MSM8974 Linux Android Power Debugging and Optimization Guide

80-NA157-246 B

August 1, 2014

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

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Contents

Figures

Tables

Revision history

Revision	Date	Description
A	Mar 2014	Initial release
B	Aug 2014	Updated Sections 4.2, 4.7, 4.8, 4.9, and 4.13. Added Section 4.9.2

1 Introduction

1.1 Purpose

This guide outlines the generic and use case-specific approaches for debugging OEM multimedia-related power issues.

1.2 Scope

This document applies to the MSM8974 family of chipsets and is intended for OEMs who need to debug multimedia-related power issues, using Linux®.

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:`**.

Button and key names appear in bold font, e.g., click **Save** or press **Enter**.

1.4 References

Reference 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 Reference documents that are no longer applicable are deleted from this table; therefore, reference numbers may not be sequential.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 Technologies		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*Power Consumption Measurement Procedure for MSM (Android-Based)/ MDM Devices*	80-N6837-1
Q3	*Application Note: MSM8x74 and MSM8x74AB Clock Plan and Process Voltage Scaling*	80-NA157-3
Q4	*Application Note: MSM8974 Modem/Multimedia Use Case Details*	80-NE724-1
Q5	*MSM8974 Linux Android Current Consumption Data*	80-NA437-7
Q6	*MSM8960 LA Server and Hardware Component Setup for Video Streaming and Browser Test Case*	80-N8520-1
Q7	*Presentation: MSM8974 Power Debugging*	80-NA157-68

1.5 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://support.cdmatech.com/.

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

1.6 Acronyms

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

2 Generic Approach to Debugging

This chapter details the tools when used as a generic debugging of the power gap. The commonly used tools for power debugging are:

- Powertop
- Top
- Ftrace/Pytime chart
- Systrace
- Clock dump debugging
- Node Power Architecture (NPA) dump debugging
- MSM™ bus request debugging
- XO shutdown debugging
- Govenor parameter check
- Waveform comparison
- Wakelocks comparison
- msmpmstat analysis
- Rail-level comparison
- Bandwidth request comparison
- PX3 current

2.1 Powertop

The powertop tool identifies specific components of kernel and user space applications that frequently wake the CPU. The power savings are higher as the CPU can be in idle state most of the time. Powertop provides CPU profiling information of the system as shown in Figure 2-1. It is advisable to capture the powertop data and battery power measurement simultaneously to ensure powertop data is aligned with the battery power measurements on the device.

Powertop logs using the -d option capture information for 15 sec. To capture powertop data for a specific duration use:

```
powertop -d –t 60
```

Where -t indicates time in seconds. If the -d option is supplied alone, the data is captured for 15 sec. All msmpmstats Idle state information is captured out of this duration; for more detailed information msm_pm_stats logs can be captured.

The CPU power difference depends on how long a CPU was busy, the residency at each frequency level, and the duration for which the CPU was resisdent under each C0...C3 low power states.

Figure 2-1 Powertop log explanation

Figure 2-2 demonstrates how to analyze powertop data on good, i.e., a QTI reference device, and bad, i.e., a device with comparatively high power consumption, devices for a static image display use case.

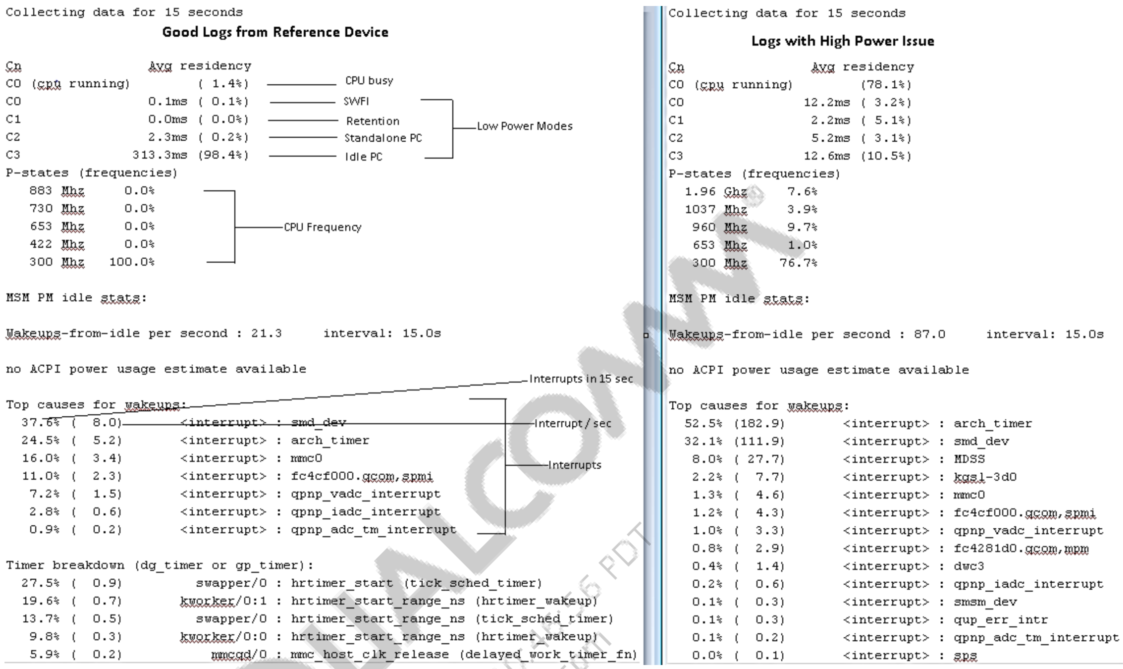


Figure 2-2 Powertop analysis for static image display use case

Analysis

To analyze the powertop data:

1. Compare the powertop data between the QTI reference device and the customer device.
2. Check C0 (CPU running), which signifies the total percentage of duration that the CPU was busy.
3. Check the percentage of time the device was in different C states, i.e., C0, C3, etc., of Low Power modes.
4. Check the difference in the number of wake-ups from idle per second; these wake-ups prevent the device from entering power collapse.
5. Check the top causes of wake-ups, i.e., wake-up interrupts, for the differences. The top cause of the wake-up is vital information, since it lists the interrupts that are preventing the CPU from entering power collapse; optimizing these interrupts result in power savings.
6. Check the CPU frequency by observing the P-states.
   - If the CPU frequency is higher on a device, this may lead to using high power consumption PLLs/CX, thus more power consumption may be observed.This needs to be checked in conjunction with C0, i.e., CPU running.
7. Check the wake-up from the Idle field that indicates how many times the CPU wakes up from idle every second due to the interrupts as discussed above.

Limitations

Powertop logs provide CPU status comprising all CPU cores, as such it is difficult to identify how many CPU cores were running.

2.2 Top

This utility captures the threads and process level information. System % gives the overall CPU busy information and IOW is the input/output wait time for which the CPU is waiting.

Analysis

To analyze the top data:

1. Check the System % and IOW % and match the percent differences with the reference device top data.
2. Check for the process consuming the majority of CPU cycles and also for any processes that are not supposed to run during a particular use case.

Figure 2-3 is an example of the top statistics showing the process level information for every second.

```
User 2%, System 18%, IOW 0%, IRQ 0%
User 9 + Nice 0 + Sys 79 + Idle 326 + IOW 4 + IRQ 1 + SIRQ 1 = 420

  PID   TID PR CPU% S     VSS     RSS PCY UID      Thread          Proc
 2084  2084  0   8% D      0K      0K     root     irq/341-synapti
15126 15126  1   5% R   1736K    980K     root     top             top
 1384  1402  0   1% S   6612K    872K     root     mpdecision      /system/bin/mpdecision
  865   974  0   1% S 579796K  55336K  fg system   UEventObserver  system_server
  865   960  0   0% S 579796K  55336K  fg system   ActivityManager system_server
  140   140  0   0% S      0K      0K     root     kworker/0:3
 1029  1029  0   0% S 491776K  48880K  fg u0_a60   ndroid.systemui com.android.systemui
    3     3  0   0% S      0K      0K     root     ksoftirqd/0
  395   677  0   0% S  69824K  10620K  fg system   EventThread
/system/bin/surfaceflinger
 2195  2195  0   0% S      0K      0K     root     kworker/0:0


User 0%, System 24%, IOW 0%, IRQ 1%
User 0 + Nice 0 + Sys 83 + Idle 250 + IOW 0 + IRQ 5 + SIRQ 0 = 338

  PID   TID PR CPU% S     VSS     RSS PCY UID      Thread          Proc
 2084  2084  0  22% D      0K      0K     root     irq/341-synapti
15126 15126  0   6% R   1744K    992K     root     top             top
 1384  1402  0   3% S   6612K    872K     root     mpdecision      /system/bin/mpdecision
    3     3  0   0% S      0K      0K     root     ksoftirqd/0
  140   140  0   0% S      0K      0K     root     kworker/0:3
   19    19  0   0% S      0K      0K     root     kworker/0:1H
  126   126  0   0% S      0K      0K     root     mmcqd/0
  865   974  0   0% S 579796K  55336K  fg system   UEventObserver  system_server
15094 15094  0   0% S      0K      0K     root     kworker/u:2
  413   413  0   0% S   1180K    624K     system   qrngd           /system/bin/qrngd
```

Figure 2-3 Top statistics

2.3 Ftrace/Pytime chart

Ftrace is a kernel function tracer that helps debug kernel function calls. The pytime chart is a tool to visualize the collected ftrace logs. Powertop details interrupt information, but it does not provide the cause of the interrupt at more granular level, in terms of what causes the interrupt. Ftrace allows further debugging of the cause of the interrupts.

Analysis

Ftrace analysis is done in a graphical format as shown in Figure 2-4 and Figure 2-5.

Figure 2-4 shows an ftrace analysis for a current increase caused by kgsl interrupts and the mdss_fb_commit_wq_handler thread.

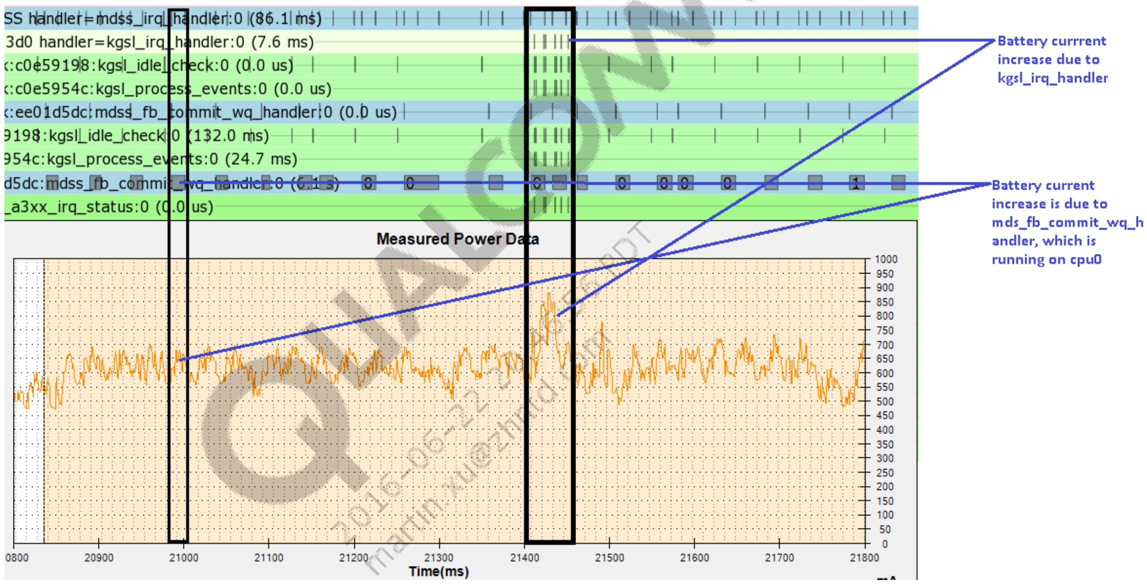


Figure 2-4 Ftrace log analysis – Static display use case with high current caused by kgsl interrupt

Figure 2-5 shows an ftrace analysis of a static display use case with a current increase with the device not entering Idle power collapse caused by a touch driver interrupt.

