Graphics Development

User Guide

Issue	**01**
Date	**2018-05-15**

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 provides one schemes for graphics development. The schemes include scheme description, derivative scheme, development process, application scenarios, and advantages and limitations.

Related Versions

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

Product Name	**Version**
Hi3536	V100
Hi3521A	V100
Hi3520D	V300
Hi3531A	V100
Hi3531D	V100
Hi3521D	V100
Hi3536C	V100
Hi3536D	V100
Hi3520D	V400

Intended Audience

This document is intended for:

- Technical support personnel
- Software development engineers

Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
⚠ DANGER	Alerts you to a high risk hazard that could, if not avoided, result in serious injury or death.
⚠ WARNING	Alerts you to a medium or low risk hazard that could, if not avoided, result in moderate or minor injury.
⚠ CAUTION	Alerts you to a potentially hazardous situation that could, if not avoided, result in equipment damage, data loss, performance deterioration, or unanticipated results.
TIP	Provides a tip that may help you solve a problem or save time.
NOTE	Provides additional information to emphasize or supplement important points in the main text.

Change History

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

Issue 01(2018-05-15)

This issue is the first official release.

The sction 1.2.3 is modified.

Issue 00B07(2017-11-20)

This issue is the seventh draft release, which incorporates the following changes:

In section 1.2, table 1-1 to table 1-4 are updated.

Issue 00B06(2017-09-08)

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

The description of the Hi3536D V100 is added.

Issue 00B05 (2017-04-10)

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

The description of the Hi3536C V100 is added.

Issue 00B04 (2015-10-10)

This issue is the fourth draft release.

The section 1.2.3 is modified.

Issue 00B03 (2015-07-29)

This issue is the third draft release.

The description of the Hi3531A is added.

Issue 00B02 (2015-03-02)

This issue is the second draft release.

The contents related to the Hi3521A and Hi3520D V300 are added.

Issue 00B01 (2015-01-14)

This issue is the first draft release.

Contents

Figures

Tables

1 Introduction to Graphics Layers

1.1 Overview

The HiSilicon digital media processing platform (HiMPP) provides a set of mechanisms for developing graphical user interfaces (GUIs). The mechanisms consist of:

- Two-dimensional engine (TDE). It processes graphics through hardware acceleration.
- HiSilicon frame buffer (HiFB). It manages graphics layers. Besides the basic functions of the Linux FB, it also provides the extended functions such as inter-layer colorkey and inter-layer alpha.

NOTE

- For details on how to use the TDE, see the *TDE API Reference*.
- For details on how to use the HiFB, see the *HiFB Development Guide* and *HiFB API Reference*.
- Unless otherwise stated, Hi3521A and Hi3520D V300 contents are consistent.
- Unless otherwise stated, Hi3521DV100 and Hi3520D V400 contents are consistent.

1.2 Architecture of Graphics Layers

1.2.1 Architecture of Graphics Layers for Hi3536

The Hi3536 supports two HD display devices (DHD0 and DHD1), one SD display device (DSD0), and five graphics layers (G0−G4). G3 and G4 are cursor layers.

NOTE

For details about the interfaces and timings supported by each output device, see section 11.2 "VDP" in the *Hi3536 H.265 Decoder Processor Data Sheet* .

Table 1-1 shows the relationships among the FB device files of the Hi3536, graphics layers, and output devices.

Table 1-1 Relationships among the FB device files of the Hi3536, graphics layers, and output devices

FB Device File	Graphics Layer	Corresponding Display Device
/dev/fb0	G0	G0 is displayed only on DHD0.

FB Device File	Graphics Layer	Corresponding Display Device
/dev/fb1	G1	G1 is displayed only on DHD1.
/dev/fb2	G2	G2 is displayed only on DSD0.
/dev/fb3	G3	G3 is a cursor layer and is always displayed on the top. For example, if the video layer of DHD0, G0, and G3 are overlaid, the overlay sequence from bottom to top is as follows: video layer < G0 < G3. G3 can act as the hardware cursor layer or software cursor layer, which is determined by the driver loading parameter **softcursor**. When G3 is used as the hardware cursor layer, its usage is the same as that of other graphic layers. When G3 is used as the software cursor layer, it is operated by calling dedicated software cursor interface of the HiFB
/dev/fb4	G4	G4 is a cursor layer. It has the same features as G3.

