MSM7x30 HDMI for Android™

Feature Design Document

80-N2296-1 A

August 6, 2010

Submit technical questions at:
https://support.cdmatech.com/

Qualcomm Confidential and Proprietary

Restricted Distribution. Not to be distributed to anyone who is not an employee of either Qualcomm or a subsidiary of Qualcomm without the express approval of Qualcomm's Configuration Management.

Not to be used, copied, reproduced in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm.

Qualcomm reserves the right to make changes to the product(s) or information contained herein without notice. No liability is assumed for any damages arising directly or indirectly by their use or application. The information provided in this document is provided on an "as is" basis.

This document contains Qualcomm confidential and proprietary information and must be shredded when discarded.

QUALCOMM is a registered trademark of QUALCOMM Incorporated in the United States and may be registered in other countries. Other product and brand names may be trademarks or registered trademarks of their respective owners. CDMA2000 is a registered certification mark of the Telecommunications Industry Association, used under license. ARM is a registered trademark of ARM Limited. QDSP is a registered trademark of QUALCOMM Incorporated in the United States and other countries.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

QUALCOMM Incorporated
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.

Copyright © 2010 QUALCOMM Incorporated.
All rights reserved.

Contents

1 Introduction... 7

1.1 Purpose.. 7
1.2 Scope.. 7
1.3 Conventions 7
1.4 References.. 7
1.5 Technical assistance.. 8
1.6 Acronyms.. 8

2 HDMI Overview 9

3 HDMI High-Level Design 11

3.1 Design considerations.. 11
3.1.1 Maximum concurrency vs. consistent behavior.. 11
3.1.2 Backward compatibility.. 11
3.1.3 Limitations.. 11
3.2 PMEM requirements.. 11
3.2.1 Frame buffer 11
3.2.2 Video 12
3.2.3 Audio 12
3.3 High-level design proposal 13
3.3.1 Architecture 13
3.3.2 HDMI connected/disconnected events 14
3.3.3 UI/graphics mirroring (landscape/portrait).. 14
3.3.4 Video mirroring (landscape only).. 15

4 Android Framework... 16

4.1 Applications using Qualcomm on/off settings.. 16
4.2 HDMI service 16
4.2.1 isHDMIConnected.. 16
4.2.2 setHDMIOutput.. 16
4.2.3 Intents that are broadcast 17
4.3 HDMI daemon 17
4.4 SurfaceFlinger.. 18
4.4.1 orientationChanged.. 18
4.4.2 videoOverlayStarted 18
4.5 Gralloc HAL 19
4.5.1 orientationChanged.. 19
4.5.2 videoOverlayStarted 19
4.6 Overlay HAL public members.. 20

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4.6.1 startChannel ... 20
4.6.2 closeChannel ... 20
4.6.3 setPosition ... 21
4.6.4 setParameter ... 21
4.6.5 setFd ... 21
4.6.6 queueBuffer ... 22
4.6.7 Audio routing ... 22

5 Software Call Flows ... 25

5.1 UI/video mirroring/HDMI events ... 25

6 Audio Driver Design ... 29

6.1 Concurrency combination selection ... 29
6.2 Backward compatibility ... 29
6.3 PCM driver APIs ... 29
6.4 mI2S sound device driver APIs ... 30
6.5 Software components call sequence ... 30

7 HDMI Driver Description ... 32

8 HDMI Driver Capabilities ... 34

8.1 Video output formats ... 34
8.2 Audio formats ... 34
8.3 Hot plug detection ... 34
8.4 EDID ... 34
8.5 HDCP ... 34
8.6 HDMI driver IOCTLs ... 34
8.7 HDMI driver response to events ... 35
8.8 Software component call sequence ... 35

9 Overlay Engine ... 36

9.1 Video/UI ... 36
9.2 Concurrency combination selection ... 36
9.3 Backward compatibility ... 36
9.4 Driver APIs ... 36
9.5 Software components call sequence ... 37

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

Figures

Figure 2-1 HDMI block diagram 9
Figure 3-1 Android HDMI architecture 13
Figure 3-2 MDP 4.1 overlay pipes 15
Figure 4-1 Audio block diagram 22
Figure 4-2 Device switching with HDMI connected 23
Figure 4-3 In-call device switching 24
Figure 5-1 HDMI HPD and EDID events 25
Figure 5-2 HDMI UI display 26
Figure 5-3 HDMI video display 27
Figure 5-4 Audio routing call sequence 28
Figure 6-1 Audio subsystem block diagram 30
Figure 6-2 Audio driver call flow 31
Figure 7-1 HDMI driver (EDID) flow chart 32
Figure 7-2 MDP architecture 33

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

Tables

Table 1-1 Reference documents and standards 7

Revision history

Revision	Date	Description
A	Aug 2010	Initial release

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

1 Introduction

1.1 Purpose

This document describes the software design of the High Definition Multimedia Interface (HDMI) feature in the Android™ platform and Linux kernels on the MSM7630™ chipset. The HDMI feature allows user space applications to project content (graphics/video/audio) on high-definition TV via the HDMI cable.

1.2 Scope

This document is relevant to the Android framework, Android HAL, and kernel drivers. It is assumed that readers have a general knowledge of Android and Linux development.

1.3 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font, e.g., `#include`.

Code variables appear in angle brackets, e.g., <number>.

Commands to be entered appear in a different font, e.g., **`copy a:*.* b:`**.

Parameter types are indicated by arrows:

- → Designates an input parameter
- ← Designates an output parameter
- ↔ Designates a parameter used for both input and output

1.4 References

Reference documents, which may include QUALCOMM®, standards, and resource documents, are listed in Table 1-1. Reference documents that are no longer applicable are deleted from this table; therefore, reference numbers may not be sequential.

Table 1-1 Reference documents and standards

Ref.	Document	
Qualcomm		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

1.5 Technical assistance

For assistance or clarification on information in this guide, submit a case to Qualcomm CDMA Technologies at https://support.cdmatech.com/.

If you do not have access to the CDMATech Support Service website, register for access or send email to support.cdmatech@qualcomm.com.

1.6 Acronyms

For definitions of terms and abbreviations, see [Q1].

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

2 HDMI Overview

The HDMI is a compact audio/video interface for transmitting uncompressed digital data. The HDMI specification defines the protocols, signals, electrical interfaces, and mechanical requirements of the standard. Figure 2-1 shows the overall architecture of the HDMI feature.

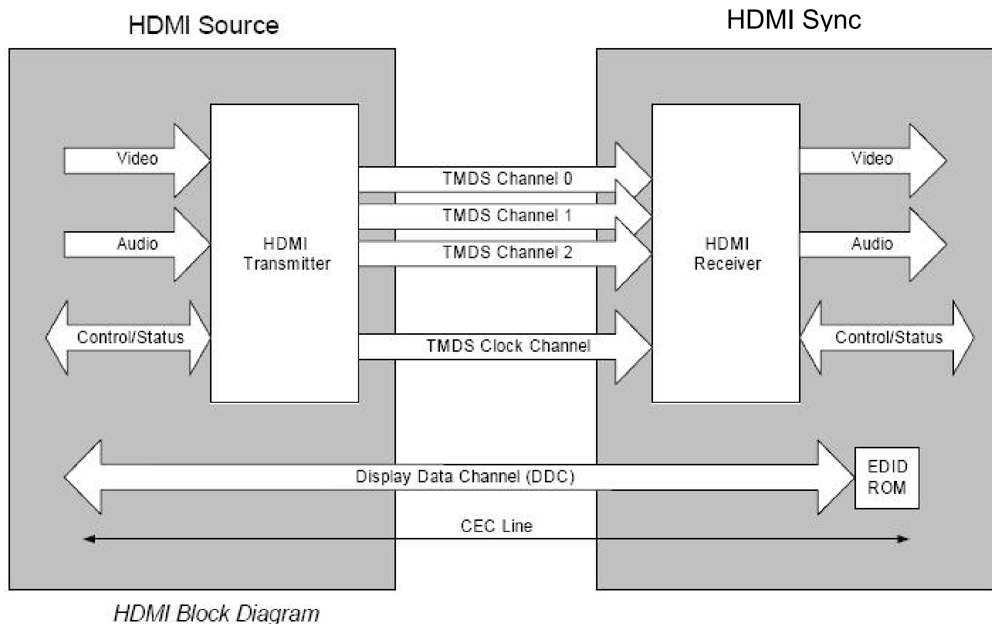


Figure 2-1 HDMI block diagram

The HDMI can carry high-quality multichannel audio data, and all standard and high-definition consumer electronics video formats. Content protection technology is available. The HDMI can also carry control and status information in both directions.