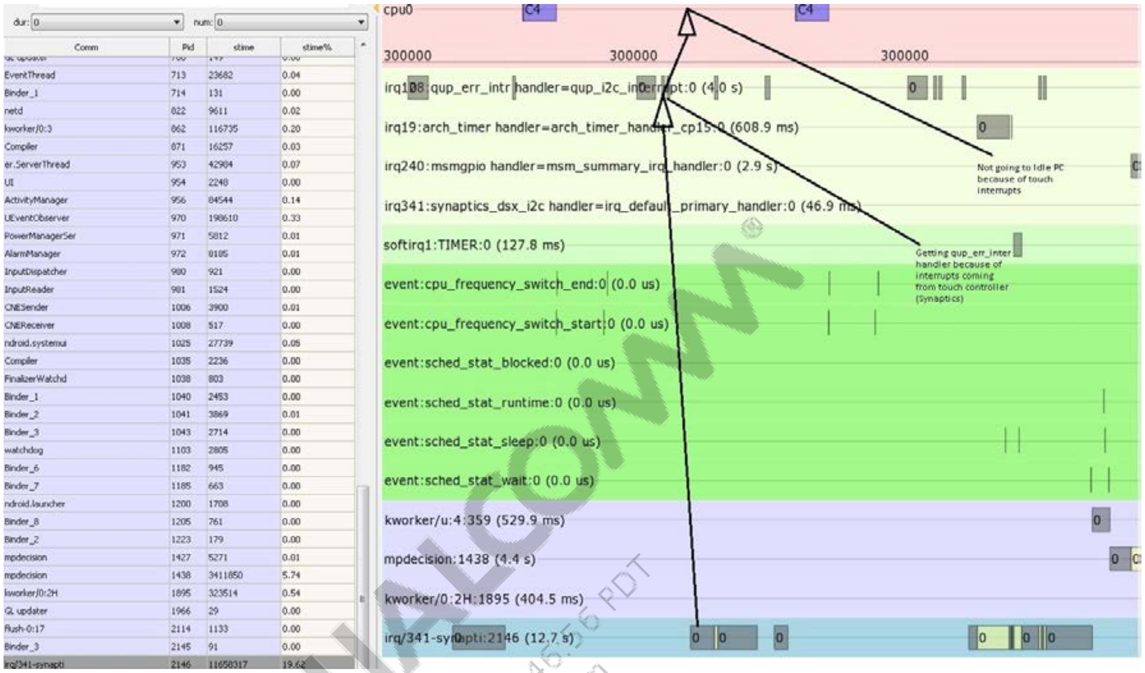


Figure 2-5 Ftrace logs analysis – Static display use case with high current caused by synaptics touch interrupt

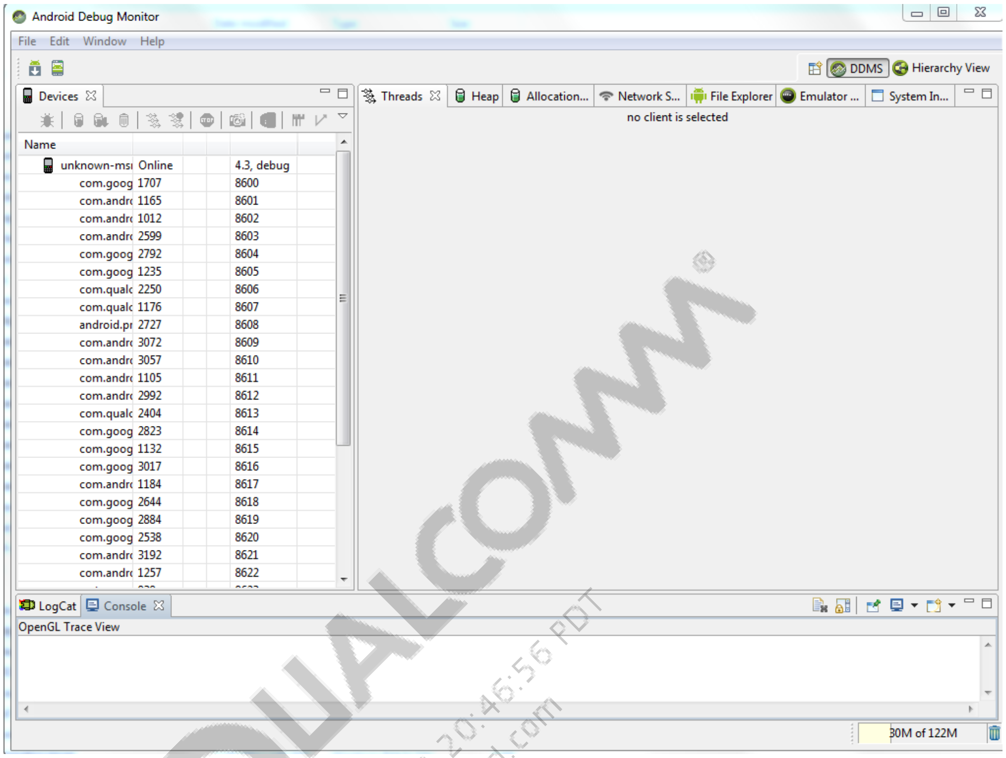
2.4 Systrace

The systrace tool helps analyze the performance of an application by capturing and displaying execution times of the application’s processes and other Android™ system processes. Systrace is helpful in debugging current increase issues related to the application or launcher. Systrace provides function-level detail, i.e., it provides causes based on which function call thread is running.

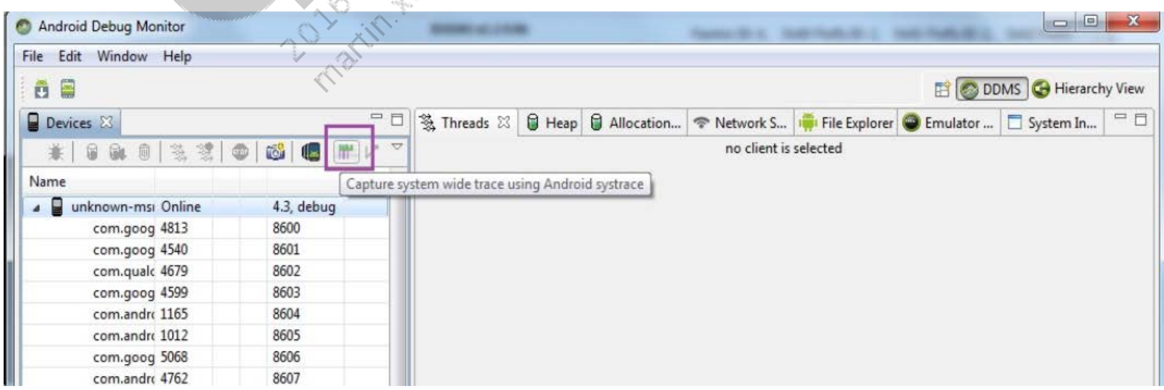
To run systrace after prerequisite installation is done:

1. Power on the device and make sure it is rooted; if the device is not rooted, enter the adb root command in the command prompt. The USB must be connected for systrace capturing.
2. Go to adt-bundle-windows-x86_64-20130729\sdk\tools.

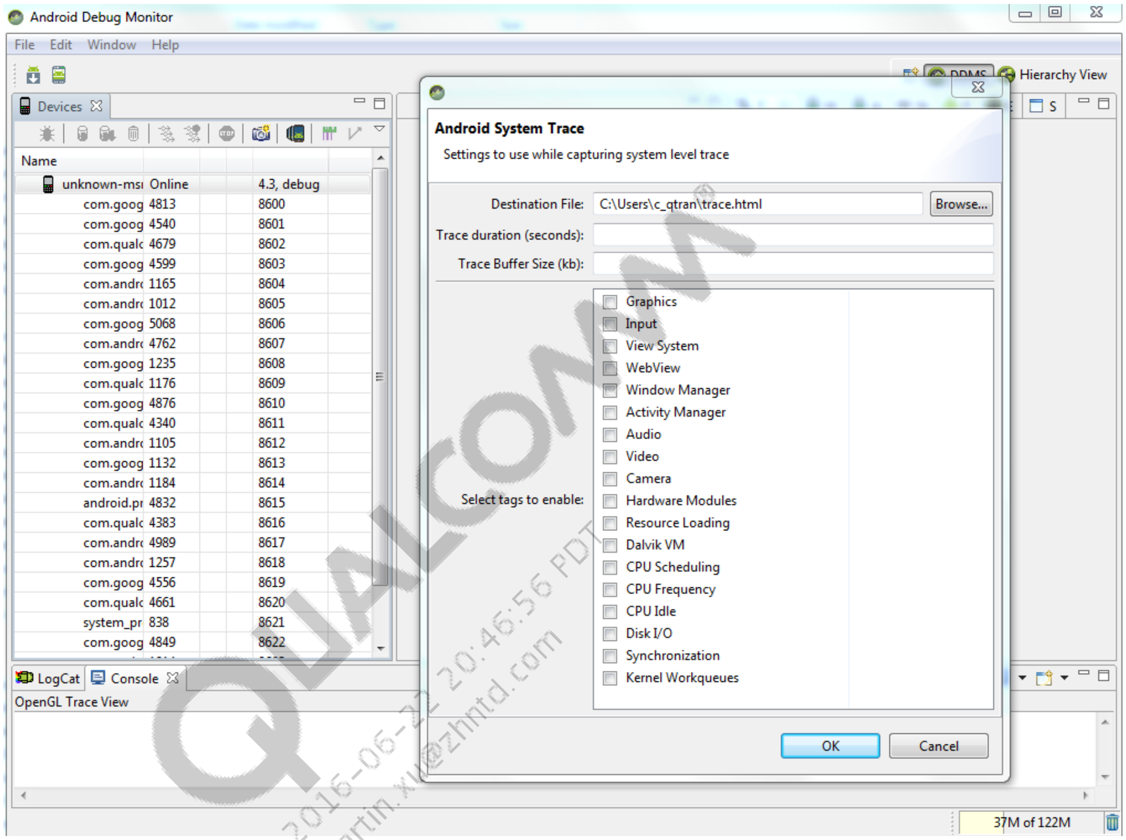
3. Click **Monitor**; the following window appears.



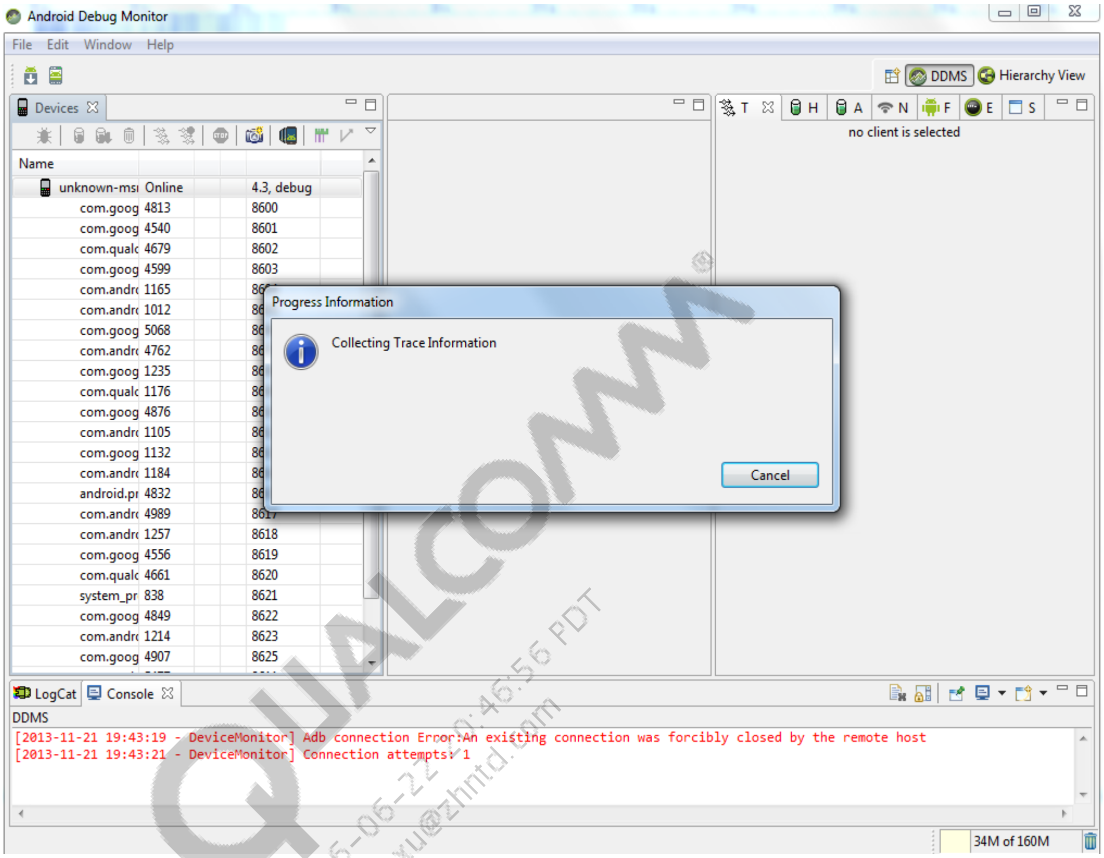
4. Click the highlighted button shown below.