NOTE

To display graphics layers, you must configure and enable VO devices by calling the interfaces of the VOU, and operate graphics layers by calling the interfaces of the Hi3536 HiFB.

1.2.2 Architecture of Graphics Layers for Hi3521A

The Hi3521A supports one HD display devices (DHD0), one SD display device (DSD0), and three graphics layers (G0−G2). G2 is cursor layers.

NOTE

For details about the interfaces and timings supported by each output device, see section 10.2 "VDP" in the Hi3521A/Hi3520DV300 H.264 CODEC Processor Data Sheet

Table 1-2 shows the relationships among the FB device files of the Hi3521A, graphics layers, and output devices.

Table 1-2 Relationships among the FB device files of the Hi3521A, graphics layers, and output devices

FB Device File	Graphics Layer	Corresponding Display Device
/dev/fb0	G0	G0 is displayed only on DHD0.
/dev/fb1	G1	G1 is displayed only on DSD0.
/dev/fb2	G2	G2 is a cursor layer and is always displayed on the top. For example, if the video layer of DHD0, G0, and G2 are overlaid, the overlay sequence from bottom to top is as follows: video layer < G0 < G2. G2 can act as the hardware cursor layer or software cursor layer, which is determined by the driver loading parameter **softcursor**. When G2 is used as the hardware cursor layer, its usage is the same as

FB Device File	Graphics Layer	Corresponding Display Device
		that of other graphic layers. When G2 is used as the software cursor layer, it is operated by calling dedicated software cursor interface of the HiFB

NOTE

To display graphics layers, you must configure and enable VO devices by calling the interfaces of the VOU, and operate graphics layers by calling the interfaces of the Hi3521A HiFB.

1.2.3 Architecture of Graphics Layers for Hi3531A/Hi3531D V100/Hi3521D V100/Hi3536C V100

The Hi3531A/Hi3531D V100/Hi3521D V100/Hi3536C V100 supports two HD display devices (DHD0 and DHD1), one SD display device (DSD0), and five graphics layers (G0−G4). G3 and G4 are cursor layers.

NOTE

- For details about the interfaces and timings supported by each output device, see section 10.2 "VDP" in the Hi3531A H.264 CODEC Processor Data Sheet.
- For details about the interfaces and timings supported by each output device, see section 11.2 "VDP" in the Hi3531D V100 H.265 CODEC Processor Data Sheet
- For details about the interfaces and timings supported by each output device, see section 11.2 "VDP" in the Hi3521D V100 H.265 CODEC Processor Data Sheet
- For details about the interfaces and timings supported by each output device, see section 11.2 "VDP" in the Hi3520D V400 H.265 CODEC Processor Data Sheet
- For details about the interfaces and timings supported by each output device, see section 11.1 "VDP" in the Hi3536C V100 H.265 CODEC Processor Data Sheet

Table 1-3 shows the relationships among the FB device files of the Hi3531A/Hi3531D V100/Hi3521D V100/Hi3536C V100, graphics layers, and output devices.

Table 1-3 Relationships among the FB device files of the Hi3531A/Hi3531D V100/Hi3521D V100/Hi3536C V100, graphics layers, and output devices

FB Device File	Graphics Layer	Corresponding Display Device
/dev/fb0	G0	G0 is displayed only on DHD0.
/dev/fb1	G1	G1 is displayed only on DHD1.
/dev/fb2	G2	G2 is displayed only on DSD0.

