

DRM KMS overview

Stable: 02.10.2019 - 14:50 / Revision: 02.10.2019 - 14:46

This article gives information about the Linux[®] DRM/KMS display, graphic and composition framework.

Contents

1 Framework purpose	2
2 System overview	3
2.1 Component description	3
2.2 API description	4
3 Configuration	4
3.1 Kernel configuration	4
3.2 Device tree configuration	5
4 How to use the framework	5
4.1 modetest (DRM/KMS test tool)	5
4.1.1 modetest help	5
4.1.2 Show display overall status	6
4.1.3 Set a particular video mode	10
4.1.4 Send a test pattern to a display connector	11
4.1.5 Test the display of 2 layers	11
4.1.6 Test Vsync	11
4.2 kmscube (DRM/KMS OpenGL GPU test tool)	12
4.3 Tips	12
4.3.1 How to get the name and current status of a DRM connector	12
5 How to trace and debug the framework	13
5.1 How to monitor	13
5.1.1 How to monitor with modetest	13
5.1.2 How to monitor with debugfs	13
5.1.3 How to monitor with /sys/class/drm	14
5.1.4 How to monitor the display framerate	14
5.1.5 How to monitor the DMA-BUF and CMA memory usage	15
5.2 How to trace	16
5.2.1 Enable DRM/KMS traces with the sysfs	16
5.2.2 Enable DRM/KMS traces with the kernel dynamic debug	17
5.3 How to debug	17
5.3.1 Errors	17
6 Source code location	17
6.1 Kernel space	17
6.2 User space	18
7 To go further	18
8 References	18

1 Framework purpose

The purpose of this article is to provide a quick overview of the Direct Rendering Manager^[1] & Kernel Mode Setting^[2] Linux[®] framework, called the "**DRM/KMS**" framework for short, giving some hints on its architecture, configuration, usage, debug and related use cases.

The DRM/KMS framework is dedicated to the management of the display, graphic and composition subsystems. With the help of other Linux multimedia frameworks and applications, the DRM/KMS framework is typically used:

- to compose animated contents taking advantages of the hardware acceleration.
- to control both display interfaces and external displays including their settings (resolution, frequency, multi-screen...).
- to display this animated contents on display panels or HDMI outputs.

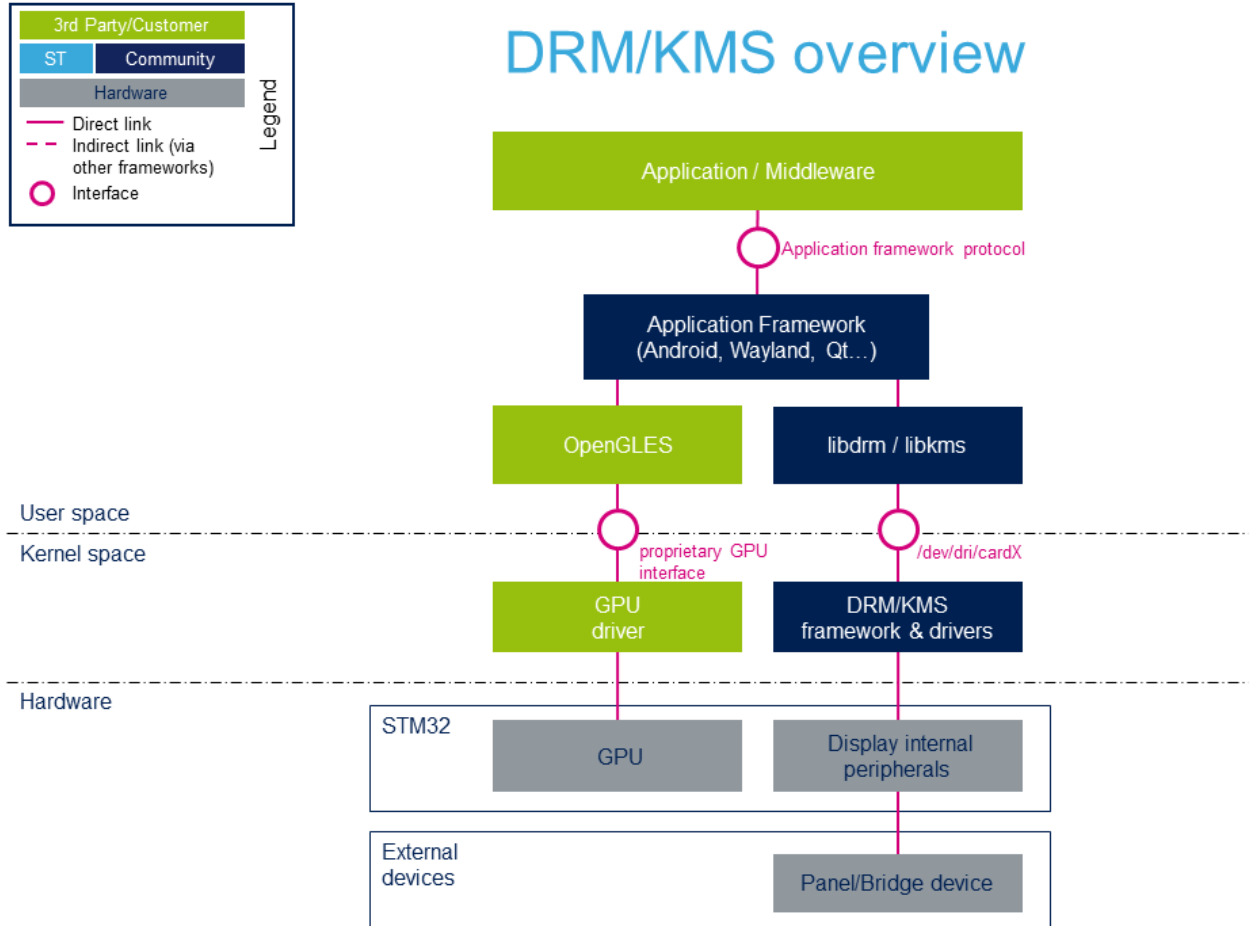
The DRM/KMS framework offers:

- a **kernel level interface driver** for display, graphic and composition internal and external hardware peripherals.
- a **userland level interface**, with the help of libdrm/libkms libraries, to get access to the related hardware features, configuration and hardware acceleration.

Examples of use cases involving the DRM/KMS framework:

- Display an animated content on a DSI or RGB panel.
- Configure the HDMI output to a given resolution and frame rate.
- Activate the HDMI output among several available display outputs.
- Create and manage memory buffers dedicated to the display, which contains graphical content (animations, 3D objects, videos...).
- Get display timing information to efficiently adjust the speed of animated content.

2 System overview



2.1 Component description

From Application/Middleware to Hardware

- **Application/Middleware** (User space)

The Application/Middleware "client" relying on the application framework and its related protocol. This client can be either a traditional application or a specific middleware.

- **Application Framework** (User space)

The application framework (Android, Wayland, Qt...) libraries implementing the related application framework protocol, including the display framework with the libdrm/libkms libraries and the GPU-based composition with the [OpenGLES](#) libraries.

- **OpenGLES** (User space)

The [OpenGL ES](#) libraries used by the application framework for the GPU-based composition and by the application for creating animated contents.

- **libdrm/libkms** (User space)

The libdrm and libkms libraries are used by the application framework to configure, control and refresh the display paths and content. Incorporated wrapper functions are used to access the DRM/KMS ioctls.

- **GPU driver** (Kernel space)

The Linux kernel driver used to transfer OpenGL ES instructions to the [GPU hardware block](#).

- **DRM/KMS framework & drivers** (Kernel Space)

The DRM/KMS Linux kernel framework and related drivers used to access the [display hardware block](#) and the related user space API implementation.

- **GPU** (Hardware)

The [GPU hardware block](#).

- **Display internal peripherals** (Hardware)

All internal display peripheral hardware blocks like the [LTDC internal peripheral](#) and the [DSI internal peripheral](#).

- **Panel/Bridge device** (Hardware)

The physical [Panel or Bridge device](#) to display contents.

2.2 API description

The DRM/KMS framework userland API documentation:

- for the libdrm/libkms libraries : See [libdrm source code and documentation](#)^[3]
- for the kernel/userland (uapi): See "[Linux GPU Driver Developer's Guide / Userland interfaces](#)"^[4]

The internal DRM/KMS kernel framework API is documented in the official "[Linux GPU Driver Developer's Guide / DRM Internals](#)"^[5]

3 Configuration

The objective of this chapter is to explain how to configure the Linux kernel and device tree to have the DRM /KMS framework and related drivers activated.

3.1 Kernel configuration

The DRM/KMS framework and related drivers are activated by default in all ST deliveries. Nevertheless, if a specific configuration is needed, this section indicates how the DRM/KMS framework and drivers can be activated/deactivated in the kernel.

Activate the DRM/KMS framework in kernel configuration with Linux Menuconfig tool: [Menuconfig or how to configure kernel](#)

```
Device Drivers --->
  Graphics support --->
    [*] Direct Rendering Manager (XFree86 4.1.0 and higher DRI support) --->
```

Note: Most of the important kernel configuration flags related to the DRM/KMS framework are enabled by default in **multi_v7_defconfig** file.

The internal display peripherals (LTDC, DSI) and the related display panels/bridges, connected external peripherals must also be enabled. See the example below:

```
Device Drivers --->
  Graphics support --->
    [*] DRM Support for STMicroelectronics SoC Series
    [*] STMicroelectronics specific extensions for Synopsys MIPI DSI
  Display Panels --->
    [*] support for simple panels
    [*] Orise Technology otm8009a 480x800 dsi 2dl panel
    [*] Raydium RM68200 720x1280 DSI video mode panel
  Display Interface Bridges --->
    [*] Silicon Image sii902x RGB/HDMI bridge
```

3.2 Device tree configuration

Refer to the [LTDC device tree configuration](#) and [DSI device tree configuration](#) articles for the complete documentation of the LTDC, DSI, display panel and bridge configuration supported by the Linux kernel device tree mechanism.

4 How to use the framework

4.1 modetest (DRM/KMS test tool)

The tool **modetest** provided by the libdrm library^[3] is useful to:

- List all display capabilities: CRTCs, encoders & connectors (DSI, DPI, HDMI, ...), planes, modes, etc...
- Perform basic tests: display a test pattern, display 2 layers, perform a vsync test
- Specify the video mode: resolution and refresh rate



You can **get all display capabilities with modetest** whatever was running on top of DRM/KMS Linux framework (like [Wayland](#) [Weston](#)).

But if you want to **test displays with modetest**, the framework has to be stopped.

For instance, you can [stop Weston](#) with the following command:

```
Board $> systemctl stop weston
```

4.1.1 modetest help

You can get the help of **modetest** with the following command:

```
Board $> modetest --help
```

Result example:

```
~# modetest --help
usage: modetest [-acDdefMPpsCvw]

Query options:
  -c      list connectors
  -e      list encoders
  -f      list framebuffers
  -p      list CRTCs and planes (pipes)

Test options:
  -P <plane_id>@<crtc_id>:<w>x<h>[+<x>+<y>][*<scale>][@<format>] set a plane
  -s <connector_id>[,<connector_id>][@<crtc_id>]:<mode>[-<vrefresh>][@<format>] se
  -C      test hw cursor
  -v      test vsynced page flipping
  -w <obj_id>:<prop_name>:<value> set property
  -a      use atomic API

Generic options:
  -d      drop master after mode set
  -M module      use the given driver
  -D device      use the given device

Default is to dump all info.
```



If you want to know more about DRM/KMS encoders, connectors, CRTCs and planes, refer to the presentations suggested in the [To go further](#) chapter.

4.1.2 Show display overall status

Get the status of all display capabilities:

```
Board $> modetest -M stm
```

Result example:

```
~# modetest -M stm
Encoders:
id      crtc   type   possible crtcs  possible clones
28      33      DPI    0x00000001     0x00000000
30      0       DSI    0x00000001     0x00000000

Connectors:
id      encoder status      name          size (mm)    modes  encoders
29      28      connected   HDMI-A-1     700x390     10     28
  modes:
    name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
```

```

1920x1080 30 1920 2008 2052 2200 1080 1084 1089 1125 74250 flags: phsync, pvsync; type:
1920x1080 25 1920 2448 2492 2640 1080 1084 1089 1125 74250 flags: phsync, pvsync; type:
1920x1080 24 1920 2558 2602 2750 1080 1084 1089 1125 74250 flags: phsync, pvsync; type:
1280x720 60 1280 1390 1430 1650 720 725 730 750 74250 flags: phsync, pvsync; type: drive
1280x720 60 1280 1390 1430 1650 720 725 730 750 74250 flags: phsync, pvsync; type: drive
1280x720 50 1280 1720 1760 1980 720 725 730 750 74250 flags: phsync, pvsync; type: drive
1280x720 50 1280 1720 1760 1980 720 725 730 750 74250 flags: phsync, pvsync; type: drive
800x600 75 800 816 896 1056 600 601 604 625 49500 flags: phsync, pvsync; type: driver
720x576 50 720 732 796 864 576 581 586 625 27000 flags: nhsync, nvsync; type: driver
720x480 60 720 736 798 858 480 489 495 525 27000 flags: nhsync, nvsync; type: driver
props:
  1 EDID:
    flags: immutable blob
    blobs:
    value:
      00ffffffffffff004c2d920900000000
      0a160103804627780aee91a3544c9926
      0f5054bdef80714f81c0810081809500
      a9c0b3000101023a801871382d40582c
      4500a05a0000001e662156aa51001e30
      468f3300a05a0000001e000000fd0018
      4b0f5117000a202020202020000000fc
      0053414d53554e470a20202020200152
      020330f14d901f041305140312202122
      07162309070783010000e2000f72030c
      001000b82d20d0080140073f405090a0
      011d80d0721c1620102c2580a05a0000
      009e011d8018711c1620582c2500a05a
      0000009e011d00bc52d01e20b8285540
      a05a0000001e011d007251d01e206e28
      5500a05a0000001e0000000000000097
  2 DPMS:
    flags: enum
    enums: On=0 Standby=1 Suspend=2 Off=3
    value: 0
  5 link-status:
    flags: enum
    enums: Good=0 Bad=1
    value: 0
  6 non-desktop:
    flags: immutable range
    values: 0 1
    value: 0
  19 CRTC_ID:
    flags: object
    value: 33
31 0 connected DSI-1 52x86 1 30
modes:
name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
480x800 50 480 600 664 784 800 814 828 842 33000 flags: ; type: preferred, driver
props:
  1 EDID:
    flags: immutable blob
    blobs:
    value:
  2 DPMS:
    flags: enum
    enums: On=0 Standby=1 Suspend=2 Off=3
    value: 0
  5 link-status:
    flags: enum
    enums: Good=0 Bad=1
    value: 0
  6 non-desktop:
    flags: immutable range

```

```

        values: 0 1
        value: 0
19 CRTC_ID:
    flags: object
    value: 0

CRTC's:
id      fb      pos      size
33      0      (0,0)   (1280x720)
1280x720 60 1280 1390 1430 1650 720 725 730 750 74250 flags: phsync, pvsync; type: drive
props:
20 ACTIVE:
    flags: range
    values: 0 1
    value: 1
21 MODE_ID:
    flags: blob
    blobs:

    value:
        0a22010000056e05960572060000d002
        d502da02ee0200003c00000005000000
        40000000313238307837323000000000
        00000000000000000000000000000000
        00000000
18 OUT_FENCE_PTR:
    flags: range
    values: 0 18446744073709551615
    value: 0
25 GAMMA_LUT:
    flags: blob
    blobs:

    value:
26 GAMMA_LUT_SIZE:
    flags: immutable range
    values: 0 4294967295
    value: 256

Planes:
id      crtc      fb      CRTC x,y      x,y      gamma size      possible crtcs
32      0      0      0,0      0,0      0      0      0x00000001
formats: AR24 XR24 RG24 RG16 AR15 XR15 AR12 XR12 C8
props:
7 type:
    flags: immutable enum
    enums: Overlay=0 Primary=1 Cursor=2
    value: 1
16 FB_ID:
    flags: object
    value: 0
17 IN_FENCE_FD:
    flags: signed range
    values: -1 2147483647
    value: -1
19 CRTC_ID:
    flags: object
    value: 0
12 CRTC_X:
    flags: signed range
    values: -2147483648 2147483647
    value: 0
13 CRTC_Y:
    flags: signed range
    values: -2147483648 2147483647
    value: 0
14 CRTC_W:

```



```

        flags: range
        values: 0 2147483647
        value: 1280
15 CRTC_H:
        flags: range
        values: 0 2147483647
        value: 720
8 SRC_X:
        flags: range
        values: 0 4294967295
        value: 0
9 SRC_Y:
        flags: range
        values: 0 4294967295
        value: 0
10 SRC_W:
        flags: range
        values: 0 4294967295
        value: 83886080
11 SRC_H:
        flags: range
        values: 0 4294967295
        value: 47185920
34 0 0 0,0 0,0 0 0x00000001
formats: AR24 RG24 RG16 AR15 AR12 C8
props:
7 type:
        flags: immutable enum
        enums: Overlay=0 Primary=1 Cursor=2
        value: 0
16 FB_ID:
        flags: object
        value: 0
17 IN_FENCE_FD:
        flags: signed range
        values: -1 2147483647
        value: -1
19 CRTC_ID:
        flags: object
        value: 0
12 CRTC_X:
        flags: signed range
        values: -2147483648 2147483647
        value: 0
13 CRTC_Y:
        flags: signed range
        values: -2147483648 2147483647
        value: 0
14 CRTC_W:
        flags: range
        values: 0 2147483647
        value: 0
15 CRTC_H:
        flags: range
        values: 0 2147483647
        value: 0
8 SRC_X:
        flags: range
        values: 0 4294967295
        value: 0
9 SRC_Y:
        flags: range
        values: 0 4294967295
        value: 0
10 SRC_W:
        flags: range
        values: 0 4294967295

```

```

value: 0
11 SRC_H:
    flags: range
    values: 0 4294967295
    value: 0


Frame buffers:
id      size  pitch

```

In the above output, the following results can be seen:

- 2 encoders
 - DPI
 - DSI
- 2 connectors
 - HDMI-A-1
 - the related encoder is DPI (id 28)
 - HDMI cable status is "connected"
 - many resolutions are supported (through the edid data)
 - DSI-1
 - no encoder is connected to the DSI panel
 - a single resolution is supported (480x800)
- 1 CRTC
- 2 planes with various supported color formats

Note: id values may differ depending on the application display hardware configuration.

	If you want to know more about DRM/KMS encoders, connectors, CRTCs and planes, please have a look at the presentations suggested in the To go further chapter.
---	--

4.1.3 Set a particular video mode

Using the **modetest -M stm** command (see [Show display overall status](#)), let's assume that the application returns the following results:

- the HDMI-A-1 is on connector id **29**, is **connected** and supports a resolution of **1280x720-50**.
- the CRTC is on id **33**

The command to set the **1280x720-50** video:

```
Board $> modetest -M stm -s 29@33:1280x720-50 -d
```

By using the modetest option **-d** (drop master after mode set) in the above command, the next DRM-based application that will start will detect the current video mode and use it directly. For instance, with [Weston](#), use "mode=current" in the weston.ini file to set preferred DRM video mode with the above command and restart weston to use it.

4.1.4 Send a test pattern to a display connector

Using the **modetest -M stm** command (see [Show display overall status](#)), let's assume you have identified that:

- the HDMI-A-1 is on connector id **29**, is **connected** and supports a resolution of **1280x720-50**.
- the CRTC is id **33**

So the command to send a test pattern is:

```
Board $> modetest -M stm -s 29@33:1280x720-50
```

The SMPTE color bars test pattern^[6] will appear on the HDMI screen.

If you want to test another DRM/KMS connector, simply change the connector and resolution values in the command (in the example below the connector id **31** is associated with the connector DSI-1 supporting the 480x800 mode)

```
Board $> modetest -M stm -s 31@33:480x800
```

4.1.5 Test the display of 2 layers

A test using 2 layers can be run using the **-s** and **-P** options. For instance, the following command displays a SMPTE color bars test pattern in full screen on a first layer and a 256x256 test pattern on a 2nd layer positioned on top of the 1st layer at coordinates (100, 100) in AR24-fourcc color format:

```
Board $> modetest -M stm -s 31@33:480x800 -P 34@33:256x256+100+100@AR24
```

If a 2 layer **DRM atomic mode**^[7] display test is needed using the modetest option **-a**, use the following command:

```
Board $> modetest -M stm -s 31@33:480x800 -P 32@33:480x480@AR24 -P 34@33:256x256@AR24 -a
```



- Check the list of supported plane color formats by referring to the [Show display overall status](#) chapter.
- Two planes may not work on higher display resolutions or /and on small systems (low DDR...).

4.1.6 Test Vsync

The following command flips between 2 buffers: the SMPTE pattern and a black/white screen. The flip is synchronized with the display Vsync. The screen continuously blinks, and the framerate is displayed in the console:

```
Board $> modetest -M stm -s 31@33:480x800 -v
```

Example results:

```
~# modetest -M stm -s 31@33:480x800 -v
setting mode 480x800-50Hz@XR24 on connectors 31, crtc 33
freq: 50.16Hz
freq: 49.99Hz
freq: 49.99Hz
```

If you want to test the Vsync with the **DRM atomic mode**^[7] thanks to the modetest option **-a**, use the following command:

```
Board $> modetest -M stm -s 31@33:480x800 -P 32@33:256x256@AR24 -v -a
```

4.2 kmscube (DRM/KMS OpenGL ES GPU test tool)

The small application **kmscube** (rotating cube) provided by Mesa^[8] demonstrates that the GPU is running on top of the DRM/KMS framework.

Here are useful kmscube commands:

```
Board $> systemctl stop weston # required only if Weston is running
Board $> kmscube --help
Board $> kmscube
Board $> kmscube -A # use drm/kms atomic modesetting and fencing

Board $> kmscube --mode=rgba # rgba textured cube
Board $> kmscube --mode=nv12-2img # yuv textured (color conversion in shader)
Board $> kmscube --mode=nv12-1img # yuv textured (single nv12 texture)

Board $> kmscube -V /dev/video0 # camera preview on the cube. /!\ depends on the camera
Board $> kmscube -V video.mkv # video playback on the cube. /!\ performances could be
```

4.3 Tips

4.3.1 How to get the name and current status of a DRM connector

Use the following command to get the DRM connector names and associated status:

```
Board $> for p in /sys/class/drm/*/status; do con=${p%/status}; echo -n "${con#*/card?-}:
```

Result example:

```
DSI-1: connected
HDMI-A-1: connected
```

5 How to trace and debug the framework

The objective of this chapter is to give methods to monitor, trace and debug the DRM/KMS framework.

5.1 How to monitor

5.1.1 How to monitor with modetest

You can get the running configuration with **modetest**, please refer to the [Show display overall status](#) chapter.

5.1.2 How to monitor with debugfs

The DRM/KMS framework provides information with the [Debugfs](#).

```
Board $> ls /sys/kernel/debug/dri/0/  
DSI-1 HDMI-A-1 clients crtc-0 framebuffer gem_names internal_clients name state
```

Get the status with the command:

```
Board $> cat /sys/kernel/debug/dri/0/state
```

Result example:

```
~# cat /sys/kernel/debug/dri/0/state  
plane[32]: plane-0  
    crtc=crtc-0  
    fb=36  
        allocated by = weston  
        refcount=2  
        format=XR24 little-endian (0x34325258)  
        modifier=0x0  
        size=480x800  
        layers:  
            size[0]=480x800  
            pitch[0]=1920  
            offset[0]=0  
            obj[0]:  
                name=0  
                refcount=3  
                start=00010177  
                size=1536000  
                imported=yes  
        crtc-pos=480x800+0+0  
        src-pos=480.000000x800.000000+0.000000+0.000000  
        rotation=1  
        normalized-zpos=0  
        color-encoding=ITU-R BT.601 YCbCr  
        color-range=YCbCr limited range  
        user_updates=0fps  
plane[34]: plane-1  
    crtc=(null)
```

```

fb=0
crtc-pos=0x0+0+0
src-pos=0.000000x0.000000+0.000000+0.000000
rotation=1
normalized-zpos=0
color-encoding=ITU-R BT.601 YCbCr
color-range=YCbCr limited range
user_updates=0fps
crtc[33]: crtc-0
enable=1
active=1
planes_changed=1
mode_changed=0
active_changed=0
connectors_changed=0
color_mgmt_changed=0
plane_mask=1
connector_mask=2
encoder_mask=2
mode: 0:"480x800" 50 33000 480 600 664 784 800 814 828 842 0x48 0x0
connector[29]: HDMI-A-1
crtc=(null)
connector[31]: DSI-1
crtc=crtc-0

```



More information related to debugfs are available in the [Debugfs](#) article.

5.1.3 How to monitor with `/sys/class/drm`

The DRM/KMS framework provides information from the `/sys/class/drm` directory.

```

Board $> ls /sys/class/drm
card0 card0-DSI-1 card0-HDMI-A-1 version

```

Examples of the available information:

- the status of a given connector (*connected* or *disconnected*). The following example gives the status of the DRM connector named "HDMI-A-1":

```

Board $> cat /sys/class/drm/card0-HDMI-A-1/status
connected

```

- the modes available on a given connector. The following example gives the available modes of the DRM connector named "DSI-1":

```

Board $> cat /sys/class/drm/card0-DSI-1/modes
480x800

```

5.1.4 How to monitor the display framerate

When an animation is running on the display, the related framerate can be monitored from the [display driver](#) level thanks to the command:

```
Board $> (while true; do export fps=`cat /sys/kernel/debug/dri/0/state | grep fps -m1 | gr
```

The display framerate is then periodically output in the user console in "fps" (**f**rames **p**er **s**econd):

```
display 50fps
display 50fps
display 50fps
```

Notes:

- Stop monitoring the framerate with the command "kill -9 `ps -o ppid= -C sleep`".
- Adjust the framerate update period by modifying the "sleep" value (4 seconds in the example).
- Use the command "dmesg -n8" to mix both user and kernel console outputs.
- [Debugfs](#) configuration needs to be enabled.



The DRM/KMS framework does not offer natively a display framerate monitoring, the above solution is powered by the STMicroelectronics DRM/KMS implementation.

5.1.5 How to monitor the DMA-BUF and CMA memory usage

The DRM/KMS framework uses both DMA-BUF and CMA memory managers^[9] as display dedicated buffers.

- CMA

```
Board $> cat /proc/meminfo | grep -i cma
```

Result example:

```
CmaTotal:      131072 kB
CmaFree:       85644 kB
```

- DMA-BUF

```
Board $> cat /sys/kernel/debug/dma_buf/bufinfo
```

Result example:

```
Dma-buf Objects:
size      flags      mode      count      exp_name
03522560  00000002  00080007  00000005  galcore
  Attached Devices:
  5a001000.display-controller
Total 1 devices attached

03522560  00000002  00080007  00000005  galcore
  Attached Devices:
  5a001000.display-controller
Total 1 devices attached
```

```

03522560      00000002      00080007      00000005      galcore
    Attached Devices:
    5a001000.display-controller
Total 1 devices attached

03686400      00000002      00080007      00000006      galcore
    Attached Devices:
    5a001000.display-controller
Total 1 devices attached

03686400      00000002      00080007      00000006      galcore
    Attached Devices:
    5a001000.display-controller
Total 1 devices attached

03686400      00000002      00080007      00000006      galcore
    Attached Devices:
    5a001000.display-controller
Total 1 devices attached

03686400      00000002      00080007      00000006      galcore
    Attached Devices:
    5a001000.display-controller
Total 1 devices attached

Total 7 objects, 25313280 bytes

```

5.2 How to trace

5.2.1 Enable DRM/KMS traces with the sysfs

The DRM/KMS traces can be enabled / disabled dynamically using the sysfs debug entry (**/sys/module/drm/parameters/debug**). The DRM/KMS traces are directed to the [dmesg](#) log.

The debug value to set is an OR combination of some debug levels defined in [include/drm/drm_print.h](#) . See below an extract of this file:

```

/*
 * The following categories are defined:
 *
 * CORE: Used in the generic drm code: drm_ioctl.c, drm_mm.c, drm_memory.c, ...
 *       This is the category used by the DRM_DEBUG() macro.
 *
 * DRIVER: Used in the vendor specific part of the driver: i915, radeon, ...
 *         This is the category used by the DRM_DEBUG_DRIVER() macro.
 *
 * KMS: used in the modesetting code.
 *      This is the category used by the DRM_DEBUG_KMS() macro.
 *
 * PRIME: used in the prime code.
 *        This is the category used by the DRM_DEBUG_PRIME() macro.
 *
 * ATOMIC: used in the atomic code.
 *          This is the category used by the DRM_DEBUG_ATOMIC() macro.
 *
 * VBL: used for verbose debug message in the vblank code
 *       This is the category used by the DRM_DEBUG_VBL() macro.
 *
 * Enabling verbose debug messages is done through the drm.debug parameter,
 * each category being enabled by a bit.

```



```
*
* drm.debug=0x1 will enable CORE messages
* drm.debug=0x2 will enable DRIVER messages
* drm.debug=0x3 will enable CORE and DRIVER messages
* ...
* drm.debug=0x3f will enable all messages
*
* An interesting feature is that it's possible to enable verbose logging at
* run-time by echoing the debug value in its sysfs node:
* # echo 0xf > /sys/module/drm/parameters/debug
*/
```

Examples:

- Enable all the DRM/KMS logs:

```
Board $> echo 0xff > /sys/module/drm/parameters/debug
```

- Enable only driver DRM/KMS logs:

```
Board $> echo 0x02 > /sys/module/drm/parameters/debug
```

- Disable all DRM/KMS logs:

```
Board $> echo 0 > /sys/module/drm/parameters/debug
```

5.2.2 Enable DRM/KMS traces with the kernel dynamic debug

It is possible to have a partial or full trace of the DRM/KMS driver with the [kernel dynamic debug](#).

5.3 How to debug

5.3.1 Errors

Errors are unconditionally traced in the kernel [dmesg](#) log.

```
Board $> dmesg
[ 2898.424338] [drm] ltcd fifo underrun: please verify display mode
```

6 Source code location

6.1 Kernel space

Hereafter a list of the most important DRM/KMS Linux kernel source code locations:

- DRM/KMS Linux kernel:
 - [main source code in drivers/gpu/drm directory](#) .
 - [panel drivers in drivers/gpu/drm/panel directory](#) .

- [bridge drivers \(drivers/gpu/drm/bridge directory\)](#) .
- [stm drivers in drivers/gpu/drm/stm directory](#) .
- DRM/KMS upstream:
 - [DRM kernel graphics driver development tree](#).
 - [Kernel DRM miscellaneous fixes and cross-tree changes](#).

6.2 User space

Hereafter a list of the most important DRM/KMS user space libraries source code locations:

- DRM/KMS libraries official git: [libdrm Direct Rendering Manager library and headers](#).
 - [libkms](#).
 - [modetest](#).

7 To go further

You can find good overviews of the DRM/KMS framework in the following presentations:

- ["DRM Driver Development For Embedded Systems"](#), Inki Dae, 2011.
- ["The DRM/KMS subsystem from a newbie's point of view"](#), Boris Brezillon, 2014.
- ["Kernel Recipes 2017 - An introduction to the Linux DRM subsystem"](#), Maxime Ripard, 2017.

You can also refer to the official DRM/KMS documentation:

- ["Linux GPU Driver Developer's Guide"](#) and the following sub-chapters:
 - [Userland Interfaces](#).
 - [Kernel Mode Setting](#).

8 References

1. ↑ [Direct Rendering Manager article on Wikipedia](#)
2. ↑ [Kernel Mode Setting article on Wikipedia](#)
3. ↑ [^{3.0} ^{3.1} libdrm source code and documentation](#)
4. ↑ [Linux GPU Driver Developer's Guide / Userland interfaces](#)
5. ↑ [Linux GPU Driver Developer's Guide / DRM Internals](#)
6. ↑ [SMPTE Color Bars test pattern on Wikipedia](#)
7. ↑ [^{7.0} ^{7.1} Linux GPU Driver Developer's Guide / Kernel Mode Setting \(KMS\) / Atomic Mode Setting](#)
8. ↑ [kmscube source code in Mesa](#)
9. ↑ [Linux GPU Driver Developer's Guide / DRM Memory Management](#)