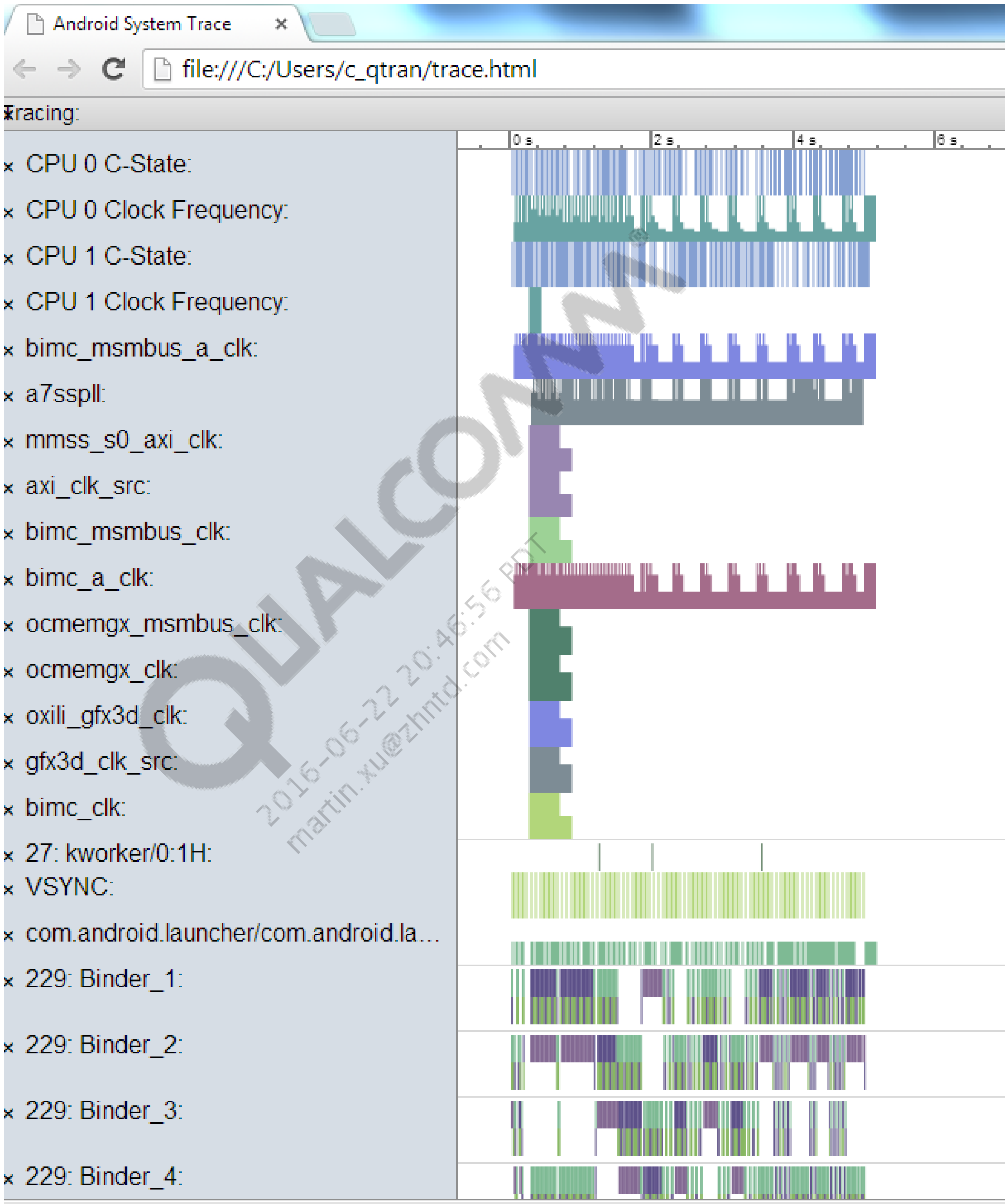
5. Select the area of interest, specify the destination location where data will be saved, and/or specify the trace duration in seconds; the default trace duration is 5 sec. Once you are done, click **OK**.



6. After clicking OK; a window displaying progress information appears.



7. After data has been captured, open the trace.html file in Google Chrome to display the output.



8. Capture the same data on a reference device and compare for differences.

2.5 Clock dump debugging

Clock dump debugging provides the entire list of clocks in the system with their respective frequencies. Clock dump information is used to ensure that device clocks are aligned with the reference target. When clocks are high, it can result in extra uses of PLLs or it can push CX into higher power modes, i.e., nominal, turbo, etc., than expected. The respective PLLs and CX voting by the clocks can be found in [Q4].

It is recommented to capture at least three different instances of clock dumps from RPM using JTAG. While comparing the clock dumps of two different devices, ensure that the CPU clocks are aligned on two out of three instances captured. If there are no matching instances, the closest CPU clocks are preferred.

For the expected clock plan for dashboard use cases, see [Q4]. Table 2-1 shows the clock plan mapping.

Table 2-1 MSM8974 clock maping

System Network on Chip (NoC)	gcc_sys_noc_axi_clk
Multimedia Subsystem (MMSS) NoC	mmss_mmssnoc_axi_clk
Configuration NoC	gcc_cfg_noc_ahb_clk
Peripheral NoC	gcc_periph_noc_ahb_clk
On-Chip Memory (OCMEM) NoC	ocmemnoc_clk
MMSS ahb clock	mmss_mmssnoc_ahb_clk
Bus Integrated Memory Controller (BMIC)	gcc_bimc_clk
Mobile Display Processor (MDP)	mdss_mdp_clk
Video core	venus0_vcodec0_clk
Graphics Processing Unit (GPU)	oxili_gfx3d_clk
Camera Post Processing (CPP)	camss_vfe_cpp_clk
Video Front-End (VFE)	camss_vfe_vfe0_clk

Debuging examples

The device does not enter XO shutdown during the static display use case in Command mode.

The clock dumps show:

```
mmss_mmssnoc_axi_clk                    ON             100.000939
cxo_clk_src                             ON             19.200585
```

This indicates that the mmssnoc_axi clock is running at 100 MHz and CXO is on and running at 19.2 MHz, which prevents XO shutdown. To debug further which client is voting for CXO, NPA dumps are collected and analyzed.

Clock debugging can also be done through adb for clocks that are not Application Processor Subsystem (APSS) controlled, e.g., oxili_gfx3d_clk, mdss_mdp_clk, mmss_mmssnoc_axi_clk, etc. This can be done from the node /d/clk. To continuously dump the clock state, write a loop as shown below.

```
adb shell
while
do cat /d/clk/mmss_mmssnoc_axi_clk/measure
sleep .1 (this is how often you want the state information in terms of secs
         and can be changed appropriately)
done
```

2.6 Node Power Architecture (NPA) dump debugging

The NPA dumps provide information about which client is voting for the shared NPA resources. In the NPA RPM dumps, look for the following string:

```
(type: NPA_CLIENT_REQUIRED) (request: 1)
```

The client name corresponding to this string is the one voting for CXO.

An example is shown for the problem described in Section 2.5. The device does not enter XO shutdown during the static display use case in Command mode.

The following NPA logs indicate that the APSS is requesting CXO.

```
npa_resource (name: "/xo/cxo") (handle: 0x196aa0) (units: Enable) (resource
max: 1) (active max: 1) (active state: (active headroom: 0) (request state:
1)
        npa_client (name: MPSS) (handle: 0x19ce70) (resource: 0x196aa0)
(type: NPA_CLIENT_LIMIT_MAX) (request: 1)
        npa_client (name: MPSS) (handle: 0x19ce30) (resource: 0x196aa0)
(type: NPA_CLIENT_REQUIRED) (request: 0)
        npa_client (name: WCSS) (handle: 0x19cd58) (resource: 0x196aa0)
(type: NPA_CLIENT_LIMIT_MAX) (request: 1)
        npa_client (name: WCSS) (handle: 0x19cd18) (resource: 0x196aa0)
(type: NPA_CLIENT_REQUIRED) (request: 0)
        npa_client (name: LPASS) (handle: 0x19c940) (resource: 0x196aa0)
(type: NPA_CLIENT_LIMIT_MAX) (request: 1)
        npa_client (name: LPASS) (handle: 0x19c900) (resource: 0x196aa0)
(type: NPA_CLIENT_REQUIRED) (request: 0)
        npa_client (name: APSS) (handle: 0x196c80) (resource: 0x196aa0)
(type: NPA_CLIENT_REQUIRED) (request: 1)
        npa_change_event (name: "sleep") (handle: 0x198f68) (resource:
0x196aa0)
        end npa_resource (handle: 0x196aa0)
```

To further debug which APSS client is requesting CXO, MSM bus debug client data is analyzed.

2.7 MSM bus request debugging

The MSM bus debug information is collected using the adb interface.

To capture bandwidth requests from all clients run:

```
adb root
adb remount
adb shell
cd  /d/msm-bus-dbg/client-data
cat  *
```

To capture a bandwidth request from a specific client, the client name can be provided instead of *, i.e., cat * becomes cat mdss_mdp.

```
adb shell
cd d/msm-bus-dbg/client-data
ls
Acpuclock
grp3d
mdss_mdp
mdss_pp
msm-rng-noc
pil-venus
qcom,dec-ddr-ab-ib
qcom,enc-ddr-ab-ib
qseecom-noc
 scm_pas
 sdhc1
 sdhc2
 update-request
 usb


  cat mdss_mdp
 157306.118032626
 curr   : 1
 masters: 22
 slaves : 512
 ab     : 890265600
 ib     : 3200000000
```

```
157306.127250803
curr   : 2
masters: 22
slaves : 512
ab     : 1335398400
ib     : 3200000000

157308.351429170
curr   : 1
masters: 22
slaves : 512

slaves : 512
ab     : 445132800
ib     : 333849600
```

In the output listed above, the arbitrated (ab) and instantaneous (ib) bandwidth information is observed. The last entry in the output is observed. If ab and ib values are 0, it means that the client is not requesting any bandwidth or has relinquished the bandwidth. If the entries are nonzero, as in the above example, the client is holding the bus.

The master and slave ID information is specific to each target.The msm_bus_board.h header describes the master and slave enumerations. The header file is located under kernel/arch/arm/mach-msm/include/mach.