On the MSM7630 chipset, the following HDMI features are supported:

- The HDMI driver supporting analog device ADV7520NK, which is a high-speed, low-power HDMI 1.3-compliant transmitter, operating at up to 80 MHz. The maximum resolution supported is 720p.
- Video playback is mirrored on the primary/HDMI display. Only landscape orientation is supported.
- UI/graphics are mirrored on the primary/HDMI display. The primary display supports WVGA resolution and is scaled to 720p in Landscape mode. In Portrait mode, WVGA resolution on the primary display is 404p x 720p, with black pillars on the side.
- Audio routing is supported. All audio streams except voice calls/ringer are routed to the HDMI. Voice calls can be routed to a phone speaker, headset, and Bluetooth® SCO device.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

- Hot Plug Detect (HPD), HDCP, and Extended Display Identification Data (EDID) are supported by the HDMI driver.
- Supported audio configurations:
  - Channel mode (stereo only)
  - Bits/sample (16 bits)
  - Sample rate up to 48 KHz
  - Channels – Two-channel
  - Codec – All formats that can be supported in Android

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

3 HDMI High-Level Design

3.1 Design considerations

3.1.1 Maximum concurrency vs. consistent behavior

- HDMI audio and FM cannot coexist.
- Video and UI cannot be displayed simultaneously on the HDMI.

3.1.2 Backward compatibility

Not applicable

3.1.3 Limitations

Software limitations of the HDMI solution on MSM7x30 chipsets:

- HDMI-only and LCD-only features are not supported. If the HDMI is connected and enabled, UI/video are mirrored on both the primary and HDMI displays.
- Five-channel audio is not supported.
- Coexistence with other I2S audio devices is not supported.
- The CEC is not supported.
- No end-to-end support is provided for EDID.

3.2 PMEM requirements

3.2.1 Frame buffer

- Primary display (/dev/fb0)
  - WVGA display is supported and requires (800 x 480) x 2 (number of buffers) x 4 (32 bpp color format) = 0x2EE000 (decimal 3072000 bytes).
- HDMI (/dev/fb1)
  - /dev/fb1 must be allocated because of the FB driver initialization requirement; however, it is used when UI/graphics are mirrored on the HDMI. FB1 contains the black pixels needed to create side pillars to display the UI in Portrait mode.
  - (/dev/fb1) requires (1280 x 720) x 1 (number of buffers) x 2 (16 bpp color format) = 0x1C2000 (decimal 1843200 bytes).

- Rotator buffers
  - UI/graphics – Two WVGA buffers are allocated from the existing A-DSP region, and require (800 x 480) x 2 (number of buffers) x 4 (32 bpp color format) = 0x2EE000 (decimal 3072000 bytes).
  - These buffers are used in Landscape mode to rotate buffers before they are sent to the HDMI.
- Video mirroring – Two 720p YUV buffers are allocated from the A-DSP region. These are common between primary and secondary displays. The already allocated PMEM buffers are sufficient. No additional buffers are required for HDMI.

3.2.2 Video

No additional PMEM is required to support HDMI.

3.2.3 Audio

No additional PMEM is required.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

3.3 High-level design proposal

3.3.1 Architecture

Figure 6-1 shows the high-level architecture of the HDMI feature on the Android platform.

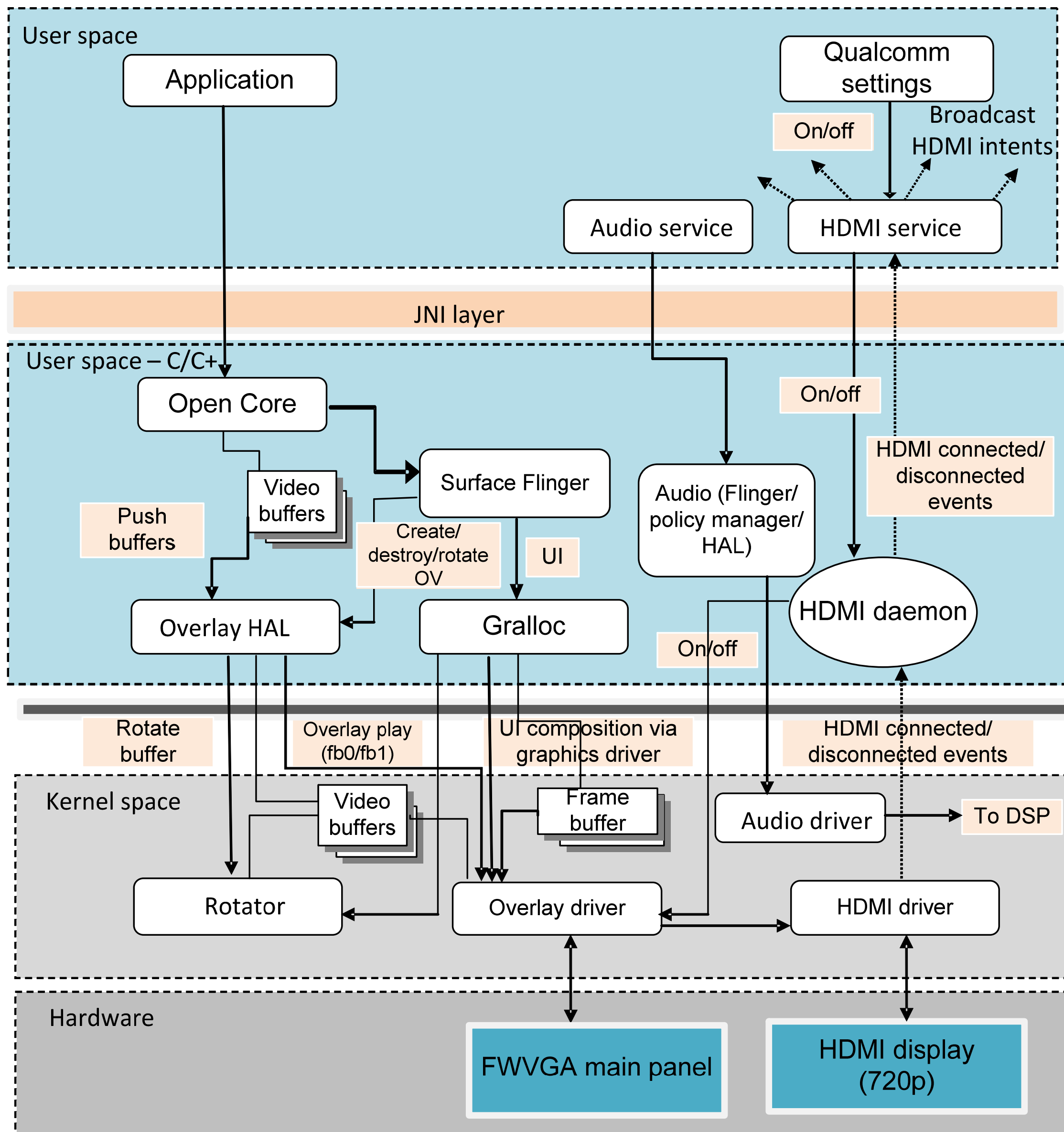


Figure 3-1 Android HDMI architecture

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

3.3.2 HDMI connected/disconnected events

When HDTV is connected to the HDMI port, an HDMI connected event is sent by the HDMI driver to the user space. The HDMI daemon is a background process that is registered for this event. This daemon then forwards these events to the HDMI user space HDMI service. The HDMI service is a Java® service that broadcasts these events as intents. Applications interested in this event can intercept it by registering for it.

Similarly, HDMI disconnected events are also sent to the user space, then broadcast. These events are broadcast by the HDMI service only if the user has enabled the HDMI by selecting the **HDMI On** option from the Qualcomm settings application.

3.3.3 UI/graphics mirroring (landscape/portrait)

The Android application uses a view class object that receives two 32-bit drawable surfaces from SurfaceFlinger. These double buffers are constantly swapped as the application draws and pushes updates to the screen. SurfaceFlinger receives a request from each application and renders each surface in the order received, then composes the final surface.

When an application pushes this surface to SurfaceFlinger, it then uses Open GL® (or 2D library, which is not formally supported in Éclair) to render it to the frame buffer. This process is called surface composition.

When the final surface is composed and ready to be pushed to the screen, it calls the post-frame buffer to the frame buffer driver (/dev/graphics/fb0). This initiates the DMA engine to render its frame buffer to the panel screen. Since the image is now updated to the primary screen, the frame buffer is sent as an input to the rotator hardware to copy to the HDMI overlay buffer (32-bit) for UI mirroring.

