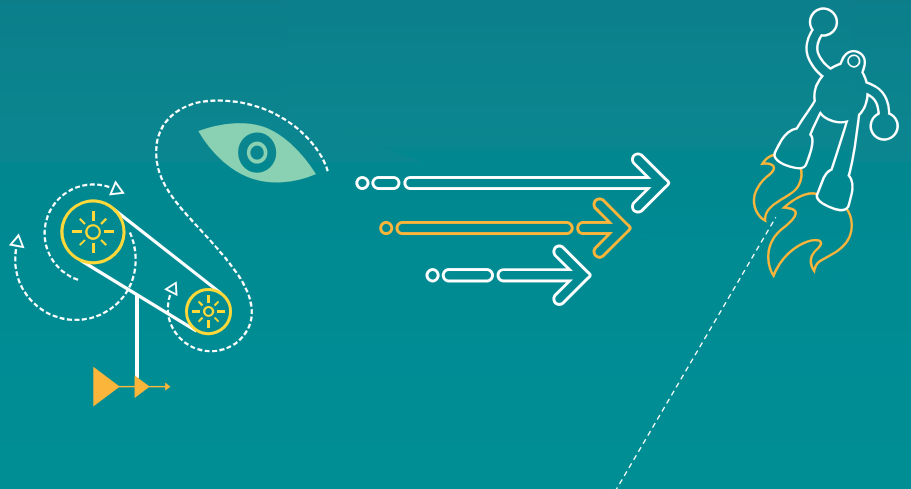
高通多媒体技术期刊　20140806

QUALCOMM®

Qualcomm Technologies, Inc.



Revision History

Revision	Date	Description
A	Aug 2014	Initial release

Contents

- Display
- Graphics
- Video

Display

如何动态debug 显示 Kernel 驱动

- 对于MDSS，主要包括下面几大模块 (参考文档：80-NL239-15)
  - Source Surface Processor(ViG pipe, RGB pipe, DMA pipe - SSPP)
  - Layer Mixer(LM)
  - Destination Surface Processor(DSPP)
  - Write-Back/Rotation(WB)
  - Display Interface

  具体的驱动文件在 //kernel/drivers/video/msm/mdss 目录下。

- 动态调试显示驱动，执行下面命令：
  - adb root
  - adb remount
  - adb shell
  - mount -t debugfs none /sys/kernel/debug/
  - root@android:/sys/kernel/debug # cd dynamic_debug
  - root@android:/sys/kernel/debug/ dynamic_debug # echo “file mdss_mdp_overlay.c +p” > control
  - root@android:/sys/kernel/debug/ dynamic_debug # echo “file mdss_mdp_ctl.c +p” > control
  - root@android:/sys/kernel/debug/ dynamic_debug # echo “file mdss_mdp_pipe.c +p” > control
  - root@android:/sys/kernel/debug/ dynamic_debug # echo “file mdss_mdp_util.c +p” > control
  - root@android:/sys/kernel/debug/ dynamic_debug # echo “file mdss_mdp_intf_video.c +p” > control

  另外，可以通过echo更多的文件进行调试。这样，通过 “cat /proc/kmsg”命令， 可以获取每个函数的调用流程， 以及参数的值，便于分析问题的关键所在。

- 参考Solution：00028591
  - 可以阅读 //kernel/Documentation/dynamic-debug-howto.txt

如何使用XLOG去调试 MDSS crash 问题

- 在开发过程中，会遇到MDSS crash的问题，可能是由于MDSS Ping-Pong timeout 引起的system crash 或者 MDP IOMMOU page fault 导致system crash。
- 对于这类型的crash问题：
- 1. 首先，需要提供下面的logs
  - 完整的ramdump logs，对应的vmlinux 文件，以及所使用的具体基线的ID。把所有上面信息通过case提供给Qualcomm，Qualcomm内部分析后，会尽快您一个反馈。
- 2. 另外一种比较好的方法是 使能 XLOG，可以实时 dump所需要的信息。
  - Add debugfs entry for xlog to enable or disable logging and register dumping