FB Device File	Graphics Layer	Corresponding Display Device
/dev/fb3	G3	G3 is a cursor layer and is always displayed on the top. For example, if the video layer of DHD0, G0, and G3 are overlaid, the overlay sequence from bottom to top is as follows: video layer < G0 < G3. G3 can act as the hardware cursor layer or software cursor layer, which is determined by the driver loading parameter **softcursor**. When G2 is used as the hardware cursor layer, its usage is the same as that of other graphic layers. When G2 is used as the software cursor layer, it is operated by calling dedicated software cursor interface of the HiFB
/dev/fb4	G4	G4 is a cursor layer. It has the same features as G3.

NOTE

To display graphics layers, you must configure and enable VO devices by calling the interfaces of the VOU, and operate graphics layers by calling the interfaces of the Hi3531A/Hi3531D V100/Hi3521D V100/Hi3536C V100 HiFB.

1.2.4 Architecture of Graphics Layers for Hi3536D V100

The Hi3536D V100 supports one HD display device (DHD0) and two graphics layers (G0 and G1). G1 is a cursor layer.

NOTE

For details about the interfaces and timings supported by each output device, see section 9.1 "VDP" in the Hi3536D V100 H.265/H.264 Decoder Processor Data Sheet

Table 1-4 shows the relationships among the FB device files of the Hi3536D V100, graphics layers, and output devices.

Table 1-4 Relationships among the FB device files of the Hi3536D V100, graphics layers, and output devices

FB Device File	Graphics Layer	Corresponding Display Device
/dev/fb0	G0	G0 is displayed only on DHD0.
/dev/fb1	G1	G1 is a cursor layer and is always displayed on the top. G1 can act as the hardware cursor layer or software cursor layer, which is determined by the driver loading parameter **softcursor**. When G1 is used as the hardware cursor layer, its usage is the same as that of other graphic layers. When G1 is used as the software cursor layer, it is operated by calling dedicated software cursor interface of the HiFB.

NOTE

To display graphics layers, you must configure and enable VO devices by calling the interfaces of the VOU, and operate graphics layers by calling the interfaces of the Hi3536D V100 HiFB.

2 Recommended Schemes for Graphics Development

2.1 Overview

In the surveillance field, the GUI contents of an output device include:

- Backend OSD: It includes the dividing lines of the displayed picture, channel IDs, and time that specify the display layout of multiple pictures.
- GUI: It includes various menus and progress bars. You can configure devices through the GUI.
- Cursor: It provides user-friendly and convenient menus.

The preceding GUI contents can be implemented through one or more graphics layers. As the Hi3536, Hi3521A, Hi3531A, Hi3531D V100, Hi3521D V100, or Hi3536C V100 provides multiple graphics layers, this chapter describes the following recommended schemes to instruct you to use those graphics layers in a correct, reasonable, and effective manner, satisfying the requirements in various output GUI applications.

2.2 GUI Scheme with a Single Graphics Layer

2.2.1 Introduction

In this scheme, each device displays the backend OSD, GUI, and cursor through one graphics layer. The cursor can also be displayed at a cursor layer.

To be specific, each output device displays its backend OSD and GUI through one graphics layer. The GUI is drawn in an independent buffer, the backend OSD is drawn in the display buffer, and then alpha blending is performed by using the TDE. The cursor is displayed on a cursor layer or the shared graphics layers of the backend OSD and GUI. When the shared graphics layer is used, the cursor can be drawn in the GUI buffer.

The following mechanisms are used in this scheme:

- The backend OSD of each device is drawn in its display buffer.

  For example, the dividing lines, channel ID, or time is drawn in the FB corresponding to each graphics layer.
- Each device has a GUI canvas that is updated partially when the GUI changes.

That is, each device draws the GUI by using an independent buffer (also known as GUI canvas). This canvas needs to be updated partially when the GUI changes.

- The GUI canvas is entirely transferred to the display buffer of the corresponding graphics layer.

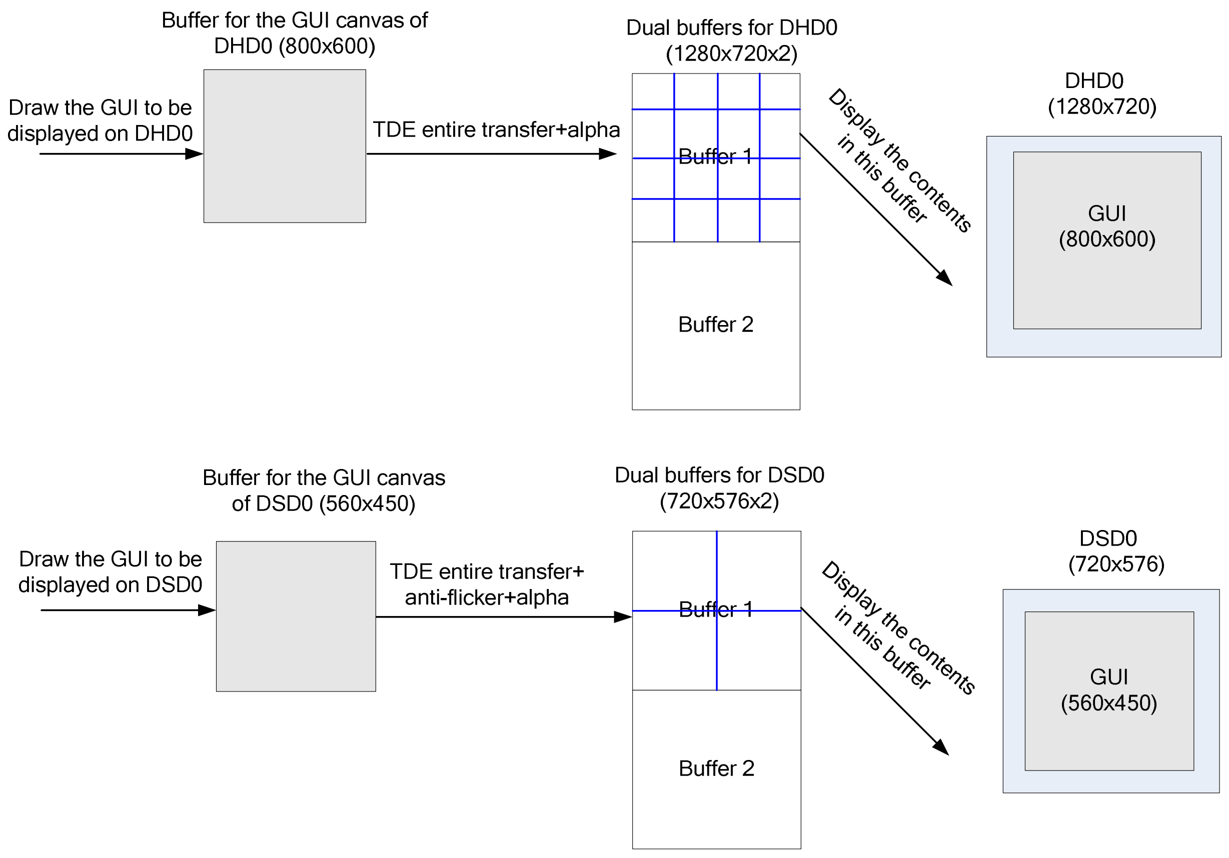
  When a drawn canvas is entirely transferred to the corresponding display buffer, transparent blending between the GUI and OSD can be implemented through the TDE. Each time the GUI or OSD changes, the blended area between the GUI and OSD does not need to be calculated based on the local information, because blending is performed on the entire canvas and OSD.

- Dual display buffers

  To avoid the drawing process being visible when a display buffer is used for drawing and displaying concurrently, dual display buffers, HIFB_LAYER_BUF_DOUBLE or HIFB_LAYER_BUF_DOUBLE_IMMEDIATE in the extended mode is recommended. That is, the HiFB module assigns dual display buffers with the same size for drawing and displaying alternatively. For example, when buffer 2 is displayed on the VOU, buffer 1 is used for drawing. For the FB standard mode, after PAN_DISPLAY and FBIOFLIP_SURFACE of the HiFB are called, the VO device is notified that buffer 1 should be displayed. For the FB extended mode, after FBIO_REFRESH of the HiFB is called, the VO device is notified that buffer 1 should be displayed.