The content of the msm_bus_board.h file is:

```
Master: 22 =  MSM_BUS_MASTER_MDP_PORT0
Slave: 512 =  DDR.

enum msm_bus_fabric_master_type {
203     MSM_BUS_MASTER_FIRST = 1,
204     MSM_BUS_MASTER_AMPSS_M0 = 1,
205     MSM_BUS_MASTER_AMPSS_M1,
206     MSM_BUS_APPSS_MASTER_FAB_MMSS,
207     MSM_BUS_APPSS_MASTER_FAB_SYSTEM,
209     MSM_BUS_SYSTEM_MASTER_FAB_APPSS,
210     MSM_BUS_MASTER_SPS,
211     MSM_BUS_MASTER_ADM_PORT0,
212     MSM_BUS_MASTER_ADM_PORT1,
213     MSM_BUS_SYSTEM_MASTER_ADM1_PORT0,
214     MSM_BUS_MASTER_ADM1_PORT1,
215     MSM_BUS_MASTER_LPASS_PROC,
216     MSM_BUS_MASTER_MSS_PROCI,
217     MSM_BUS_MASTER_MSS_PROCD,
```

```
218     MSM_BUS_MASTER_MSS_MDM_PORT0,
219     MSM_BUS_MASTER_LPASS,
220     MSM_BUS_SYSTEM_MASTER_CPSS_FPB,
221     MSM_BUS_SYSTEM_MASTER_SYSTEM_FPB,
222     MSM_BUS_SYSTEM_MASTER_MMSS_FPB,
223     MSM_BUS_MASTER_ADM1_CI,
224     MSM_BUS_MASTER_ADM0_CI,
225     MSM_BUS_MASTER_MSS_MDM_PORT1,
227     MSM_BUS_MASTER_MDP_PORT0,
228     MSM_BUS_MASTER_MDP_PORT1,

enum msm_bus_fabric_slave_type {
315     MSM_BUS_SLAVE_FIRST = SLAVE_ID_KEY,
316     MSM_BUS_SLAVE_EBI_CH0 = SLAVE_ID_KEY,
317     MSM_BUS_SLAVE_EBI_CH1,
318     MSM_BUS_SLAVE_AMPSS_L2,
319     MSM_BUS_APPSS_SLAVE_FAB_MMSS,
320     MSM_BUS_APPSS_SLAVE_FAB_SYSTEM,
322     MSM_BUS_SYSTEM_SLAVE_FAB_APPS,
323     MSM_BUS_SLAVE_SPS,
324     MSM_BUS_SLAVE_SYSTEM_IMEM,
325     MSM_BUS_SLAVE_AMPSS,
326     MSM_BUS_SLAVE_MSS,
327     MSM_BUS_SLAVE_LPASS,
328     MSM_BUS_SYSTEM_SLAVE_CPSS_FPB,
329     MSM_BUS_SYSTEM_SLAVE_SYSTEM_FPB,
330     MSM_BUS_SYSTEM_SLAVE_MMSS_FPB,
331     MSM_BUS_SLAVE_CORESIGHT,
332     MSM_BUS_SLAVE_RIVA,
```

2.8 XO shutdown/VDD min debugging

When there are no CXO client votes, the XO can be shutdown to save power. The system might also choose to enter VDD-min by keeping the CX and MX rails at the retention voltage to save more power. Issues that potentially interfere with the ability to enter these low power states are:

- High idle sleep current; for more information, see [Q7]
- High static display current with smart panel
- High suspended sleep current when the device is suspended during an active test case, i.e., mp3 playback or video playback

2.8.1 Sample XO-shutdown/VDD-min issues

- mp3 playback with screen off→mp3 playback finishes and there is no audio playback→the current is higher than the sleep current.
- Video playback, i.e., active use case with screen on→pause playback→suspend the device, i.e., turn off the screen→the current is higher than the sleep current.

In either case, it is necessary to be in VDD_Min and achieve the correct sleep current. If this is not the case, it is possibly because the XO-shutdown was blocked. To confirm the XO-shutdown was blocked:

1. Check RPM counts.
   a. Connect a USB and check VDDmin counts using the following adb commands:

```
adb root
adb remount
adb shell
cat /d/rpm_stats
```

   b. Disconnet the USB.
   c. Wait 30 sec and connect the USB.
   d. Get the RPM stats by running:

```
cat /d/rpm_stats
```

2. Check whether the VDDmin count has increased.

   If it has not increased then VDD_min was blocked. To debug this issue:

   a. Restart the device.
   b. Check the sleep current.
      i If the sleep current is high, debug the sleep current.
      ii If the sleep current is good but VDD_Min was blocked, check the following after the device is suspened and any test cases have stopped or finished.
         (a) Check the NPA dump on RPM as explained in Section 2.6 if CXO is being voted for by APSS or LPASS. For multimedia test cases, either of these subsystems could have blocked XO-shutdown/VDD_min. If it is LPASS, LPASS needs to relinquish the CXO vote.
         (b) If the NPA dump on RPM shows APSS as voting for CXO, check the msm-bus-dbg/client-data as explained in Section 2.7 to determine if any multimedia cores have not relinquished the bandwidth vote. If any core shows an active bandwidth request, this should be corrected.
         (c) Check the clock state of all multimedia cores and the LPASS. If any core shows an active clock (turned on), this should be corrected.

NOTE: Wakelock, as explained in Section 2.13, only blocks XO-shutdown/VDD_min. XO-shutdown is independent of suspended power collapse.

In the case of a high static display current with a smart panel, VDD_min cannot be achieved if MDP DSI-PHY is kept on, since MDP DSI-PHY needs SVS voltage. However, since XO-shutdown can still be achieved, check the XO-shutdown count from rpm_stats to confirm the high current is caused by XO-shutdown being blocked.

XO-shutdown can be blocked by any interrupt or clock being left enabled. To check the interrupts and clocks, capture the kernel logs. To capture the kernel logs:

1. Connect the USB.
2. Enable the following masks by running:

```
adb root
adb remount
adb shell
mount -t debugfs none /sys/kernel/debug
echo 1 > /sys/kernel/debug/clk/debug_suspend
echo 32 > /sys/module/msm_pm/parameters/debug_mask
echo 8 > /sys/module/mpm_of/parameters/debug_mask
```

3. On the same boot up, start capturing the kernel logs and disconnect the USB after issuing the following adb command and turning off the display:

```
adb shell
cat /proc/kmsg > /data/kmsg.txt &
```

4. Connect the USB after 30 to 60 sec and kill the kernel log collection process using its process id (pid) and the following adb commands:

```
adb shell
kill <pid>
```

5. Pull the logs by running the following adb command:

```
adb pull /data/kmsg.txt c:\temp\kmsg.txt
```

2.8.2 Kernel message analysis

1. Search for any enabled clocks in the kernel log.

   In the following example log, the mmss_mmssnoc_axi_clk is holding CXO. The cxo_lpm_clk also shows that there is at least one interrupt enabled that is preventing XO-shutdown.

```
<6>[  346.101268] msm_pm_enter
<6>[  346.101268] msm_pm_enter: power collapse
<6>[  346.101268] Enabled clocks:
<6>[  346.101268] cxo_mmss:1:1 [1000] -> cxo_clk_src:2:2 [19200000]
<6>[  346.101268] cxo_lpm_clk:1:1 [1000] -> cxo_clk_src:2:2 [19200000]
<6>[  346.101268] cxo_a_clk_src:1:1 [19200000]
<6>[  346.101268] cnoc_msmbus_clk:1:1 [100000] -> cnoc_clk:1:1 [100000]
<6>[  346.101268] snoc_msmbus_clk:1:1 [100000] -> snoc_clk:1:1 [100000]
<6>[  346.101268] snoc_msmbus_a_clk:1:1 [100000000] -> snoc_a_clk:1:1
[100000000]
<6>[  346.101268] pnoc_msmbus_clk:1:1 [100000] -> pnoc_clk:1:1 [100000]
<6>[  346.101268] bimc_msmbus_clk:1:1 [100000] -> bimc_clk:1:1 [100000]
<6>[  346.101268] bimc_msmbus_a_clk:1:1 [75000000] -> bimc_a_clk:1:1
[75000000]
<6>[  346.101268] pnoc_keepalive_a_clk:1:1 [19200000] -> pnoc_a_clk:1:1
[19200000]
<6>[  346.101268] snoc_clk:1:1 [100000]
<6>[  346.101268] pnoc_clk:1:1 [100000]
<6>[  346.101268] cnoc_clk:1:1 [100000]
<6>[  346.101268] bimc_clk:1:1 [100000]
<6>[  346.101268] snoc_a_clk:1:1 [100000000]
<6>[  346.101268] pnoc_a_clk:1:1 [19200000]
<6>[  346.101268] bimc_a_clk:1:1 [75000000]
<6>[  346.101268] cnoc_msmbus_clk:1:1 [100000] -> cnoc_clk:1:1 [100000]
<6>[  346.101268] snoc_msmbus_clk:1:1 [100000] -> snoc_clk:1:1 [100000]
<6>[  346.101268] snoc_msmbus_a_clk:1:1 [100000000] -> snoc_a_clk:1:1
[100000000]
<6>[  346.101268] pnoc_msmbus_clk:1:1 [100000] -> pnoc_clk:1:1 [100000]
<6>[  346.101268] bimc_msmbus_clk:1:1 [100000] -> bimc_clk:1:1 [100000]
<6>[  346.101268] bimc_msmbus_a_clk:1:1 [75000000] -> bimc_a_clk:1:1
[75000000]
<6>[  346.101268] gpll0_ao_clk_src:1:1 [600000000]
<6>[  346.101268] gcc_usb30_sleep_clk:1:1 [0]
<6>[  346.101268] gcc_usb2a_phy_sleep_clk:1:1 [0]
<6>[  346.101268] gcc_usb2b_phy_sleep_clk:1:1 [0]
<6>[  346.101268] gcc_usb2b_phy_sleep_clk:1:1 [0]
<6>[  346.101268] gcc_mss_q6_bimc_axi_clk:1:1 [0]
<6>[  346.101268] gcc_mss_cfg_ahb_clk:1:1 [0]
```

```
<6>[  346.101268] gcc_boot_rom_ahb_clk:1:1 [0]
<6>[  346.101268] gcc_mmss_noc_cfg_ahb_clk:1:1 [0]
<6>[  346.101268] gcc_ocmem_noc_cfg_ahb_clk:1:1 [0]
<6>[  346.101268] gcc_lpass_q6_axi_clk:1:1 [0]
<6>[  346.101268] axi_clk_src:2:2 [19200000, 1] -> cxo_clk_src:1:1 [0] -
> cxo_mmss:1:1 [1000] -> cxo_clk_src:2:2 [19200000]
<6>[  346.101268] mmss_mmssnoc_axi_clk:1:1 [0] -> axi_clk_src:2:2
[19200000, 1] -> cxo_clk_src:1:1 [0] -> cxo_mmss:1:1 [1000] ->
cxo_clk_src:2:2 [19200000]
<6>[  346.101268] mmss_s0_axi_clk:1:1 [19200000] -> axi_clk_src:2:2
[19200000, 1] -> cxo_clk_src:1:1 [0] -> cxo_mmss:1:1 [1000] ->
cxo_clk_src:2:2 [19200000]
<6>[  346.101268] mmss_s0_axi_clk:1:1 [19200000] -> axi_clk_src:2:2
[19200000, 1] -> cxo_clk_src:1:1 [0] -> cxo_mmss:1:1 [1000] ->
cxo_clk_src:2:2 [19200000]
<6>[  346.101268] acpu_aux_clk:2:2 [300000000] -> gpll0_ao_clk_src:1:1
[600000000]
<6>[  346.101268] krait0_sec_mux_clk:1:1 [300000000] -> acpu_aux_clk:2:2
[300000000] -> gpll0_ao_clk_src:1:1 [600000000]
<6>[  346.101268] l2_sec_mux_clk:1:1 [300000000] -> acpu_aux_clk:2:2
[300000000] -> gpll0_ao_clk_src:1:1 [600000000]
<6>[  346.101268] krait0_pri_mux_clk:1:1 [300000000] ->
krait0_sec_mux_clk:1:1 [300000000] -> acpu_aux_clk:2:2 [300000000] ->
gpll0_ao_clk_src:1:1 [600000000]
<6>[  346.101268] l2_pri_mux_clk:1:1 [300000000] -> l2_sec_mux_clk:1:1
[300000000] -> acpu_aux_clk:2:2 [300000000] -> gpll0_ao_clk_src:1:1
[600000000]
<6>[  346.101268] l2_clk:1:1 [300000000, 1] -> l2_pri_mux_clk:1:1
[300000000] -> l2_sec_mux_clk:1:1 [300000000] -> acpu_aux_clk:2:2
[300000000] -> gpll0_ao_clk_src:1:1 [600000000]
<6>[  346.101268] krait0_clk:1:1 [300000000, 1] ->
krait0_pri_mux_clk:1:1 [300000000] -> krait0_sec_mux_clk:1:1 [300000000]
-> acpu_aux_clk:2:2 [300000000] -> gpll0_ao_clk_src:1:1 [600000000]
<6>[  346.101268] l2_clk:1:1 [300000000, 1] -> l2_pri_mux_clk:1:1
[300000000] -> l2_sec_mux_clk:1:1 [300000000] -> acpu_aux_clk:2:2
[300000000] -> gpll0_ao_clk_src:1:1 [600000000]
<6>[  346.101268] krait0_clk:1:1 [300000000, 1] ->
krait0_pri_mux_clk:1:1 [300000000] -> krait0_sec_mux_clk:1:1 [300000000]
-> acpu_aux_clk:2:2 [300000000] -> gpll0_ao_clk_src:1:1 [600000000]
<6>[  346.101268] Enabled clock count: 47
```

To further debug which MMSS client is causing this shared resource, collect the bus client debug information as described in Section 2.7.