    adb root，adb remount，adb shell

    mount -t debugfs none /d
  - Enable XLOG:

    echo 1 > /d/mdp/xlog/enable
  - Enable register dump:

    echo 1 > /d/mdp/xlog/reg_dump
  - Enable panic on error:

    echo 1 > /d/mdp/xlog/panic
  - 当crash产生后，然后执行下面命令，得到xlogs 和 registers dump，提供给Qualcomm分析。

    cat /d/mdp/xlog/dump > xlog_and_mdss_register.txt
- 具体XLOG的patch如下：

https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/commit/?h=LNX.LA.3.7&id=dcac801bd7b3b5a26627db1391c673c988770f4c

ESD solution问题

- 对于ESD，目前有很多规避方法，如下：
  - 让每一行数据都进入LP11 --- 对于质量比较好的panel，一般可以不需要Recovery机制
  - BTA check --- 出现BTA error时，调用blank/unblank 去recovery LCD
  - 读LCD的状态寄存器 --- 出现寄存器返回值错误时，调用blank/unblank 去recovery LCD

  注意：blank/unblank 是在HWC HAL的reset_panel 函数里进行recovery操作。

- 1. 如何让每一行都进行LP11，在LCD panel驱动的dtsi文件中 增加下面flags
  - qcom,mdss-dsi-hfp-power-mode
  - qcom,mdss-dsi-hbp-power-mode
  - qcom,mdss-dsi-hsa-power-mode

  这样，对于比较好的屏，不需要Recovery机制，但绝大多数情况，仍然需要Recovery机制。

- 2. Check BTA
  - 默认，每隔5秒，查看一次BTA

    #**define** STATUS_CHECK_INTERVAL_MS 5000
  - 下面是一个例子：

```
--- a/arch/arm/boot/dts/qcom/dsi-panel-nt35590-720p-video.dtsi
+++ b/arch/arm/boot/dts/qcom/dsi-panel-nt35590-720p-video.dtsi
qcom,mdss-dsi-on-command-state = "dsi_lp_mode";
qcom,mdss-dsi-off-command-state = "dsi_hs_mode";
+ qcom,mdss-dsi-panel-status-check-mode = “bta_check”; // 添加这行代码
qcom,mdss-dsi-h-sync-pulse = <1>;
qcom,mdss-dsi-traffic-mode = "burst_mode";
qcom,mdss-dsi-bllp-eof-power-mode;
```

ESD solution问题 — 续一

- 3. 读LCD的状态寄存器
  - 在进行ESD测试时，通过读取LCD的状态寄存器的正确与否，来决定是否需要reset。
  - 下面是一个例子：

```
--- a/arch/arm/boot/dts/qcom/dsi-panel-innolux-720p-video.dtsi
+++ b/arch/arm/boot/dts/qcom/dsi-panel-innolux-720p-video.dtsi
qcom,mdss-dsi-on-command-state = "dsi_lp_mode";
qcom,mdss-dsi-off-command-state = "dsi_hs_mode";
+ qcom,mdss-dsi-panel-status-command = [06 01 00 01 05 00 02 0A 08]; // 发送的命令
+ qcom,mdss-dsi-panel-status-command-state = “dsi_lp_mode”; // 使用 LP mode去读取
+ qcom,mdss-dsi-panel-status-check-mode = "reg_read";
+ qcom,mdss-dsi-panel-status-value = <0x9c>;  // 寄存器的返回值
qcom,mdss-dsi-h-sync-pulse = <1>;
qcom,mdss-dsi-traffic-mode = "burst_mode";
```

- 4. 具体CAF链接如下(一般默认代码里面都有这个links)：
  - msm: mdss: Add support to enable ESD check through dtsi entry

https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/commit/?h=LNX.LA.3.7&id=6827717f961b10081e1ff849d56ae29377224e0a

  - ARM: dts: msm: enable ESD check for 8916 CDP with 720p and qrd-skuh

https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/commit/?h=LNX.LA.3.7&id=92e275c4aaf15ed1f1e421126ea44312d23076d4

  - msm: mdss: Add support for checking panel status through register read

https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/commit/?h=LNX.LA.3.7&id=5c687a4ca66f3b3728cadb871288ca25a73495b5

  - ARM: dts: msm: configure panel esd status check method for panels

https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/commit/?h=LNX.LA.3.7&id=db3b47cb5a7f90acbd1c281de8327490adf9342a

ESD solution问题 — 续二

- 5. 如何设置EOT packet
  当读取寄存器时，不同的panel需求不一致：
  - 有些panel，不需要设置EOT packet，就可以成功地读取LCD的寄存器
  - 有些panel，必须需要设置EOT packet后，才能有效地读取LCD寄存器
- 5.1 在kernel display 去设置EOT packet
  - 方法一
    添加 qcom,mdss-dsi-tx-eot-append 到LCD panel 驱动的dtsi文件中
  - 方法二
    在mdss_dsi_host_init 函数中，以MSM8916为例：

```
data = 0;
if (pinfo->rx_eot_ignore)
        data |= BIT(4);
+++ pinfo->tx_eot_append = 1; // 添加这行代码
if (pinfo->tx_eot_append)
        data |= BIT(0);
MIPI_OUTP((ctrl_pdata->ctrl_base) + 0x00cc, data); /* DSI_EOT_PACKET_CTRL */
```

- 5.2 在LK display 去设置EOT packet
  - 在mdss_dsi_cmd_mode_config 函数中，以MSM8916举例如下：

```
writel(0x14000000, ctl_base + COMMAND_MODE_DMA_CTRL);
writel(0x10000000, ctl_base + MISR_CMD_CTRL);
+++ writel(0x1, ctl_base + EOT_PACKET_CTRL);   // 添加这个代码
```

- 6. 针对一些panel，即使LCD的状态寄存器返回值为 正确，但有时候也会出现下面一些logs：

```
197.805697:   <6> mdss_dsi_isr: ndx=0 isr=3310003
197.808853:   <6> mdss_dsi_fifo_status: status=44441000
197.813539:   <6> mdss_dsi_ack_err_status: status=1008000
197.818397:   <6> mdss_dsi_timeout_status: status=1
```

上面这些logs表明，100800 是来自LCD panel反馈，没有什么坏处。如果想继续研究，请联系LCD vendor FAE去check。

设置MIPI DSI Clock 和Data lanes的LP11在LCD HW Reset 之前

- 对于有些特殊的LCD panel，在LCD 硬件Reset之前，需要使能DSI CLK和Data lanes(LP11)。
- 具体实现：
- 在LCD panel dtsi 文件中，添加如下参数
  - qcom,mdss-dsi-lp11-init
  - qcom,mdss-dsi-init-delay-us = <50000>; // 根据不同的panel，可以修改

- 参考的CAF links代码 (以MSM8916为例)：
- 在kernel display，
  - msm: mdss: Enable hardware reset line based on panel setting
  - https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/commit/?h=LNX.LA.3.7&id=699406258812f7fa25dfd2a2ac60e86e731ef012
  - ARM: dts: msm: enable LP11 configuration for ssd2080m
  - https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/commit/?h=LNX.LA.3.7&id=67ea2af11374f4f395d86076cd9a365313445fda

