertical coordinates of the hot spot of the software cursor must be greater than or equal to 0 and must be less than or equal to the width and height of the cursor bitmap.

[Example]

None

[See Also]

None

FBIOGET_CURSOR_INFO

[Purpose]

To obtain the information about the cursor layer.

[Syntax]

```
int ioctl (int fd, FBIOGET_CURSOR_INFO, HIFB_CURSOR_S *pstCursor)
```

[Description]

This API is used to obtain the information about the cursor layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOGET_CURSOR_INFO	ioctl number	Input

Parameter	Description	Input/Output
pstCursor	Information about the software cursor layer	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

None

[Example]

None

[See Also]

None

FBIOPUT_CURSOR_ATTCHCURSOR

[Purpose]

To bind the software cursor to a graphics layer.

[Syntax]

```
int ioctl (int fd,
            FBIOPUT_CURSOR_ATTCHCURSOR,
            HI_U32 *pu32LayerId)
```

[Description]

After the software cursor is bound to a graphics layer, the contents of the software cursor are displayed on the graphics layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOPUT_CURSOR_ATTCHCURSOR	ioctl number	Input
pu32LayerId	Identifier of the graphics layer to be bound	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

- Before this API is called, the graphics layer to be bound must be opened.
- A cursor can be bound to a graphics layer for multiple times, but multiple cursors cannot be bound to a graphics layer at the same time. If a graphics layer is bound to a cursor, you must unbind the graphics layer before binding it to another cursor. Otherwise, an error occurs.
- You must set the information about the cursor layer before binding the software cursor to a graphics layer. You cannot bind the software cursor to other cursor layers.

[Example]

None

[See Also]

None

FBIOPUT_CURSOR_DETACHCURSOR

[Purpose]

To unbind the software cursor from a graphics layer.

[Syntax]

```
int ioctl (int fd,
          FBIOPUT_CURSOR_DETACHCURSOR,
          HI_U32 *pu32LayerId)
```

[Description]

After the software cursor is unbound from a graphics layer, the contents of the software cursor are not displayed.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOPUT_CURSOR_DETACHCURSOR	ioctl number	Input

Parameter	Description	Input/Output
pu32LayerId	Identifier of a graphics layer	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

None

[Example]

None

[See Also]

None

FBIOPUT_CURSOR_STATE

[Purpose]

To set the display status of the software cursor.

[Syntax]

```
int ioctl (int fd, FBIOPUT_CURSOR_STATE, HI_BOOL *pbShow)
```

[Description]

This API is used to set the display status of the software cursor.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOPUT_CURSOR_STATE	ioctl number	Input
pbShow	Pointer to the display status	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

After the software cursor is bound, it is hidden by default. The software cursor is displayed only when you call this API to set the display status.

[Example]

None

[See Also]

None

FBIOGET_CURSOR_STATE

[Purpose]

To obtain the display status of the software cursor.

[Syntax]

```
int ioctl (int fd, FBIOGET_CURSOR_STATE, HI_BOOL *pbShow)
```

[Description]

This API is used to obtain the display status of the software cursor.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOPUT_CURSOR_STATE	ioctl number	Input
pbShow	Pointer to the display status	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

The software cursor is hidden by default.

[Example]

None

[See Also]

None

FBIOPUT_CURSOR_POS

[Purpose]

To set the display position of the software cursor at the bound graphics layer.

[Syntax]

```
int ioctl (int fd, FBIOPUT_CURSOR_POS, HIFB_POINT_S *pstPos)
```

[Description]

This API is used to set the display position of the software cursor at the bound graphics layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOPUT_CURSOR_POS	ioctl number	Input
pstPos	Information about the display position	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

None

[Example]

None

[See Also]

None

FBIOGET_CURSOR_POS

[Purpose]

To obtain the display position of the software cursor at the bound graphics layer.

[Syntax]

```
int ioctl (int fd, FBIOGET_CURSOR_POS, HIFB_POINT_S *pstPos)
```

[Description]

This API is used to obtain the display position of the software cursor at the bound graphics layer.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOPUT_CURSOR_POS	ioctl number	Input
pstPos	Information about the display position	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

None

[Example]

None

[See Also]

None

FBIOPUT_CURSOR_COLORKEY

[Purpose]

To set the colorkey information about the software cursor.

[Syntax]

```
int ioctl (int fd, FBIOPUT_CURSOR_ COLORKEY, HIFB_COLORKEY_S * pstColorKey)
```

[Description]

This API is used to set the colorkey information about the software cursor.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOPUT_CURSOR_COLORKEY	ioctl number	Input
pstColorKey	Pointer to the colorkey data type	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

None

[Example]

None

[See Also]

None

FBIOGET_CURSOR_COLORKEY

[Purpose]

To obtain the colorkey information about the software cursor.

[Syntax]

```
int ioctl (int fd, FBIOGET_CURSOR_ COLORKEY, HIFB_COLORKEY_S * pstColorKey)
```

[Description]

This API is used to obtain the colorkey information about the software cursor.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOGET_CURSOR_COLORKEY	ioctl number	Input
pstColorKey	Pointer to the colorkey data type	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

None

[Example]

None

[See Also]

None

FBIOPUT_CURSOR_ALPHA

[Purpose]

To set the alpha blending information about the software cursor.

[Syntax]

```
int ioctl (int fd, FBIOPUT_CURSOR_ALPHA,
HIFB_ALPHA_S *pstAlphaInfo)
```

[Description]

This API is used set the alpha blending information about the software cursor.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input

Parameter	Description	Input/Output
FBIOPUT_CURSOR_ALPHA	ioctl number	Input
pstAlphaInfo	Alpha blending information	Input

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

None

[Example]

None

[See Also]

None

FBIOGET_CURSOR_ALPHA

[Purpose]

To obtain the alpha blending information about the software cursor.

[Syntax]

```
int ioctl (int fd, FBIOGET_CURSOR_ALPHA,
HIFB_ALPHA_S *pstAlphaInfo)
```

[Description]

This API is used to obtain the alpha blending information about the software cursor.