2. Check for any enabled interrupts.

   The following log snippet shows the gic preventing XO-shutdown as interrupts 18, 19, and 25 are enabled. To fix this issue, these interrupts need to be put on the mpm bypass list if these interrupts can never be fired when the APSS is power collapsed.

```
6] msm_mpm_interrupts_detectable(): gic preventing system sleep modes during idle
6]     hwirq: 18
6]     hwirq: 19
6]     hwirq: 25
```

2.9 Govenor parameter check

Check the interactive governor parameters if the CPU frequency is not compatible with the QTI reference device.

The parameters located at /sys/devices/system/cpu/cpufreq/interactive are:

- above_hispeed_delay
- boost
- boostpulse
- boostpulse_duration
- go_hispeed_load
- hispeed_freq
- io_is_busy
- min_sample_time
- sampling_down_factor
- sync_freq
- target_loads
- timer_rate
- timer_slack
- up_threshold_any_cpu_freq
- up_threshold_any_cpu_load

The parameters located at /sys/devices/system/cpu/cpu0/cpufreq/ are:

- affected_cpus
- cpu_utilization
- cpuinfo_cur_freq
- cpuinfo_max_freq
- cpuinfo_min_freq
- cpuinfo_transition_latency
- related_cpus
- scaling_available_frequencies

- scaling_available_governors
- scaling_cur_freq
- scaling_driver
- scaling_governor
- scaling_max_freq
- scaling_min_freq
- scaling_setspeed
- stats
    - time_in_state
    - total_trans

The tuneable values for this governor are:

- target_loads – This is the CPU load values used to adjust speed to influence the current CPU load toward that value.
- min_sample_time – This is the minimum amount of time to spend at the current frequency before ramping down.
- hispeed_freq – An intermediate high-speed at which to initially ramp when the CPU load hits the value specified in go_hispeed_load. If the load stays high for the amount of time specified in above_hispeed_delay, the speed may be increased.
- go_hispeed_load – This is the CPU load at which to ramp to hispeed_freq.
- above_hispeed_delay – When speed is at or above hispeed_freq, wait for this amount of time before raising the speed in response to a continued high load.
- timer_rate – This is the sample rate for reevaluating the CPU load when the CPU is not idle.
- timer_slack – This is the maximum additional time to defer handling the governor sampling timer beyond timer_rate when running at speeds above the minimum.
- boost – If nonzero, immediately boost the speed of all CPUs to at least hispeed_freq until zero is written to this attribute; if zero, allow CPU speeds to drop below hispeed_freq according to the load as usual.
- boostpulse – On each write, immediately boost the speed of all CPUs to hispeed_freq for at least the period of time specified by boostpulse_duration, after which speeds are allowed to drop below hispeed_freq according to the load as usual.
- boostpulse_duration – This is the length of time to hold the CPU speed at hispeed_freq on a write to boostpulse before allowing the speed to drop according to the load as usual.

2.10 sync_threshold comparison

Compare the sync_threshold of the device with the QTI reference device. This can be checked by running the following command:

```
cat /sys/module/cpu_boost/parameters/sync_threshold
```

sync_threshold determines the frequency of the destination core when a thread migrates from a source core to the destination core. If the source core frequency is higher than the sync_threshold, the destination core frequency will be ramped up to the sync_threshold frequency. If the source core frequency is lower than the sync_threshold, the destination core frequency will match the source core frequency. Configuring sync_threshold to a higher value results in better performance and higher power consumption.

2.11 Check for perf-locks

Spurious peaks observed in battery waveform can be due to sudden jumps in CPU frequency caused by perf-locks being held by certain services or modules. When a perf-lock is held, the scaling_min_freq is raised above its default value of 300 MHz. To check this in real time, track any changes to the scaling_min_freq node by writing an infinite loop as follows:

```
adb shell
while
do cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
sleep .1
done
```

mpctl logging can be checked to monitor all perf-locks being acquired and released. To monitor the perf-locks, add the debug.trace.perf = 1property in build.prop, and chmod 644 the build.prop after adding it.

```
adb shell
adb pull /system/build.prop
add the property debug.trace.perf = 1 to build.prop
adb push build.prop /system/build.prop
chmod 644 /system/build.prop
adb reboot
```

After the target reboots, start the test case and run the following:

```
adb shell
logcat | grep ANDR-PERF-MPCTL
```

```
I/ANDR-PERF-MPCTL( 1584): perf_lock_acq: client_pid=955, client_tid=1940, inupt handle=0, duration=0 ms, num_args=2, list=0xECD 0x1400
I/ANDR-PERF-MPCTL( 1584): perf_lock_acq: client_pid=955, client_tid=1940, inupt handle=0, duration=0 ms, num_args=2, list=0xECD 0x1400 0xECD 0x140
I/ANDR-PERF-MPCTL( 1584): perf_lock_acq: output handle=2
I/ANDR-PERF-MPCTL( 1584): perf_lock_rel: input handle=2
```

The above log shows perf-lock being acquired by client with pid 955 and release identified by the handle. The arguments represent the number of resources being used. All the resources are defined in qc-performance.h. The first two bits of the list correspond to the numerical value of the enum defined in the qc-performance.h file, and the next two bits are the arguments passed to that resource.

2.12 Waveform comparison

The waveforms during any use case can be compared with QTI reference waveforms to see the pattern of the waveforms difference.

The point where the current spike is relatively high can be debugged while analyzing the powertop/clock dump to locate which additional clocks wake up interrupts.

The baseline power difference can also be attributed to the power gap at the battery. The root cause of a baseline power difference needs to be debugged using the clock dumps.

Figure 2-6 shows the waveform captured on a QTI reference device during mp3 playback with high current consumption.

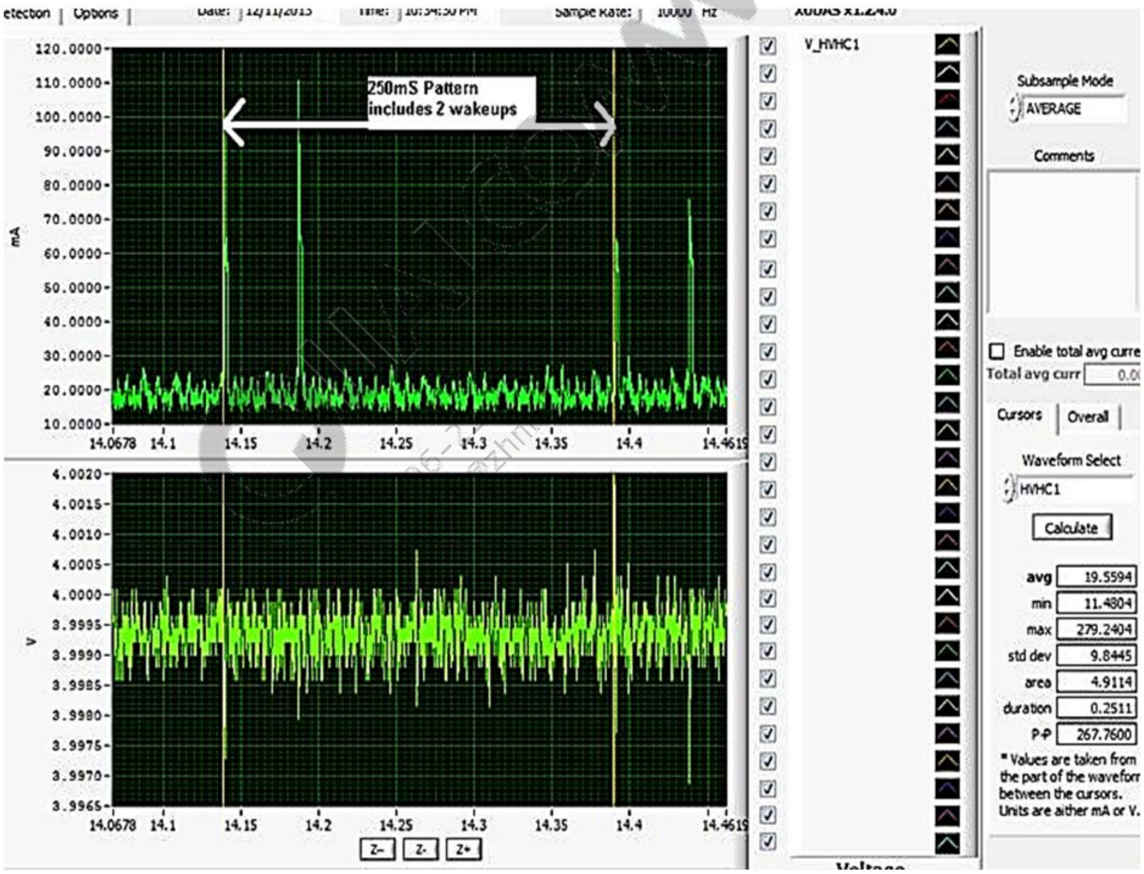


Figure 2-6 mp3 playback on reference device A

Figure 2-7 shows the waveform captured on device B during mp3 playback with power regression enabled.

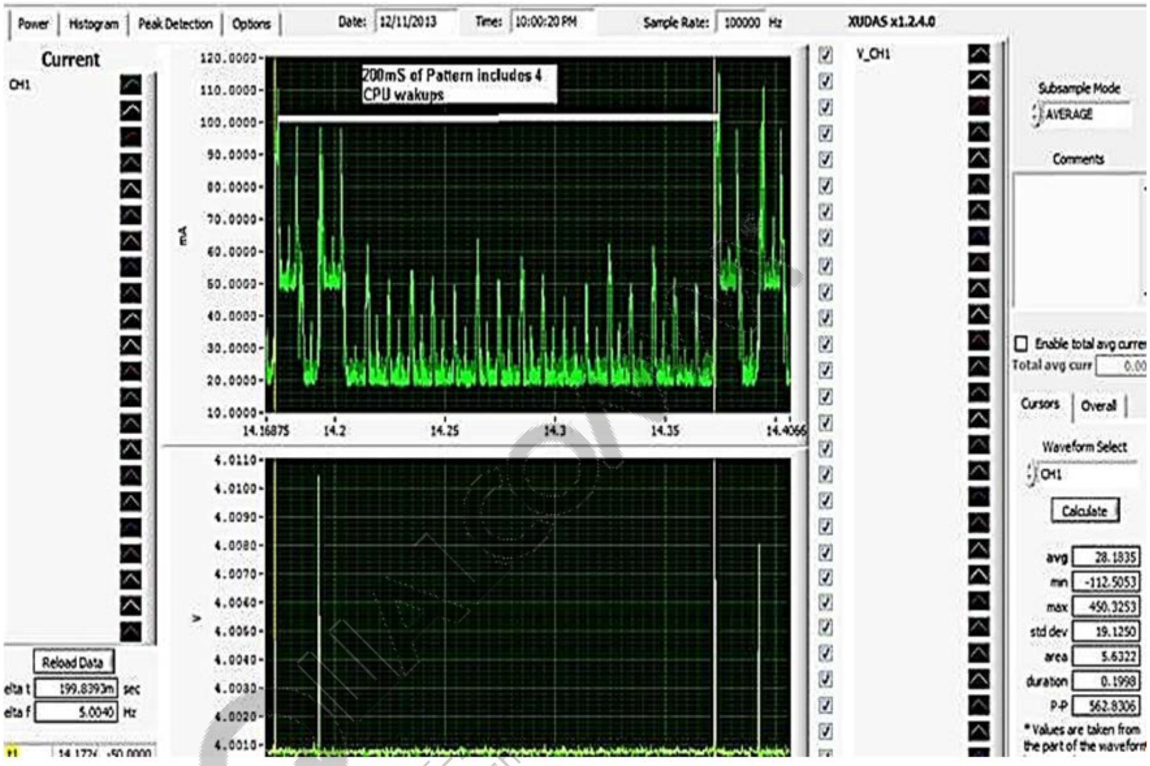


Figure 2-7 mp3 playback on device B with power regression

Table 2-2 Waveform analysis

	QTI reference device A	Another device with power regression (device B)	Comments
Power	19.88 mA	30.2 mA	Difference is 10.32 mA
Number of Krait™ wake-ups	2	4	
Pattern period (as highlighted in the snippet)	250 ms	200 ms	▪ Capture powertop, top, and msmpmstats ▪ Need to check latency and mpdecision on device B
CPU running (power impact on overall power)	~1.08 mA	~4.5 mA	
CPU in SPC (power impact on overall power)	—	3.9	
Baseline power (CPU is in Idle power collapse)	~18 mA	~21.5 mA	▪ CX is nominal on device B while device A operates in SVS mode ▪ Debug RPM/DSP side to find the root cause of RPM running at 171 MHz

2.13 Wakelocks comparison

Wakelocks can be checked by checking wakeup_sources and dumpsys power as follows.