Because only one rotator hardware exists and due to scaling limitations in the MDP overlay engine, UI mirroring is only enabled when there is no video playback. This prevents any conflict in using the rotator hardware for UI mirroring.

At system initialization, Gralloc sets the UI mirror overlay (MSMFB_SET_OVERAY), which is disabled by the video overlay. The UI overlay is disabled during video playback in the overlay library/HAL for the reason described above. It is enabled when video playback finishes.

When the HDMI overlay buffer is ready, the buffer (MSMFB_OVERLAY_PLAY) is pushed to the MDP video and graphics pipe to scale to the aspect ratio in Portrait mode, or to the full screen when it is in Landscape mode.

In doing so, the 160 x 90 (16-bit) RGB buffer is used to scale to 1280 x 720 as a background for a black color bar for Portrait mode only. In Landscape mode, this RGB buffer is fetched by the hardware because the foreground HDMI overlay buffer is fully opaque. Also in Landscape mode, the UI must be rotated by an additional 90- degrees for the HDMI. To accomplish this, the two WVGA buffers from the ADSP region are used. These buffers are then fed to the video/graphic pipe on the secondary panel. In Portrait mode, the existing FB0 buffers can be directly used.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

3.3.4 Video mirroring (landscape only)

Video mirroring is much simpler than UI mirroring. The rotator is not used in video mirroring because video is always in Landscape mode (no need to rotate) and is completely abstracted inside the overlay library without the need to interact with SurfaceFlinger or the Gralloc library.

The video renderer, e.g., Open Core, then creates an overlay object and registers it in SurfaceFlinger, which then notifies the overlay HAL with the destination output size and its rotation. When the next video frame is ready to be rendered to the screen, the video renderer simply pushes the buffer to the overlay. If rotation is needed, an offline rotator is used and is then output to a separate rotator buffer via the rotator hardware. This buffer is then used as an input to the primary overlay engine.

For HDMI output, the source video input is intercepted inside the overlay library/HAL and sent directly to the secondary overlay engine to be scaled to the best matching aspect ratio.

Both UI and video mirroring have a separate user setting to control (on/off) the output.

When video playback is finished, SurfaceFlinger destroys the video overlay, and the overlay HAL resumes the UI overlay for HDMI mirroring.

Figure 3-2 illustrates the MDP overlay pipes.

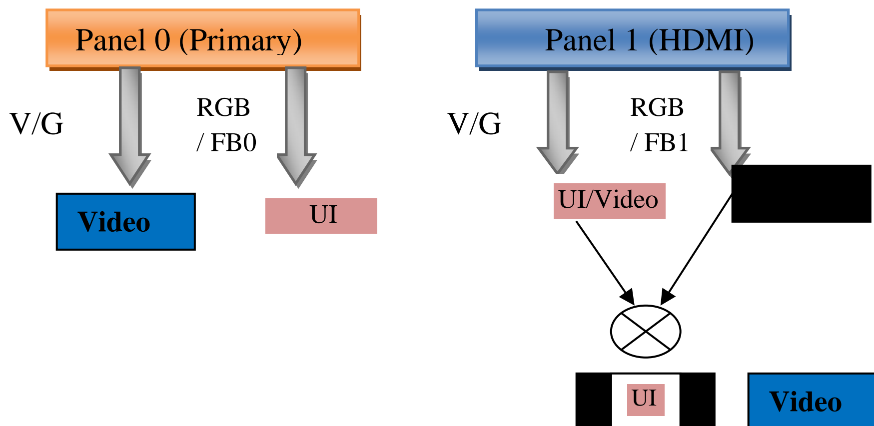


Figure 3-2 MDP 4.1 overlay pipes

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4 Android Framework

4.1 Applications using Qualcomm on/off settings

The Qualcomm settings are in a list menu for choosing display output. The settings use APIs provided by the HDMI service to disable/enable HDMI/LCD output.

4.2 HDMI service

The HDMI service provides HDMI display options. It broadcasts intent HDMI_CONNECTED, HDMI_DISCONNECTED, HDMI_CABLE_CONNECTED, and HDMI_CABLE_DISCONNECTED. The service provides the following APIs through the IHDMIService IDL for applications to use. It is not yet decided whether the service checks the permissions before executing these commands.

4.2.1 isHDMIConnected

This function returns the current state of the HDMI connection.

Parameters

```
boolean isHDMIConnected(void)
```

Return value

This function returns the following values:

- True – Indicates that the HDMI cable is connected
- False – Indicates that the HDMI cable is disconnected

4.2.2 setHDMIOutput

This function sets the HDMI display output.

Parameters

```
void setHDMIOutput(boolean enable)
```

→	enable	Enables the HDMI display output; values are: ▪ True – HDMI display output is enabled ▪ False – HDMI display output is disabled

Return value

None

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4.2.3 Intents that are broadcast

Action: HDMI_CABLE_CONNECTED

Category: Intent.CATEGORY_DEFAULT

Description: This intent is broadcast when the HDMI cable is connected.

Action: HDMI_CABLE_DISCONNECTED

Category: Intent.CATEGORY_DEFAULT

Description: This intent is broadcast when the HDMI cable is disconnected.

Action: HDMI_CONNECTED

Category: Intent.CATEGORY_DEFAULT

Description: This intent is broadcast when users select the HDMI ON option in the Qualcomm settings and the HDMI cable is connected.

Action: HDMI_DISCONNECTED

Category: Intent.CATEGORY_DEFAULT

Description: This intent is broadcast when:

- The user selects the HDMI ON option in the Qualcomm settings and the HDMI cable is disconnected.
- The user selects the HDMI OFF option in the Qualcomm settings.

4.3 HDMI daemon

The HDMI daemon is started by init and remains until Android shuts down. The HDMI daemon catches the events from the kernel and filters for MSM FB (HDMI) events. It also has a server socket with which it listens for incoming requests from the framework.

The framework opens a client on the hdmid socket, which is generated by HDMIService.

The following commands are processed from the framework:

- disable_hdmi
- enable_hdmi

The daemon sends the following commands upon receiving HDMI connected/disconnected events from the kernel:

- hdmi_connected
- hdmi_disconnected

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4.4 SurfaceFlinger

4.4.1 orientationChanged

This function sends a notification to the frame buffer device about orientation changes.

Parameters

```
void orientationChanged(int orientation)
```

→	`orientation`	Specifies the orientation value

Description

SurfaceFlinger notifies the Gralloc frame buffer device about orientation changes. The orientation change is intercepted in the setOrientation function. This function is added in the FramebufferNativeWindow class to notify the Gralloc frame buffer device about the orientation change.

Return value

None

4.4.2 videoOverlayStarted

This function sends a notification to the frame buffer device about the video overlay start.

Parameters

```
void videoOverlayStarted (bool started)
```

→	`started`	Indicates that video overlay has started

Description

To share the PMEM between the HDMI UI and video mirroring, a callback is provided by the frame buffer device to notify it for video overlay start so that it disables the UI output to the HDMI. SurfaceFlinger and the frame buffer native window are changed to accommodate this API.

Return value

None

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4.5 Gralloc HAL

The Gralloc HAL provides the following API for accepting notifications about orientation changes. This API is part of the frame buffer device.

A new thread for HDMI Display output is spawned by the frame buffer device. This thread pushes the UI output onto the TV.

4.5.1 orientationChanged

This function is the frame buffer device's function callback method for notification about orientation changes.

Parameters

```
int (*orientationChanged) (struct framebuffer_device_t* dev,
                           int orientation)
```

→	dev	Specifies the pointer of the frame buffer device
→	orientation	Specifies the orientation value

Return value

This function returns the following values:

- 0 –Success
- -1 – Failure

4.5.2 videoOverlayStarted

This function is the frame buffer device's function callback method for notification about a video overlay start.

Parameters

```
int (*videoOverlayStarted) (struct framebuffer_device_t* dev, int started)
```

→	dev	Specifies the pointer of the frame buffer device
→	started	Specifies whether the video overlay is started

Return value

This function returns the following values:

- 0 –Success
- -1 – Failure

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4.6 Overlay HAL public members

The overlay HAL mirrors the video output to the primary display, as well as the HDMI output.