[Parameter]

Parameter	Description	Input/Output
fd	FD of an FB device	Input
FBIOGET_CURSOR_ALPHA	ioctl number	Input
pstAlphaInfo	Alpha blending information	Output

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

Header file: hifb.h

[Note]

None

[Example]

None

[See Also]

None

2.5 Error Codes

Table 2-2 lists all the error codes that may appear when the return value is smaller than zero. These error codes are the standard Linux error codes. For detailed definitions, see the source code errno_base.h of the Linux Kernel. Print the standard Linux error code errno to query the error codes, or use strerror (errno) to print the error information.

Table 2-2 Error codes

Error Code	Macro Definition	Description
1	EPERM	The operation is not supported.
12	ENOMEM	The memory is insufficient.
14	EFAULT	The address of the input parameter pointer is invalid.
22	EINVAL	The input parameter is invalid.

3 Data Types

3.1 Standard Data Types

struct fb_bitfield

[Description]

Defines the bit field information to set the pixel format.

[Definition]

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

[Member]

Member	Description	Supported or Not
offset	Start bit of the color component	Supported
length	Bit length of the color component	Supported
msb_right	Whether the bit on the right is the highest valid bit	The bit can only be zero. In other words, the bit on the left is the highest valid bit.

[Note]

Take the ARGB1555 format as an example, the values of its bit field are as follows:

```
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};
```

[See Also]

None

struct fb_var_screeninfo

[Description]

Defines the variable screen information.

[Definition]

```
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 pixel clock (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 */
};
```

[Member]

Member	Description	Supported or Not
xres	Visible screen width (in pixel)	Supported. The default value of fb0 or fb1 is 1280, and the default value of fb2 or fb3 is 720.
yres	Visible screen height (in pixel)	Supported. The default value of fb0 or fb1 is 720, and the default value of fb2 or fb3 is 576.
xres_virtual	Virtual screen width (image width in the display buffer). When the value is smaller than xres, xres is modified so that it equals the value.	Supported. The default value of fb0 or fb1 is 1280, and the default value of fb2 or fb3 is 720.
yres_virtual	Virtual screen height (image height in the display buffer). When the value is smaller than yres, yres is modified so that it equals the value. In conjunction with xres_virtual, it can be used to quickly move images horizontally or vertically.	Supported. The default value of fb0 or fb1 is 720, and the default value of fb2 or fb3 is 576.
xoffset	Offset pixel count in the horizontal direction	Supported. The default value is 0.
yoffset	Offset pixel count in the vertical direction	Supported. The default value is 0.
bits_per_pixel	Bit counts occupied by a pixel	Supported. The default value is 16.
grayscale	Gray scale	Not supported. The default value is 0, representing color.
red	Bit field information of the red component	Supported. The default value is (10, 5, 0).
green	Bit field information of the green component	Not supported. The default value is (5, 5, 0).
blue	Bit field information of the blue component	Supported. The default value is (0, 5, 0).
transp	Bit field information of the alpha component	Supported. The default value is (15, 1, 0).
nonstd	Whether it is the standard pixel format	Not supported. The default value is 0, indicating that the standard pixel format is supported.

Member	Description	Supported or Not
activate	Activation time	Not supported. The default value is FB_ACTIVATE_NOW, indicating that the configuration is activated right now.
height	Screen height, in the unit of mm	Not supported. The default value is –1.
width	Screen width, in the unit of mm	Not supported. The default value is –1.
accel_flags	The acceleration flag	Not supported. The default value is –1.
pixclock	Time required for displaying a pixel, in the unit of ns	Not supported. The default value is –1.
left_margin	Left margin, right margin, and horizontal synchronization duration respectively. The sum of the three values equals the horizontal scanning duration, in the unit of pixel clock.	Not supported. The default value is –1.
right_margin		
hsync_len		
upper_margin	Upper margin, lower margin, and vertical synchronization duration respectively. The sum of the three values equals the vertical scanning duration, in the unit of pixel clock.	Not supported. The default value is –1.
lower_margin		
vsync_len		
sync	Synchronization signal mode	Not supported. The default value is –1.
vmode	Scan mode	Not supported. The default value is –1.
rotate	Rotated degree in the clockwise direction	Not supported. The default value is 0, indicating no rotation.

[Note]

The default resolution of the HD device graphics layer is 1280 x 720, the default resolution of SD device graphics layer is 720 x 576, and the default resolution of the cursor layer is 128 x 128. The default pixel format is ARGB1555.

[See Also]

- struct fb_bitfield
- FBIOGET_VSCREENINFO
- FBIOPUT_VSCREENINFO

struct fb_fix_screeninfo

[Description]

Defines the fixed screen information.

[Definition]

```
struct fb_fix_screeninfo
{
    char id[16];  /* identification string eg "TT Builtin" */
    unsigned long smem_start;   /* Start of FB mem (physical address) */
    __u32 smem_len;             /* Length of frame buffer mem */
    __u32 type;                 /* see FB_TYPE_* */
    __u32 type_aux;             /* Interleave for interleaved Planes */
    __u32 visual;               /* see FB_VISUAL_* */
    __u16 xpanstep;             /* zero if no hardware panning */
    __u16 ypanstep;             /* zero if no hardware panning */
    __u16 ywrapstep;            /* zero if no hardware ywrap */
    __u32 line_length;          /* length of a line in bytes */
    unsigned long mmio_start;   /* Start of Memory Mapped I/O (physical
                                   address) */
    __u32 mmio_len;             /* Length of Memory Mapped I/O */
    __u32 accel; /* Indicate to driver which specific chip/card we have */
    __u16 reserved[3];          /* Reserved for future compatibility */
};
```

[Member]

Member	Description	Supported or Not
id	Name of the device driver	Supported
smem_start	Physical start address of the display buffer	Supported
smem_len	Size of the display buffer	Supported
type	Type of the display adapter	The value is FB_TYPE_PACKED_PIXELS permanently, indicating packed pixels.
type_aux	Auxiliary type	Not supported. The value is invalid when the video adapter is the FB_TYPE_PACKED_PIXELS type.
visual	Color mode	Not supported. The default value is FB_VISUAL_TRUECOLOR, true color.
xpanstep	Whether the PAN display in the horizontal direction is supported • 0: not supported. • Non-zero: supported. The value represents the pixel counts of each step in the horizontal direction.	The value is 1 permanently.