2.13.1 wakeup_sources

To check if additional wakelocks are held, run:

```
cat /sys/kernel/debug/wakeup_sources
```

If idle wakelocks are held, the APSS does not go into idle power-collapse and increases the baseline power at the battery; the log snippets to identify wakelocks are shown in Table 2-3 and Table 2-4. The second column shows the wakelock count, and the sixth column (active_since) shows if there is an active wakelock. A nonzero active_since value indicates there is an active wakelock. When a USB is connected, there is always a suspend wakelock held by the USB, which prevents the apps processor from suspending the idle power collapse. The USB wakelock is shown in Table 2-3 and Table 2-4 as msm_dwc3.

Table 2-3 wakeup_sources log snippet for rock bottom sleep use case

Name	active_count	event_count	wakeup_count	expire_count	active_since	total_time	max_time	last_change	prevent_ suspend_ time
PowerManagerService. Broadcasts	7	7	0	0	0	4181	902	771175	720
ipc00000087_Loc_hal_ worker	1	1	0	0	0	2	2	119175	2
ipc00000086_Loc_hal_ worker	22	23	0	0	0	1	0	740451	0
ipc00000081_rvices.loc ation	0	0	0	0	0	0	0	44645	0
ipc0000007e_Loc_hal_ worker	1	1	0	0	0	2	2	119175	2
ipc0000007d_Loc_hal_ worker	2	2	0	0	0	0	0	44519	0
ipc0000007c_Loc_hal_ worker	0	0	0	0	0	0	0	44457	0

Name	active_count	event_count	wakeup_count	expire_count	active_since	total_time	max_time	last_change	prevent_suspend_time
PowerManagerService.WakeLocks	661	661	1	0	0	535259	150322	12061131	512274
msm_dwc3	2	2	0	0	11377854	11380874	11377854	731340	11339187

Table 2-4 wakeup_sources log snippet for mp3 use case

Name	active_count	event_count	wakeup_count	expire_count	active_since	total_time	max_time	last_change	prevent_suspend_time
PowerManagerService.Broadcasts	11	11	0	0	0	6753	902	17129069	1253
ipc00000087_Loc_hal_worker	1	1	0	0	0	2	2	119175	2
ipc00000086_Loc_hal_worker	22	23	0	0	0	1	0	740451	0
ipc00000081_rvices.location	0	0	0	0	0	0	0	44645	0
ipc0000007e_Loc_hal_worker	1	1	0	0	0	2	2	119175	2
ipc0000007d_Loc_hal_worker	2	2	0	0	0	0	0	44519	0
ipc0000007c_Loc_hal_worker	0	0	0	0	0	0	0	44457	0
PowerManagerService.WakeLocks	757	757	1	0	7505	693124	150322	17123682	659659
msm_dwc3	2	2	0	0	16399852	16402872	16399852	731340	16339360

2.13.2 Dumpsys power

Held wakelocks can also be captured by collecting the dumpsys power logs. To check if a wakelock is held, look for a value of 1, which indicates that a wakelock is active. Example log snippets are shown below.

- dumpsys power log snippet during rockbottom sleep use case

```
Suspend Blockers: size=4
PowerManagerService.WakeLocks: ref count=0
PowerManagerService.Display: ref count=0
PowerManagerService.Broadcasts: ref count=0
PowerManagerService.WirelessChargerDetector: ref count=0
```

- dumpsys power log snippet during mp3 use case

```
Suspend Blockers: size=4
PowerManagerService.WakeLocks: ref count=1
PowerManagerService.Display: ref count=0
PowerManagerService.Broadcasts: ref count=0
PowerManagerService.WirelessChargerDetector: ref count=0
```

2.14 msmpmstat analysis

The msmpmstat log shows details of the various power states of each CPU core. Figure 2-8 shows an example msmpmstat log.

```
  count:       26
  total_time: 0.003914477
   <      0.000062500:       12 (5313-58698)
   <      0.000250000:        8 (63385-228750)
   <      0.001000000:        6 (281562-753073)
   <      0.004000000:        0 (0-0)
   <      0.016000000:        0 (0-0)
   <      0.064000000:        0 (0-0)
   <      0.256000000:        0 (0-0)
   <      1.024000000:        0 (0-0)
   <      4.096000000:        0 (0-0)
  >=      4.096000000:        0 (0-0)
[cpu 0] retention:
  count:       49
  total_time: 0.024395728
   <      0.000062500:        9 (7812-55105)
   <      0.000250000:        9 (64688-210364)
   <      0.001000000:       29 (253333-684219)
   <      0.004000000:        0 (0-0)
   <      0.016000000:        2 (4625312-4640052)
   <      0.064000000:        0 (0-0)
   <      0.256000000:        0 (0-0)
   <      1.024000000:        0 (0-0)
   <      4.096000000:        0 (0-0)
  >=      4.096000000:        0 (0-0)
[cpu 0] idle-standalone-power-collapse:
  count:      213
  total_time: 0.203523001
   <      0.000062500:        0 (0-0)
   <      0.000250000:        2 (117604-211302)
   <      0.001000000:      177 (256875-996145)
   <      0.004000000:       32 (1001094-3740208)
   <      0.016000000:        2 (4086354-4814374)
   <      0.064000000:        0 (0-0)
   <      0.256000000:        0 (0-0)
   <      1.024000000:        0 (0-0)
   <      4.096000000:        0 (0-0)
  >=      4.096000000:        0 (0-0)

[cpu 0] idle-power-collapse:
 count:     1278
 total_time: 4.649192899
  <      0.000062500:        0 (0-0)
  <      0.000250000:        0 (0-0)
  <      0.001000000:       40 (366354-964636)
  <      0.004000000:      867 (1001719-3997031)
  <      0.016000000:      371 (4004063-10663177)
  <      0.064000000:        0 (0-0)
  <      0.256000000:        0 (0-0)
  <      1.024000000:        0 (0-0)
  <      4.096000000:        0 (0-0)
 >=      4.096000000:        0 (0-0)
[cpu 0] suspend:
 count:        0
 total_time: 0.000000000
  <      1.000000000:        0 (0-0)
  <      4.000000000:        0 (0-0)
  <     16.000000000:        0 (0-0)
  <     64.000000000:        0 (0-0)
  <    256.000000000:        0 (0-0)
  <   1024.000000000:        0 (0-0)
  <   4096.000000000:        0 (0-0)
  < 16384.000000000:        0 (0-0)
  < 65536.000000000:        0 (0-0)
 >= 65536.000000000:        0 (0-0)
```

Figure 2-8 msmpmstats log snippet

2.15 Rail-level comparison

The power grid/rail-level breakdown can be found in [Q5].

Compare the major rails VDD MX1, VDD CX1, VREG S2A, VREG S3A, VREG S4A, VREG S4B, VDD ADSP, VDD SC1, VDD SC0, LCD, BL, touch panel.

The rail-level power differences from the QTI reference device can be caused by extra hardware components, extra PLLs used, or Scorpion running with high frequency. High power numbers can be observed on digital rails CX/MX/LDOs due to PLLs, since they are running in Turbo mode. The digital rails may show a high power difference due to GPU or other cores running with higher frequency.

2.16 Bandwidth request comparison

Check the devices-8974.c file to see if the AB/IB requests have been changed for any use case, e.g., Figure 2-9 shows the structure that should be checked for vga decode.

```
static struct msm_bus_vectors vidc_vdec_vga_vectors[] = {
    {
        .src = MSM_BUS_MASTER_HD_CODEC_PORT0,
        .dst = MSM_BUS_SLAVE_EBI_CH0,
        .ab  = 40894464,
        .ib  = 327155712,
    },
    {
        .src = MSM_BUS_MASTER_HD_CODEC_PORT1,
        .dst = MSM_BUS_SLAVE_EBI_CH0,
        .ab  = 48234496,
        .ib  = 192937984,
    },
    {
        .src = MSM_BUS_MASTER_AMPSS_M0,
        .dst = MSM_BUS_SLAVE_EBI_CH0,
        .ab  = 500000,
        .ib  = 2000000,
    },
    {
        .src = MSM_BUS_MASTER_AMPSS_M0,
        .dst = MSM_BUS_SLAVE_EBI_CH0,
        .ab  = 500000,
        .ib  = 2000000,
    },
};
```

Figure 2-9 device8974.c snippet

2.17 PX3 current

If the PX3 current is high then the high power can be attributed to GPIO configurations. The GPIO configuration can be checked by dumping the GPIO configuration through Trace32 (T32).

For an APQ GPIO dump in a live RPM T32 session, run:

```
>> do <Boot build> boot_images\core\systemdrivers\tlmm\t32\
tlmm_gpio_hw.cmm
```

It is necessary to identify and avoid conflicts.

To compare the GPIO configuration with the QTI reference device:

1. Pull the register configuration.
3. Check the GPIO direction.
4. Pull the register and external components' states.
5. Compare the driving strength

NOTE: PX3 also powers DDR PHY so it is expected to see peaks on the PX3 rail when there is a DDR access.

3 Use Case-Specific Debugging

This chapter describes additional debugging information to compare device data against the QTI reference device beyond following the general power debugging approach.

3.1 Audio – mp3

The mp3 dashboard use case uses Tunnel mode audio playback. The steps to check playback mode, i.e., Tunnel or non-Tunnel mode, are:

1. Capture user-space (logcat) logs using the following adb command:

   ```
   adb logcat > c:\temp\logcat.txt
   ```

2. Play mp3 and capture logs for ~30 sec of mp3 playback duration.
6. In the logcat logs, look for the following message, which ensures that Tunnel mode playback is ongoing.

   ```
   music_offload_avg_bit_rate=128000;music_offload_sample_rate=44100
   ```

   If the deep-buffer-playback message is found in the log, it indicates that non-tunnel, i.e., normal, playback is ongoing.

To most effectively debug:

- Check the clocks on the device and whether they are aligned with the clock plan mentioned in [Q4].
- Use the same AU4.mp3 test clip for power measurement and follow the QTI test procedure to measure the mp3 power; this ensures a proper comparison with the QTI-released measurement.
- Account for hardware and software factors that potentially impact mp3 playback power consumption when comparing the devices measurements with the QTI measurements.

3.1.1 Audio power waveform analysis

Figure 3-1 shows an example waveform for mp3 playback in Tunnel mode. When using battery waveform analysis check the base current and the wake-up while debugging a high current issue during mp3 playback.