The overlay HAL uses a new basic overlay function for creating and using the overlay channel. The following classes are provided:

- Overlay – This class is provided so single threads can use the overlay.
- OverlayControlChannel – This class is provided for creating and using an overlay control channel. Overlay control device functions are implemented by the OverlayControl channel.
- OverlayDataChannel – This class is provided for creating and using the overlay data channel. Overlay data device functions are implemented by the OverlayControl channel.

OverlayControlChannel and OverlayDataChannel objects manage overlay connections, while the Gralloc uses the Overlay class to push buffers to the HDMI.

The following functions are part of the Overlay class.

4.6.1 startChannel

This function starts an overlay channel (control and data channel in one thread).

Parameters

```
bool startChannel(int w, int h, int format, int fbnum);
```

→	w	Width of the video
→	h	Height of the video
→	format	Format
→	fbnum	Frame buffer number on which to overlay the buffer

Return value

This function returns True upon success; otherwise, it returns False.

4.6.2 closeChannel

This function closes the overlay channel (control and data).

Parameters

```
bool closeChannel(void);
```

Return value

This function returns True upon success; otherwise, it returns False.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4.6.3 setPosition

This function sets the destination rectangle to be displayed.

Parameters

```
bool setPosition(int x, int y, uint32_t w, uint32_t h);
```

→	x	X axis
→	y	Y axis
→	uint32_t w	Top left width
→	uint32_t h	Top left height

Return value

This function returns True upon success; otherwise, it returns False.

4.6.4 setParameter

This function sets the orientation of the destination.

Parameters

```
bool setParameter(int param, int value);
```

→	param	Rotation parameter
→	value	Value

Return value

This function returns True upon success, otherwise, False.

4.6.5 setFd

This function sets the source file descriptor.

Parameters

```
bool setFd(int fd);
```

→	fd	Integer representing the file descriptor

Return value

This function returns True upon success, otherwise, False.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4.6.6 queueBuffer

This function sets the queue buffer, with an offset.

Parameters

```
bool queueBuffer(uint32_t offset);
```

→	uint32_t offset	Offset for queue buffer

Return value

This function returns True upon success, otherwise, False.

4.6.7 Audio routing

Android user space modules are responsible for routing the appropriate audio streams to the HDMI devices when connected. Figure 4-1 shows the high-level architecture of the audio routing.

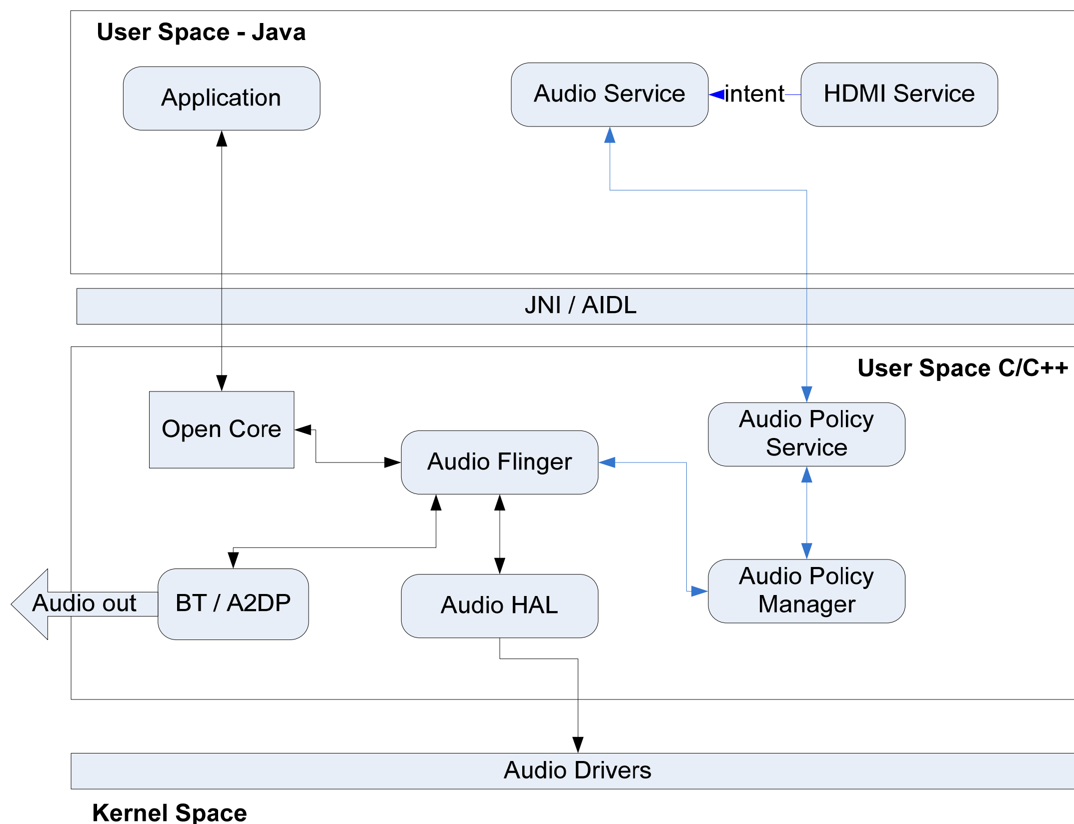


Figure 4-1 Audio block diagram

The following components contribute to proper audio routing.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4.6.7.1.1 Audio service

The audio service is one of the runtime services started by the system server and managed by the service manager. The audio service registers for the HDMI intents, HDMI_CONNECTED and HDMI_DISCONNECTED. On receipt of an intent from the HDMI service, this module notifies the corresponding native components to take the appropriate action.

4.6.7.1.2 Audio policy service

The audio policy service is a corresponding audio service in the native space. This component receives requests from the audio service over the JNI interface and passes them to the audio policy manager for further processing.

4.6.7.1.3 Audio policy manager

The audio policy manager is the brain of audio routing. It maintains a repository of all connected devices and their priorities. This module also maintains the current phone state/strategy, e.g., in-call. The audio policy manager also communicates with AudioFlinger and Bluetooth devices to set the active output device by routing the bit stream to the active output device. Figure 4-2 and Figure 4-3 show the logic for device switching when HDMI is connected.

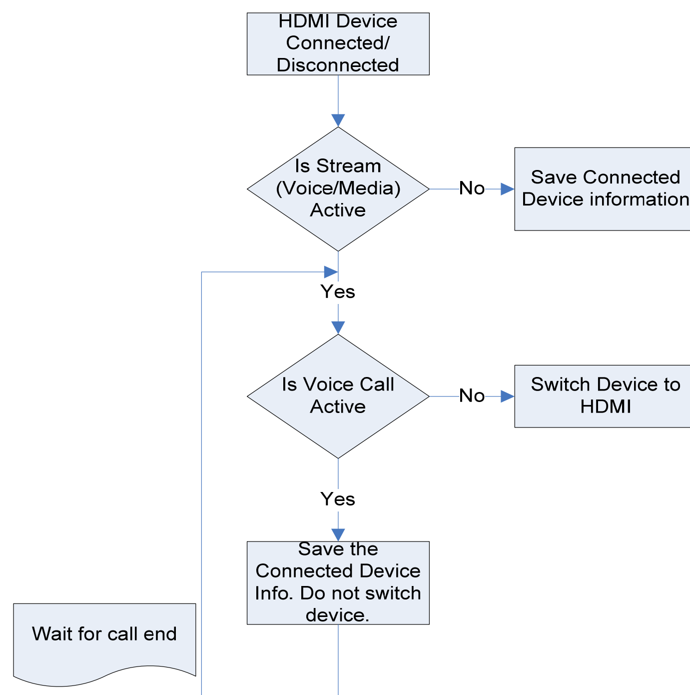


Figure 4-2 Device switching with HDMI connected

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

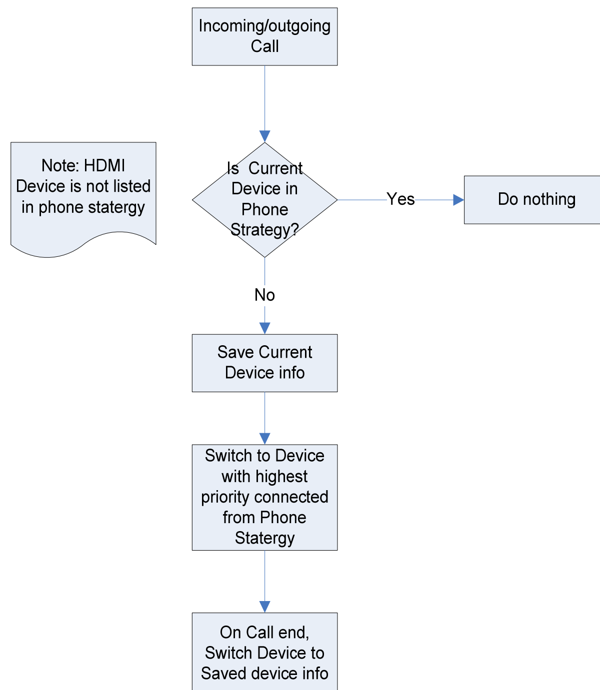


Figure 4-3 In-call device switching

4.6.7.1.4 AudioFlinger

AudioFlinger is the main interface between the audio bit stream and output devices. This module interacts with the mixer to mix all of the audio streams and render PCM audio to the active output device via the audio HAL. AudioFlinger also registers with the audio policy manager to be notified when the active output device changes. These changes are propagated to the HAL layer for actual rerouting.