Member	Description	Supported or Not
ypanstep	Whether the PAN display in the vertical direction is supported • 0: not supported. • Non-zero: supported. The value represents the pixel counts of each step in the vertical direction.	The value is 1 permanently.
ywrapstep	Similar to ypanstep. The difference is that the display is from the start place of the display buffer when the bottom is reached in the ywrapstep mode.	Not supported. The default value is 0.
line_length	Count of bytes in a row	Supported
mmio_start	Start of the memory mapped I/O	Not supported. The default value is 0.
mmio_len	Length of the memory mapped I/O	Not supported. The default value is 0.
accel	Supported hardware acceleration devices	Not supported. The default value is FB_ACCEL_NONE. There is no acceleration device.
reserved	Reserved	Not supported. The default value is 0.

[Note]

None

[See Also]

FBIOGET_FSCREENINFO

3.2 Extended Data Types

HIFB_COLOR_FMT_E

[Description]

Defines the set of the pixel formats supported by the HiFB.

[Definition]

```
typedef enum
{
    HIFB_FMT_1BPP = 0,          /* 1bpp */
    HIFB_FMT_2BPP,              /* 2bpp */
    HIFB_FMT_4BPP,              /* 4bpp */
    HIFB_FMT_8BPP,              /* 8bpp */
    HIFB_FMT_KRGB444,           /* RGB444 */
    HIFB_FMT_KRGB555,            /* RGB555 */
    HIFB_FMT_RGB565,             /* RGB565 */
    HIFB_FMT_ARGB4444,           /* RGB4444 */
    HIFB_FMT_ARGB1555,           /* RGB1555 */
    HIFB_FMT_KRGB888,            /* RGB888 */
    HIFB_FMT_ARGB8888,           /* RGB8888 */
    HIFB_FMT_BUTT
}HIFB_COLOR_FMT_E;
```

[Member]

Member	Description
HIFB_FMT_1BPP	Index format 1 bpp
HIFB_FMT_2BPP	Index format 2 bpp
HIFB_FMT_4BPP	Index format 4 bpp
HIFB_FMT_8BPP	Index format 8 bpp
HIFB_FMT_KRGB444	RGB444 format
HIFB_FMT_KRGB555	RGB555 format
HIFB_FMT_RGB565	RGB565 format
HIFB_FMT_ARGB4444	ARGB4444 format
HIFB_FMT_ARGB1555	ARGB1555 format
HIFB_FMT_KRGB888	RGB888 format
HIFB_FMT_ARGB8888	ARGB8888 format
HIFB_FMT_BUTT	Invalid pixel format

[Note]

None

[See Also]

None

HIFB_CAPABILITY_S

[Description]

Defines the capability of an overlay layer.

[Definition]

```
typedef struct
{
    HI_BOOL bKeyRgb;
    HI_BOOL bKeyAlpha;          /* whether support colorkey alpha */
    HI_BOOL bGlobalAlpha;       /* whether support global alpha */
    HI_BOOL bCmap;              /* whether support color map */
    HI_BOOL bColFmt[HIFB_FMT_BUTT]; /* support which color format */
    HI_U32  u32MaxWidth;        /* the max pixels per line */
    HI_U32  u32MaxHeight;       /* the max lines */
    HI_U32  u32MinWidth;        /* the min pixel per line */
    HI_U32  u32MinHeight;       /* the min lines */
    HI_U32  u32VDefLevel;       /* vertical anti-flicker level, less than 2
                              means vertical anti-flicker is unsupported */
    HI_U32  u32HDefLevel;       /* horizontal anti-flicker level, less than
2
                              means horizontal anti-flicker is unsupported */
    HI_BOOL  bDcmp;
    HI_BOOL  bPreMul;
}HIFB_CAPABILITY_S;
```

[Member]

Member	Description
bKeyRgb	Whether the color component supports the colorkey operation
bKeyAlpha	Whether the colorkey with alpha is supported
bGlobalAlpha	Whether the global alpha and the pixel alpha overlay are supported
bCmap	Whether the palette mode is supported
bColFmt	Supported pixel formats For example, the equation bColFmt[HIFB_FMT_ARGB1555] = 1 indicates that the ARGB1555 format is supported.
u32MaxWidth	Maximum resolution width
u32MaxHeight	Maximum resolution height
u32MinWidth	Minimum resolution width
u32MinHeight	Minimum resolution height
u32VDefLevel	Maximum vertical anti-flicker level. The vertical anti-flicker is not supported when the value is smaller than two.
u32HDefLevel	Maximum horizontal anti-flicker level. The horizontal anti-flicker is not supported when the value is smaller than two.
bDcmp	Whether the compression mode is supported
bPreMul	Whether the pre-multiply mode is supported

[Note]

- bGlobalAlpha = 1

  Overlaying between the global alpha and the pixel alpha is supported. When the overlay layer is in the alpha channel mode, the overlay alpha is the sum of the global alpha and the pixel alpha.
- bGlobalAlpha = 0

  Overlaying between the global alpha and the pixel alpha is not supported. When the overlay layer is in the alpha channel mode, the overlay alpha is equal to the global alpha.

[See Also]

- HIFB_COLOR_FMT_E
- FBIOGET_CAPABILITY_HIFB

HIFB_POINT_S

[Description]

Defines the coordinates.

[Definition]

```
typedef struct
{
   HI_U32 u32PosX;         /* horizontal position */
   HI_U32 u32PosY;         /* vertical position */
}HIFB_POINT_S;
```

[Member]

Member	Description
u32PosX	Horizontal coordinate
u32PosY	Vertical coordinate

[Note]

None

[See Also]

- FBIOGET_SCREEN_ORIGIN_HIFB.
- FBIOPUT_SCREEN_ORIGIN_HIFB.

HIFB_MIRROR_MODE_E

[Description]

Defines the mirror modes.