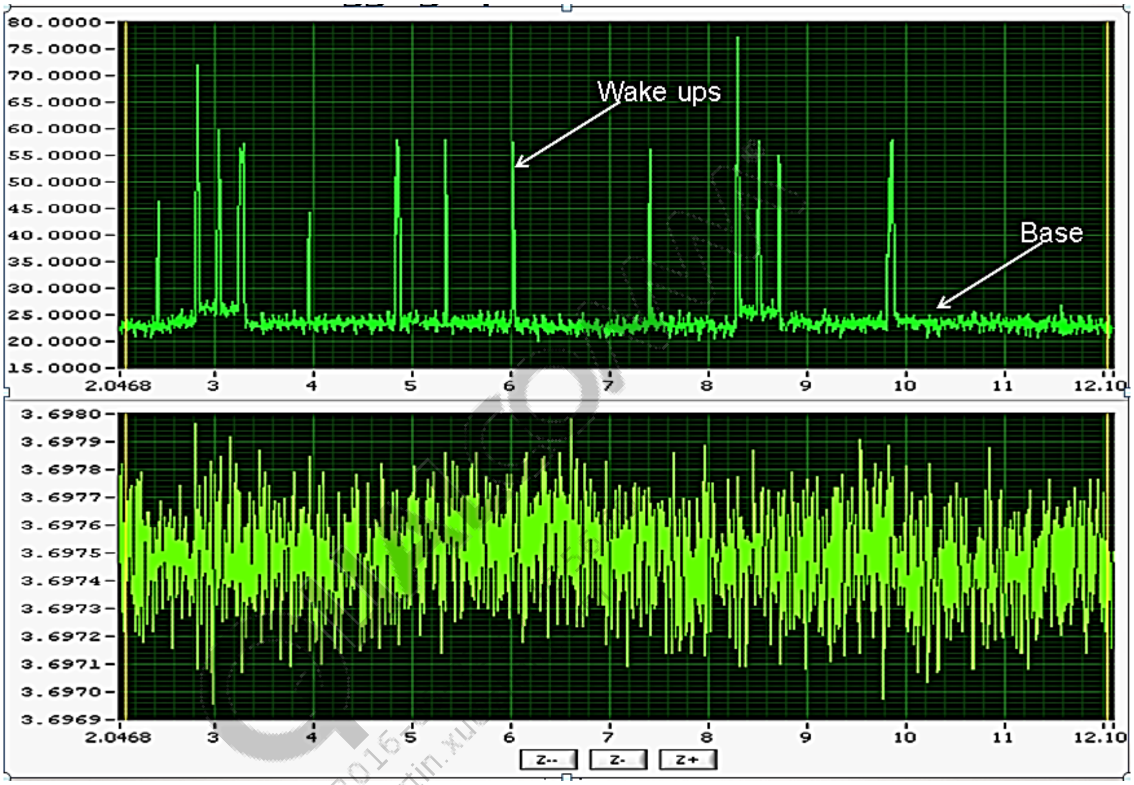


Figure 3-1 mp3 playback waveform analysis

3.2 Video – 720p and 1080p playback

- Ensure that Tunnel mode for audio is enabled during video playback.
- To check Tunnel mode:
  a. Pull build.prop from the system using the following adb command:

```
adb pull /system/build.prop > c:\temp\
```

  - Check the following property in the build.prop file:

```
av.offload.enable  [for video + audio use case]
```

If the above is true then audio playback occurs in Tunnel or Normal mode. The following build.prop snippet shows audio playback occurring in Tunnel mode:

```
#Enable offload audio video playback by default
av.offload.enable=true
```

- Check the clocks on the device and whether they are aligned with the clock plan mentioned in [Q3].
- Use the same qtc77.mp4 (720p video clip) and qtc88 (1080p video clip) test clip for power measurement and follow the QTI test procedure to measure video playback power consumption; this ensures a proper comparision with the QTI-released measurement.
- Account for hardware and software factors that potentially impact video decode power consumption when comparing with the QTI measurement.
- The default gallery app, Gallery2.apk, is used for power profiling.
- Check if OCMEM is being used; disabling OCMEM increases DDR traffic.

3.3 Camera – 1080p encode

- Follow the power measurement procedure and the settings mentioned in [Q2].
- Check the clocks the device and whether they are aligned with the clock plan mentioned in [Q4].
- Account for hardware and software factors that potentially impact video decode power consumption when comparing with the QTI measurement.
- Hardware and software differences such as camera sensor configuration, ISP, 3A algorithm (AF, AE, AWB) can impact power, and as such it is highly recommended to account for the power difference while debugging the power gap between the device and the QTI reference device.
- Ensure the fps is 30. Low light conditions may reduce the frame rate during encoding and previewing. To monitor the fps through adb, run the following commands:

```
adb shell
setprop persist.debug.sf.showfps 1
adb logcat | grep showfps
```

3.4 Display – Static image display

- Check the clocks on the device and whether they are aligned with the clock plan mentioned in [Q4].
- Account for hardware and software factors that potentially impact video decode power consumption when comparing with the QTI measurement.
- The following properties can be checked in the build.prop file, which is under system, or it can be checked using the adb getprop command as shown below.
  - Composition type

```
adb shell getprop | grep debug.composition.type
```

- Composition bypass enable/disable

```
adb shell getprop | grep persist.hwc.mdpcomp. kgsl interrupts check
```

- To get the kgsl interrupts count, use the following adb command. While running the use case, print it for multiple instances and get the difference.

```
adb shell cat /proc/interrupts | grep kgsl
```

- To check the number of hardware layers – While capturing dumpsys, make sure none of the layers are being updated. If the status bar is getting updated because of USB charging then the wrong information will be seen. To avoid this, run the command in the background after a device sleep of 10 to 15 sec as shown below.

```
adb shell dumpsys SurfaceFlinger
```

To capture dumpsys in the background after the USB is disconnected, run:

```
adb shell sleep 10 && dumpsys SurfaceFlinger > /data/sf.txt
```

3.5 Gfx – PowerLift and Egypt

- Check the clocks on the device and whether they are aligned with the clock plan mentioned in [Q4].
- Account for any hardware and software factors that potentially impact video decode power consumption when comparing with the QTI measurement.
- Mount debugfs by using the following adb command:

```
adb shell
mount –t debugfs none /sys/kernel/debug
```

- Always measure 3D clocks from debugfs real measurements supplied directly by the clock driver.
- kgsl thinks the active clock rate should be gpuclk as defined in /sys/class/kgsl/kgsl-3d0.
- Check the parameters, i.e., gpu_available_frequencies, idle_timer, gpubusy, gputop, gpu governor, etc.
  - gpubusy
    - In gpubusy, there are two values; the first value is the GPU busy time and the second is the total system time (~1 sec)
    - (first_value/second_value)*100 gives percentage of the last second the GPU core was busy

- gputop
  - gputop provides information of how long the GPU stayed in each state, i.e., busy_time, total_time (~1 sec), time_in_TURBO, time_in_NOMINAL, time_in_SVS, and time_in_SLEEP.
  - The time spent in each state is depicted sequentially, as shown in Figure 3-2
  - gputop updates the per-GPU clock statistics every second
  - To capture gputop and its output run:

```
adb shell
cd /sys/class/kgsl/kgsl-3d0
cat gputop
```

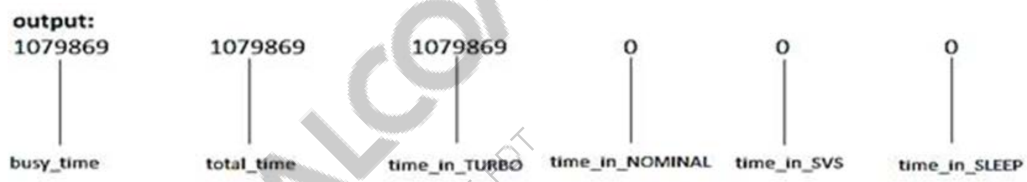


Figure 3-2 Example gputop output

- gpu governor
  - To check gpu governor, under adb shell go to /sys/class/kgsl/kgsl-3d0/pwrscale.
- gpu frequency
  - To check gpu frequency, use the following adb command:

```
cat /sys/kernel/debug/clk/oxili_gfx3d_clk/measure
```

- Check gpu power levels in the msm8974-gpu.dtsi file.
- The following properties can be checked in the build.prop file, which is under system, or it can be checked using the adb getprop command as shown below.
  - To check composition type

```
adb shell getprop | grep debug.composition.type
```

  - To check composition bypass enable/disbale

```
adb shell getprop | grep persist.hwc.mdpcomp.enable
```

- To retrieve the the kgsl interrupts count, use the the following adb command; while running the use case, print it for multiple instances and get the difference.

```
adb shell cat /proc/interrupts | grep kgsl
```

- To check the number of hardware layers, use the following adb command; while capturing dumpsys, make sure none of the layers are being updated. If the status bar is getting updated due to USB charging, run the command in the background after a device sleep of 10 to 15 sec as shown below.

```
adb shell dumpsys SurfaceFlinger
```

To capture dumpsys in the background after the USB is disconnected, run:

```
adb shell sleep 10 && dumpsys SurfaceFlinger > /data/sf.txt
```

3.6 Streaming/browser – 720p video streaming and web browser

- Check the clocks on the device and whether they are aligned with the clock plan as described in [Q4].
- Use the same setup as mentioned in [Q6] for power measurement and follow the QTI test procedure in [Q2] to measure the browser power to correctly compare with the QTI-released measurement.
- Account for any hardware and software factors that potentially impact the browser power consumption when comparing measurements with the QTI-released measurement.
- For browser and video streaming use cases, power analysis needs to be done in both the active and idle period; the active period is when the CPU wakes up and the idle period is when the CPU is in the Idle state. Example browser use case waveforms are shown in Figure 3-3 and Figure 3-4.

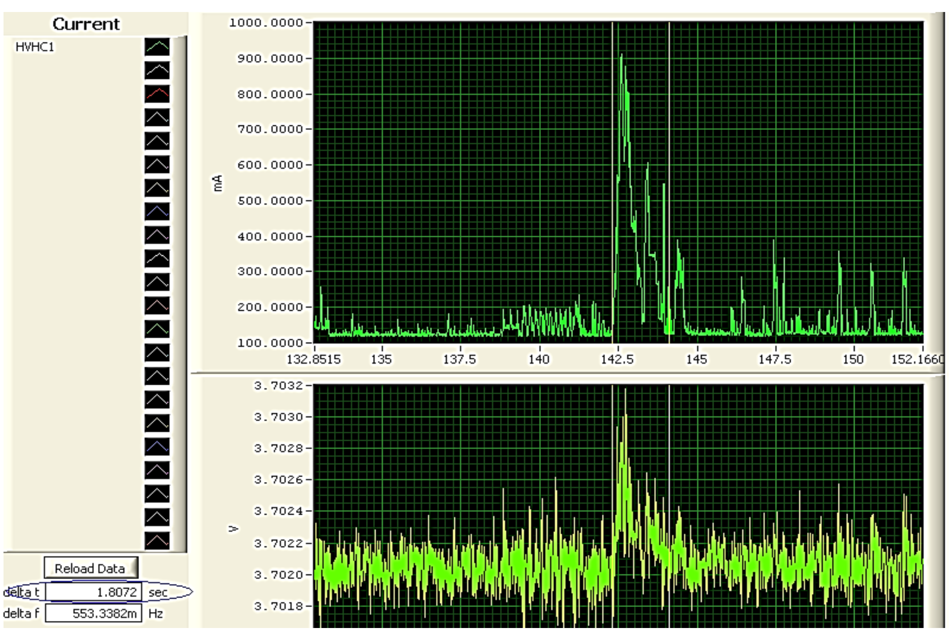


Figure 3-3 Browser waveform (active period ~1.8 sec)

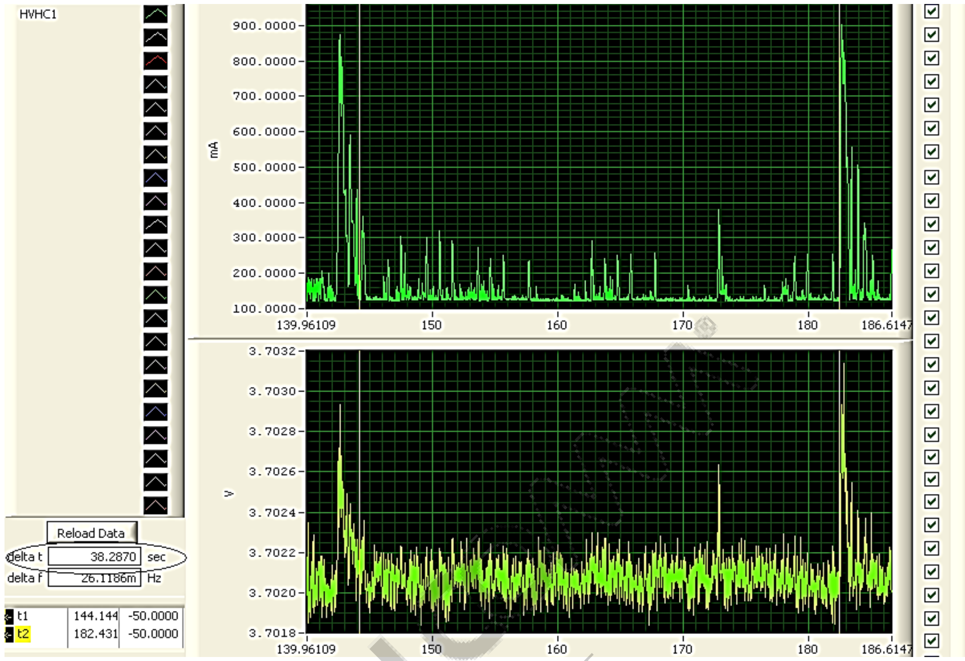


Figure 3-4 Browser waveform (idle period ~38.28 sec)

4 Capturing Debugging Logs

4.1 Powertop

To collect the powertop debugging log:

1. Connect a USB cable from the device to a PC.
2. Use the following adb commands:

```
adb root
adb remount
```

3. Push the powertop binary to the device data folder and make it executable by using the following adb commands:

```
adb push <powertop binary location>\powertop /data/
adb shell "chmod 777 /data/powertop"
```

NOTE: It may not be necessary to push the binary since some builds already include it.