4.6.7.1.5 Audio HAL

This module mainly communicates with audio drivers and manages the I/O devices. This module initializes and opens output devices to render the audio bit streams received from AudioFlinger. This module is also responsible for switching physical devices. On receipt of the device switch request (setParameter()) from AudioFlinger, this module deactivates the current device, activates the new device, and begins rendering new data to the activated device.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

5 Software Call Flows

5.1 UI/video mirroring/HDMI events

Figure 5-1, Figure 5-2, Figure 5-3, and Figure 5-4 explain the call flows from the applications to the drivers.

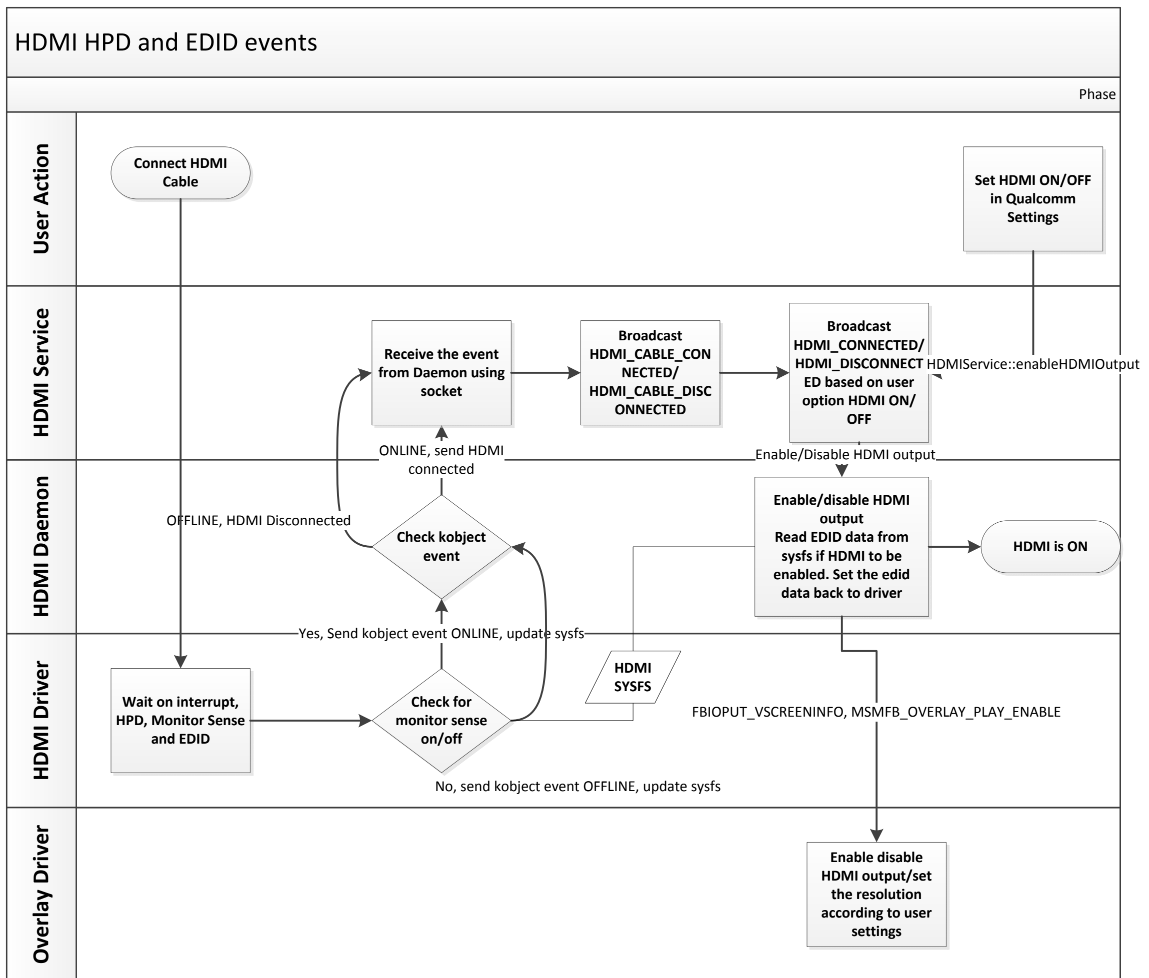


Figure 5-1 HDMI HPD and EDID events

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

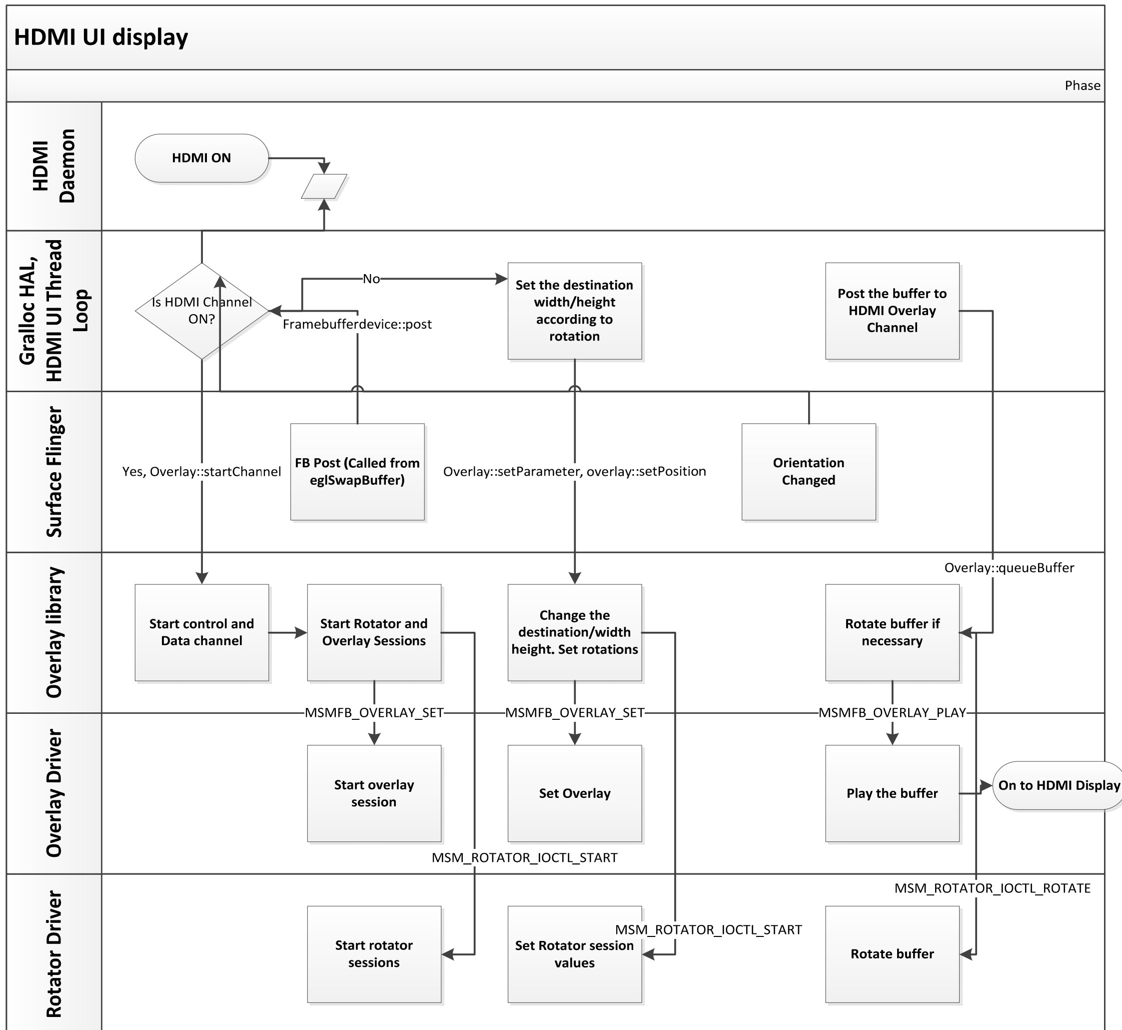


Figure 5-2 HDMI UI display

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

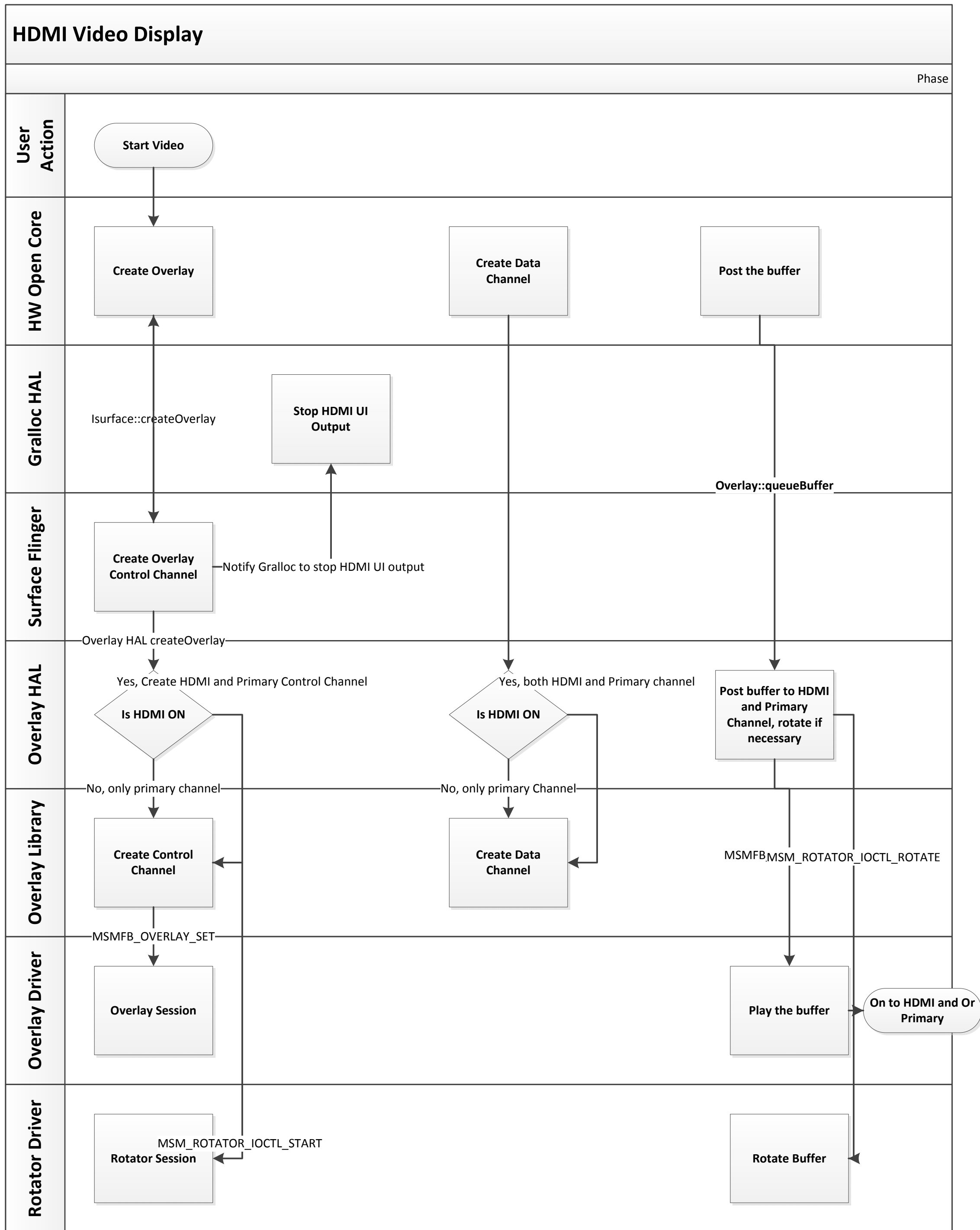


Figure 5-3 HDMI video display

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

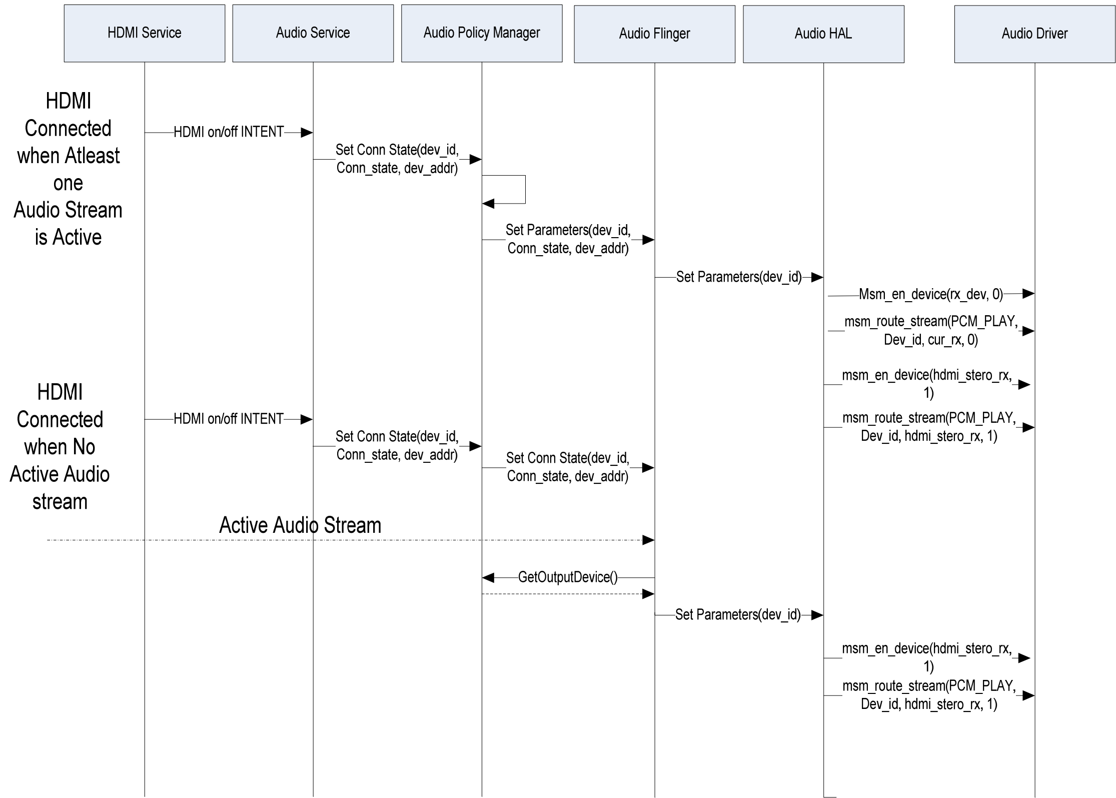


Figure 5-4 Audio routing call sequence

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

6 Audio Driver Design

The audio bitstream is decoded by the software decoders. The AudioFlinger sends the mixed PCM audio to the driver for rendering via the HDMI device. Two drivers are used to configure and control the audio path:

- The PCM driver sends the PCM data to the DSP for rendering to an output device.