- 在LK display，
  - platform: msm_shared: Support LP11 configuration
  - https://www.codeaurora.org/cgit/quic/la/kernel/lk/commit/?h=LNX.LA.3.7&id=e2e6b71f1916fa3dd0a851a39b6fdf1c55a88fd8
  - dev: gcdb: Support post power API to handle LP11 panel config
  - https://www.codeaurora.org/cgit/quic/la/kernel/lk/commit/?h=LNX.LA.3.7&id=c9611a9621f480c7096e7c03dc511fe8327586f2
  - platform: msm_shared: Support post power API in display
  - https://www.codeaurora.org/cgit/quic/la/kernel/lk/commit/?h=LNX.LA.3.7&id=680f04f8fca25fcf9ba4b2d634b90b9359cc7947
  - platform: msm_shared: support panel pre-initialize function
  - https://www.codeaurora.org/cgit/quic/la/kernel/lk/commit/?h=LNX.LA.3.7&id=4c7e37f0b3d797b613f87de15b2a6b8a4ee819d2
  - dev: gcdb: implement pre-initialize function
  - https://www.codeaurora.org/cgit/quic/la/kernel/lk/commit/?h=LNX.LA.3.7&id=eb462f61fa747f773169fab8c3a27780cc0adbd3
  - dev: gcdb: enable LP11 configuration for ssd2080m
  - https://www.codeaurora.org/cgit/quic/la/kernel/lk/commit/?h=LNX.LA.3.7&id=2f9e5228b02dddab8826c811b983e846a9e7dbb0
  - dev: gcdb: Remove lp11 init duplicate definition
  - https://www.codeaurora.org/cgit/quic/la/kernel/lk/commit/?h=LNX.LA.3.7&id=729aa97087a45aea3ed5876b50168454f21e5131

设置MIPI DSI Clock一直在HS mode

- 对于有些特殊的LCD panel，需要DSI CLK一直工作在HS模式。
- 下面两部分，分别介绍了在LK 和kernel 如何设置 DSI CLK一直为HS mode：

1. 在kernel display，mdss_dsi.c 文件中， mdss_dsi_on 函数：

```
+++ mipi->force_clk_lane_hs = 1;  // 强制HS mode
if (mipi->force_clk_lane_hs)
{
  u32 tmp;
  tmp = MIPI_INP((ctrl_pdata->ctrl_base) + 0xac);
  tmp |= (1<<28);
  MIPI_OUTP((ctrl_pdata->ctrl_base) + 0xac, tmp);
  wmb();
}
```

2. 在 LK display，mipi_dsi.c 文件中，mdss_dsi_host_init 函数：

```
writel(broadcast << 31 | EMBED_MODE1 << 28 | POWER_MODE2 << 26
        | PACK_TYPE1 << 24 | VC1 << 22 | DT1 << 16 | WC1,
         MIPI_DSI0_BASE + COMMAND_MODE_DMA_CTRL);

+++ writel(1 << 28, MIPI_DSI0_BASE + 0xac); // 强制HS mode，对应的寄存器，可参看8916的SWI手册
writel(lane_swap, MIPI_DSI0_BASE + LANE_SWAP_CTL);
```

调整MIPI DSI 驱动能力的问题

- 对于MIPI DSI 来说，有两种模式：
  - LP mode --- 从软件层面，可以调节
  - HS mode --- 从软件层面，不可以调节

- 从软件角度，LP mode可以调节，举例如下：
  - 对于MSM8916，在文件msm8916-mdss.dtsi中

    qcom,platform-strength-ctrl = [ff 06]; // [DSIPHY_STR_LP_N, DSIPHY_STR_LP_P]

    对应的寄存器为 0x01A98684 MDSS_DSI_0_PHY_DSIPHY_STRENGTH_CTRL_0

  - 芯片软件接口手册(Software Interface Manual)，具体文档号：80-NK807-2X，下载文档的网址为 https://downloads.cdmatech.com。

- 从软件角度，HS mode不可以调节，DSI PHY HW自动地进行调整。如果，满足不了需求，请联系HW team去查看PCB的阻抗匹配等。

MIPI DSI PHY的 LDO mode 和 DC-DC mode

- 对于MIPI DSI PHY，有两种提供电源的方式：
  - LDO 模式
  - DC-DC 模式

- DC-DC模式：
  - 能提供很好的电源功效，但需要额外的电感元件，从而增加了HW PCB成本。

- LDO模式：
  - 与DC-DC模式相比，电源功效差些，但不需要额外的电感元件，从而节省了成本。

- 具体代码，参考CAF links (MSM8916为例)：
  - msm: mdss: enable LDO mode for DSI PHY regulator

https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/commit/?h=LNX.LA.3.7&id=18901f18865b10ff836cc85f546cff6ffc7852dc

  - msm: mdss: support LDO mode for DSI PHY regulator

https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/commit/?h=LNX.LA.3.7&id=c91dc70dfa5b1477ce85c565b60844f44c117945

Graphics

GPU hang FAQ (solution#00028668)

- **Q1) 我碰到一个issue，怀疑是GPU hang的问题。但是我不熟悉GPU hang, 怎么来确认是GPU hang呢?**
- **A1)** 虽然GPU hang 问题不容易解决，但是知道是否是GPU hang 相对容易。
- 如果你在kernel log里面看到如下类似的log，那么这就是一个GPU hang。
- [1144.957632 / 03-25 15:31:58.510] kgsl kgsl-3d0: ndroid.keyguard[1226]: gpu fault ctx 4 ts 9899 status E5678011 rb 0bca/0bca ib1 3fa8a900/03b5 ib2 3f66f000/0000
- 你可以找到"gpu fault" 字符串，后面跟着"status", "rb", "ib1" and ib2"等字符串。
-
- NOTE: 这个错误log内容在以后可能会改变，以增强GPU hang 恢复或检测。
- NOTE2：如果你在kgsl log中看到"time out"字符串，那么这就不是一个GPU hang，而是Long IB detection。
- Long IB detection 不同于GPU hang, 会有单独的Question 表述。
- <3>[40879.596442] c0 2200 kgsl kgsl-3d0: ogle.android.gm[21367]: gpu timeout ctx 8 ts 1220
- <3>[40879.596479] c0 2200 kgsl kgsl-3d0: ogle.android.gm[21367]: gpu failed ctx 8 ts 1220

- **Q2) GPU fault log的具体含义是什么?**
- **A2)** 你需要着重关注"status"的值， 我们用这个值作为Signature，来区分不同类型的GPU hang，因为它反映了GPU内部在GPU hang发生时刻的状态。

- [1144.957632 / 03-25 15:31:58.510] kgsl kgsl-3d0: ndroid.keyguard[1226]: gpu fault ctx 4 ts 9899 status E5678011 rb 0bca/0bca ib1 3fa8a900/03b5 ib2 3f66f000/0000

- 在"rb"字符串后面的值是Ringbuffer 读和写操作指针的位置。
- 在"ib1"和"ib2"字符串后面的值是IB buffer 的基地址，以及还没有被GPU执行的commands的大小。
- NOTE: 这并不意味相同的status值就表示相同类型的GPU hang 根本原因。但是它还是很有用的以区分不同类型的GPU hang，用作Signature。

- **Q3) 我已经确认一个GPU hang，还需要提供哪些信息去缩小GPU hang的范围?**
  A3) 下面是用来缩小GPU hang 范围的基本信息

A. GPU snapshot
- 这个是很关键信息用来缩小GPU hang的范围。没有GPU snapshot 我们没法缩小一个GPU hang的范围。
- GPU snapshot 是一个二进制文件，包含GPU在hang时刻的所有resource, 包括
- * GPU registers
- * Command buffers
- Shader binaries
- 以及其他
- 这些可以帮助我们理解GPU在Hang时刻在做什么，为什么卡住了。
- 如果你现在没有GPU snapshot，请一定提取出GPU snapshot来。再次重申，没有GPU snapshot我们没有办法找到GPU hang的根本原因。

GPU hang FAQ (solution#00028668)

- B. 你当前使用的QCOM Build
- 对GPU, 我们需要APSS build 信息。如果没有APSS build, 那么Meta build 也可以，我们可以从Meta build中查到对应的APSS build。但是很难从其他build, 比如MPSS 或者RPM 匹配对应的APSS build。
- 例如) LNX.LA.3.6-00720-8084.0-1 (APSS build, 很好)
- F8084AAAAANLYD121121A (Meta build, 也可以)
- MPSS.BO.1.0.r6-00011-M9635TAARANAZM-1(MPSS build, 这个不行，无法找到对应的APSS build)
- 正确的APSS build信息可以帮助我们去检查在你当前的build上是否缺少一些已知的GPU hang问题的修复。如果我们发现有任何缺少的change, 就可以立刻提供一个Test SBA。

- -你的GPU hang 是否可以重现吗?
- 如果是的，我们可以很容易地接近GPU hang问题。最好能知道GPU hang发生的频率，是100%，还是15分钟一次，等等。

- **Q4) 我知道GPu snapshot是缩小GPU hang问题所必须的，那么怎么提取GPU snapshot呢？**
- **A4)** 可以用一下几种方法：
- 方法 1
- ========
- GPU snapshot 被保存在sys文件系统默认的虚拟文件节点上，但是重启手机后就删除了。
- 如果在GPU hang发生时手机还是活着，可以用以下方法提取GPU snapshot.

- - 确认GPU snapshot是否已经产生了？

- >adb shell cat sys/class/kgsl/kgsl-3d0/snapshot/timestamp
- 如果不为0，表示GPU snapshot 已经产生了
- 从虚拟节点上面保存GPU snapshot到一个文件
- >adb pull /sys/class/kgsl/kgsl-3d0/snapshot/dump GPUsnapshot.bin
- 如果保存成功了，提供GPUsnapshot.bin 文件给我们分析。

- NOTE: 通过这种方法得到的GPU snapshot是第一次GPU snapshot.
- 如果你有多个GPU hang, 只有最后一次GPU hang影响用户使用，这里提取的GPU snapshot可能不包含最后一次GPU hang的信息。
- NOTE2: 不是每一个GPU hang都会影响用户使用，因为有些GPU hang是可以恢复的。

GPU hang FAQ (solution#00028668)

- 方法 2
- ========
- 在很多情况下，在GPU hang发生后可能不太容易device 还是活着状态。
- 有些GPu hang引起系统不稳定，最终导致手机crash。
- 如果你碰到这种情况，而且可以得到RAMDUMP, RAMDUMP将是提取GPU snapshot的第二条选择。
- 把kernel log和RAMDUMP一起提供给我们。
- kernel log应该有以下的信息：

- <3>[1406.544189 / 04-24 13:53:47.698] kgsl kgsl-3d0: |kgsl_device_snapshot| snapshot created at pa 0x25880000 size 447960
- 这里列出了GPU snapshot的物理地址和大小。有了RAMDUMP和这个信息，我们可以尝试从RAMDUMP中提取GPU snapshot。
- 然而请注意，这是第二条选择，因为可能提取GPU snapshot会失败，或者没有我们想要的全部的信息。

- 方法 3
- ========
- Adreno 库支持把GPU snapshot保存到一个OEM指定的路径
- 如果你可以在你的device上重现问题，下面是最容易提取GPU snapshot的方法

- 怎么设置GPU snapshot 保存的路径
- - 创建一个名为adreno_config.txt的文件文件， 文件名很重要，Adreno 库通过这个文件来加载配置选项。
- - 在这个文件里添加下面一行
- gpuSnapshotPath={一个你想要保存GPU snapshot的路径, 必须有写权限}
- - 把这个adreno_config.txt 文件推送到设备的/data/local/tmp
- 重启设备，在Adreno 库初始化时，就会把这个路径设置为GPU snapshot保存路径。
- 下面是一个设置GPU snapshot保存路径的例子，在GPU hang发生时，GPU snapshot 文件名为dump_xxxxxxxx就会被保存在/data/local/tmp
- >vi adreno_config.txt
- gpuSnapshotPath=/data/local/tmp
- >adb push adreno_config.txt /data/local/tmp
- >adb shell sync

- >adb reboot
- >adb wait-for-device
- >adb shell chmod 777 /data/local/tmp

GPU hang FAQ (solution#00028668)

- **Q5) 你在前面的问题里提到GPU hang　recovery，那么什么是GPU hang recovery，它是怎么工作的呢？**
- **A5)** QCOM把它称作GPU hang recovery 或者 GPU fault tolerance。
- 我们知道最好是避免GPU hang发生，但是如果不能避免，退而求其次在GPU hang发生后能够避免对用户产生影响也很好。
- GPU fault tolerance (GFT)的目的就是即使GPU hang发生了减少对用户的影响。
- 终端用户不用关心GPU hang发生，即便是在很多GFT的情况下GPU hang都发生了。

- GFT工作的方式如下：
-
- - KGSL (kernel part of GPU driver) detects GPU hang rapidly.
- - Reset GPU HW
- - Issue last GPU command again.
- - Check GPU hang
- - If no GPU hang, move forward to next command.
- - If GPU hang detects again, skip the command and issue next command.
- - If no GPU hang, move forward to next command.
- If GPU hang detects again, mark context bad and this eventually kills the process from user mode.

- **Q6) 前面你提动long IB detection 不同于GPU hang。那么什么是long IB detection，需要提供什么信息?**
- **A6)** Long IB detection 是一种情况，GPU 处理一批GPU command花费了太长的时间。
- NOTE: IB (indirect buffer)里面包含的是GPU commands，Long IB 意思是GPU command buffer 太长了。
- 虽然这不是GPU hang，GPU一直在运行中处理GPU command, KGSL driver等了太长时间去等待这项任务完成。
- 如果GPU在很长的时间还不能完成一个任务，终端用户就会看到绘图卡住了。
- 当前Long IB detection超时时间是2秒钟，对于大多数情况都是足够了。
- 如果KGSL 没有定义超时时间，用户感觉到系统UI都卡住了，因为GPU可能被一个context的超长IB一直占用着。
- Long IB的典型原因：
- - 应用程序bug
- 一个典型的会产生Long IB 的情况就是在shader中的无限或者很长的循环。如果应用程序产生了无效的shader code，就会引起Long IB.

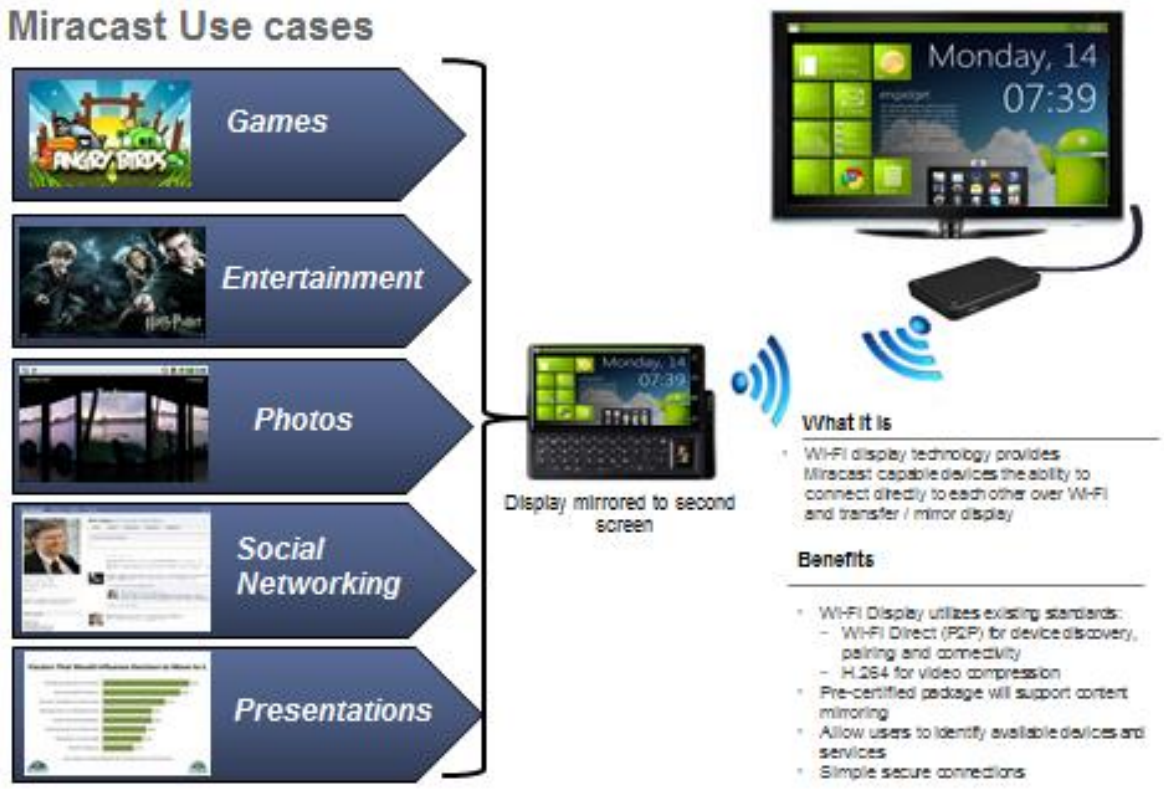
- - 针对桌面电脑的Web GL应用网站
- 一些WebGL 网站会产生Long IB因为这些网站是针对高性能的桌面系统GPU。一些Web GL网站甚至把桌面电脑弄得很慢，因为他们对GPU产生了太多繁重的操作。
- 在这种情况下，浏览器进程可以被Long IB detection杀掉而退出。

- 为了确认是否是Long IB detection引起的问题， 你可以从下面方法禁止Long IB detection。
- >adb shell "echo 0 > > /sys/class/kgsl/kgsl-3d0/ft_long_ib_detect“

Video

WFD (WiFi-Display) 概述

WFD 又称为 Miracast 。它是Wi-Fi联盟推出的基于P2P的一个多媒体应用标准。下图是WFD 应用的一种场景。



WFD 支持情况

Feature	8926	8974 AB	8X16	8939	8x10/12
Sink/Source	Yes	Yes	Yes	Yes	No
Video (Miracast Source)	720P 30 fps	1080P 60fps	720P 30 fps	1080P 30fps	N/A
Video (Miracast Sink)	1080P 30 fps	1080P 60 fps	1080P 30 fps	1080P 60 fps	N/A

WFD Debug

- How to debug

如果在WFD 播放过程中出现了异常。我们需要相关的调试信息。请参考下面的调试方法。

1. 基本的调试方法，参见solution 28489
2. Sink 端调试方法,参见solution 28488
3. Sink 端出现马赛克，播放不流畅，黑屏等的调试方法，参加solution 28464
4. Sink 端出现连接错误。参见 solution 28467
5. 在source 和Sink 端使能HDCP ， 参见 solution 28468
6. Source 端基本调试，参见solution 28486
7. Audio only ， 参见 solution 28527
8. P2P 断开连接，如果WFD 在播放过程中断开连接，请 参见solution 28463
9. Keepalive timeout – 有时SRC 不能获取SRC 端到Sink 端的keep alive command的response，参见 solution 28465
10. Sink 端突然端口连接，参见solution 28466

WFD Debug

- How to debug

11. 使能Audio 加密，参见solution 28445
12. 如果采用高通的方案 WFD 方案请参见 solution 28443
13. WFD 的 bring up ,请参见solution 28806

14. Sink 端出现马赛克，播放不流畅，黑屏等的调试方法，参加solution 28464
15. WFD 认证。参见 solution 28444
16. WFDMiracast/Wifi-Display Certification test case 5.1.10 A failure - Steps to Try在source 和Sink 端使能 HDCP ， 参见 solution 28468

Questions?

You may also submit questions to:
https://support.cdmatech.com