4. Start the use case, i.e., play an mp3.
5. Run the following adb commands:

```
adb shell
sleep 10 && /data/powertop -d > /data/powertop.txt &
sleep 10 &&  powertop -d > /data/powertop.txt &
```

6. Disconnect the USB cable within 10 sec of running the above commands.
7. After the 10 sec period, powertop data will start capturing in the background for the next 15 sec. At this point, take a quick measurement to verify the power number is as expected.
8. After a total of 25 sec, reconnect the USB cable and extract the powertop log using the following adb command:

```
adb pull /data/powertop.txt <local location>
```

4.2 Top

To collect the top debugging log:

1. Connect a USB cable from the device to a PC.
2. Run the following adb commands:

```
adb root
adb remount
```

3. Start the use case, e.g., play an mp3.
4. Run the following adb commands:

```
adb shell
sleep 10 && top -t -m 10 -d 10 > /data/top.txt &
```

5. Disconnect the USB cable within 10 sec of running the above commands; after the 10 sec period, top data will start capturing.
6. Take a quick power measurement; the power measurement will be little higher than expected because the top process is running.
7. After a total of 20 sec, reconnect the USB cable and extract the top log using the following adb command:

```
adb pull /data/top.log <local location>
```

4.3 Ftrace

To collect the ftrace log:

1. Connect a USB cable from the device to a PC.
2. Run the following adb commands:

```
adb root
adb remount
```

3. Start the use case, i.e., play an mp3.

4. Run the following adb commands:

```
adb shell
mount -t debugfs nodev /d/
sleep 10 && echo 16384 > /d/tracing/buffer_size_kb && echo "" >
/d/tracing/set_event &&
echo "" > /d/tracing/trace && echo "irq:* sched:* power:*
msm_low_power:* kgsl:*" > /d/tracing/set_event && sleep 60 && cat
/d/tracing/trace > /data/local/ftrace.txt &
```

5. Disconnect the USB cable within 10 sec of running the above commands and ensure the display is off for mp3 use case.
6. Wait 2 min for data collection.
7. Reconnect the USB cable and extract the ftrace log by running the following adb command:

```
adb pull /data/local/ftrace.txt <local location>
```

4.4 msmpmstats

To collect the msmpmstats log:

1. Connect a USB cable from the device to a PC.
2. Run the following adb commands:

```
adb root
adb remount
```

3. Start the use case, i.e., play an mp3.
4. Run the following adb commands:

```
adb shell
sleep 10 && echo reset > /proc/msm_pm_stats && sleep 25 && cat
/proc/msm_pm_stats > /data/msmpmstats.txt &
```

5. Disconnect the USB cable within 10 sec of running the above commands.
6. After a total of 40 sec, reconnect the USB cable and extract the msmpmstats log by running the following adb command; this captures data for 25 sec of the duration.

```
adb pull /data/msmpmstats.txt <local location>
```

If the background command does not work:

1. Follow the same process to step 2 and start the use case.
2. Run the following adb command and save the stats in any textpad.

```
adb shell
echo reset > /proc/msm_pm_stats
cat /proc/msm_pm_stats
```

3. Disconnect the USB cable.
4. After a total of 25 sec, reconnect the USB cable and extract the msmpmstats log by running the following adb command; this captures the data for 25 sec of the duration and time duration can be set depending on need.

```
adb shell
cat /proc/msm_pm_stats
```

5. Get the difference of stats counts and duration in each state collected in step 1 and step 2.

4.5 Wakelock (dumpsys power)

To collect the wakelock debugging log:

1. Connect a USB cable from the device to a PC.
2. Run the following adb commands:

```
adb root
adb remount
```

3. Start the use case, i.e., play an mp3.
4. Run the following adb commands:

```
adb shell
sleep 10 && dumpsys power > /data/dumpsys.txt &
```

5. Disconnect the USB cable within 10 sec of running the above command.
6. After a total of 20 sec, reconnect the USB cable and extract the dumpsys log by running the following adb command:

```
adb pull /data/dumpsys.txt <local location>
```

To check wakeup_sources for wakelocks:

1. Follow the previous process through step 2 and run the use case.
2. Run the following adb commands:

```
adb shell
cat sys/kernel/debug/wakeup_sources
```

3. Look for active_since.

4.6 SurfaceFlinger

To collect the SurfaceFlinger debugging log:

1. Connect a USB cable from the device to a PC.
2. Run the following adb commands:

```
adb root
adb remount
```

3. Start the use case, i.e., static display.
4. Run the following adb commands:

```
adb shell
sleep 10 && dumpsys SurfaceFlinger > /data/sf.txt &
```

5. Disconnect the USB cable within 10 sec of running the above commands.
6. After a total of 10 sec, reconnect the USB cable and extract the dumpsys SurfaceFlinger log by running the following adb command:

```
adb pull /data/sf.txt <local location>
```

4.7 MDP stats

To collect the MDP stats debugging log:

1. Connect a USB cable from the device to a PC.
2. Run the following adb commands:

```
adb root
adb remount
adb shell
mount -t debugfs none /sys/kernel/debug
cd /sys/kernel/debug/mdp
```

3. Start the use case, i.e., static image display.
4. With the USB cable connected, run the following adb commands two to three times:

```
adb shell
while
do cat stat
sleep 1
done
```

5. Copy and paste the results into a text file.
6. Look for the count increments for each pipe.

4.8 Interrupts

To collect the interrupts debugging log:

1. Connect a USB cable from the device to a PC.
2. Run the following adb commands:

```
adb root
adb remount
```

3. Start the use case.
4. With the USB cable connected, run the following adb commands multiple times:

```
adb shell
while
do cat /proc/interrupts
sleep 1
done
```

   - To capture only kgsl interrupts, run the following adb commands:

```
adb shell
while
do cat /proc/interrupts |grep kgsl
sleep 1
done
```

5. Copy and paste the results into a text file.
6. See the increment in counts of the interrupts.

4.9 Clock dump

There are two methods for collecting a clock dump. The recommended method is to use JTAG.

4.9.1 Clock dump capture using JTAG

To collect the clock dump debugging log:

1. Start T32 RPM and APPS_CORE0 by running t32start.cmd in <meta build location>/common/t32.
2. In RPM, load the .elf file by going to RPM Commands→Load Symbols and selecting the .elf file.
3. On the home screen, disable the PMIC watchdog by running the following commands in RPM:

```
Break [pause button]
v pmic_wdog_enable = 0
Go [Play symbol]
```

4. Start the use case.
5. Quickly take a power measurement for 10 to 15 sec to verify that the power number is within the expected range.
6. Press **Pause** to break randomly.
7. Go to <modem build>/modem_proc/core/systemdrivers/clocks/scripts/msm8974 and drag and drop the testclock.cmm file to a T32 RPM command line; press **Enter**.
8. When prompted, type **all** to start the clock data dump.
9. Wait for all data to process.
10. Once data is finished processing, scroll to the top of the data, click the top left corner, and select **To Clipboard all** to copy data.
11. Paste data into a text file.
12. Reboot the device and repeat; take a total of three clock dumps.

4.9.2 Clock dump capture using adb

NOTE: This section was added to this document revision.

To collect the clock dump debugging log through adb:

1. Connect USB to the device.
2. Run the test scenario.

3. Run the following command:

```
adb shell
mount -t debugfs none /sys/kernel/debug
sleep 10 && while true; do echo
\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=\=
\=\=\=\=\=\=\=\=\=\=\=\=\=; cat /proc/uptime; cd /sys/kernel/debug/clk;
for i in *; do if [ -d $i ]; then if [ "$(cat $i/enable)" == "1" ]; then
if [ -e $i/measure ]; then echo $i \=\> enable:`cat $i/enable`
measure:`cat $i/measure`; else echo $i \=\> enable:`cat $i/enable`
rate:`cat $i/rate`; fi; fi; fi; done; echo \-\-\-\-\-\-\-\-\-\-\-\-\-\-\-
\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-; cd
/sys/class/regulator; for i in *; do if [ -d $i ]; then if [ -e $i/state
]; then if [ "$(cat $i/state)" == "enabled" ]; then if [ -e
$i/microvolts ]; then echo $i \=\> name:`cat $i/name` state:`cat
$i/state` microvolt:`cat $i/microvolts`; else echo $i \=\> name:`cat
$i/name` state:`cat $i/state` microvolt: N\/A; fi; fi; fi; fi; done;
sleep 2; done > /data/dumpclk.txt &
```

4. Disconnect the USB when the PID is displayed in the command prompt window.
5. Wait 30 seconds, then plug the USB back in. Kill the process with the PID displayed in the previous step, and pull the dumpclk.txt file.

```
adb shell "kill <PID>"
adb pull /data/dumpclk.txt
```


4.10 PLL dump

To collect the PLL dump debugging log:

1. Start T32 RPM and APPS_CORE0 by running the t32start.cmd in <meta build location>/common/t32.
2. In RPM, load the .elf file by going to RPM Commands→Load Symbols and selecting the .elf file.
3. On the home screen, disable the PMIC watchdog by entering the following commands in RPM:

```
Break [pause button]
v pmic_wdog_enable = 0
Go [Play symbol]
```

4. Start the use case.
5. Quickly take a power measurement for 10 to 15 sec to verify that the power number is within the expected range.
6. Press **Pause** to break randomly.

7. Go to <modem build>/modem_proc/core/systemdrivers/clocks/scripts/msm8974 and drag and drop the testpll.cmm file to the T32 RPM command line; press **Enter**.
8. When prompted, type **all** to start clock data dump.
9. Wait for all data to process.
10. Once data is finished processing, scroll to the top of data, click the top left corner, and select **To Clipboard all** to copy data.
11. Paste data into a text file.
12. Reboot the device and repeat; take a total of three PLL dumps.

4.11 PMIC dump

To collect the PMIC dump debugging log:

1. Start T32 RPM and APPS_CORE0 by running the t32start.cmd in <meta build location>/common/t32.
2. In RPM, load the .elf file by going to RPM Commands→Load Symbols and selecting the .elf file.
3. Disable the PMIC watchdog by entering the following commands:

```
Break [pause button]
v pmic_wdog_enable = 0
Go [Play symbol]
```

4. Start the use case.
5. Quickly take a power measurement to verify that the power number is within the expected range.
6. Press **Pause** to break randomly.
7. Type the following in the T32 RPM window:

```
CD.DO rpm_proc\core\systemdrivers\pmic\ scripts\PMICDump.cmm
```

8. Change the location of the log file, if desired.
9. Select the correct platform.
10. Press **Dump**.
11. Wait for the data to process; by default, the raw PMIC dump file is saved in c:\temp\pmicdump.xml.
12. Use the following command to convert the raw PMIC register setting to a human-readable file:

```
cd rpm_proc\core\systemdrivers\pmic\scripts
python PMICDumpParser.py --flat=pm8941/v3/CORE_ADDRESS_FILE_
CUSTOMER.FLAT --file=c:\temp\pmicdump.xml > pmicdump.txt
```

4.12 GPIO dump

All GPIOs in the hardware use the following script, which can be run from the RPM T32 window, to read the current configuration of all GPIOs in the hardware:

```
do <Modem_Build>\modem_proc\core\systemdrivers\tlmm\scripts\
tlmm_gpio_hw.cmm
```

4.13 MSM bus driver debug – Client data

1. Connect a USB cable from the device to a PC.
2. Run the following adb commands:

```
adb root
adb remount
```

3. Start the use case.
4. With the USB cable connected, run the following adb commands multiple times:

```
adb shell
cd /d/msm-bus-dbg/client-data/
ls
cat mdss_mdp
```

   This gives an output of mdss_mdp client data. In this same way, bandwidth requests from other clients can be captured.

5. Copy and paste the results into a text file.