- The mI2S sound device driver configures the mI2S lines that send the data to the HDMI device.

6.1 Concurrency combination selection

[TBD]

6.2 Backward compatibility

[TBD]

6.3 PCM driver APIs

The PCM driver exposes the following interfaces to control the data path:

- open – Opens the decoder instance
- release – Releases the decoder instance
- write – Sends the PCM data to the DSP
- ioctl – Controls the input/output interface; the following IOCTLs are supported:
  - AUDIO_START – Starts decoding
  - AUDIO_STOP – Stops decoding
  - AUDIO_FLUSH – Flushes the buffers
  - AUDIO_GET_SESSION_ID – Gets the decoder identifier
  - AUDIO_GET_CONFIG – Gets the default values of the driver’s buffer size, buffer count, sampling rate, and channel mode
  - AUDIO_SET_CONFIG – Sets the driver’s buffer size, buffer count, sampling rate, and channel mode
  - AUDIO_SET_VOLUME – Sets the volume configuration parameter
  - AUDIO_SET_PAN – Sets the pan configuration parameter

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

6.4 mI2S sound device driver APIs

From a user's perspective, HDMI is just another output device. The mI2S driver performs a series of steps that enable the hardware block, configure the DSP, and apply the calibration values. The mI2S driver registers with the audio routing driver to export its control interface, as well as the names of the devices under its control, in this case HDMI. When the audio HAL issues a command to the HDMI device, the audio routing driver propagates the command through the control interface.