[Definition]

```
typedef enum
{
    HIFB_MIRROR_NONE = 0x0,
    HIFB_MIRROR_HORIZONTAL = 0x1,
    HIFB_MIRROR_VERTICAL = 0x2,
    HIFB_MIRROR_BOTH= 0x3,
    HIFB_MIRROR_BUTT
}HIFB_MIRROR_MODE_E;
```

[Member]

Member	Description
HIFB_MIRROR_NONE	No mirror
HIFB_MIRROR_HORIZONTAL	Horizontal mirror
HIFB_MIRROR_VERTICAL	Vertical mirror
HIFB_MIRROR_BOTH	Horizontal and vertical mirror
HIFB_MIRROR_BUTT	Invalid mirror mode

[Note]

None

[See Also]

FBIOGET_MIRROR_MODEFBIOPUT_MIRROR_MODE

HIFB_ALPHA_S

[Description]

Defines the alpha information.

[Definition]

```
typedef struct
{
    HI_BOOL bAlphaEnable;        /* alpha enable flag */
    HI_BOOL bAlphaChannel;       /* alpha channel enable flag */
    HI_U8 u8Alpha0;              /* alpha0 value */
    HI_U8 u8Alpha1;              /* alpha1 value */
    HI_U8 u8GlobalAlpha;         /* global alpha value */
    HI_U8 u8Reserved;
}HIFB_ALPHA_S;
```

[Member]

Member	Description
bAlphaEnable	Enable signal of the alpha overlay. The default value is 1.

Member	Description
bAlphaChannel	Enable signal of the alpha channel. The default value is 0.
u8Alpha0	Value of alpha 0, ranging from 0 to 255. The default value is 255. When the highest bit is 0 in the RGB1:5:5:5 format, the value is the overlay alpha value.
u8Alpha1	Value of alpha 1, ranging from 0 to 255. The default value is 255. When the highest bit is 1 in the RGB1:5:5:5 format, the value is the overlay alpha value.
u8GlobalAlpha	Global alpha value, ranging from 0 to 255. The default value is 255. It takes effect when the alpha channel is enabled.
u8Reserved	Reserved

[Note]

After being enabled, the alpha overlay can be performed; otherwise, the lower layer is overlaid with the upper layer.

- When the alpha channel is enabled, the global alpha takes part in the overlay.
  - For the chip that does not support the overlay of the global alpha and the pixel alpha, the formula for calculating the overlay alpha is as follows: $\alpha = u8GlobalAlpha$
  - For the chip that supports the overlay of the global alpha and the pixel alpha, the formula for calculating the overlay alpha is as follows: $\alpha = u8GlobalAlpha * \alpha_{pixel}$
- When the alpha channel is disabled, the overlay alpha equals the pixel alpha as follows: $\alpha = \alpha_{pixel}$

[See Also]

- FBIOGET_ALPHA_HIFB
- FBIOPUT_ALPHA_HIFB

HIFB_COLORKEY_S

[Description]

Defines the information to set the colorkey attribute.

[Definition]

```
typedef struct
{
   HI_BOOL bKeyEnable;       /*Colorkey enable*/
   HI_U32 u32Key;
}HIFB_COLORKEY_S;
```

[Member]

Member	Description
bKeyEnable	Colorkey enable TRUE: enabled FALSE: disabled
u32Key	Colorkey value

[Note]

None

[See Also]

- FBIOGET_COLORKEY_HIFB
- FBIOPUT_COLORKEY_HIFB

HIFB_DEFLICKER_S

[Description]

Defines the anti-flicker information to set or obtain the anti-flicker status of an overlay layer.

[Definition]

```
typedef struct hiHIFB_DEFLICKER_S
{
   HI_U32  u32HDfLevel;    /* horizontal anti-flicker level */
   HI_U32  u32VDfLevel;    /* vertical anti-flicker level */
   HI_U8   *pu8HDfCoef;    /* horizontal anti-flicker coefficient */
   HI_U8   *pu8VDfCoef;    /* vertical anti-flicker coefficient */
}HIFB_DEFLICKER_S;
```

[Member]

Member	Description
u32HDfLevel	Level of horizontal anti-flicker
u32VDfLevel	Level of vertical anti-flicker
pu8HDfCoef	Horizontal anti-flicker coefficient. The number of coefficients is the level of horizontal anti-flicker minus 1.
pu8VDfCoef	Vertical anti-flicker coefficient. The number of coefficients is the level of vertical anti-flicker minus 1.

[Note]

The anti-flicker level is the number of pixels in a row or column that are involved in operations. In general, the higher the anti-flicker level is, the better the anti-flicker effect is, but the more blurred the picture is.

[See Also]

- FBIOGET_DEFLICKER_HIFB
- FBIOPUT_DEFLICKER_HIFB

HIFB_SURFACEEX_S

[Description]

Defines the surface information to set the attributes of two surfaces in dual-buffer mode.

[Definition]

```
typedef struct
{
    HI_U32 u32PhyAddr;
    HIFB_ALPHA_S stAlpha;
    HIFB_COLORKEY_S stColorkey;
}HIFB_SURFACEEX_S;
```

[Member]

Member	Description
u32PhyAddr	Physical address of a surface
stAlpha	Alpha attributes of a surface
stColorkey	Colorkey attributes of a surface

[Note]

The surface physical address must be within the address range of the display buffer configured at the overlay layer and must be 16-byte aligned.

[See Also]

FBIOFLIP_SURFACE

HIFB_LAYER_INFO_S

[Description]

Defines the graphics layer information.

[Definition]

```
typedef struct
{
    HIFB_LAYER_BUF_E BufMode;
    HIFB_LAYER_ANTIFLICKER_LEVEL_E eAntiflickerLevel;
    HI_S32 s32XPos;          /**<  the x pos of origin point in screen */
    HI_S32 s32YPos;          /**<  the y pos of origin point in screen */
    HI_U32 u32CanvasWidth;    /**<  the width of canvas buffer */
    HI_U32 u32CanvasHeight;   /**<  the height of canvas buffer */
    HI_U32 u32DisplayWidth;   /**<  the width of display buf in fb */
    HI_U32 u32DisplayHeight;  /**<  the height of display buf in fb. */
    HI_U32 u32ScreenWidth;    /**<  the width of screen */
    HI_U32 u32ScreenHeight;   /**<  the height of screen */
    HI_BOOL bPreMul;         /**<  The data drawn in buffer is premultiplied 
data or not.*/
    HI_U32 u32Mask;           /**<  param modify mask bit*/
}HIFB_LAYER_INFO_S;
```

[Member]

Member	Description
BufMode	Refresh mode in extended mode
eAntiflickerLevel	Anti-flicker level of a graphics layer
s32XPos	Origin horizontal coordinate of a graphics layer on the screen
s32YPos	Origin vertical coordinate of a graphics layer on the screen
u32CanvasWidth	Canvas buffer width
u32CanvasHeight	Canvas buffer height
u32DisplayWidth	Display buffer width
u32DisplayHeight	Display buffer height
u32ScreenWidth	Display screen width
u32ScreenHeight	Display screen height
bPreMul	Whether the data in the FB is premultiplied data
u32Mask	Parameter modification mask bit when the graphics layer information is set

[Note]

- The Hi35xx does not support layer scaling. When the display buffer resolution or the screen display resolution changes, the actual display resolution also changes. The display buffer resolution or the screen display resolution must be less than or equal to the device resolution.
- When the pixel format of the graphics layer is ARGB1555 or ARGB4444, the pre-multiplication mode is not supported.
- When the global alpha of the graphics layer is 1, the pre-multiplication mode is not supported.

[See Also]

- FBIOPUT_LAYER_INFO
- FBIOGET_LAYER_INFO

HIFB_LAYER_ANTIFLICKER_LEVEL_E

[Description]

Defines the anti-flicker level of a graphics layer.

[Definition]

```
typedef enum
{
    HIFB_LAYER_ANTIFLICKER_NONE = 0x0,  /**<No anti-flicker*/
    HIFB_LAYER_ANTIFLICKER_LOW = 0x1,   /**<Low level*/
    HIFB_LAYER_ANTIFLICKER_MIDDLE = 0x2,/**<Medium level*/
    HIFB_LAYER_ANTIFLICKER_HIGH = 0x3, /**<High level*/
    HIFB_LAYER_ANTIFLICKER_AUTO = 0x4, /**<Automatic*/
    HIFB_LAYER_ANTIFLICKER_BUTT
}HIFB_LAYER_ANTIFLICKER_LEVEL_E;
```

[Member]

Member	Description
HIFB_LAYER_ANTIFLICKER_NONE	No anti-flicker
HIFB_LAYER_ANTIFLICKER_LOW	Low-level anti-flicker
HIFB_LAYER_ANTIFLICKER_MIDDLE	Medium-level anti-flicker
HIFB_LAYER_ANTIFLICKER_HIGH	High-level anti-flicker
HIFB_LAYER_ANTIFLICKER_AUTO	Automatic anti-flicker
HIFB_LAYER_ANTIFLICKER_BUTT	Invalid

[Note]

If this data type is not set, automatic anti-flicker is used by default.

[See Also]

- FBIOPUT_LAYER_INFO
- FBIOGET_LAYER_INFO

HIFB_LAYER_BUF_E

[Description]

Defines the graphics layer refresh type.

[Definition]

```
typedef enum
{
  HIFB_LAYER_BUF_DOUBLE = 0x0,
  HIFB_LAYER_BUF_ONE  = 0x1,
  HIFB_LAYER_BUF_NONE = 0x2,
  HIFB_LAYER_BUF_DOUBLE_IMMEDIATE = 0x3,
  HIFB_LAYER_BUF_BUTT
} HIFB_LAYER_BUF_E;
```

[Member]

Member	Description
HIFB_LAYER_BUF_DOUBLE	Dual-buffer mode
HIFB_LAYER_BUF_ONE	Single-buffer mode
HIFB_LAYER_BUF_NONE	Non-buffer mode
HIFB_LAYER_BUF_DOUBLE_IMMEDIATE	Dual-buffer immediate mode
HIFB_LAYER_BUF_BUTT	Invalid

NOTE

For details about each refresh type, see the description of the refresh mode of graphics layers in section 1.2 in the *HiFB Development Guide*.

[Note]

- Because the drawn contents are transferred from the canvas buffer to the display buffer by using the TDE, the TDE determines whether scaling is supported. When contents are transferred from the display buffer to a VO device, the VO device determines whether scaling is supported. The VO device of the Hi35xx does not support scaling. Therefore, the display buffer resolution is always the same as the screen display resolution.
- The difference between HIFB_LAYER_BUF_DOUBLE and HIFB_LAYER_BUF_DOUBLE_IMMEDIATE is as follows: If a refresh operation is performed by calling HIFB_LAYER_BUF_DOUBLE_IMMEDIATE, the API is returned only after the refreshed contents are displayed. If a refresh operation is performed by calling HIFB_LAYER_BUF_DOUBLE, the API is returned immediately after HIFB_LAYER_BUF_DOUBLE is called.

[See Also]

- FBIOPUT_LAYER_INFO
- FBIOGET_LAYER_INFO

HIFB_LAYER_INFO_MASKBIT

[Description]

Identifies the updated members of HIFB_LAYER_INFO_S.

[Definition]

```
typedef enum
{
    HIFB_LAYERMASK_BUFMODE = 0x1,
    HIFB_LAYERMASK_ANTIFLICKER_MODE = 0x2,
    HIFB_LAYERMASK_POS = 0x4,
    HIFB_LAYERMASK_CANVASSIZE = 0x8,
    HIFB_LAYERMASK_DISPSIZE = 0x10,
    HIFB_LAYERMASK_SCREENSIZE = 0x20,
    HIFB_LAYERMASK_BMUL = 0x40,
    HIFB_LAYERMASK_BUTT
}HIFB_LAYER_INFO_MASKBIT;
```

[Member]

Member	Description
HIFB_LAYERMASK_BUFMODE	Whether the buffer mode in HIFB_LAYER_INFO_S is valid mask
HIFB_LAYERMASK_ANTIFLICKER_MODE	Whether the anti-flicker mode is valid mask
HIFB_LAYERMASK_POS	Whether the graphics layer position is valid mask
HIFB_LAYERMASK_CANVASSIZE	Whether canvassize is valid mask
HIFB_LAYERMASK_DISPSIZE	Whether displaysize is valid mask
HIFB_LAYERMASK_SCREENSIZE	Whether screensize is valid mask
HIFB_LAYERMASK_BMUL	Whether premultiplication is valid mask
HIFB_LAYERMASK_BUTT	Invalid

[Note]

After setting the attributes of an item, you must set the corresponding mask. Otherwise, the settings do not take effect.

[See Also]

- FBIOPUT_LAYER_INFO
- FBIOGET_LAYER_INFO

HIFB_BUFFER_S

[Description]

Defines the canvas information and refresh region of a graphics layer for drawing and refreshing.

[Definition]

```
typedef struct
{
    HIFB_SURFACE_S stCanvas;
    HIFB_RECT UpdateRect; /*Refresh region*/
}HIFB_BUFFER_S;
```

[Member]

Member	Description
stCanvas	Canvas information about a graphics layer
UpdateRect	Refresh region of a graphics layer

[Note]

None

[See Also]

- FBIO_REFRESH
- FBIOGET_CANVAS_BUFFER

HIFB_SURFACE_S

[Description]

Defines the surface information to set the attributes of two surfaces in dual-buffer mode.

[Definition]

```
typedef struct
{
   HI_U32  u32PhyAddr;     /**<  start physical address */
   HI_U32  u32Width;      /**<  width pixels */
   HI_U32  u32Height;     /**<  height pixels */
   HI_U32  u32Pitch;      /**<  line pixels */
   HIFB_COLOR_FMT_E enFmt;  /**<  color format */
}HIFB_SURFACE_S;
```

[Member]

Member	Description
u32PhyAddr	Physical address of a surface
u32Width	Surface width
u32Height	Surface height
u32Pitch	Row stride of the storage area
enFmt	Pixel format

[Note]

None

[See Also]

- HIFB_BUFFER_S
- HIFB_CURSOR_S

HIFB_CURSOR_S

[Description]

Defines the cursor information including the information about the software cursor.

[Definition]

```
typedef struct
{
    HIFB_SURFACE_S stCursor;
    HIFB_POINT_S stHotPos;
} HIFB_CURSOR_S;
```

[Member]

Member	Description
stCursor	Canvas information about the software cursor
stHotPos	Hot spot position of the software cursor

[Note]

The hot spot of the software cursor is the reference point in the software cursor bitmap that is used to perform the offset operation when the offset position of the software cursor is specified at the graphics layer by calling FBIOPUT_CURSOR_POS. Note that the hot spot is not the start point (0, 0). The horizontal and vertical coordinates of the hot spot must be greater than or equal to 0 and must be less than or equal to the width and height of the cursor bitmap.

[See Also]

- FBIOPUT_CURSOR_INFO
- FBIOGET_CURSOR_INFO

HIFB_DDRZONE_S

[Description]

Defines the DDR detection attributes including the start zone and number of zones for DDR detection.

[Syntax]

```
typedef struct
{
    HI_U32 u32StartSection;
    HI_U32 u32ZoneNums;
} HIFB_DDRZONE_S;
```

[Member]

Member	Description
u32StartSection	Start zone for DDR detection
u32ZoneNums	Number of DDR detection zones

[Note]

A maximum of 32 DDR detection zones are supported. The total number of the start zone and other zones cannot be greater than 32.

[See Also]

- FBIOPUT_MDDRDETECT_HIFB
- FBIOGET_MDDRDETECT_HIFB

4 Auxiliary Interfaces for Graphics Development

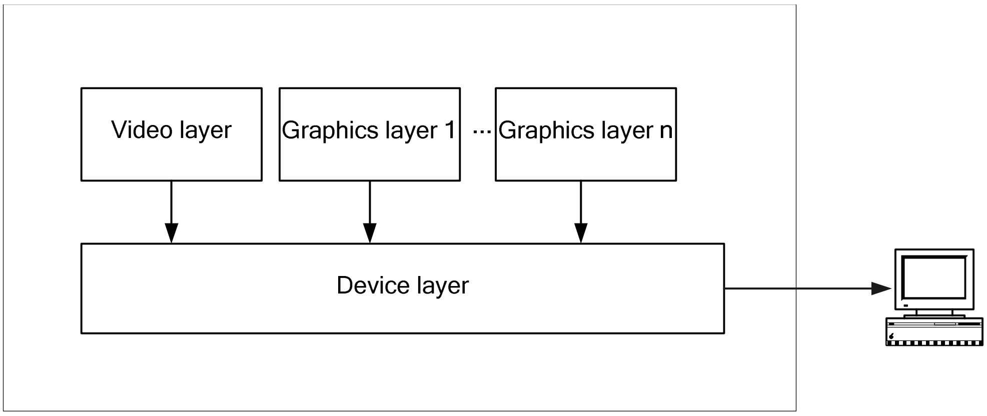
4.1 Overview

4.1.1 Introduction

The video output unit (VOU) consists of the device layer, video layer, and graphics layer, as shown in Figure 4-1. The details are as follows:

- The device layer is the basis of the video layer and graphics layers. Based on the configuration, the device layer outputs timings to enable its connected display device to display videos and graphics. In addition, the device layer determines the output resolution of the device, that is, the device layer limits the display resolutions of the video layer and graphics layers.
- Because of the preceding architecture, before performing any operation on the device layer, you must close the video layer and all the graphics layers to ensure that the videos and graphics can be displayed properly. For example,
  - Before closing the device layer, close the video layer and graphics layers.
  - When the attributes of the device layer change such as the switching of the output resolution of the device, close the video layer, graphics layers, and device layer in sequence, and then reconfigure and restart the device layer, video layer, and graphics layers in sequence.

Figure 4-1 Basic architecture of the VOU



4.1.2 Guidelines

Note the following when developing the graphics layers:

Displaying a Graphics Layer on the Display Device

To display a graphics layer on the display device properly, you must configure and start the device layer before calling the open("/dev/fbn") function.

Each display device supports multiple output timings. By default, the configurations of device layers are not provided in the SDK and the device layer is not started when the HiFB module is inserted. You can view the display result only after enabling the device layer by calling the related APIs and then operating the graphics layers.

The SDK controls the device layer by using the VOU. The VOU provides the APIs for controlling the device layer and video layer. The APIs for operating the device layer include HI_MPI_VO_Enable, HI_MPI_VO_Disable, and HI_MPI_VO_SetPubAttr/HI_MPI_VO_GetPubAttr.

NOTE

For details about the VOU APIs, see section 4.3 in the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

Switching a Graphics Layer Between Devices

Table 4-1 describes how to switch a graphics layers between devices.

Table 4-1 Switching a graphics layer between devices.

Chip	Description
Hi3516A/Hi3518E V200/Hi3518E V200/Hi3519 V100/Hi3519 V101	The Hi3516A supports one graphics layer (G0). G0 is always bound to DSD0.

NOTE

Before switching a graphics layer, you must unbind and close it. However, you do not need to disable the display device.

4.2 API Reference

HI_MPI_VO_BindGraphicLayer

[Purpose]

To bind a graphics layer to a specified VO device.

[Syntax]

```
HI_S32 HI_MPI_VO_BindGraphicLayer(GRAPHIC_LAYER GraphicLayer, VO_DEV VoDev)
```

[Description]

This API is used to bind a graphics layer to a VO device.

[Parameter]

Parameter	Description	Input/Output
GraphicLayer	Graphics layer ID	Input
VoDev	VO device ID	Input

[Difference]

Chip	Value Range of VoDev
Hi3516A/Hi3518E V200/Hi3518E V200/Hi3519 V100/Hi3519 V101	This API is not supported.

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

Before calling this API, ensure that the graphics layer is unbound and disabled.

[Example]

None

[See Also]

HI_MPI_VO_UnBindGraphicLayer

HI_MPI_VO_UnBindGraphicLayer

[Purpose]

To unbind a specified graphics layer from a device.

[Syntax]

```
HI_S32 HI_MPI_VO_UnBindGraphicLayer(GRAPHIC_LAYER GraphicLayer, VO_DEV
VoDev)
```

[Description]

This API is used to unbind a specified graphics layer from a device.

[Parameter]

Parameter	Description	Input/Output
GraphicLayer	Graphics layer ID	Input
VoDev	VO device ID	Input

[Difference]

Chip	Value Range of VoDev
Hi3516A/Hi3518E V200/Hi3518E V200/Hi3519 V100/Hi3519 V101	This API is not supported.

[Return Value]

Return Value	Description
0	Success
–1	Failure

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Before calling this API, ensure that the graphics layer is disabled.
- Currently, **VoDev** is meaningless and is set to **0** typically.
- If you unbind a graphics layer that is not bound before, a code indicating success is returned. That is, a graphics layer can be unbound for multiple times.

[Example]

None

[See Also]

HI_MPI_VO_UnBindGraphicLayer

HI_MPI_VO_SetPubAttr

[Purpose]

To set the public attributes of a VO device, including the interface type and timing.

[Syntax]

```
HI_S32 HI_MPI_VO_SetPubAttr(VO_DEV VoDev, const VO_PUB_ATTR_S *pstPubAttr)
```

[Description]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Parameter]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Return Value]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Note]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Example]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[See Also]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

HI_MPI_VO_GetPubAttr

[Purpose]

To query the public attributes of a VO device, including the interface type and timing.

[Syntax]

```
HI_S32 HI_MPI_VO_GetPubAttr(VO_DEV VoDev, VO_PUB_ATTR_S *pstPubAttr)
```

[Description]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Parameter]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Return Value]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or

HiMPP IPC V3.0 Media Processing Software Development Reference.

[Note]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*

[Example]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[See Also]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

HI_MPI_VO_Enable

[Purpose]

To enable a VO device.

[Syntax]

```
HI_S32 HI_MPI_VO_Enable (VO_DEV VoDev)
```

[Description]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Parameter]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Return Value]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Note]

To display a graphics layer on the display device properly, you must enable a VO device by calling this API before calling the open("/dev/fbn") function.

[Example]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[See Also]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

HI_MPI_VO_Disable

[Purpose]

To disable a VO device.

[Syntax]

```
HI_S32 HI_MPI_VO_Disable(VO_DEV VoDev)
```

[Description]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Parameter]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Return Value]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Note]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[Example]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

[See Also]

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

4.3 Data Types

VO_DEV

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

VO_PUB_ATTR_S

For details, see the *HiMPP IPC V2.0 Media Processing Software Development Reference* or *HiMPP IPC V3.0 Media Processing Software Development Reference*.

5 Proc Debugging Information

5.1 Mapping Between IDs of the Graphics Layers and System Devices of an FB

Note the following:

- The Hi3516A/Hi3518E V200/Hi3518E V200/Hi3519 V100/Hi3519 V101 HIFB manages at most one graphics layer (G0) that correspond to the devices files **/dev/fb0.**
- View the status of each graphics layer by running **cat /proc/umap/hifb*n***. *n* indicates the graphics layer ID.

5.2 Debugging Information About a Single Graphics Layer

[Debugging Information]

```
# cat /proc/umap/hifb0
layer name                  :layer_0
Open count                  :0
Show state                  :OFF
Start position              :(0, 0)
xres, yres                  :(1280, 720)
xres_virtual, yres_virtual  :(1280, 1440)
xoffset, yoffset            :(0, 720)
fix.line_length             :2560
Mem size:                   :8100 KB
Layer Scale (hw):           :NO
ColorFormat:                :ARGB1555
Alpha Enable                :ON
AlphaChannel Enable         :OFF
Alpha0, Alpha1              :0, 255
Alpha Global                :0
Colorkey Enable             :OFF
Colorkey value              :0xfc00
Deflicker Mode:             :NONE
Deflicker Level:            :AUTO
Display Buffer mode         :unkown
Displaying addr (register)  :0x841d2000
display buffer[0] addr      :0x84010000
display buffer[1] addr      :0x841d2000
displayrect                 :(1280, 720)
screenrect                  :(1280, 720)
device max resolution       :1280, 720
IsNeedFlip(2buf)            :NO
BufferIndexDisplaying(2buf) :1
refresh request num(2buf)   :0
switch buf num(2buf)        :0
union rect (2buf)           :(0,0,0,0)
canavas updated addr        :0x841d2000
canavas updated (w, h)      :1280,720
canvas width                :1280
canvas height               :720
canvas pitch                :2560
canvas format               :ARGB1555
IsCompress                  :NO
Is DDR Dettect               :NO
DDR Detect Zones            :0
```

[Analysis]

This section records the memory configuration and display information about the graphics layer corresponding to the current device.

[Parameter Description]

Parameter		**Description**
Basic attributes of a graphics layer	layer name	The layer names of G0 to G3are layer_0, layer_1, layer_2, layer_3.
	Open Count	Count of opening the graphics layer. This number is increased by 1 when open( ) is called and is decreased by 1 when close( ) is called. After the first user calls open( ), the graphics layer of the VOU is opened actually; after the last user calls close(), the graphics layer is closed actually.
	Show State	Display status of the graphics layer. Value range: {OFF: hide; ON: show} After struct fb_var_screeninfo is configured successfully, the graphics layer is automatically displayed and its status value is changed to 1. When FBIOPUT_SHOW_HIFB is called to hide or show the graphics layer, the status value is changed accordingly.

Parameter		Description
	Start Position	Start display position of the graphics layer on the display device, in pixel. For example, (100, 50) indicates that the start display position x is 100 and y is 50. The default value is (0, 0). You can call FBIOPUT_SCREEN_ORIGIN_HIFB to update the display position.
	Layer Scale (hw)	Whether the graphics layer supports hardware scaling. Value: {NO: not supported; YES: supported} This parameter is fixed at NO for the Hi35xx.
	ColorFormat	Format of the graphics layer. Value range: Hi3516A//Hi3519 V100/Hi3519 V101: {ARGB1555, ARGB4444, ARGB8888} Hi3518E V200: {ARGB1555, ARGB4444} After the system is loaded, the default format is ARGB1555. You can update the format after configuring the format item of struct fb_var_screeninfo.
	AlphaEnable	Whether to enable alpha of the graphics layer. Value range: {OFF: no; ON: yes}. The default value is ON. All the alpha information in Proc is updated when FBIOPUT_ALPHA_HIFB is updated. If Alpha Enable is disabled, the pixel alpha configurations become invalid. If Alpha Enable is enabled but AlphaChannel is disabled, only the pixel alpha is valid (that is, Alpha0 and Alpha1 are valid for the ARGB1555 format). If both Alpha Enable and AlphaChannel are enabled, the pixel alpha and global alpha are valid.
	AlphaChannel Enable	Control whether the global alpha is valid. Value range: {OFF: no; ON: yes}. The default value is ON. Alpha Global is valid only when AlphaChannel Enable is enabled.
	Alpha0	In ARGB1555 format, if the most significant bit (MSB) is 0, you can select alpha0 as the alpha value of alpha blending. The value ranges from 0 to 255 and the default value is 0.

Parameter		Description
	Alpha1	In ARGB1555 format, if the MSB is 1, you can select alpha1 as the alpha value of alpha blending. The value ranges from 0 to 255 and the default value is 255.
	Alpha Global	Global alpha. The value ranges from 0 to 255 and the default value is 255.
	Colorkey Enable	Whether to enable the colorkey function of the graphics layer. Value range: {OFF: no; ON: yes}. The default value is OFF.
	Colorkey Value	Value of the transparent pixel that is consistent with the current pixel format of the graphics layer.
	Deflicker Mode	Anti-flicker mode.
	Deflicker Level	Anti-flicker level.
	device max resolution	Current display resolution of the display device where the graphics layer is located.
	IsCompress	Whether the compression function is enabled.
	DDR Detect Zones	Number of DDR detection zones
Information about the display buffer of a graphics layer	fix.smem_start	Start physical address of the display buffer that is allocated for the graphics layer. The display buffer is allocated when the HiFB module is loaded.
	fix.smem_len	Size of the display buffer allocated for the graphics layer, in byte. The minimum size is 256 bytes and the maximum size depends on the size of the MMZ. The HiFB display buffer is allocated from the MMZ. The MMZ are divided into blocks by 4096 bytes. The size of a display buffer must be an integral multiple of 4096 bytes. For example, if you set vramX_size to 256 when loading the HiFB module, the actual size of the allocated buffer is 4096 bytes, that is, fix.smem_len is 4096.

Parameter		Description
	fix.line_length	Stride of a display buffer, in byte. The stride of a display buffer is calculated by multiplying var.xres_virtual (set by configuring struct fb_var_screeninfo) by the number of bytes occupied by each pixel. In addition, the stride is automatically 8-byte aligned upwards. You can view the stride of a display buffer by querying struct fb_fix_screeninfo.
	var.xres_virtual	Width of the virtual screen, in pixel. See Figure 2-1. The default value is 720. (xres_virtual, yres_virtual): virtual screen area that indicates the maximum area that can be operated by using the HiFB. The actual display area is specified by (xres, yres). Note that the size of the virtual screen area cannot greater than that of the display buffer. (xres, yres): size of the current display area. It can be a part of the size specified by (xres_virtual, yres_virtual). (xoffset, yoffset): start position of the current display area in the area specified by (xres_virtual, yres_virtual).
	var.yres_virtual	Height of the virtual screen, in pixel. See Figure 2-1. The default value is 576.
	var.xoffset	Start x coordinate of the actual display area in the virtual screen area, in pixel. The default value is 0. You can adjust the position of the display area in the display buffer by calling FBIOPAN_DISPLAY.
	var.yoffset	Start y coordinate of the actual display area in the virtual screen area, in pixel. The default value is 0. You can adjust the position of the display area in the display buffer by calling FBIOPAN_DISPLAY.
	var.xres	Width of the actual display area, in pixel. See Figure 2-1 The default value is 720.
	var.yres	Height of the actual display area, in pixel. See Figure 2-1. The default value is 576.

Parameter		Description
	Display Buffer mode	Refresh mode. The mapping between refresh modes and display contents is as follows: HIFB_LAYER_BUF_DOUBLE - triple HIFB_LAYER_BUF_ONE - double HIFB_LAYER_BUF_NONE - single DOUBLE_IMMEDIATE - triple(no frame is discarded) HIFB_LAYER_BUF_BUTT - unknown

5.3 Graphics Layers That Can Be Dynamically Bound

To view the graphics layers that can be dynamically bound, run **cat /proc/umap/vo**. You can check the last lines as follows:

```
-----GRAPHICS LAYER------------------------------------------------------------
Layer BindDev
HC0      0
```

[Parameter Description]

Parameter		Description
GRAPHICS LAYER	Layer	Graphics layers that can be dynamically bound.
	BindDev	ID of the VO device to which a graphics layer is bound.