Figure 2-1 shows the schematic diagram of the scheme with a graphics layer

Figure 2-1 Schematic diagram of the scheme with a graphics layer


Buffer for the GUI canvas of DHD0 (800x600)
Dual buffers for DHD0 (1280x720x2)
DHD0 (1280x720)
Draw the GUI to be displayed on DHD0
TDE entire transfer+alpha
Buffer 1
Display the contents in this buffer
GUI (800x600)
Buffer 2
Buffer for the GUI canvas of DSD0 (560x450)
Dual buffers for DSD0 (720x576x2)
DSD0 (720x576)
Draw the GUI to be displayed on DSD0
TDE entire transfer+ anti-flicker+alpha
Buffer 1
Display the contents in this buffer
GUI (560x450)
Buffer 2

When the backend OSD or GUI changes, the OSD or GUI needs to be drawn again in the display buffer.

- When the backend OSD changes (such as the 16-picture dividing line is switched to the 9-picture dividing line), you need to empty the display buffer, draw a new OSD, and then transfer the entire GUI to the display buffer.
- When the GUI changes, you also need to empty the display buffer, draw a new OSD, and then entirely transfer the new GUI to the display buffer.

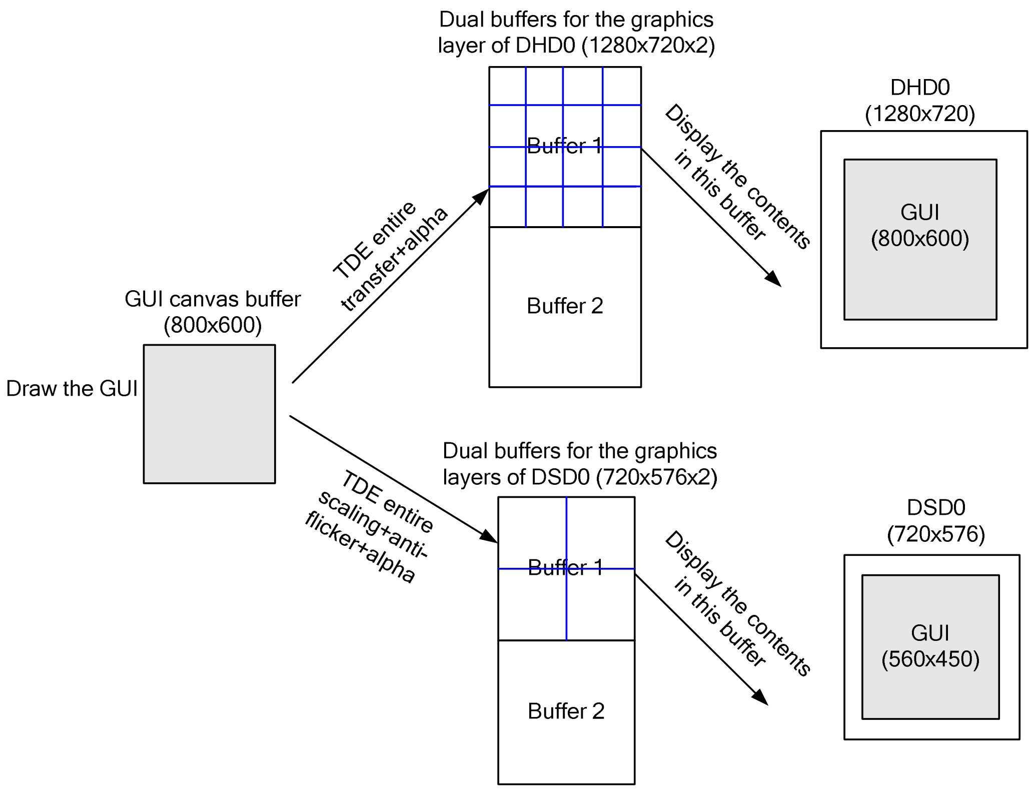
2.2.2 Derivative Scheme

When DSD0 and DHD0 display the same GUI, the preceding scheme can be simplified as a derivative scheme that requires only a GUI canvas buffer.

- The canvas size (800x600) is the same as that of DHD0 GUI. You can prepare a set of pictures based on the specifications (800x600) of DHD0 GUI. Each time the GUI changes, you need to draw the canvas partially. DSD0 GUI is obtained after scaling and anti-flicker operations are performed on the entire canvas. The DSD GUI is not as clear as the DHD GUI.
- For the HD device, each time after the canvas is updated, only the entire transfer operation by using the TDE is required, because the canvas size is the same as the GUI size. For DSD0, each time after the canvas is updated, the entire canvas needs to be scaled by using the TDE based on the size of display buffer for the graphics layer bound to DSD0, because DSD0 is an interlaced device. In addition, anti-flicker is performed.

Figure 2-2 shows the schematic diagram of the derivative scheme.

Figure 2-2 Schematic diagram of the derivative scheme


Dual buffers for the graphics layer of DHD0 (1280x720x2)
Buffer 1
Buffer 2
DHD0 (1280x720)
GUI (800x600)
Display the contents in this buffer
TDE entire transfer+alpha
GUI canvas buffer (800x600)
Draw the GUI
TDE entire scaling+anti-flicker+alpha
Dual buffers for the graphics layers of DSD0 (720x576x2)
Buffer 1
Buffer 2
DSD0 (720x576)
GUI (560x450)
Display the contents in this buffer

2.2.3 Development Process

Development Process of the Scheme with a Graphics Layer

This section takes the GUIs and OSDs of DHD0 and DSD0 as examples. The dividing lines can divide the entire screen into 16 even pictures for DHD0 or four even pictures for DSD0. In addition, DHD0 and DSD0 display the same GUI concurrently.

If the GUI changes, the scheme is implemented as follows:

Step 1 Clear the free buffers for the graphics layers corresponding to DHD0 and DSD0.

This example assumes that the free buffer is buffer 1 and the VO device displays the contents in buffer 2.

Step 2 Draw 16-picture dividing lines in buffer 1 for the graphics layer corresponding to DHD0.

Step 3 Draw 4-picture dividing lines in buffer 1 for the graphics layer corresponding to DSD0.

Step 4 Refresh the canvas partially.

Step 5 Transfer the entire canvas to buffer 1 for the graphics layer corresponding to DHD0 by using the TDE.

During this process, alpha transparency blending is supported for making the GUI semi-transparent.

Step 6 Scale the entire canvas by using the TDE based on the size of buffer 1 for the graphics layer corresponding to DSD0.

During this process, anti-flicker and alpha transparency blending are supported for making the GUI semi-transparent.

Step 7 Call PAN_DISPLAY to instruct DHD0 to display the contents in buffer 1 for the graphics layer bound to DHD0.

Step 8 Call PAN_DISPLAY to instruct DSD0 to display the contents in buffer 1 for the graphics layer bound to DSD0.

----End

2.2.4 Application Scenarios

This scheme applies to the following scenarios:

- Each device has its backend OSD. For example, the backend OSD is 16-picture layout for DHD0, 8-picture layout for DHD1, 4-picture layout for DSD0.
- Two or more output devices display the same or different GUIs.

2.2.5 Advantages and Limitations

The advantages of the scheme with a single graphics layer are as follows:

- Displaying GUIs on multiple devices at the same time.
- Updating the GUI canvas partially, which saves bus bandwidth and improves the TDE capability.

- Implementing transparency blending between the GUI and OSD through an easy-to-use process. Each time the GUI or OSD changes, the blended area between the GUI and OSD does not need to be calculated based on the local information, because blending is performed on the entire canvas and OSD.
- In the derivative scheme, only a set of GUI pictures are required. In this case, the GUI requirements of the devices with different solutions are met and the space of the flash memory is saved.

The limitation on the derivative scheme is as follows:

The GUI displayed on the SD device is not as clear as that on the HD device, because the GUI displayed on the SD device is obtained after being scaled.