The mI2S driver exposes the following interface, which is similar to the existing sound devices:

- open – Opens the mI2S driver
- close – Closes the mI2S driver
- set_freq – Sets the sampling rate; currently, only 48 KHz is supported

6.5 Software components call sequence

The interaction between the audio modules is shown in Figure 6-1.

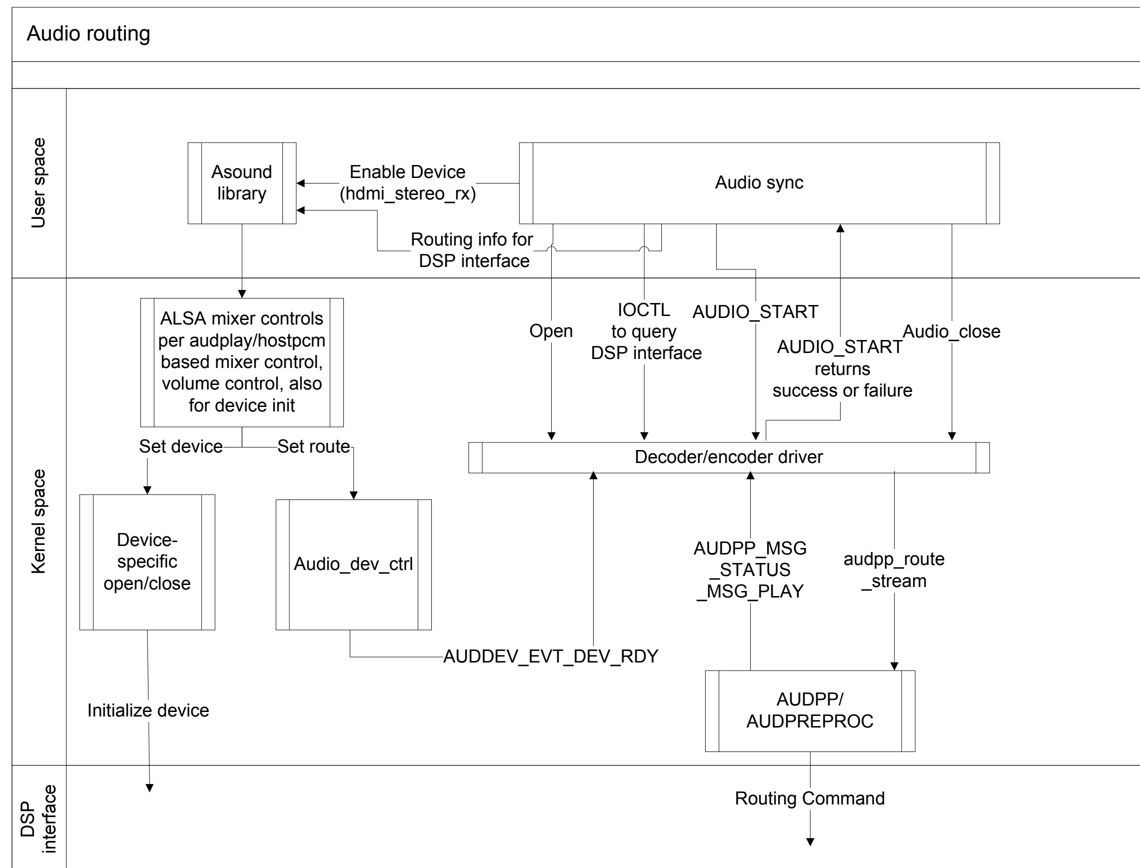
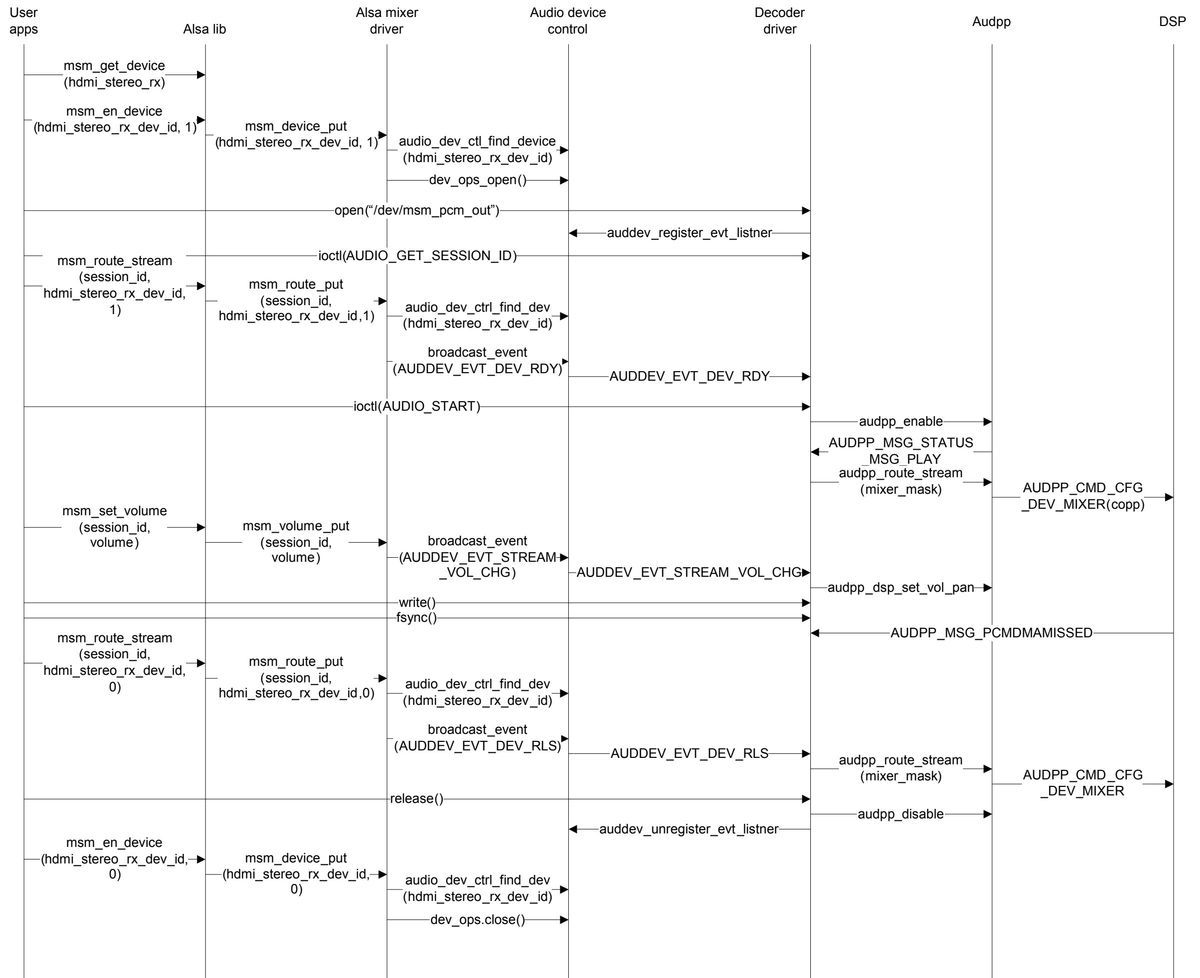


Figure 6-1 Audio subsystem block diagram

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

1 The call flow to set up and configure the PCM driver and the mI2S driver is shown in Figure 6-2.



2

3 **Figure 6-2 Audio driver call flow**

4

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

7 HDMI Driver Description

The ADV7520NK HDMI chip is connected on the Lumos LCD card and is connected through the I2C interface to the MSM7x30 chipset. It is connected at the I2C slave address of 0x72.

To operate the AD9387NK/ADV7520NK chip, it is necessary to monitor the HPD signal and power up the part when the HPD becomes high. The best method to determine when the HPD is high is to use the interrupt system. After the HPD is detected and the AD9387NK/ADV7520NK chip is powered up, it buffers segment 0 of the EDID. The EDID memory address is programmable and controlled by register 0x43 of the main register map. The default setting is 0x7E. Based on the EDID read, it sets up audio/video system configuration parameters, such as audio channels, monitor resolution, and pixel frequency.

The HDMI driver/EDID process is shown in Figure 7-1.

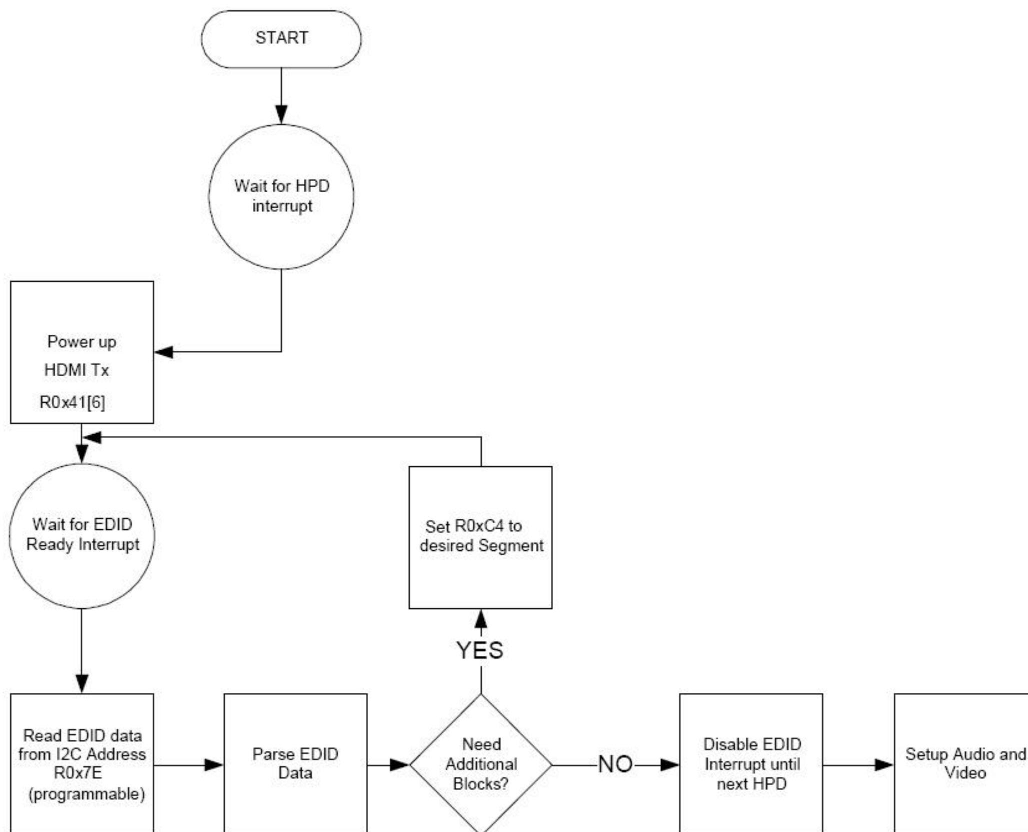


Figure 7-1 HDMI driver (EDID) flow chart

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

If a device, e.g., HDTV, fails to provide EDID, the Android user space is notified with HPD detected. In this case, the driver uses the default resolution (720p) and passes it to the HDMI daemon in the Android user space.

The HDMI device driver is registered as a frame buffer driver on the RGB interface in the current Linux MSM frame buffer architecture. Figure 7-2 shows the MDP architecture.

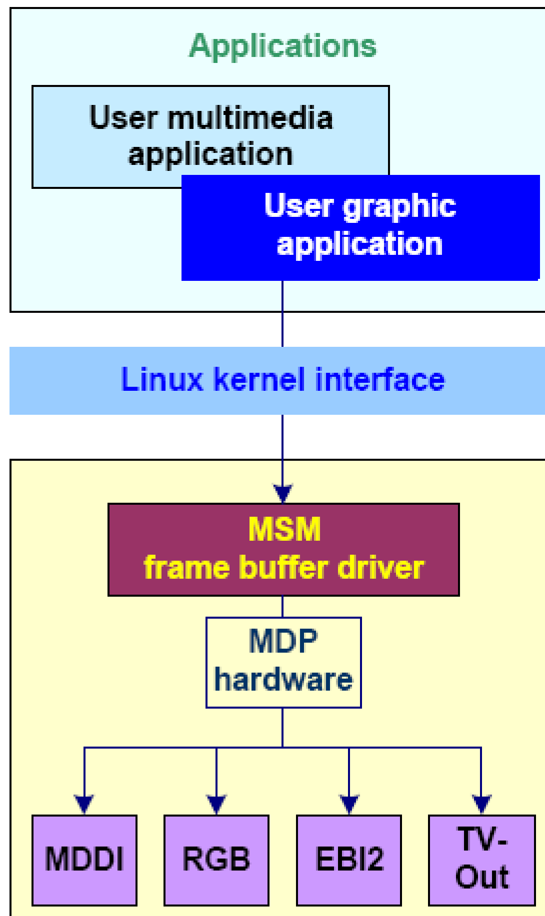


Figure 7-2 MDP architecture

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

8 HDMI Driver Capabilities

8.1 Video output formats

The HDMI driver currently supports a 24-bit RGB444 input format at 480p/720p resolution.

8.2 Audio formats

The HDMI driver currently supports dual-channel audio at a frequency of 44.1 KHz.

8.3 Hot plug detection

The HDMI driver supports the detection of hot-plug interrupts on cable connect/disconnect events and on powering on/off the HDMI chip based on the same.

8.4 EDID

The HDMI driver supports reading EDID data from the display. Once the HPD is detected and ADV7520 is powered up, it buffers segment 0. The system can request other segments by programming register R0xC4. An interrupt bit (R0 x96[2]) indicates the completion of EDID rebuffering. A normal HDMI system will have only two EDID blocks and will only use segment 0. The first EDID block is always a base EDID 1.3 structure; the second EDID block is usually the CEA extension defined in the CEA 861D.

The EDID information passed to the Android user space is as follows:

- Horizontal resolution
- Vertical resolution

Additional data can easily be added to kObject and passed to the user space, if required.

8.5 HDCP

High-bandwidth Digital Content Protection (HDCP) support is not yet implemented.

8.6 HDMI driver IOCTLs

The HDMI driver exports two IOCTLs to the user space application:

- GET EDID – Gives EDID information to the user space
- GET_HPD – Reads the status of the Hot Plug Detect Interrupt register

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

8.7 HDMI driver response to events

The HDMI driver sends kObject events to the HDMI daemon in the user space when HPD detect and EDID read events occur. The driver uses the following kObject events:

- KOBJ_ADD – Init KObject
- KOBJ_REMOVE – Deinit kobjectKOBJ _ONLINE, HPD connected
- KOBJ_OFFLINE – HPD disconnected

8.8 Software component call sequence

See Chapter 5.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

9 Overlay Engine

The MDP4 overlay engine provides two mixers (mixer1 and mixer2) with four blending pipes, two RGB pipes, and two VG pipes. The RGB pipe can only carry graphic image data; however, the VG pipe can carry either video or graphic image data. Currently, only one blending stage exists per mixer.

RGB1 and VG1 are blended at mixer1, and RGB2 and VG2 are blended at mixer2. The RGB pipe is the base layer and the VG pipe is the ground layer during the blending process. Mixer1 output is fed into the embedded primary display, while mixer2 output is fed into the external HDMI TV panel.

9.1 Video/UI

The RGB2 pipe is the base layer (background) that carries RGB format data. The VG2 pipe is the foreground layer, which is shared by the UI and video clip. Both the UI and video clip share the VG2 pipe in a mutually exclusive manner. Since video clips have a higher priority than the UI, the UI can be preempted any time by a video clip. The UI is a mirror of the primary display.

9.2 Concurrency combination selection

[TBD]

9.3 Backward compatibility

None

9.4 Driver APIs

The overlay driver exports three IOCTLs to the user space application:

- Overlay_set can be used for new play or existing play with new updated parameters. Overlay_set performs parameter checking and allocates the pipe for future play if the request is for new play and no failures occur during parameter checking. A new play ID is returned to the user space application for new play requests. An existing play request only performs parameter checking and returns the same play ID if no failures occur during parameter checking.
- Overlay_play configures the pipe and blending engine according to parameters embedded at the play request. The command kicks off the blend engine, which triggers the DMA to start fetching data from the frame buffer. Fetched data will undergo pipe processing and a mixer operation. The blended data feeds directly to the display through an LCDC or MDDI link.
- Overlay_unset tears down the blending engine and frees the pipe back to the pool.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

9.5 Software components call sequence

See Chapter 5.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION