Power Consumption Optimization and Debugging for MSM8916 Devices

80-NL239-48 C

October 10, 2014

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

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

Contents

1 Introduction 7
1.1 Purpose 7
1.2 Conventions 7
1.3 References 7
1.4 Technical assistance 8
1.5 Acronyms 8

2 Use Case-Specific Debugging 9
2.1 Optimizing Sleep/Rockbottom current 9
2.1.1 Possible culprits impacting sleep current 9
2.1.2 Test setup-related issues impacting power 9
2.1.3 Debug methods for higher sleep current 11
2.1.4 Ensuring APSS power collapse 11
2.1.5 Ensuring modem sleep 12
2.1.6 Ensuring WCNSS power collapse 13
2.1.7 Ensuring all masters are power-collapsed 14
2.1.8 Ensuring overall system entering low power modes 14
2.2 Optimizing standby current 17
2.2.1 Debugging high standby current – Awake current issue 18
2.2.2 Test setup-related issues impacting power 18
2.2.3 Inspecting and optimizing average current during page wakeups 18
2.3 Optimizing talk current 18
2.3.1 Test setup-related issues impacting power 18
2.3.2 Inspecting clocks 18
2.3.3 Component breakdown 19
2.4 Optimizing data call current 19
2.4.1 Test setup-related issues impacting power 19
2.4.2 Inspecting clocks 19
2.4.3 Component breakdown 19
2.4.4 APPS debugging 19
2.5 Audio – mp3 20
2.5.1 Audio power waveform analysis 21
2.6 Video – 720p and 1080p playback 21
2.7 Camera – 1080p encode 22
2.8 Display – Static image display 22
2.9 Gfx – PowerLift and Egypt 23
2.10 Streaming/browser – 720p video streaming and web browser 25

3 Dumps/Logs and Tools for Debugging 27
3.1 PowerTop 27

3.2 Top 30
3.3 PerfTop 31
3.4 Ftrace/Pytime chart 32
3.5 Systrace 33
3.6 Clock dump debugging 38
3.7 NPA dump debugging 40
3.8 MSM bus request debugging 41
3.9 Governor parameter check 42
3.10 Check for perf-locks 44
3.11 Waveform comparison 45
3.12 Wakelocks comparison 47
3.12.1 wakeup_sources 47
3.12.2 Dumpsys power 49
3.13 msmpmstat analysis 50
3.14 Rail-level comparison 51
3.15 Bandwidth request comparison 52

4 Logs/Dumps Collection Procedure 53

4.1 JTAG license information 53
4.2 Starting T32 session 53
4.3 Clock dumps 54
4.3.1 Clock dumps using JTAG 54
4.3.2 Clock dumps using the adb shell 54
4.4 PMIC dumps 55
4.5 PowerTop 55
4.6 Top 56
4.7 Ftrace 57
4.8 msmpmstats 57
4.9 Wakelock (dumpsys power) 58
4.10 SurfaceFlinger 59
4.11 MDP statistics 59
4.12 Interrupts 60
4.13 GPIO dump 60
4.14 MSM bus driver debug – Client data 61
4.15 Perf tool 61

5 Additional Debug Details and Logging Procedures 62

5.1 RPM debug 62
5.1.1 Saving RPM RAM dumps 62
5.1.2 Loading RPM dumps onto T32 and extracting logs 62
5.1.3 RPM log (alternate way – Using ADB) 64
5.2 Modem debug logs 64

6 Debugging APSS 66

6.1 Low Power mode debugging 66
6.1.1 LPM debugging – VDD_min/XO shutdown 67
6.1.2 Kernel message analysis 69

6.2 Enabling/disabling various modes 70

7 Power Features Verification 73

7.1 To check XO shutdown and VDD minimization 73

7.1.1 To verify XO shutdown 73

7.1.2 To verify VDD minimization 73

7.2 MPDecision 73

7.3 CPU hotplug 74

7.4 GDSC status 74

7.5 To check voltage corners for CX and MX 75

7.6 Supported retention voltages for CX and MX 76

7.7 Check CX RBCPR enablement and type 76

7.8 Check APSS RBCPR enablement 77

7.9 To check modem DCVS 77

Figures

Figure 2-1 mp3 playback waveform analysis 21
Figure 2-2 Example gputop output 24
Figure 2-3 Browser waveform (active period ~1.8 sec) 26
Figure 2-4 Browser waveform (idle period ~38.28 sec) 26
Figure 3-1 PowerTop log explanation 28
Figure 3-2 PowerTop analysis for static image display use case 29
Figure 3-3 Ftrace log analysis – Static display use case with high current caused by kgsl interrupt 32
Figure 3-4 Ftrace logs analysis – Static display use case with high current caused by synaptic touch interrupt 33
Figure 3-5 mp3 playback on reference device A 45
Figure 3-6 mp3 playback on device B with power regression 46

Tables

Table 1-1 Reference documents and standards 7
Table 3-1 MSM8916 clock mapping 38
Table 3-2 Waveform analysis 46
Table 3-3 wakeup_sources log snippet for rock bottom sleep use case 47
Table 3-4 wakeup_sources log snippet for mp3 use case 48
Table 3-5 Rail-level comparison details 51
Table 7-1 Retention levels 73

Revision history

Revision	Date	Description
A	Jul 2014	Initial release
B	Sep 2014	Added Chapter 7
C	Oct 2014	Updated template

1 Introduction

1.1 Purpose

This document provides details on how to optimize specific power test cases and debug issues related to the RPM, applications, multimedia, and modem.

The document is intended for OEMs who need to debug core, Android, and multimedia power issues on MSM8916 chipsets.

1.2 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**.

Shading indicates content that has been added or changed in this revision of the document.

1.3 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	*Presentation: MSM8916 System Power Overview*	80-NL239-6
Q4	*MSM8916 Linux Android Software Debug Manual Software Product Document*	SP80-NL239-5
Q5	*MSM8916 Linux Android Current Consumption Data*	80-NK807-7
Q6	*MSM8960 LA Server and Hardware Component Setup for Video Streaming and Browser Test Cases*	80-N8520-1
Q7	*Presentation: MSM8974 Power Debugging*	80-NA157-68

1.4 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.

1.5 Acronyms

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

2 Use Case-Specific Debugging

This chapter provides details on how to debug and optimize current consumption for each of the QTI standard power dashboard use cases. The steps to debug high power consumption as compared to the QTI reference power data by collecting relevant logs and their interpretation is discussed.

2.1 Optimizing Sleep/Rockbottom current

2.1.1 Possible culprits impacting sleep current

Usually, if the sleep current is substantially high, it is likely that one or more of the following has occurred:

- One or more major subsystems on the MSM™ chipset failed to go into their lowest power mode preventing XO shutdown/VDD_min. On the MSM8916, these subsystems may be:
    - Applications Processor Subsystem (APSS) not going into power collapse
    - Modem Peripheral Subsystem (MPSS) not going into sleep
    - Wireless Connect Subsystem (WCNSS) not going into sleep
- One or more peripheral devices (outside of the MSM chipset) did not going into their lowest power state.

If all the subsystems are in their lowest power states, the sleep current might only be slightly higher than expected (however, not necessarily). This extra current might be because of leakage caused by:

- GPIOs not being configured to their suggested low-power configuration before entering the Sleep state
- PMIC SMPS/LDOs are not configured to their suggested low-power configuration before entering sleep

2.1.2 Test setup-related issues impacting power

To determine whether test setup-related issues impact power, ensure that the setup is correct. Sleep current is measured either by placing the phone in Airplane mode or by connecting to a call box using desired air interface, e.g., WCDMA, LTE, etc., in Standby mode. See [Q2] for the setup procedure for various standby modes.

Airplane mode

- No wakeups are expected during Airplane mode.
- The phone must exhibit a constant current consumption behavior.
- If there are random wakeups and high current cycles, first check if the issue is caused by the timer or unknown spurious interrupts.

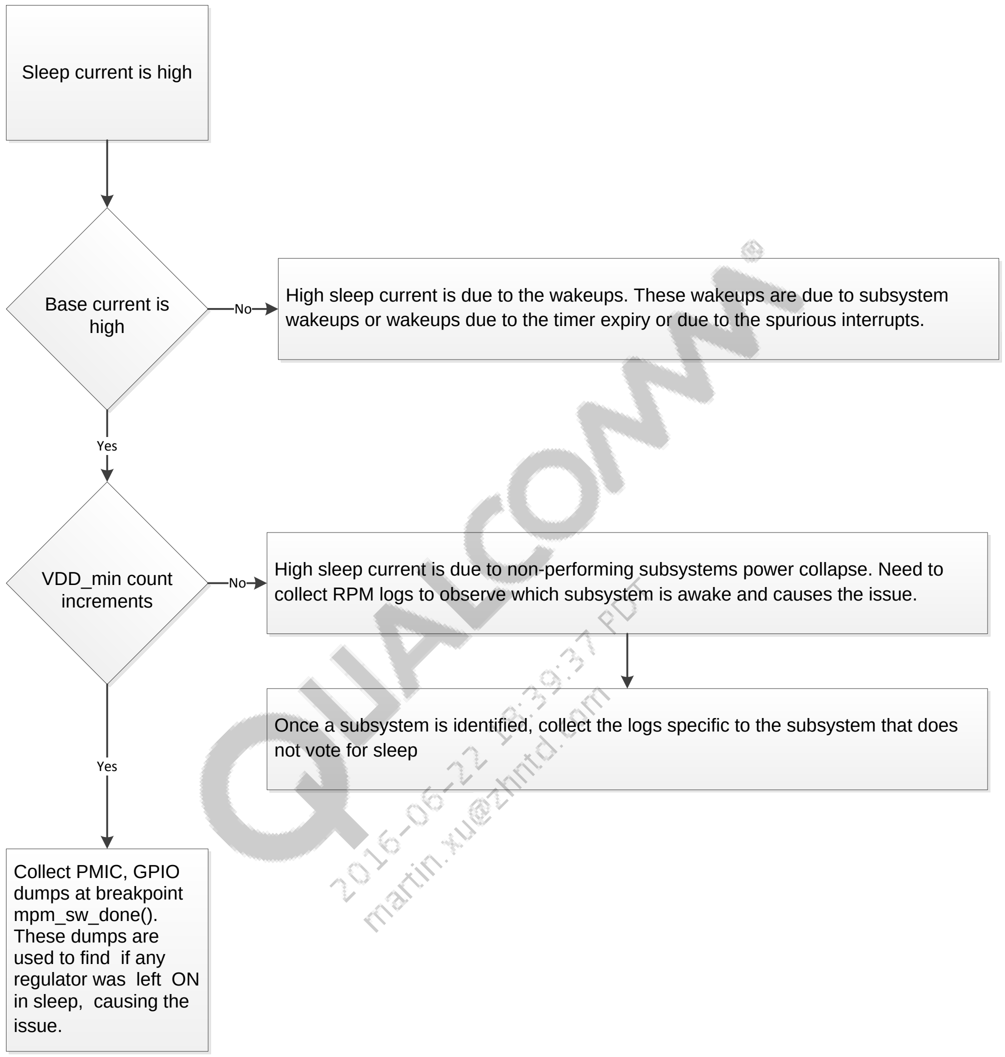
Standby mode

In Online mode, ensure that the phone is camped to the call box and is getting repeated paging cycles as expected.

System variables

Always ensure that the variable sleep_allow_low_power_modes is set to TRUE. If this variable is set to FALSE, the system does not enter low power modes.

2.1.3 Debug methods for higher sleep current


Sleep current is high
Base current is high
No
High sleep current is due to the wakeups. These wakeups are due to subsystem wakeups or wakeups due to the timer expiry or due to the spurious interrupts.
Yes
VDD_min count increments
No
High sleep current is due to non-performing subsystems power collapse. Need to collect RPM logs to observe which subsystem is awake and causes the issue.
Once a subsystem is identified, collect the logs specific to the subsystem that does not vote for sleep
Yes
Collect PMIC, GPIO dumps at breakpoint mpm_sw_done(). These dumps are used to find if any regulator was left ON in sleep, causing the issue.

2.1.4 Ensuring APSS power collapse

2.1.4.1 Check RPM external logs

APSS power collapse can be confirmed by observing the RPM external logs. Check for the following messages in the RPM logs:

- 34.521729 – rpm_shutdown_req (master: "APSS") (core: 0)
- 34.521759 – rpm_shutdown_ack (master: "APSS") (core: 0)
- 34.522064 – rpm_master_set_transition (master: "APSS") (leaving: "Active Set") (entering: "Sleep Set")

34.521xxx is the timestamp at which the respective action occurs in the above logs. As APSS is expected to go to power collapse as soon as there are no tasks to block apps Low Power Mode (LPM), if the above messages are not available in RPM logs, it can be assumed that the applications are already in power collapse, or it might not have entered in power collapse.

The correct point to break the execution (on RPM) is to see power collapse after booting the device and allow the display to power off, and then break the JTAG for taking RPM logs.

2.1.4.2 Check RPM data structure in RPM

APSS sleep can be confirmed by checking the RPM data structure on RPM. After the display is off and given enough time to go to sleep, break the Trace32 execution in RPM randomly to monitor this variable.

Check rpm.ees[0].subsystem_status, 0→APPS information

- If the applications are in Sleep mode, subsystem status of APSS is SPM_SLEEPING.
- If the applications still operate in Active mode, the subsystem status of APSS is SPM_AWAKE.
- If the applications are *not* in sleep and were expected to be, collect the required logs from APSS side to determine the cause.

2.1.5 Ensuring modem sleep

2.1.5.1 Check RPM external logs

Modem sleep can be confirmed by looking at the RPM external logs. Check for the following messages in RPM logs:

- 34.513367 – rpm_shutdown_req (master: “MSS”) (core: 0)
- 34.513397 – rpm_shutdown_ack (master: “MSS”) (core: 0)
- 34.514038 – rpm_master_set_transition (master: “MSS”) (leaving: “Active Set”) (entering: “Sleep Set”)

As the modem is expected to go to sleep as soon as there are no other high-priority tasks pending to run, if the above messages are not available in RPM external logs, it can be assumed that the modem is already in sleep, or it might not have entered sleep.

As the modem is expected to wake up during every paging cycle, the above messages are easy to capture as they appear once for every DRX cycle.

2.1.5.2 Check RPM data structure in RPM

Modem sleep can be confirmed by checking the RPM data structure on RPM. After display is OFF and given enough time to go to sleep, break the Trace32 execution in RPM randomly to monitor this variable.

Check rpm.ees[1].subsystem_status, 1→Modem information

- If the modem is in Sleep mode, the subsystem status of the modem is SPM_SLEEPING.
- If the modem still operates in Active mode, the subsystem status of the modem is SPM_Awake.
- If the modem is *not* in sleep and was expected to be, collect the required logs from the Modem side to determine the cause.

2.1.6 Ensuring WCNSS power collapse

2.1.6.1 Check RPM external logs

WCNSS power collapse can be confirmed by looking at RPM external logs. Check for the following messages in RPM logs:

- 34.513367 – rpm_shutdown_req (master: "WCNSS")
- 34.513397 – rpm_shutdown_ack (master: "WCNSS")
- 34.514038 – rpm_master_set_transition (master: "WCNSS") (leaving: "Active Set") (entering: "Sleep Set")

2.1.6.2 Check RPM data structure in RPM

WCNSS sleep can be confirmed by checking the RPM data structure on RPM. After display is off and given enough time to go to sleep, break the Trace32 execution in RPM randomly to monitor this variable.

Check rpm.ees[2].subsystem_status, 2→WCNSS information

- If WCNSS is in Sleep mode, the subsystem status of WCNSS is SPM_SLEEPING.

  If WCNSS still operates in Active mode, the subsystem status of WCNSS is SPM_AWAKE.
- If the WCNSS is *not* in sleep and was expected to be, collect the required logs from the WCNSS side to determine the cause.

2.1.7 Ensuring all masters are power-collapsed

Obtain the master (EE) status from the ee-status.txt file, which is generated by the Hansei parser tool from the RPM dump.

ee-status.txt contains information about which subsystems (and their cores) are active or sleeping.

```
*** APPS ***
status:           SPM_AWAKE
num_active_cores: 2
pending_bringups: 0x00000000
*** MSS ***
status:           SPM_SLEEPING
num_active_cores: 0
pending_bringups: 0x00000000
*** WCSS ***
status:           SPM_SLEEPING
num_active_cores: 0
pending_bringups: 0x00000000
```

2.1.8 Ensuring overall system entering low power modes

Supported low power modes are:

- RPM halt
- XO shutdown
- VDD_min

Before checking whether the system does not enter low power modes, ensure that the following variable is set properly:

- sleep_allow_low_power_modes must be set to TRUE.

To verify if the system enters XO shutdown and VDD_min, check if the following breakpoints are hit using T32:

- For XO shutdown: xo_shutdown_enter() and then mpm_sw_done()
- For VDD_min: vdd_min_enter() and then mpm_sw_done()
- Verifications of XO shutdown/VDD_min countXO shutdown using ADB and T32

To check the XO shutdown/VDD_min count using ADB:

1. Connect a USB and check VDD_min counts using the command below to print RPM statistics.
2. Disconnect the USB.
3. Wait for 30 sec and connect the USB.
4. Get the RPM statistics by running the command below.
5. Check whether the VDD_min/XO shutdown count has increased.

Commands to get the RPM statistics are:

```
mount -t debugfs none /sys/kernel/debug
cat /sys/kernel/debug/rpm_stats
```

The output of the above commands look as shown below. The count represents how many times the XO shutdown/VDD_min occurred.

```
root:/ # cat  /d/rpm_stats
cat  /d/rpm_stats
RPM Mode:xosd
count:0
time in last mode(msec):0
time since last mode(sec):791
actual last sleep(msec):0
client votes: 0x00020001
RPM Mode:vmin
count:28
time in last mode(msec):8000
time since last mode(sec):475
actual last sleep(msec):233000
client votes: 0x00000000
```

Obtain sleep count from T32.

Check if the counts of the variables below increase after each expected power collapse (only one of the two would increment based on mode selected).

- sleep_stats[0].count //XO shutdown power mode counter
- sleep_stats[1].count //VDD_min shutdown power mode counter

2.1.8.1 Node Power Architecture (NPA) log analysis for XO shutdown/VDD_min

When there are no CXO client votes, the XO can be shut down to save power. The system might choose to enter VDD_min by keeping the CX and MX rails at the retention voltage to save more power.

- Check the RPM NPA logs snippet to ensure that the system enters low power modes:
- Check the Sleep/Uber Active state from RPM NPA logs. This must be 0 in the Sleep state, otherwise, the respective resource is expected by some client.
- If the active state is 6, it means that all the EEs are in Power Collapse mode, but aggregation of VDD CX and VDD_MX is not in retention voltage. In this state, the device does not enter VDD Minimization Power mode.

In this example, everything is needed by the system. The NPA log snippet is:

```
: npa_resource (name: “/sleep/uber”) (handle: 0x196E30) (units: on/off)
(resource max: 7) (active max: 7) (active state: 7)  (active headroom: -8)
(request state: 1)
 :         npa_client (name: sleep) (handle: 0x197060) (resource: 0x196E30)
(type: NPA_CLIENT_REQUIRED) (request: 1)
 :         npa_client (name: vddmx) (handle: 0x197098) (resource: 0x196E30)
(type: NPA_CLIENT_REQUIRED) (request: 4)
 :         npa_client (name: vddcx) (handle: 0x1970D0) (resource: 0x196E30)
(type: NPA_CLIENT_REQUIRED) (request: 2)
```

Check the RPM NPA log for the following messages to ensure which subsystem votes for CXO and prevents device sleep:

Check the active state of the “/xo/cxo” node from the RPM NPA logs.

From the below NPA log snippet, MPSS and APSS are requested for CXO resource.

```
: npa_resource (name: “/xo/cxo”) (handle: 0x196DE8) (units: Enable)
(resource max: 1) (active max: 1) (active state: 1)  (active headroom: 0)
(request state: 1)
 :         npa_client (name: MPSS) (handle: 0x19C780) (resource: 0x196DE8)
(type: NPA_CLIENT_LIMIT_MAX) (request: 1)
 :         npa_client (name: MPSS) (handle: 0x19C740) (resource: 0x196DE8)
(type: NPA_CLIENT_REQUIRED) (request: 1)
 :         npa_client (name: WCSS) (handle: 0x19C620) (resource: 0x196DE8)
(type: NPA_CLIENT_LIMIT_MAX) (request: 1)
 :         npa_client (name: WCSS) (handle: 0x19C5E0) (resource: 0x196DE8)
(type: NPA_CLIENT_REQUIRED) (request: 0)
 :         npa_client (name: LPASS) (handle: 0x19C260) (resource: 0x196DE8)
(type: NPA_CLIENT_LIMIT_MAX) (request: 1)
 :         npa_client (name: LPASS) (handle: 0x19C220) (resource: 0x196DE8)
(type: NPA_CLIENT_REQUIRED) (request: 0)
 :         npa_client (name: APSS) (handle: 0x196FF0) (resource: 0x196DE8)
(type: NPA_CLIENT_REQUIRED) (request: 1)
 :         npa_change_event (name: sleep) (handle: 0x198C10) (resource:
0x196DE8)
 :         end npa_resource (handle: 0x196DE8)
```

2.1.8.2 Subsystem VDD_min voting debug example

To find a voltage that a subsystem votes for Vdd_Dig while it is asleep/sleep, check the variable in T32 using the command below:

The following variable identifies the master that did not request retention voltage:

```
railway.rail_state[1].voter_list_head.voter_link = 0x0019BDC0 -> (
    voltage_corner = RAILWAY_RETENTION,
    explicit_voltage_in_uv = 0,
    suppressible = FALSE,
    id = 2, //2:WCSS subsystem
    rail = 1, //CX power rail
    sw_enable = TRUE ,
voter_link = 0x00199490 -> (
           voltage_corner = RAILWAY_SUPER_TURBO,
           explicit_voltage_in_uv = 0,
           suppressible = FALSE,
           id = 0, //0:APPS subsystem
           rail = 1,
           sw_enable = FALSE,
           voter_link = 0x00196F98)))))
```

In this case, the WCNSS subsystem votes for Vdd_Dig minimization whereas APSS requests Vdd_dig at TURBO voltage and prevents device entering VDD minimization mode.

2.1.8.3 Sleep current deltas/leakage – Further analysis

Further ways to analyze Sleep current delta/leakage include:

- Take the component breakdown and compare against the QTI reference device. Based on the power rail differences, focus on respective power rail optimization. If it is core current, clock analysis would be helpful.
- Check the GPIO configurations for PAD current optimization. Compare the GPIO configurations device under test just before system enters Sleep state. If there are any differences in the hardware compared to QTI reference device, ensure that GPIOs are configured to correct states. This helps to reduce the PAD current leakage.
- Estimate design deltas from schematics. Analyze the extra components on device under test compared to QTI reference device and approximate their power consumption if they have a breakout point to measure.

2.2 Optimizing standby current

Standby current is a combination of sleep current and awake current averaged over N number of cycles. First, separate the issue, i.e., determine whether it is due to high sleep current or high paging awake penalty. If the issue is due to high sleep current, follow the high sleep current issue debugging explained in Section 2.1.

2.2.1 Debugging high standby current – Awake current issue

If the paging awake current is higher than it should be, the cause could be a setup issue or a software issue.

2.2.2 Test setup-related issues impacting power

To determine if the cause is a test setup-related issue:

- Check the setup (see [Q2]).
- Ensure that testing is done with a callbox. Testing in a live network has higher awake current penalty and inconsistent run-to-run results, so it is not recommended.
- Disable neighbor cells in the callbox.
- Disable all data operations/browsing in the phone through the GUI.
- Ensure power levels used in the callbox are as recommended. These power levels include Rx cell power, Tx cell power, AGC, etc.
- Ensure all the debug logging is disabled through NV items.

2.2.3 Inspecting and optimizing average current during page wakeups

Check the portion of the awake current that is longer compared to QTI reference device awake. If it is XO warmup time/RF wakeup/RF processing/RF sleep/System sleep, F3 logs are useful for this analysis.

2.3 Optimizing talk current

2.3.1 Test setup-related issues impacting power

To analyze test setup-related issues impacting power:

1. Set NV item 00010 to the modem technology under test (WCDMA only/GSM only/1X only). The phone is expected to work when this NV item is set to Automatic mode as well.
2. Ensure that Tx and Rx power is at the specified power level as in the test case definition for QTI standard power dashboard. Also make sure that the PA gain state is 0.
3. Qualcomm Debug Subsystem (QDSS) must also be disabled when measuring power.
4. See [Q2] for any test setup-related issues/NV-related issues.

2.3.2 Inspecting clocks

Clocks are a main cause of power consumption in voice call scenarios. It is recommended to take a clock dump during a voice call and ensure that all the clocks operate at recommended frequencies. The other way of doing the same is to compare clock dumps during a voice call between the device under test and the QTI reference device and analyze the clock differences.

2.3.3 Component breakdown

Take a component breakdown and compare it against the QTI reference device. Based on the power rail differences, focus on respective power rail optimization. If it is core rail (VDD_CORE), the clock analysis would be helpful.

2.4 Optimizing data call current

2.4.1 Test setup-related issues impacting power

To analyze test setup-related issues impacting power:

1. Set NV item 00010 to the respective modem technology used (WCDMA only/LTE only/HDR only). The phone is expected to work when this NV item is set to Automatic mode as well.
2. Ensure that Tx and Rx power is at the specified power level as in the test case definition for QTI standard power dashboard. Also, ensure that the PA gain state is 0.
3. QDSS must also be disabled when measuring power.
4. See [Q2] for any test setup-related issues/NV-related issues.

2.4.2 Inspecting clocks

Clocks are a main cause of power consumption in data call scenarios. It is recommended to take a clock dump during a voice call and ensure that all the clocks operate at recommended frequencies. The other way of doing the same is to compare clock dumps during a voice call between the device under test and the QTI reference device and analyze the clock differences.

2.4.3 Component breakdown

Take a component breakdown and compare it against the QTI reference device. Based on the power rail differences, focus on respective module power optimization.

2.4.4 APPS debugging

Apart from the techniques discussed above, the following debug methods can be used to optimize data current, as it also involves the applications processor.

1. Analyzing power-top logs and compare them against the QTI reference device. In power-top logs, observe the time in each C-State (low power states), time in each frequency and interrupt activity. These tests provide a rough estimate of why the applications processor consume higher current.
2. Check for any extra running processes using top data. Top data can be obtained using adb shell with the command top –m <number of processes to be displayed>, e.g., “top –m 5” gives the top five running processes. These can be compared against reference device top logs, and any unexpected processes that can be stopped or killed.

3. Ftrace logs can be helpful for analysis of processes, which are running on each cores and the amount of time each core is at a particular frequency. It also helps to check which process hogs the CPU if kworker thread activity is more.
4. Log collection and analysis is discussed in the subsequent sections.

2.5 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.
3. In the logcat logs, look for the following message, which ensures that the Tunnel mode playback is ongoing.

   ```
   D/audio_hw_primary( 1580): out_set_parameters: enter: usecase(3:
   compress-offload-playback) kvpairs: closing=true
   D/audio_hw_primary( 1580): out_set_parameters: enter: usecase(3:
   compress-offload-playback) kvpairs: exiting=1
   ”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 nontunnel, i.e., normal playback is ongoing.

To debug effectively:

1. Check the clocks on the device, whether they are aligned with the clocks of the reference device.
2. 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.
3. Account for hardware and software factors that potentially impact mp3 playback power consumption when comparing device measurements with the QTI measurements.
4. Dump the codec registers if required to check if any gain parameters changes.
   - adb shell
   - mount -t debugfs debugfs /sys/kernel/debug
   - cd /sys/kernel/debug/asoc/msm8x16-snd-card/msm8x16_wcd_codec
   - cat codec_reg > /data/reg_dump.txt

5. Load the default Headset_cal.acdb on your device to rule out the power gap is due to audio effects introduce on headset path.
   - Default acdb files (Headset_cal.acdb) are placed at

     <Apps_build_Id>\LINUX\android\vendor\qcom\proprietary\mm-audio\audcal\family-b\acdbdata\8916\MTP\
   - On the device, the acdb files are found at adb shell "#cd /etc/acdbdata/"

2.5.1 Audio power waveform analysis

Figure 2-1 shows an example waveform for mp3 playback in Tunnel mode. When using battery waveform analysis, check the base current and the wakeup while debugging a high current issue during mp3 playback.

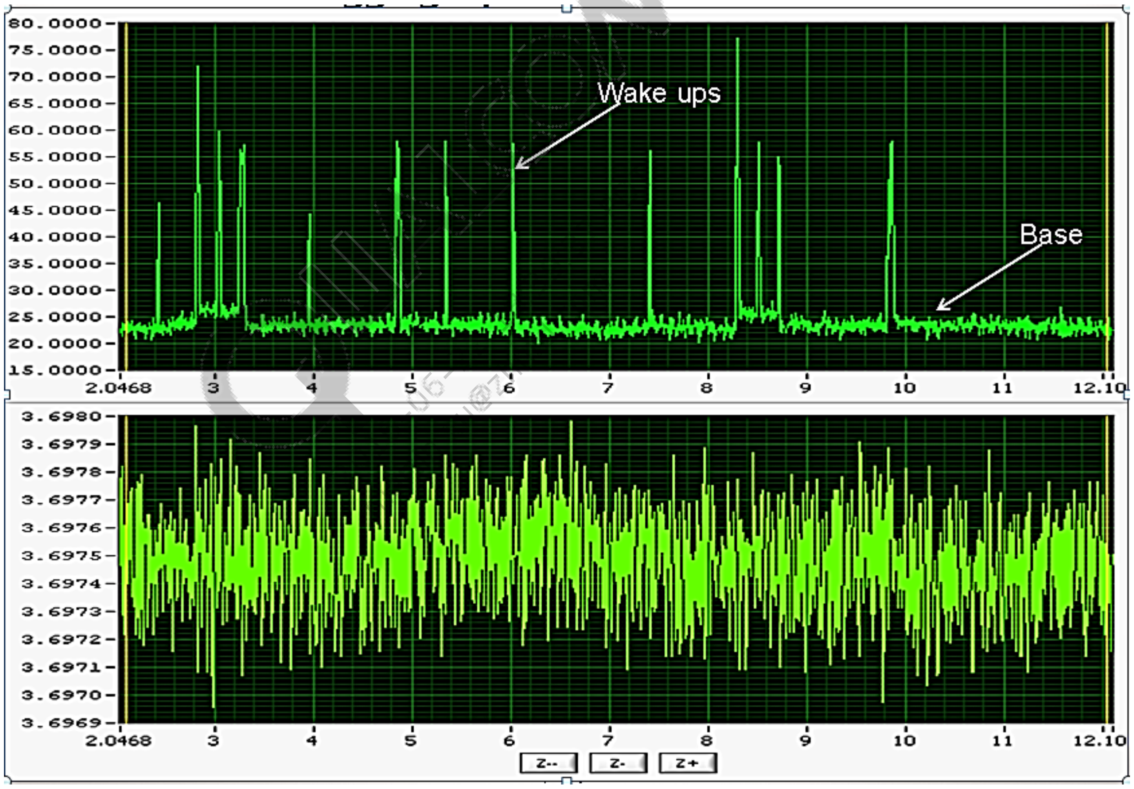


Figure 2-1 mp3 playback waveform analysis

2.6 Video – 720p and 1080p playback

On the MSM8916 chipset, the Tunnel (offload) mode for audio is not supported during video playback.

- Check the clocks on the device, whether they are aligned with the clocks of the reference device.
- 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 comparison with the QTI-released measurement.

- Account for hardware and software factors that potentially impact video decode power consumption when compared with the QTI measurement.
- The default gallery app, Gallery2.apk, is used for power profiling.

2.7 Camera – 1080p encode

1. Follow the power measurement procedure and the settings mentioned in [Q2].
2. Check the clocks on the device, whether they are aligned with the clocks of the reference device.
3. Account for hardware and software factors that potentially impact video decode power consumption when comparing with the QTI measurement.
4. Hardware and software differences such as camera sensor configuration, ISP, 3A algorithm (AF, AE, and 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.
5. Ensure that the fps is 30. Low light conditions 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
```

2.8 Display – Static image display

1. Check the clocks on the device, whether they are aligned with the clocks of the reference device.
2. Account for hardware and software factors that potentially impact video decode power consumption when comparing with the QTI measurement.
3. The following properties are 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
     ```

4. 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
```

5. To check the number of hardware layers – While capturing dumpsys, ensure that none of the layers are being updated. If the status bar gets updated because of USB charging, then incorrect information is observed. To avoid this, run the following command in the background after a device sleep of 10–15 sec:

```
adb shell dumpsys SurfaceFlinger
```

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

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

2.9 Gfx – PowerLift and Egypt

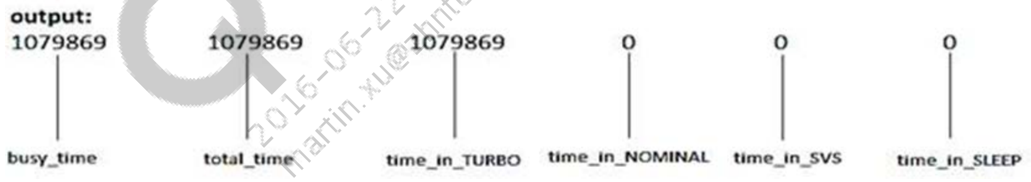
1. Check the clocks on the device, whether they are aligned with the clocks of the reference device.
2. Account for any hardware and software factors that potentially impact graphic power consumption when comparing with the QTI measurement.
3. Mount debugfs by using the following adb command:

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

4. Always measure 3D clocks from debugfs real measurements supplied directly by the clock driver.
5. kgsl thinks that the active clock rate must be gpuclk as defined in /sys/class/kgsl/kgsl-3d0.

6. 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 2-2
     - 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
```


output:

```
1079869      1079869      1079869      0      0      0
```

Figure 2-2 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
```

7. Check gpu power levels in the msm8916-gpu.dtsi file.

8. 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/disable

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

9. To retrieve 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
   ```

10. To check the number of hardware layers, use the following adb command; while capturing dumpsys, make sure that none of the layers are being updated. If the status bar gets 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
    ```

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

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

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

1. Check the clocks on the device, whether they are aligned with the clocks of the reference device.
2. 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 compare with the QTI-released measurement.
3. Account for any hardware and software factors that potentially impact the browser power consumption when comparing measurements with the QTI-released measurement.

4. For browser and video streaming use cases, power analysis must 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 2-3 and Figure 2-4.

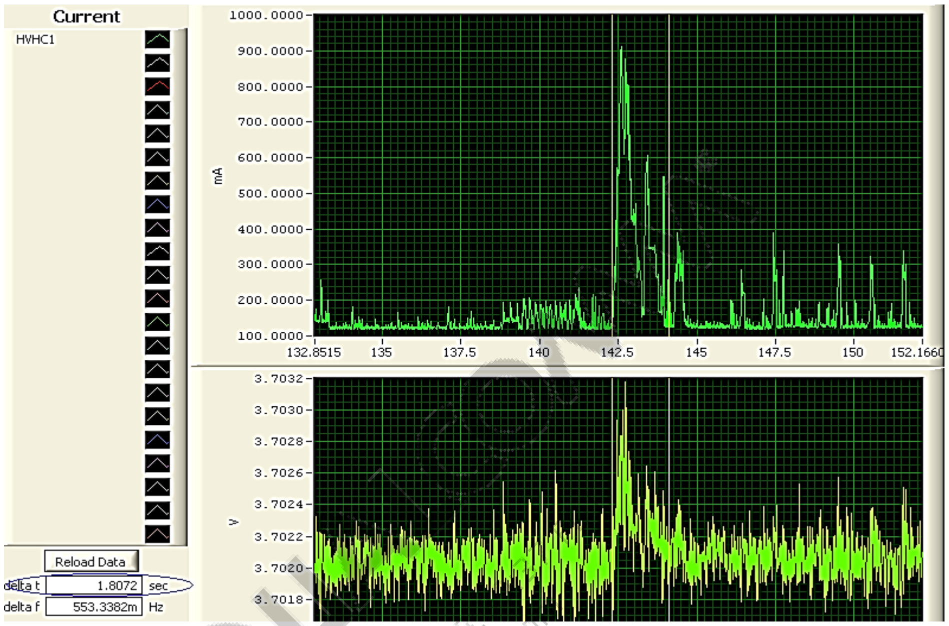


Figure 2-3 Browser waveform (active period ~1.8 sec)

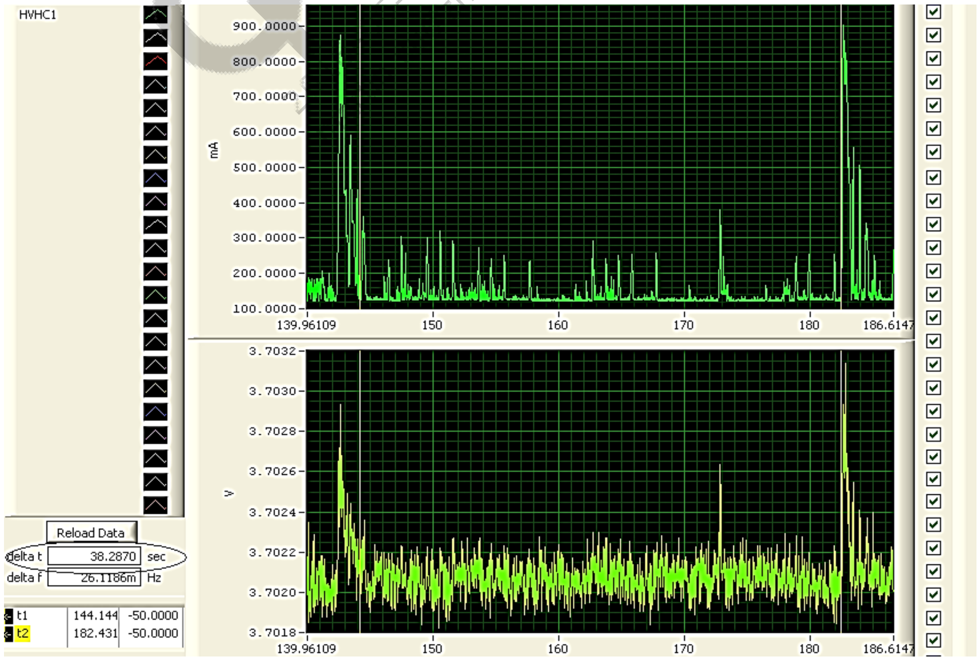


Figure 2-4 Browser waveform (idle period ~38.28 sec)

3 Dumps/Logs and Tools for Debugging

This chapter details the tools, dumps/logs used for debugging the power gap. The commonly used tools for power debugging are:

- PowerTop
- Top
- PerfTop
- Ftrace/Pytime chart
- Systrace
- Clock dump
- NPA dump
- MSM bus request debugging
- Governor parameter check
- Waveform comparison
- Wakelocks comparison
- msmpmstat analysis
- Rail-level comparison
- Bandwidth request comparison
- PX3 current

3.1 PowerTop

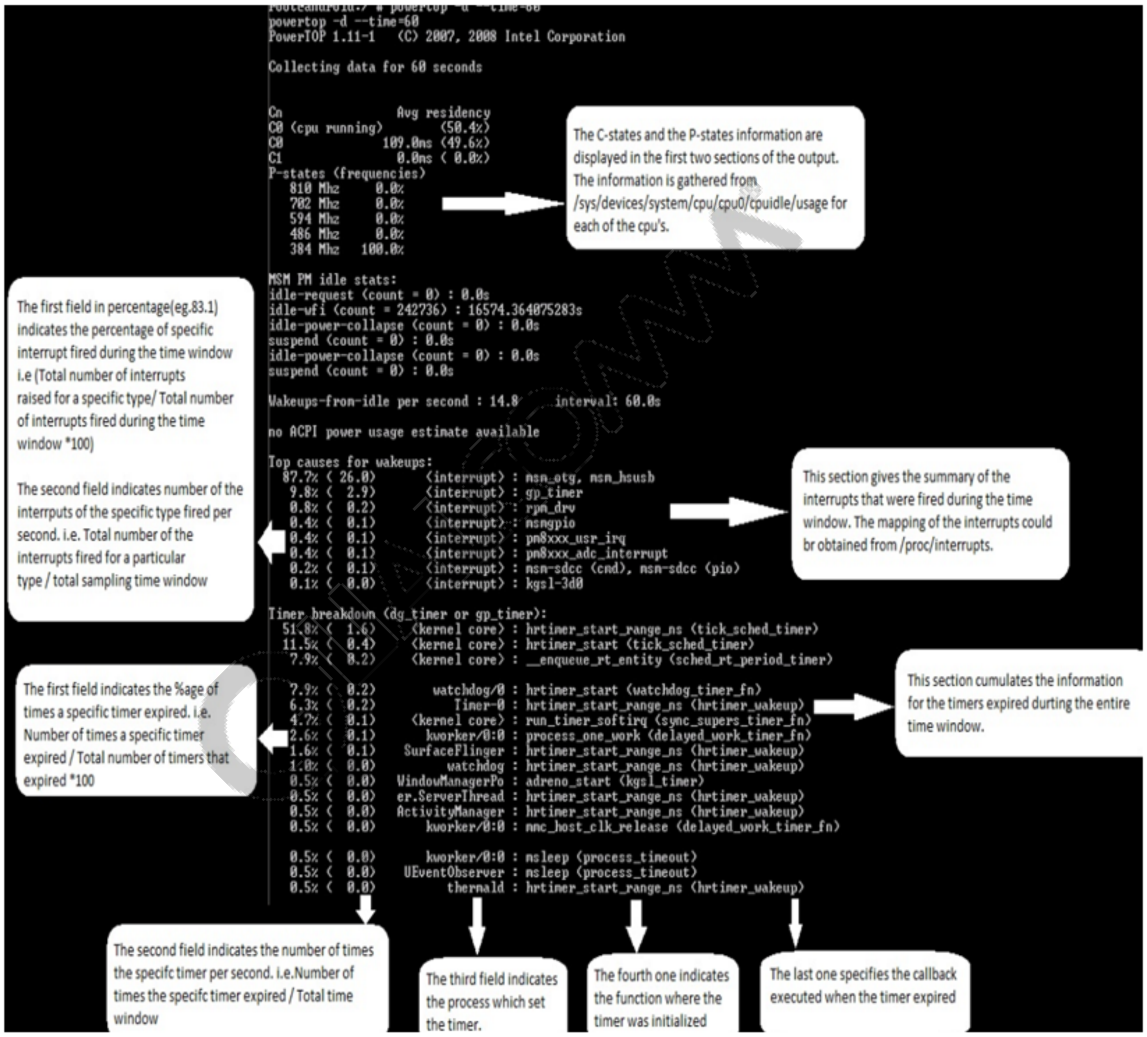
The PowerTop tool identifies specific components of kernel and user space applications that frequently wake the CPU. The power savings are higher as the CPU can be in Idle state most of the time. PowerTop provides CPU profiling information of the system as shown in Figure 3-1. It is advisable to capture the PowerTop data and battery power measurement simultaneously to ensure that 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:

```
adb shell powertop -d –t 60 or adb shell powertop > ptop_UseCaseName.txt &
```

Where -t indicates time in seconds. If the -d option is supplied alone, the data is captured for 15 sec. All msm_pm_stats 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 resident under each C0...C3 low power states.



```
root@android:/ # powertop -d --time=60
powertop -d --time=60
PowerTOP 1.11-1   (C) 2007, 2008 Intel Corporation

Collecting data for 60 seconds


Cn                Avg residency
C0 (cpu running)        (50.4%)
C0              109.0ms (49.6%)
C1                0.0ms ( 0.0%)
P-states (frequencies)
   810 Mhz     0.0%
   702 Mhz     0.0%
   594 Mhz     0.0%
   486 Mhz     0.0%
   384 Mhz   100.0%

MSM PM idle stats:
idle-request (count = 0) : 0.0s
idle-wfi (count = 242736) : 16574.364075283s
idle-power-collapse (count = 0) : 0.0s
suspend (count = 0) : 0.0s
idle-power-collapse (count = 0) : 0.0s
suspend (count = 0) : 0.0s

Wakeups-from-idle per second : 14.8     interval: 60.0s

no ACPI power usage estimate available

Top causes for wakeups:
  87.7% ( 26.0)       <interrupt> : msm_otg, msm_hsusb
   9.8% (  2.9)       <interrupt> : gp_timer
   0.8% (  0.2)       <interrupt> : rpm_drv
   0.4% (  0.1)       <interrupt> : msmgpio
   0.4% (  0.1)       <interrupt> : pm8xxx_usr_irq
   0.4% (  0.1)       <interrupt> : pm8xxx_adc_interrupt
   0.2% (  0.1)       <interrupt> : msm-sdcc (cmd), msm-sdcc (pio)
   0.1% (  0.0)       <interrupt> : kgsl-3d0

Timer breakdown (dg_timer or gp_timer):
  51.8% (  1.6)     <kernel core> : hrtimer_start_range_ns (tick_sched_timer)
  11.5% (  0.4)     <kernel core> : hrtimer_start (tick_sched_timer)
   7.9% (  0.2)     <kernel core> : __enqueue_rt_entity (sched_rt_period_timer)

   7.9% (  0.2)        watchdog/0 : hrtimer_start (watchdog_timer_fn)
   6.3% (  0.2)           Timer-0 : hrtimer_start_range_ns (hrtimer_wakeup)
   4.7% (  0.1)     <kernel core> : run_timer_softirq (sync_supers_timer_fn)
   2.6% (  0.1)       kworker/0:0 : process_one_work (delayed_work_timer_fn)
   1.6% (  0.1)    SurfaceFlinger : hrtimer_start_range_ns (hrtimer_wakeup)
   1.0% (  0.0)          watchdog : hrtimer_start_range_ns (hrtimer_wakeup)
   0.5% (  0.0)   WindowManagerPo : adreno_start (kgsl_timer)
   0.5% (  0.0)   er.ServerThread : hrtimer_start_range_ns (hrtimer_wakeup)
   0.5% (  0.0)   ActivityManager : hrtimer_start_range_ns (hrtimer_wakeup)
   0.5% (  0.0)       kworker/0:0 : mmc_host_clk_release (delayed_work_timer_fn)

   0.5% (  0.0)       kworker/0:0 : msleep (process_timeout)
   0.5% (  0.0)    UEventObserver : msleep (process_timeout)
   0.5% (  0.0)          thermald : hrtimer_start_range_ns (hrtimer_wakeup)
```

The C-states and the P-states information are displayed in the first two sections of the output. The information is gathered from /sys/devices/system/cpu/cpu0/cpuidle/usage for each of the cpu's.

The first field in percentage(eg.83.1) indicates the percentage of specific interrupt fired during the time window i.e (Total number of interrupts raised for a specific type/ Total number of interrupts fired during the time window *100)

The second field indicates number of the interrputs of the specific type fired per second. i.e. Total number of the interrupts fired for a particular type / total sampling time window

This section gives the summary of the interrupts that were fired during the time window. The mapping of the interrupts could br obtained from /proc/interrupts.

The first field indicates the %age of times a specific timer expired. i.e. Number of times a specific timer expired / Total number of timers that expired *100

This section cumulates the information for the timers expired durting the entire time window.

The second field indicates the number of times the specifc timer per second. i.e.Number of times the specifc timer expired / Total time window

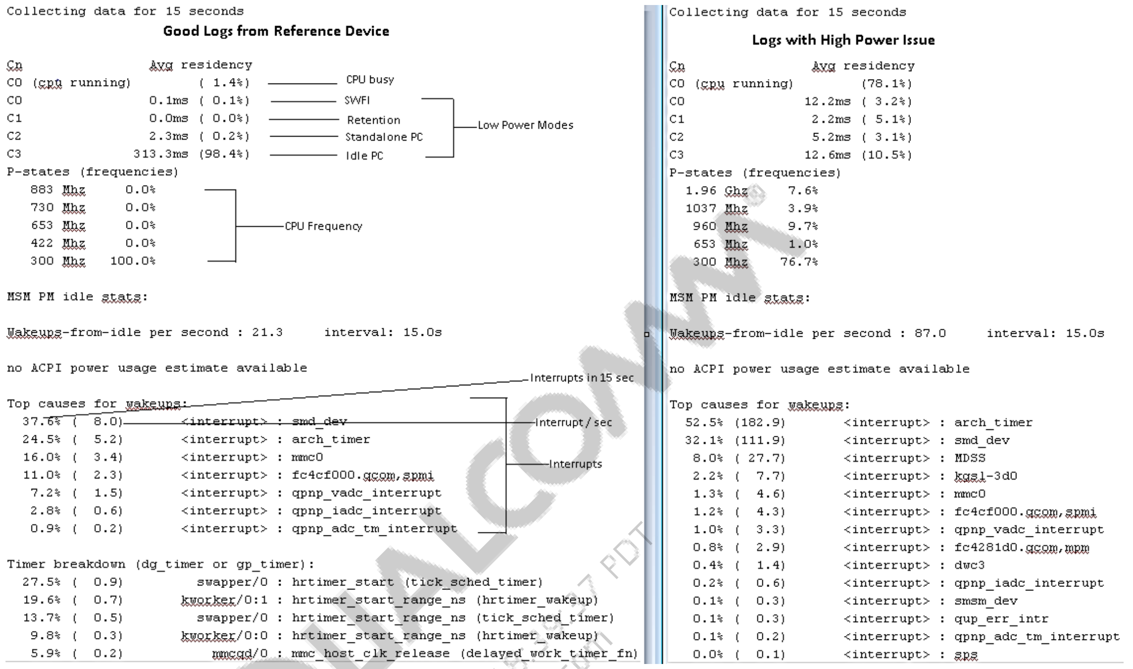
The third field indicates the process which set the timer.

The fourth one indicates the function where the timer was initialized

The last one specifies the callback executed when the timer expired

Figure 3-1 PowerTop log explanation

Figure 3-2 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.



```
Collecting data for 15 seconds

Cn                    Avg residency
C0 (cpu running)          ( 1.4%)
C0                  0.1ms ( 0.1%)
C1                  0.0ms ( 0.0%)
C2                  2.3ms ( 0.2%)
C3                313.3ms (98.4%)
P-states (frequencies)
   883 Mhz     0.0%
   730 Mhz     0.0%
   653 Mhz     0.0%
   422 Mhz     0.0%
   300 Mhz   100.0%

MSM PM idle stats:

Wakeups-from-idle per second : 21.3     interval: 15.0s

no ACPI power usage estimate available

Top causes for wakeups:
  37.6% (  8.0)       <interrupt> : smd_dev
  24.5% (  5.2)       <interrupt> : arch_timer
  16.0% (  3.4)       <interrupt> : mmc0
  11.0% (  2.3)       <interrupt> : fc4cf000.qcom,spmi
   7.2% (  1.5)       <interrupt> : qpnp_vadc_interrupt
   2.8% (  0.6)       <interrupt> : qpnp_iadc_interrupt
   0.9% (  0.2)       <interrupt> : qpnp_adc_tm_interrupt

Timer breakdown (dg_timer or gp_timer):
  27.5% (  0.9)         swapper/0 : hrtimer_start (tick_sched_timer)
  19.6% (  0.7)       kworker/0:1 : hrtimer_start_range_ns (hrtimer_wakeup)
  13.7% (  0.5)         swapper/0 : hrtimer_start_range_ns (tick_sched_timer)
   9.8% (  0.3)       kworker/0:0 : hrtimer_start_range_ns (hrtimer_wakeup)
   5.9% (  0.2)           mmcqd/0 : mmc_host_clk_release (delayed_work_timer_fn)
```

```
Collecting data for 15 seconds

Cn                    Avg residency
C0 (cpu running)          (78.1%)
C0                 12.2ms ( 3.2%)
C1                  2.2ms ( 5.1%)
C2                  5.2ms ( 3.1%)
C3                 12.6ms (10.5%)
P-states (frequencies)
  1.96 Ghz     7.6%
  1037 Mhz     3.9%
   960 Mhz     9.7%
   653 Mhz     1.0%
   300 Mhz    76.7%

MSM PM idle stats:

Wakeups-from-idle per second : 87.0     interval: 15.0s

no ACPI power usage estimate available

Top causes for wakeups:
  52.5% (182.9)       <interrupt> : arch_timer
  32.1% (111.9)       <interrupt> : smd_dev
   8.0% ( 27.7)       <interrupt> : MDSS
   2.2% (  7.7)       <interrupt> : kgsl-3d0
   1.3% (  4.6)       <interrupt> : mmc0
   1.2% (  4.3)       <interrupt> : fc4cf000.qcom,spmi
   1.0% (  3.3)       <interrupt> : qpnp_vadc_interrupt
   0.8% (  2.9)       <interrupt> : fc4281d0.qcom,mpm
   0.4% (  1.4)       <interrupt> : dwc3
   0.2% (  0.6)       <interrupt> : qpnp_iadc_interrupt
   0.1% (  0.3)       <interrupt> : smsm_dev
   0.1% (  0.3)       <interrupt> : qup_err_intr
   0.1% (  0.2)       <interrupt> : qpnp_adc_tm_interrupt
   0.0% (  0.1)       <interrupt> : sps
```

Figure 3-2 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 wakeups from idle per second; these wakeups prevent the device from entering power collapse.
5. Check the top causes of wakeups, i.e., wakeup interrupts for the differences. The top cause of the wakeup is vital information, since it lists the interrupts that prevent 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 wakeup 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.

3.2 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 waits.

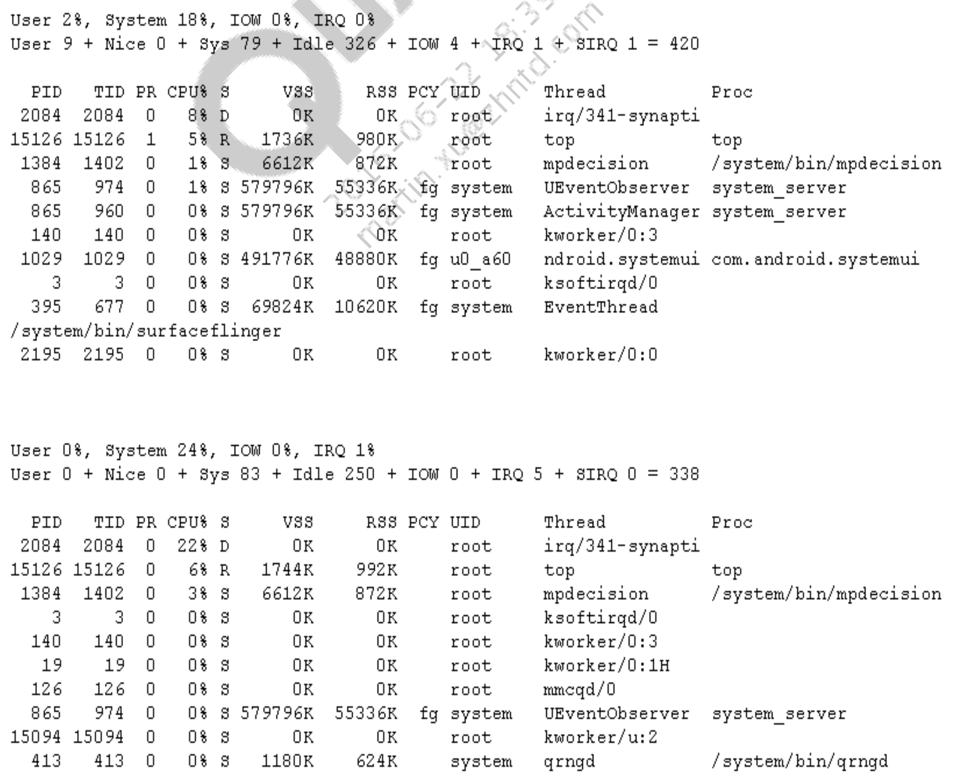
Collect the top logs in performance. The thread's CPU usage comparison is easier if the applications processor CPU is run at one frequency.

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.

Below 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
```

3.3 PerfTop

The PerfTop tool can operate in a mode similar to the Linux top tool, printing sampled functions in real time. The default sampling event is cycles and default order is in descending number of samples per symbol, thus PerfTop shows the functions where most of the time is spent. By default, PerfTop operates in processor-wide mode, monitoring all online CPUs at both user and kernel levels. It is possible to monitor only a subset of the CPUs using the `-C` option.

```
perf top
```

```
PerfTop:     260 irqs/sec  kernel:61.5%  exact:  0.0% [1000Hz
cycles],  (all, 2 CPUs)
```

```
samples  pcnt function                    DSO
            _______ _____ ______________________________
___________________________________________________________

              80.00 23.7% read_hpet                   [kernel.kallsyms]
              14.00  4.2% system_call                 [kernel.kallsyms]
              14.00  4.2% __ticket_spin_lock          [kernel.kallsyms]
              14.00  4.2% __ticket_spin_unlock        [kernel.kallsyms]
               8.00  2.4% hpet_legacy_next_event      [kernel.kallsyms]
               7.00  2.1% i8042_interrupt             [kernel.kallsyms]
               7.00  2.1% strcmp                      [kernel.kallsyms]
               6.00  1.8% _raw_spin_unlock_irqrestore [kernel.kallsyms]
               6.00  1.8% pthread_mutex_lock          /lib/i386-linux-
gnu/libpthread-2.13.so
               6.00  1.8% fget_light                  [kernel.kallsyms]
6.00  1.8% __pthread_mutex_unlock_usercnt /lib/i386-linux-gnu/libpthread-
2.13.so
               5.00  1.5% native_sched_clock          [kernel.kallsyms]
               5.00  1.5% drm_addbufs_sg
/lib/modules/2.6.38-8-generic/kernel/drivers/gpu/drm/drm.ko
```

By default, the first column shows the aggregate number of samples since the beginning of the run. By pressing the 'Z' key, this can be changed to print the number of samples since the last refresh. Recall that the cycle event counts CPU cycles when the processor is not in halted state, i.e. not idle. Therefore this is **not** equivalent to wall clock time. Furthermore, the event is also subject to frequency scaling.

It is also possible to drill down into single functions to see which instructions have the most samples. To drill down into a specify function, press the"s" key and enter the name of the function. Select the top function noploop (not shown above):

This tool is to profile the function, if single function takes larger number of samples or any new function, which is not present in the reference PerfTop logs, then find out why the number samples are larger than the reference or why the function has been added.

3.4 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 3-3 and Figure 3-4.

Figure 3-3 shows an ftrace analysis for a current increase caused by kgsl interrupts and the mdss_fb_commit_wq_handler thread.

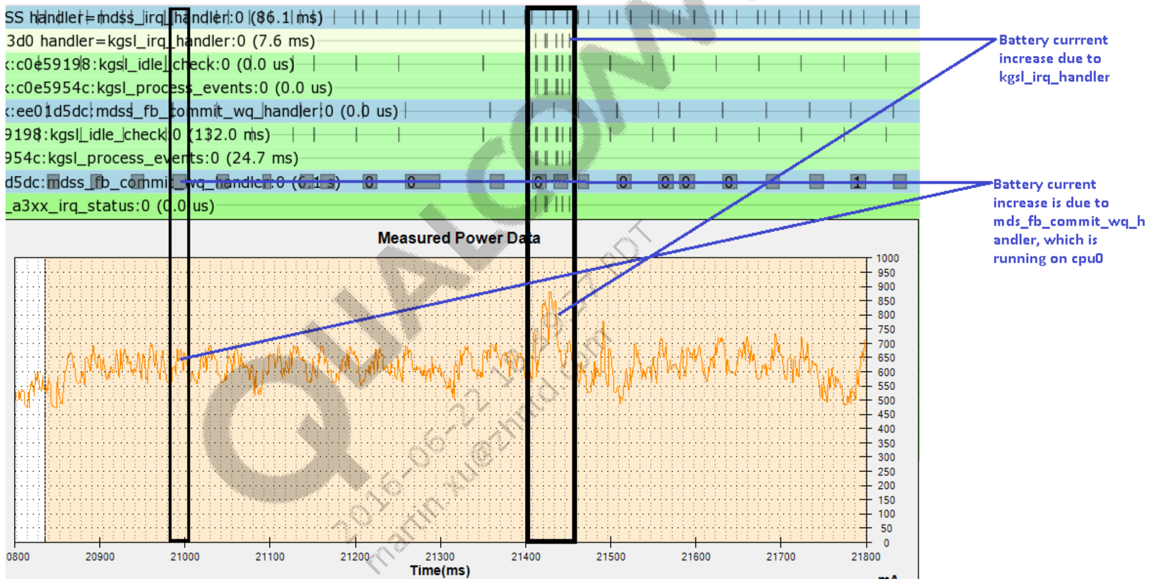
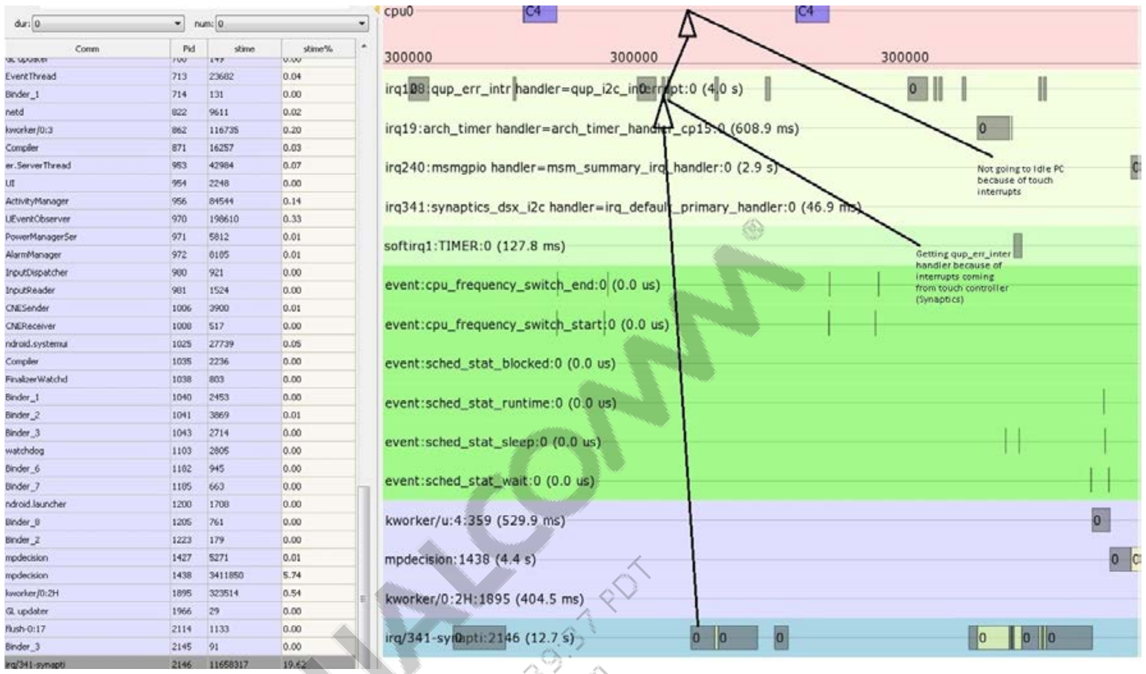


Figure 3-3 Ftrace log analysis – Static display use case with high current caused by kgsl interrupt

Figure 3-4 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.


dur: 0
num: 0
cpu0
C4
C4
300000
300000
300000
irq108:qup_err_intr handler=qup_i2c_interrupt:0 (4.0 s)
irq19:arch_timer handler=arch_timer_handler_cp15:0 (608.9 ms)
irq240:msmgpio handler=msm_summary_irq_handler:0 (2.9 s)
irq341:synaptics_dsx_i2c handler=irq_default_primary_handler:0 (46.9 ms)
softirq1:TIMER:0 (127.8 ms)
event:cpu_frequency_switch_end:0 (0.0 us)
event:cpu_frequency_switch_start:0 (0.0 us)
event:sched_stat_blocked:0 (0.0 us)
event:sched_stat_runtime:0 (0.0 us)
event:sched_stat_sleep:0 (0.0 us)
event:sched_stat_wait:0 (0.0 us)
kworker/u:4:359 (529.9 ms)
mpdecision:1438 (4.4 s)
kworker/0:2H:1895 (404.5 ms)
irq/341-synapti:2146 (12.7 s)
Not going to Idle PC because of touch interrupts
Getting qup_err_inter handler because of interrupts coming from touch controller (Synaptics)

Comm	Pid	stime	stime%
GL updater	700	149	0.00
EventThread	713	23602	0.04
Binder_1	714	131	0.00
netd	822	9611	0.02
kworker/0:3	862	116735	0.20
Compiler	871	16257	0.03
er.ServerThread	953	42984	0.07
UI	954	2248	0.00
ActivityManager	956	84544	0.14
UEventObserver	970	198610	0.33
PowerManagerSer	971	5812	0.01
AlarmManager	972	8105	0.01
InputDispatcher	980	921	0.00
InputReader	981	1524	0.00
CNESender	1006	3900	0.01
CNEReceiver	1008	517	0.00
ndroid.systemui	1025	27739	0.05
Compiler	1035	2236	0.00
FinalizerWatchd	1038	803	0.00
Binder_1	1040	2453	0.00
Binder_2	1041	3869	0.01
Binder_3	1043	2714	0.00
watchdog	1103	2805	0.00
Binder_6	1102	945	0.00
Binder_7	1105	663	0.00
ndroid.launcher	1200	1708	0.00
Binder_8	1205	761	0.00
Binder_2	1223	179	0.00
mpdecision	1427	5271	0.01
mpdecision	1438	3411850	5.74
kworker/0:2H	1895	323514	0.54
GL updater	1966	29	0.00
flush-0:17	2114	1133	0.00
Binder_3	2145	91	0.00
irq/341-synapti	2146	11658317	19.62

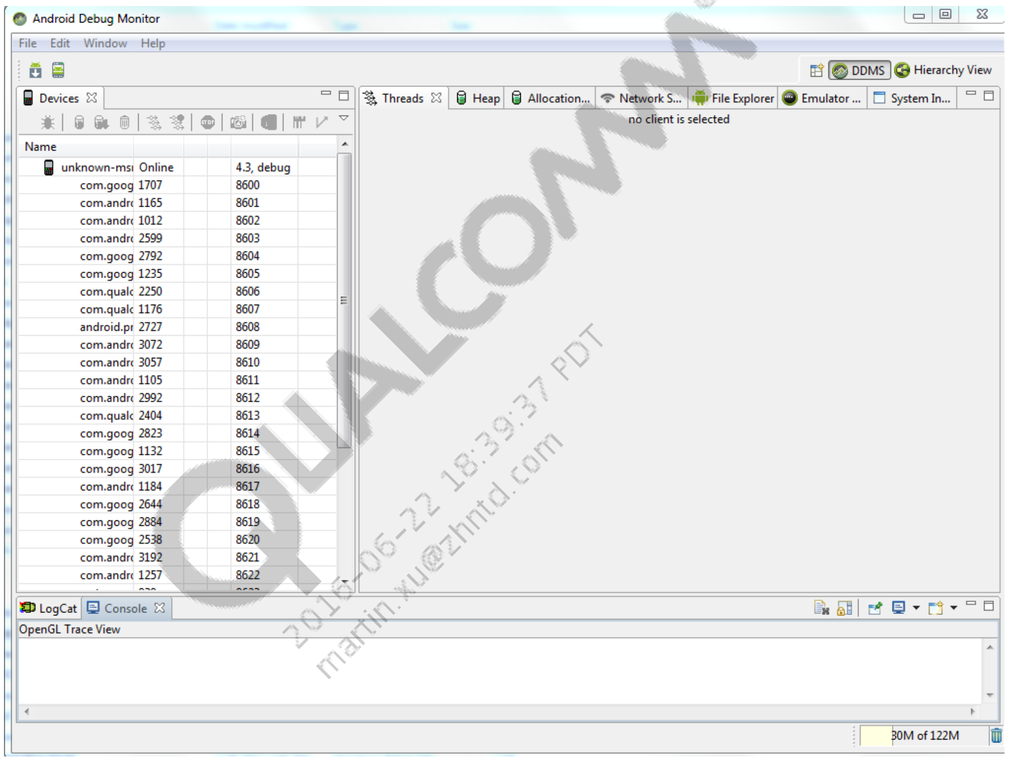
Figure 3-4 Ftrace logs analysis – Static display use case with high current caused by synaptic touch interrupt

3.5 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:

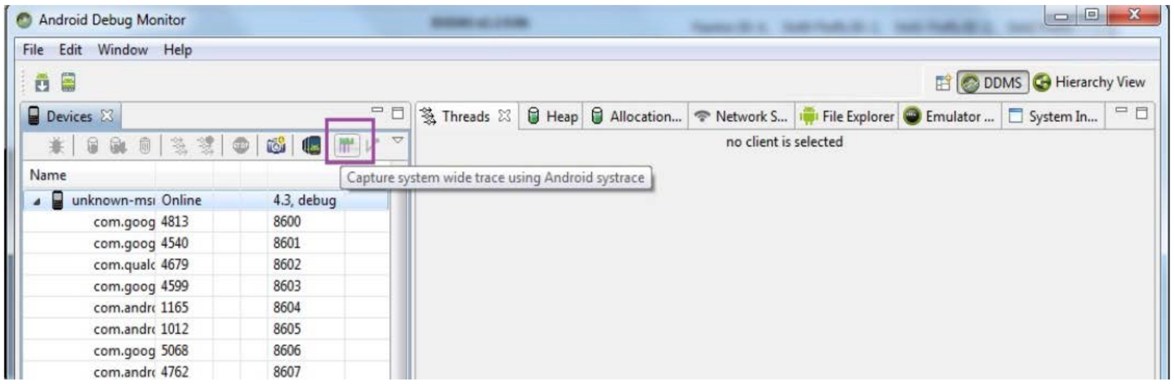
1. Power on the device and ensure that it is rooted; if the device is not rooted, enter the adb root command in the command prompt. The USB must be connected for systrace capturing.
2. Go to adt-bundle-windows-x86_64-20130729\sdk\tools.
3. Click **Monitor**; the following window appears.



no client is selected

```
Name
unknown-msı  Online      4.3, debug
com.goog     1707        8600
com.andrc    1165        8601
com.andrc    1012        8602
com.andrc    2599        8603
com.goog     2792        8604
com.goog     1235        8605
com.qualc    2250        8606
com.qualc    1176        8607
android.pr   2727        8608
com.andrc    3072        8609
com.andrc    3057        8610
com.andrc    1105        8611
com.andrc    2992        8612
com.qualc    2404        8613
com.goog     2823        8614
com.goog     1132        8615
com.goog     3017        8616
com.andrc    1184        8617
com.goog     2644        8618
com.goog     2884        8619
com.goog     2538        8620
com.andrc    3192        8621
com.andrc    1257        8622
```

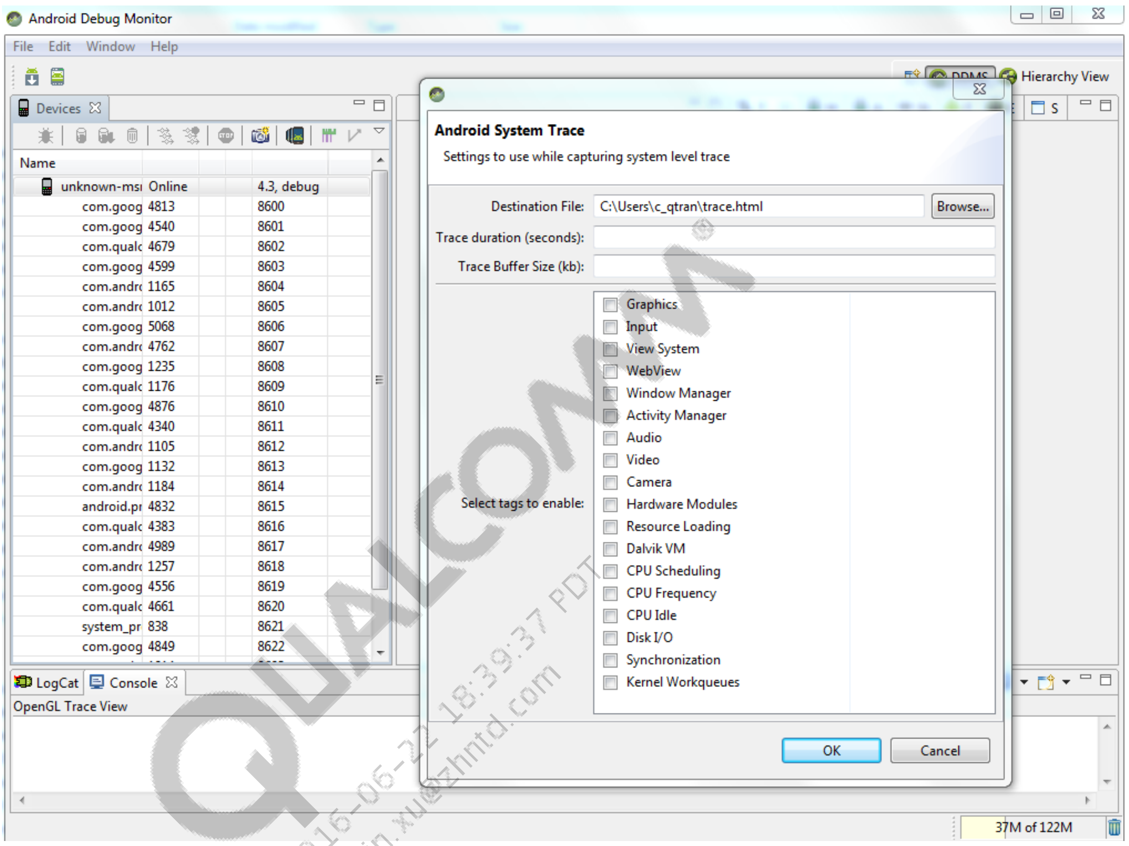
4. Click the highlighted button shown below.



no client is selected

```
Name
unknown-msı  Online      4.3, debug
com.goog     4813        8600
com.goog     4540        8601
com.qualc    4679        8602
com.goog     4599        8603
com.andrc    1165        8604
com.andrc    1012        8605
com.goog     5068        8606
com.andrc    4762        8607
```

5. Select the area of interest, specify the destination location where the data will be saved, and/or specify the trace duration in seconds; the default trace duration is 5 sec.



Name		
unknown-msı	Online	4.3, debug
com.goog	4813	8600
com.goog	4540	8601
com.qualc	4679	8602
com.goog	4599	8603
com.andrc	1165	8604
com.andrc	1012	8605
com.goog	5068	8606
com.andrc	4762	8607
com.goog	1235	8608
com.qualc	1176	8609
com.goog	4876	8610
com.qualc	4340	8611
com.andrc	1105	8612
com.goog	1132	8613
com.andrc	1184	8614
android.pr	4832	8615
com.qualc	4383	8616
com.andrc	4989	8617
com.andrc	1257	8618
com.goog	4556	8619
com.qualc	4661	8620
system_pr	838	8621
com.goog	4849	8622

Android System Trace

Settings to use while capturing system level trace

Destination File: C:\Users\c_qtran\trace.html [Browse...]

Trace duration (seconds):

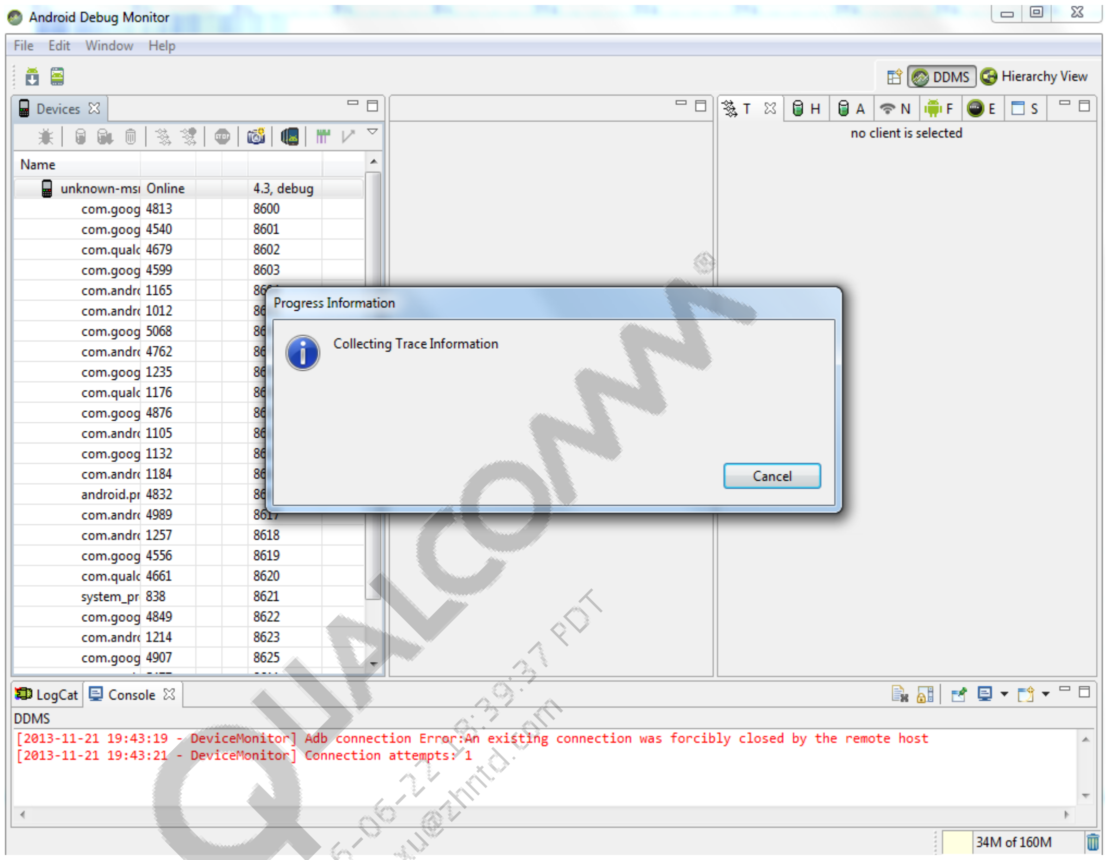
Trace Buffer Size (kb):

Select tags to enable:

- [ ] Graphics
- [ ] Input
- [ ] View System
- [ ] WebView
- [ ] Window Manager
- [ ] Activity Manager
- [ ] Audio
- [ ] Video
- [ ] Camera
- [ ] Hardware Modules
- [ ] Resource Loading
- [ ] Dalvik VM
- [ ] CPU Scheduling
- [ ] CPU Frequency
- [ ] CPU Idle
- [ ] Disk I/O
- [ ] Synchronization
- [ ] Kernel Workqueues

[OK] [Cancel]

6. Click **OK.** A window displaying progress information appears.



Name		
unknown-msı	Online	4.3, debug
com.goog	4813	8600
com.goog	4540	8601
com.qualc	4679	8602
com.goog	4599	8603
com.andrc	1165	86
com.andrc	1012	86
com.goog	5068	86
com.andrc	4762	86
com.goog	1235	86
com.qualc	1176	86
com.goog	4876	86
com.andrc	1105	86
com.goog	1132	86
com.andrc	1184	86
android.pr	4832	86
com.andrc	4989	8617
com.andrc	1257	8618
com.goog	4556	8619
com.qualc	4661	8620
system_pr	838	8621
com.goog	4849	8622
com.andrc	1214	8623
com.goog	4907	8625

no client is selected

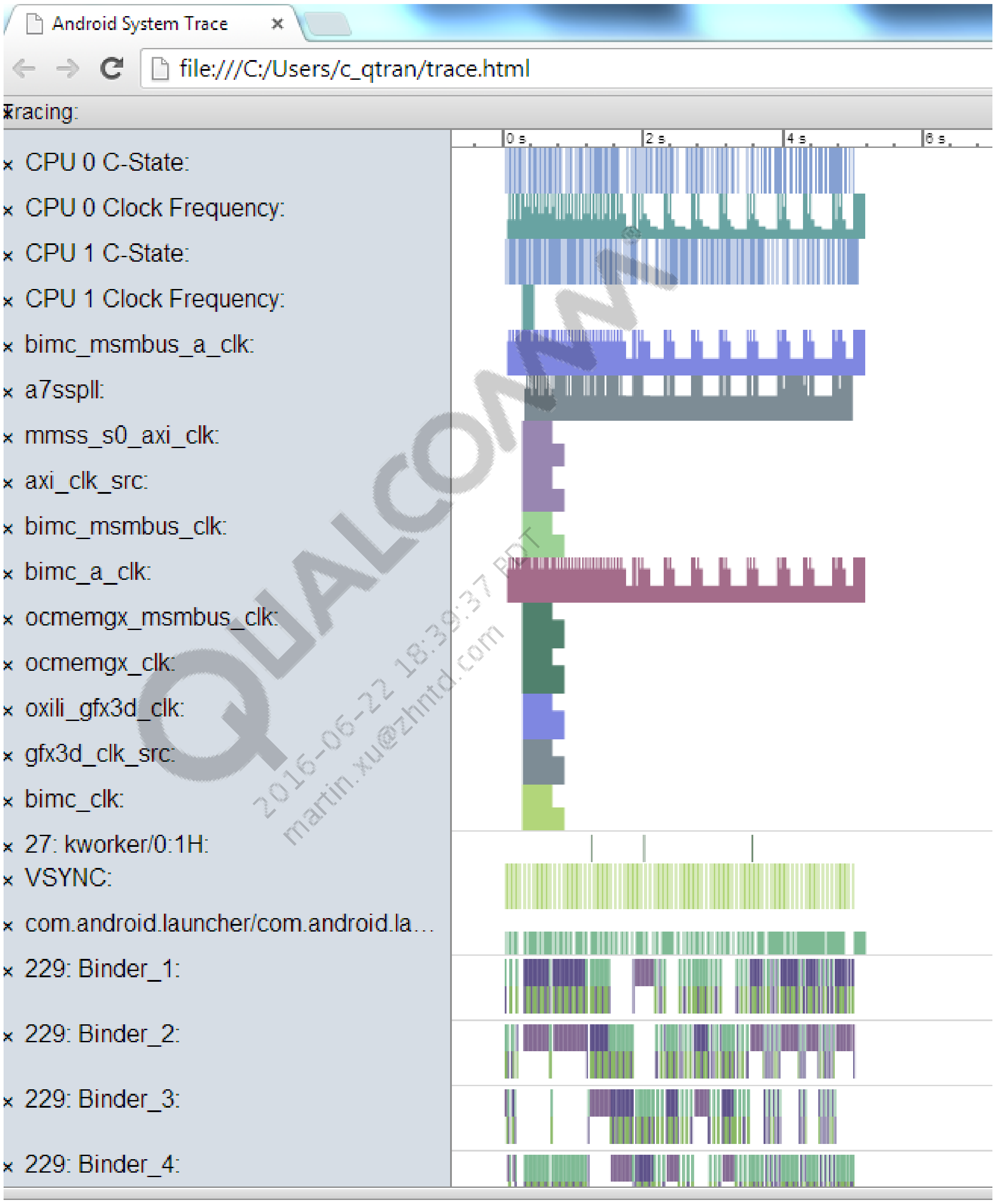
Progress Information

Collecting Trace Information

Cancel

DDMS
```
[2013-11-21 19:43:19 - DeviceMonitor] Adb connection Error:An existing connection was forcibly closed by the remote host
[2013-11-21 19:43:21 - DeviceMonitor] Connection attempts: 1
```

7. After the data is 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.

3.6 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., then expected.

It is recommended 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.

Table 3-1 shows the clock plan mapping.

Table 3-1 MSM8916 clock mapping

Clock name description	Clocks
System Network on Chip (NoC)	gcc_sys_noc_axi_clk
Peripheral and Configuration NoC	gcc_pcnoc_ahb_clk
Bus Integrated Memory Controller (BMIC)	gcc_bimc_clk
BMIC GPU	gcc_bimc_gpu_clk (RPM configures the clk and applications manage enable/disable this clk)
Mobile Display Processor (MDP)	gcc_mdss_mdp_clk
Video core	gcc_venus0_vcodec0_clk
Graphics Processing Unit (GPU)	gcc_oxili_gfx3d_clk
Camera Postprocessing (CPP)	gcc_camss_cpp_clk
Video Front-End (VFE)	gcc_camss_vfe0_clk
Camera control Interface	gcc_camss_cci_clk

Debugging examples

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

Collect the JTAG clock dumps and check for the following suspicious clocks, which can prevent XO shutdown.

```
Clocks:
gcc_bimc_sysnoc_axi_clk,
gcc_mdss_axi_clk,
gcc_sys_noc_axi_clk,
gcc_xo_clk
```

To debug further and determine which clients vote for CXO, NPA dumps must be collected and analyzed.

From the NPA dumps, if it is observed that PCNOC is holding the XO shut down and the request to PCNOC is from the apps, then find out by using the below mentioned process, which driver from the apps requests for PCNOC.

1. First collect the ftrace logs by enabling the clock set rate as mentioned below:

   ```
   mount -t debugfs none /sys/kernel/debug
   echo 1 > /d/tracing/events/power/clock_set_rate/enable
   cd /sys/kernel/debug/tracing
   echo 60000 > buffer_size_kb
   echo : > set_event
   echo 0 > tracing_on
   echo > trace
   echo 1 > tracing_on # Start trace
   disconnect the usb and run the use case.
   after 30 sec connect the usb
   adb shell
   cd /sys/kernel/debug/tracing
   echo 0 > tracing_on  # Stop trace
   cat trace > /data/log.txt # dump the trace file
   adb pull /data/log.txt
   ```

2. Search for the clock that is ON or having abnormal value or holding the XO shutdown in the file frtace logs (log.txt), and also search for “clock_set_rate”.
3. For the corresponding clock_set_rate, observe that there would be a PID. Try to find process ID.

Clock debugging can also be done through adb for clocks that are not APSS controlled, e.g., gcc_oxili_gfx3d_clk, gcc_mdss_mdp_clk, gcc_bimc_sysnoc_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/gcc_mdss_axi_clk/measure
>sleep .1
>done
```

3.7 NPA dump debugging

The NPA dumps provide information on clients that vote for the shared NPA resources. In the NPA RPM dumps, observe the following string:

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

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

The following NPA logs indicate that the APSS requests 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: 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, the MSM bus debug client data is analyzed.

3.8 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
root@msm8916_32:/d/msm-bus-dbg/client-data # ls
ls
78b6000.i2c
78b7000.spi
78b9000.i2c
78ba000.i2c
devfreq_cpubw
grp3d
mdss_mdp
mdss_pp
msm-rng-noc
msm_camera_isp – This camera client show up only when camera is active.
qcedev-noc
qseecom-noc
scm_pas
sdhc1
sdhc2
update-request
usb2
vdec-ddr
venc-ddr
```

```
cat mdss_mdp (During Display ON)
414.556255933
curr   : 1
masters: 22
slaves : 512
ab     : 448588800
ib     : 448588800

cat mdss_mdp (During Display Off)
705.577124312
curr   : 0
masters: 22
slaves : 512
ab     : 0
ib     : 0
```

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 does not request 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-ids.h header sets the master and slave values in ascending order. The header file is located under <root_director>\kernel\arch\arm\boot\dts\include\dt-bindings\msm.

The content of the msm-bus-ids.h file is:

```
Master: 22 =  MSM_BUS_MASTER_MDP_PORT0
Slave: 512 =  MSM_BUS_SLAVE_ EBI_CH0.
```

3.9 Governor 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 tunable 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.

3.10 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 device reboots, start the test case and run the following:

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

The following log shows perf-lock being acquired by the 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.

```
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
```

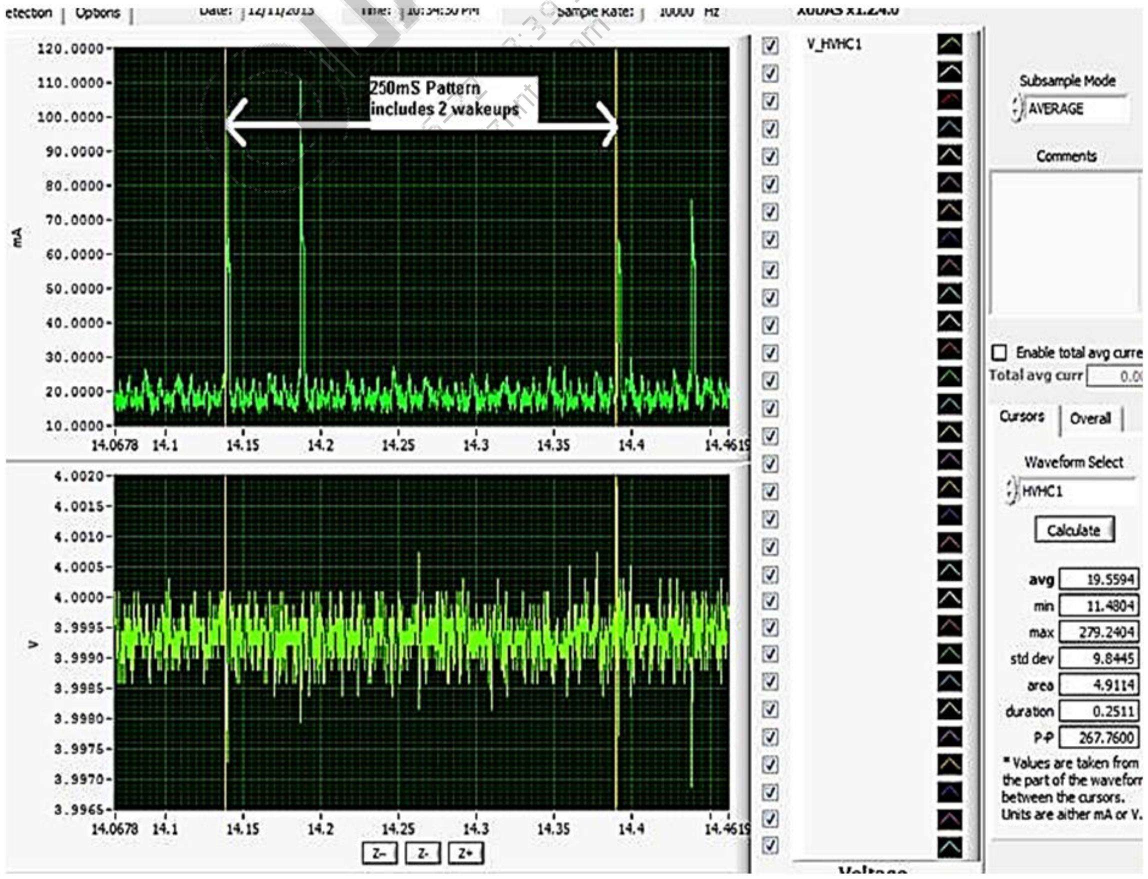
3.11 Waveform comparison

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 3-5 shows the waveform captured on a QTI reference device during mp3 playback with high current consumption.



Subsample Mode: AVERAGE

Waveform Select: HVHC1

avg	19.5594
min	11.4804
max	279.2404
std dev	9.8445
area	4.9114
duration	0.2511
P-P	267.7600

* Values are taken from the part of the wavefor between the cursors. Units are aither mA or V.

Figure 3-5 mp3 playback on reference device A

Figure 3-6 shows the waveform captured on device B during mp3 playback with power regression enabled.

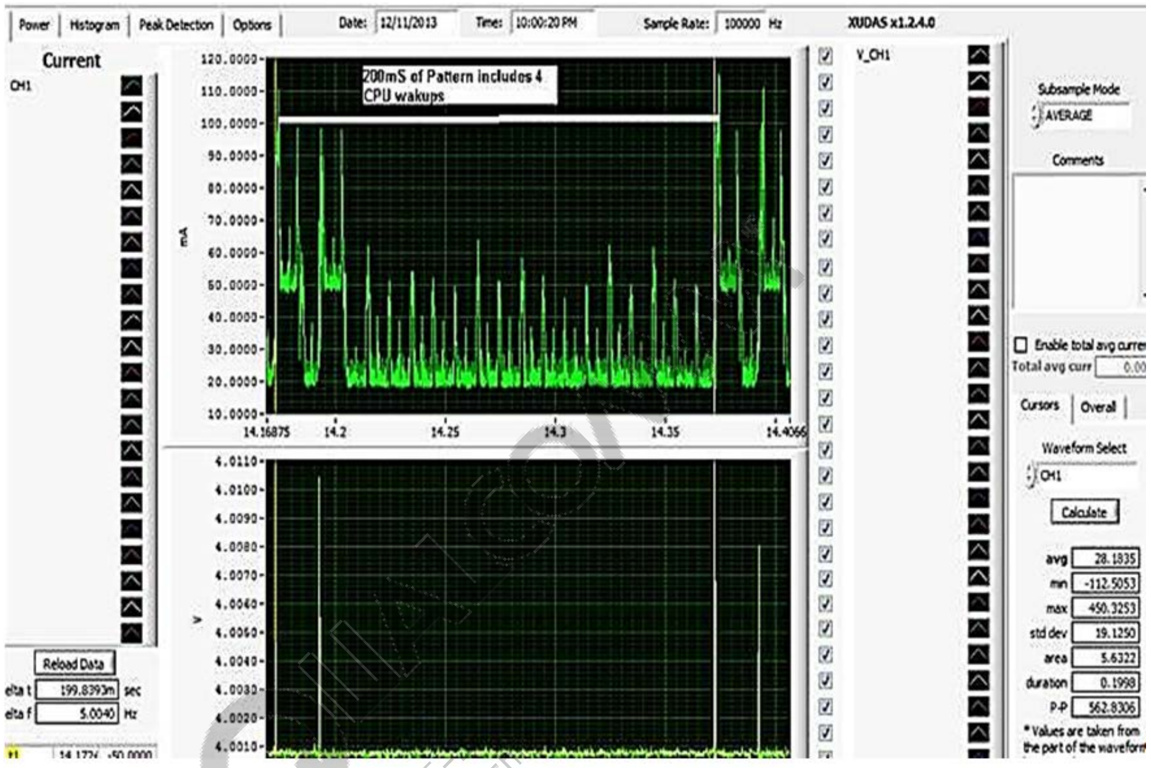


Figure 3-6 mp3 playback on device B with power regression

Table 3-2 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™ wakeups	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

3.12 Wakelocks comparison

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

3.12.1 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 3-3 and Table 3-4. 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 that 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 3-3 and Table 3-4 as msm_dwc3.

Table 3-3 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 3-4 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.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
PowerManagerService. WakeLocks	757	757	1	0	7505	693124	150322	17123682	659659
msm_dwc3	2	2	0	0	16399852	16402872	16399852	731340	16339360

3.12.2 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
```

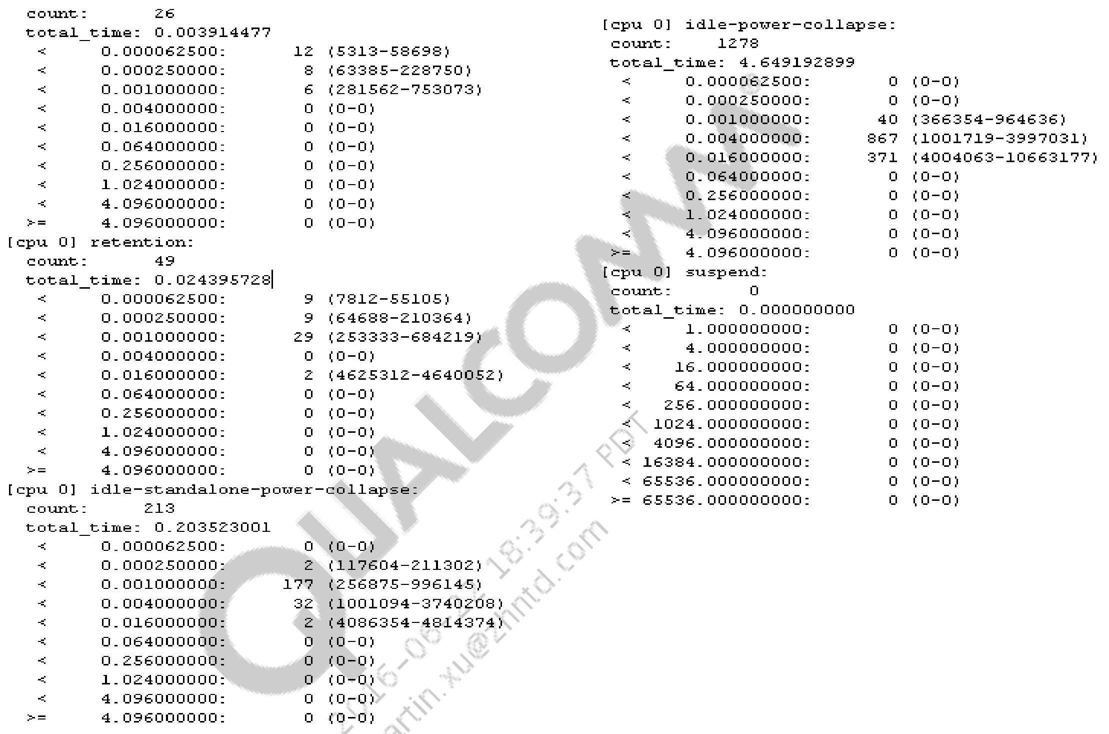
- dumpsys power log snippet during mp3 use case

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

3.13 msmpmstat analysis

The msmpmstat log shows details of the various power states of each CPU core.

As shown in below, Cpu0 is in retention for 49 times, and the total time spent in this state is 0.02439 sec. Similarly, it can be interpreted that Cpu0 goes into idle-power collapse 1278 times and the total time the cpu0 is in this state is about 4.64 sec.



```
  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)
```

3.14 Rail-level comparison

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

Table 3-5 contains the major rails and external component rails.

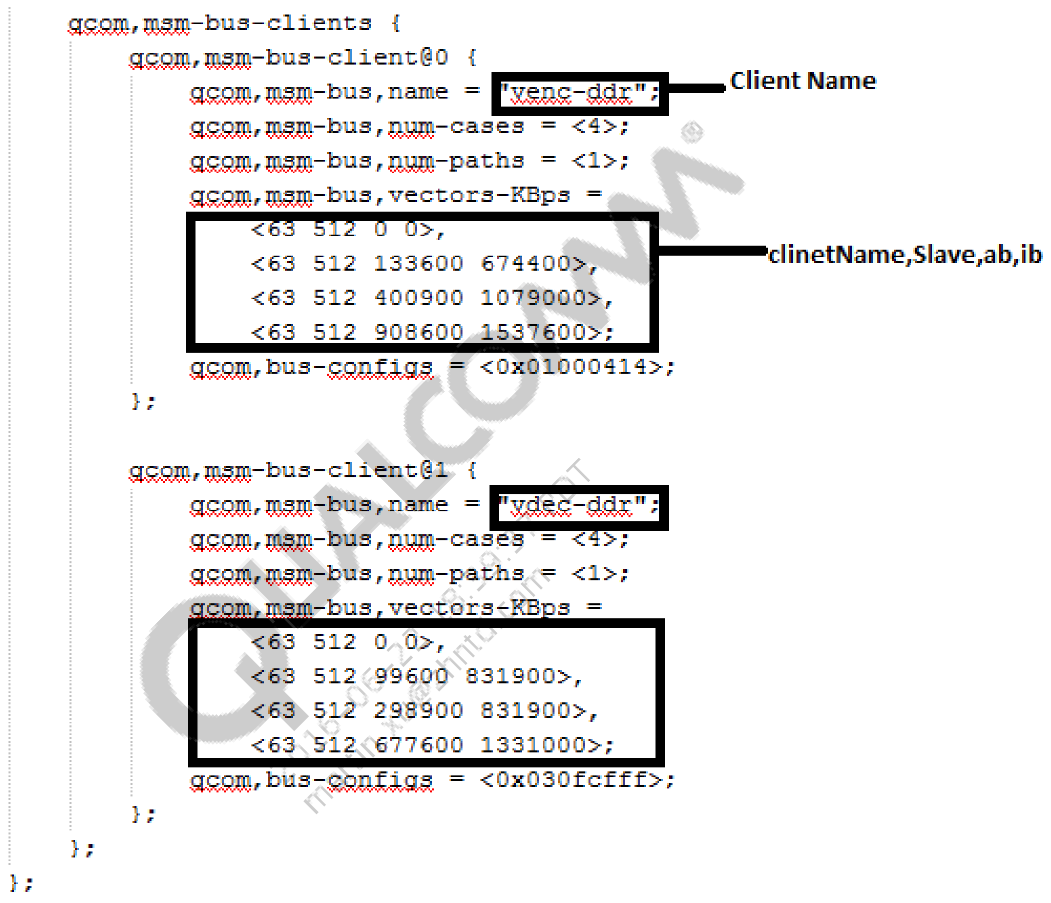
Table 3-5 Rail-level comparison details

Major rails	Comments
S1 – [CX, MSS, GFX, MDSP, CDC, SMPS, S1]	The CX rail has major subsystems included in this rail, they are modem, graphic, mdp, and audio codec.
S2 – [APC,SMPS,S2]	This rail is for APC subsystem; all the four cores are synchronous.
S3 – [MX, RF1, PX1, L1, L2_L3, LPDDR2, USB2, MIPI_CSI&DSI,SMPS,S3]	This rail is for MX, RF, caches, LPDDR memory, USB peripheral, MIPI control & data interface.
S4_OUTPUT [RF2,PX3,PX7,L4,L5,L6,L7,eMMC,USB2,SMPS,S4]	This rail is for MX, RF, caches, LPDDR memory, USB peripheral, MIPI control & data interface.
L6 LDO – CAP TS 1p8 CNTRL VDD	This is LDO for touchscreen VDD.
L17 LDO – CAP TS 2p85 CNTRL VCPIN	This is LDO for touchscreen VCPIN.
VPH_PWR LED 3p7	LED current can be obtained by summing these rails at battery.
VPH_PWR MISC 3p7	
VPH_PWR_DISP 3p7	
L17 LDO – LCD1 MIPI 2p85 VDD	
L6 LDO – LCD1 MIPI 1p8 VDDIO	
L10 LDO – MIPI 13 MP Cam VDD_AF 2p7	Camera current can be obtained by adding current of these rails at battery voltage.
L17 – MIPI 13 MP Cam VDDA 2p85	
L6 – MIPI 13 MP Cam VDDIO 1p8	
EXT L1 – MIPI 13 MP Cam VDD 1p05	

The rail-level power differences from the QTI reference device can be caused by extra hardware components, extra PLLs used, or applications 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.

3.15 Bandwidth request comparison

Check the msm8916.dtsi file to check if the AB/IB requests have been changed for any use case; e.g., the following shows the structure that should be checked for video encode and decode use cases.



```
qcom,msm-bus-clients {
        qcom,msm-bus-client@0 {
                qcom,msm-bus,name = "venc-ddr";
                qcom,msm-bus,num-cases = <4>;
                qcom,msm-bus,num-paths = <1>;
                qcom,msm-bus,vectors-KBps =
                        <63 512 0 0>,
                        <63 512 133600 674400>,
                        <63 512 400900 1079000>,
                        <63 512 908600 1537600>;
                qcom,bus-configs = <0x01000414>;
        };

        qcom,msm-bus-client@1 {
                qcom,msm-bus,name = "vdec-ddr";
                qcom,msm-bus,num-cases = <4>;
                qcom,msm-bus,num-paths = <1>;
                qcom,msm-bus,vectors-KBps =
                        <63 512 0 0>,
                        <63 512 99600 831900>,
                        <63 512 298900 831900>,
                        <63 512 677600 1331000>;
                qcom,bus-configs = <0x030fcfff>;
        };
    };
};
```

4 Logs/Dumps Collection Procedure

This chapter provides a step-by-step procedure for collecting logs and dumps required for power debugging.

4.1 JTAG license information

JTAG licenses requirements for debugging different subsystems

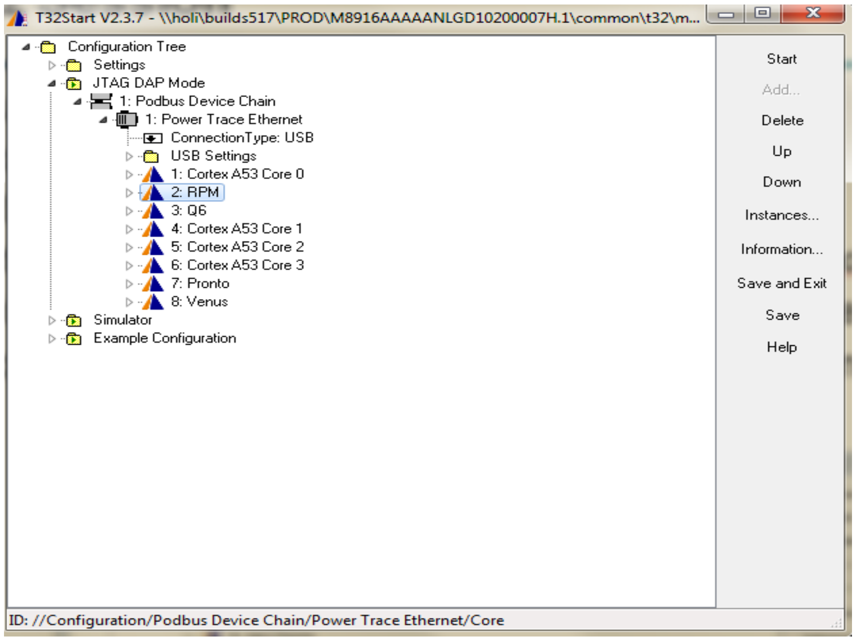
- LA-7844X or LA-7844 Cortex-M extension (for RPM) (Cortex-M3 license)
- LA-3743X Cortex A-R License Extension (For APPS A53)
- LA-3741A QDSP6 License Extension (for Modem and ADSP Hexagon)

4.2 Starting T32 session

To open T32 window:

1. T32 shortcuts can be found in the metabuild at:
2. <Metabuild>\common\t32\T32Start
3. Once the T32Start window is open, select

   Configuration Tree→JTAG DAP Mode→Podbus Device Chain →Power Trace Ethernet
4. T32 icons for RPM, modem, apps, etc., are located here.
5. Select the processor of interest and press **Start** to bring up the Trace32 window.



T32Start V2.3.7 - \\holi\builds517\PROD\M8916AAAAANLGD10200007H.1\common\t32\m...

```
Configuration Tree
  Settings
  JTAG DAP Mode
    1: Podbus Device Chain
      1: Power Trace Ethernet
        ConnectionType: USB
        USB Settings
        1: Cortex A53 Core 0
        2: RPM
        3: Q6
        4: Cortex A53 Core 1
        5: Cortex A53 Core 2
        6: Cortex A53 Core 3
        7: Pronto
        8: Venus
  Simulator
  Example Configuration
```

ID: //Configuration/Podbus Device Chain/Power Trace Ethernet/Core

4.3 Clock dumps

4.3.1 Clock dumps using JTAG

1. Open an RPM T32 window.
2. Type *sys.m.a* to attach T32 to the device under test.
3. Load the .elf file from <RPM Build>/rpm_proc/core/bsp/rpm/build/*.elf and set the breakpoint if that is desired.
4. Recreate the use case scenario.
5. Break the T32 by either of following ways:
   - Clicking **Pause** button in T32 user interface or
   - By pressing **F8**.
   - By pre-configured Breakpoint.
6. Type do <Build_Location>\rpm_proc\core\systemdrivers\clock\scripts\msm8916\testclock.cmm.
7. Type all in the pop-up window to get a dump of all clocks.
8. Clock dump is printed in the message area of T32.

4.3.2 Clock dumps using the adb shell

1. Plug the USB.
2. 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 &
```

3. Unplug the USB when PID is shown.
4. Run the test scenario <execute test scenario for at least 2 min>.

5. Plug the USB and kill the clock checking process ID.

```
adb shell ps | grep [PID]
adb shell “kill [PID]”
adb pull /data/dumpclk.txt
```

4.4 PMIC dumps

To collect PMIC dumps:

1. Open an RPM T32 window.
2. Type sys.m.a to attach T32 to the device under test.
3. Load the .elf file from <RPM Build>/rpm_proc/core/bsp/rpm/build/*.elf and set the breakpoint if that is desired.
4. Re-create the use-case scenario.
5. Break the T32 by either the following methods:
   - Click **Pause** button in T32 user interface, or
   - Pressing **F8**, or.
   - Preconfigured Breakpoint.
6. Once the breakpoint is hit, type:

```
CD.DO <RPM_Build>/rpm_proc\core\systemdrivers\pmic\scripts\PMICDump.cmm
```

7. By default, the raw PMIC dump file is saved in c:\temp\pmicdump.xml.
8. Use the command below to parse the pmicdump.xml file.

```
python PMICDumpParser.py—flat=<RPM
BUILD>\rpm_proc\core\systemdrivers\pmic\scripts\pm8916\v1_1\CORE_ADDRESS
_FILE_CUSTOMER.FLAT—file=pmicdump.xml > pmic_parsed.txt
```

4.5 PowerTop

To collect the PowerTop 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 /system/bin
adb shell “chmod 777 /system/bin/powertop”
```

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

4. Start the use case.
5. Run the following ADB commands:

```
adb shell
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 that the power number is as expected.
8. After a total of 60 sec, reconnect the USB cable and extract the PowerTop log using the following ADB command:

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

4.6 Top

To collect the top log:

1. Run the following ADB commands:

```
adb root
adb remount
```

2. Start the use case
3. Run the following adb commands:

```
adb root
adb shell
sleep 10 && top -t -m 10 > /data/top.log &
```

4. Disconnect the USB cable within 10 sec of running the above commands; after the 10 sec period, top data will start capturing.

5. Take a quick power measurement; the power measurement is a little higher than expected because the top process is running.
6. After a total of 60 sec, reconnect the USB cable and extract the top log using the following adb command:

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

4.7 Ftrace

To collect the ftrace log:

1. 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:* workqueue:* msm_low_power:* kgsl:*” >
/d/tracing/set_event && sleep 60 && cat  /d/tracing/trace >
/data/local/ftrace.txt &
```

2. Disconnect the USB and connect the USB after 2 min and pull out the ftrace logs.

4.8 msmpmstats

To collect the msmpmstats log:

1. Run the following ADB commands:

```
adb root
adb remount
```

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

4. Disconnect the USB cable within 10 sec of running the above commands.
5. After a total of 60 sec, reconnect the USB cable and extract the msmpmstats log by running the following ADB command; this captures data for 25 sec.

```
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/0 > /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 statistics counts and duration in each state collected in step 1 and step 2.

4.9 Wakelock (dumpsys power)

To collect the wakelock debugging log:

1. Run the following ADB commands:

```
adb root
adb remount
```

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

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

4. Disconnect the USB cable within 10 sec of running the above command.
5. 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.

4.10 SurfaceFlinger

To collect the SurfaceFlinger debugging log:

1. Run the following ADB commands:

```
adb root
adb remount
```

2. Start the use case, i.e., static display.
3. Run the following ADB commands:

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

4. Disconnect the USB cable within 10 sec of running the above commands.
5. 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>
```

4.11 MDP statistics

To collect the MDP statistics debugging log:

1. Run the following ADB commands:

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

2. Start the use case, i.e., static image display.

3. With the USB cable connected, run the following ADB commands two to three times:

```
adb shell
cat stat
```

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

4.12 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
cat /proc/interrupts
```

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

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

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

4.13 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
```

Select Option 14: Read All GPIO Configurations.

For default sleep configuration stored in software, use the following script to read, which is read out from TLMM.xml:

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

4.14 MSM bus driver debug – Client data

1. Run the following ADB commands:

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

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

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

   This gives an output of mdss_mdp client data.
4. Copy and paste the results into a text file.

4.15 Perf tool

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 <perf_binary_directory>\perf /data/
adb shell chmod 777 /data/perf
```

4. Command to collect the perf top logs

```
adb shell /data/perf top > perf_top.txt &
```

5 Additional Debug Details and Logging Procedures

The chapter provides additional details on debugging and log/dump-taking procedures for both the modem and RPM.

5.1 RPM debug

RPM publishes a small log into a limited area of spare message RAM. The physical format of the log is the Ulog format used for various other logs. It is a circular buffer, currently sized at about 4 kB. It is a raw log with a set of IDs and a variable number of parameters per message.

5.1.1 Saving RPM RAM dumps

In MSM8916, disabling the RPM halt is **not necessary** for keeping the JTAG daisy-chain connection.

Use the JTAG to modify the variable sleep_allow_low_power_modes to enable/disable all the RPM power modes.

- *v.v sleep_allow_low_power_modes = 0* disables all low power modes for RPM
- *v.v sleep_allow_low_power_modes = 1* enables all low power modes for RPM

Saving RPM RAM dumps

1. Open RPM T32 and attach by `sys.m.a` command.
2. Create the issue scenario where RPM dumps are needed.
3. Break T32 at the desired point and run the following script to save RPM logs:

```
do <RPM Build location>\rpm_proc\core\bsp\rpm\scripts\rpm_dump.cmm
<Location to save dumps>
```

5.1.2 Loading RPM dumps onto T32 and extracting logs

Load the RPM dumps onto the T32 simulator for further analysis. To do this:

1. Open the T32 simulator and do sys.up
2. Run the following script:

```
do <RPM Build location>\rpm_proc\core\bsp\rpm\scripts\rpm_load_dump.cmm
<Location of logs>
```

3. Load RPM.elf to start debugging.

   ```
   d.load.elf <RPM_Build>\rpm_proc\core\bsp\rpm\build\RPM_XXXXXXXXX.elf
   /nocode /noclear
   ```

- RPM external logs (RPM Ulog) using T32
  1. While attached to the RPM and in the Break state or if the RPM RAM dumps are loaded on the T32 Simulator, run the following command in T32:

     ```
     do <RPM_build>\rpm_proc\core\power\ulog\scripts\rpm_ulogdump.cmm
     <Location to save logs>
     ```

  2. This places the RPM's external logs into the desired log directory.
  3. The RPM external log requires the use of a Python parsing tool to interpret its contents. Use the following Python script on extracted logs using command prompt:

     ```
     Python \\<RPM_build_location>\rpm_proc\core\power\rpm\debug\scripts\
     rpm_log_bfam.py -f “RPM External Log.ulog”
     ```

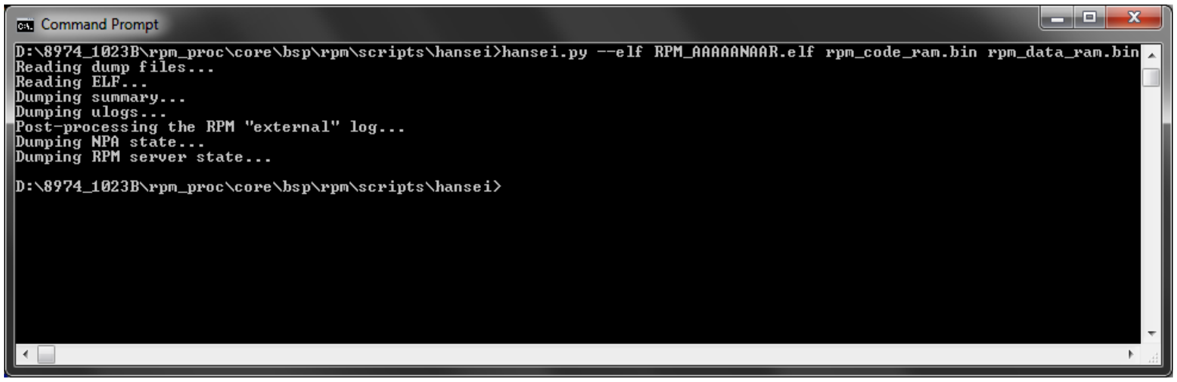
- RPM NPA logs using T32
  1. While attached to the RPM and in the Break state or if the RPM RAM dumps are loaded on the T32 Simulator, run the following command in T32:

     ```
     do <RPM_build>\rpm_proc\core\power\npa\scripts\rpm_npadump.cmm
     <Location to save logs>
     ```

- RPM logs using Hansei Script
  1. Hansei – An RPM RAM dump parser tool that can be found in the rpm_proc\core\bsp\rpm\scripts\hansei folder (Needs Python 2.7.2 Version)

     ```
     Usage – hansei.py [-h] --elf rpm.elf [--output path] dumpfile [dumpfile
     ...]
     ```

Example – hansei.py –elf rpm.elf -o . rpm_code_ram.bin rpm_data_ram.bin rpm_msg_ram.bin



```
D:\8974_1023B\rpm_proc\core\bsp\rpm\scripts\hansei>hansei.py --elf RPM_AAAAANAAR.elf rpm_code_ram.bin rpm_data_ram.bin
Reading dump files...
Reading ELF...
Dumping summary...
Dumping ulogs...
Post-processing the RPM "external" log...
Dumping NPA state...
Dumping RPM server state...

D:\8974_1023B\rpm_proc\core\bsp\rpm\scripts\hansei>
```

Summary of the output file:

- rpm-summary.txt – Contains general information about the health of the RPM, including the core dump state and various fault information
- rpm-log.txt – The postprocessed "RPM External Log"
- rpm-rawts.txt – The same log as rpm-log.txt, but with the raw timestamp
- npa-dump.txt – The standard NPA dump format
- ee-status.txt – Contains information about which subsystems (and their cores) are active or sleeping
- reqs_by_master/* – A folder containing a file for each execution environment, detailing all current requests that EE has in place with the RPM
- reqs_by_resource/* – A folder structure containing a folder for each of the resource types registered with the RPM server and, under that folder, a file containing all of the requests to each resource of that type

5.1.3 RPM log (alternate way – Using ADB)

The log can also be dumped during operation by executing the following in the ADB shell window while the USB is attached.

```
adb shell mount -t debugfs none /sys/kernel/debug/
adb shell cat /sys/kernel/debug/rpm_log > <file>
```

This log must be postprocessed similarly to the T32 logs.

Taking RPM logs using ADB can interfere with time response/performance of RPM toward system-wise requests.

5.2 Modem debug logs

- Modem Ulogs

  Execute the following script from Modem_Build in the Modem T32 window:

  ```
  do <Modem_Build>\modem_proc\core\power\ulog\scripts\UlogDump.cmm
  <Location to save logs>
  ```

  This will place the modem Ulogs in the specified directory.

- Modem NPA logs

  Execute the following script from Modem_Build in the modem T32 window:

  ```
  do <Modem_Build>\modem_proc\core\power\npa\scripts\NPADump.cmm <Location
  to save logs>
  ```

  This will place the modem NPA logs in the specified directory.

- Getting modem RAM dumps from T32
  a. Open RPM T32 and attach by sys.m.a.
  b. Open MPSS T32 and attach by sys.m.a.
  c. Create the issue scenario where modem dumps are needed.
  d. Break MPSS T32 and RPM T32 at the desired point, and in the RPM T32 window, run the following script:

```
do <Metabuild location>\common\tools\cmm\common\std_savelogs.cmm
<Location to save the dumps>
```

- Loading modem RAM dumps
  a. Open the Hexagon simulator and run the command sys.up.
  b. To load modem RAM dumps to the simulator, use the following command:

```
D.LOAD.B DDRCS0.BIN 0x80000000
```

  c. To start debugging, load the modem elf using the following command:

```
d.load.elf
<ModemBuildLocation>\modem_proc\build\ms\M8916EAAAANVAQ00311.elf
/nocode /noclear
```

6 Debugging APSS

This chapter discusses APPS Low Power mode debugging and some variables for enabling/disabling certain low power modes.

6.1 Low Power mode debugging

Following is the debug mask that can be enabled to get debug logs in the suspend path.

```
enum {
        MSM_PM_DEBUG_SUSPEND = BIT(0),
        MSM_PM_DEBUG_POWER_COLLAPSE = BIT(1),
        MSM_PM_DEBUG_SUSPEND_LIMITS = BIT(2),
        MSM_PM_DEBUG_CLOCK = BIT(3), -> clock info for SUSPEND PC
        MSM_PM_DEBUG_RESET_VECTOR = BIT(4),
        MSM_PM_DEBUG_IDLE_CLK = BIT(5),
        MSM_PM_DEBUG_IDLE = BIT(6), -> clock info for IDLE PC
        MSM_PM_DEBUG_IDLE_LIMITS = BIT(7),
        MSM_PM_DEBUG_HOTPLUG = BIT(8),
};
```

Example:

```
echo 0x0F > /sys/module/msm_pm/parameters/debug_mask
```

Following are the possible different LPM modes that can be selected:

```
enum msm_pm_sleep_mode {
MSM_PM_SLEEP_MODE_WAIT_FOR_INTERRUPT,
MSM_PM_SLEEP_MODE_RETENTION,
MSM_PM_SLEEP_MODE_POWER_COLLAPSE_STANDALONE,
MSM_PM_SLEEP_MODE_POWER_COLLAPSE,
MSM_PM_SLEEP_MODE_POWER_COLLAPSE_SUSPEND,
MSM_PM_SLEEP_MODE_NR,
MSM_PM_SLEEP_MODE_NOT_SELECTED,
};
```

Summary information displaying LPM mode is selected.

Following is the snippet of the kmesg where the selected mode is chosen. In the below example, MSM_PM_SLEEP_MODE_POWER_COLLAPSE mode is chosen during APSS suspend.

```
<4>[ 1304.154791]  [0:   kworker/u8:6: 3347] Disabling non-boot CPUs
<6>[ 1304.154942]  [0:   kworker/u8:6: 3347] CPU0:msm_cpu_pm_enter_sleep
mode:3 during suspend
```

6.1.1 LPM debugging – VDD_min/XO shutdown

It is necessary to be in VDD_Min and achieve the correct sleep current. If not, it is possible because the XO shutdown was blocked. To confirm whether the XO shutdown is blocked:

1. Check RPM counts.
   a. Connect a USB and check VDD_min counts using the following ADB commands:

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

   b. Disconnect the USB.
   c. Wait for 30 sec and connect to the USB.
   d. Get the RPM statistics by running the following:

   ```
   cat /d/rpm_stats
   ```

2. Check whether the VDD_min count has been increased. If it has not been increased, then it indicates that VDD_min is 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 is blocked, check the following after the device is suspended and any test cases have stopped or finished.
         (a) Check the NPA dump on RPM as explained in Section 2.1.8.2 if CXO is being voted for by APSS or MPSS. For multimedia test cases, either of these subsystems could have blocked XO shutdown/VDD_min. If it is MPSS, MPSS 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 4.14 to determine if any multimedia cores have not relinquished the bandwidth vote. If any core shows an active bandwidth request, this must be corrected.

(c) Check the clock state of all multimedia cores and the LPASS. If any core shows an active clock (turned on), this must be corrected.

In the case of a high static display current with a smart panel, VDD_min cannot be achieved if MIPI DSI-PHY is kept on, since MIPI 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:

a. Connect the USB.

b. 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
```

c. On the same bootup, 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 &
```

d. 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>
```

e. Pull the logs by running the following ADB command:

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

6.1.2 Kernel message analysis

Search for any enabled clocks in the kernel log. Ignore all the active clock voting :**_a_** clocks. *a* active clocks need to be applied while the APSS is active.

In the following example log, the xo_otg_clk is holding CXO.

```
<4>[ 1304.154791]  [0:   kworker/u8:6: 3347] Disabling non-boot CPUs ...
<6>[ 1304.154942]  [0:   kworker/u8:6: 3347] CPU0:msm_cpu_pm_enter_sleep
mode:3 during suspend
<6>[ 1304.154942]  [0:   kworker/u8:6: 3347] Enabled clocks:
<6>[ 1304.154942]  [0:   kworker/u8:6: 3347]    xo_a_clk_src:1:1 [19200000]
<6>[ 1304.154942]  [0:   kworker/u8:6: 3347]    bimc_msmbus_a_clk:1:1
[199884800] -> bimc_a_clk:1:1 [199884800]
<6>[ 1304.154942] xo_otg_clk:1:1 [1000] -> xo_clk_src:3:3 [19200000]
<6>[ 1304.154942]  [0:   kworker/u8:6: 3347]    pcnoc_keepalive_a_clk:1:1
[19200000] -> pcnoc_a_clk:1:1 [19200000]
<6>[ 1304.154942]  [0:   kworker/u8:6: 3347]    pcnoc_a_clk:1:1 [19200000]
<6>[ 1304.154942]  [0:   kworker/u8:6: 3347]    bimc_a_clk:1:1 [199884800]
<6>[ 1304.154942]  [0:   kworker/u8:6: 3347]    bb_clk2:1:1 [1000]
```

Check for any enabled interrupts.

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

```
<6>[   30.535446] msm_mpm_interrupts_detectable(): gpio preventing system
sleep modes during suspend
<6>[   30.535446] hwirq: 18
<6>[   30.535446] hwirq: 19
<6>[   30.535446] hwirq: 25
```

6.2 Enabling/disabling various modes

To enable/disable various modes:

- Enable standalone power collapse.
  - For Suspend:

```
echo 1 >
/sys/module/msm_pm/modes/cpu0/standalone_power_collapse/suspend_enabled
echo 1 >
/sys/module/msm_pm/modes/cpu1/standalone_power_collapse/suspend_enabled
echo 1 >
/sys/module/msm_pm/modes/cpu2/standalone_power_collapse/suspend_enabled
echo 1 >
/sys/module/msm_pm/modes/cpu3/standalone_power_collapse/suspend_enabled
```

  - For idle:

```
echo 1 >
/sys/module/msm_pm/modes/cpu0/standalone_power_collapse/idle_enabled
echo 1 >
/sys/module/msm_pm/modes/cpu1/standalone_power_collapse/idle_enabled
echo 1 >
/sys/module/msm_pm/modes/cpu2/standalone_power_collapse/idle_enabled
echo 1 >
/sys/module/msm_pm/modes/cpu3/standalone_power_collapse/idle_enabled
```

- Disable standalone power collapse.
  - For Suspend:

```
echo 0 >
/sys/module/msm_pm/modes/cpu0/standalone_power_collapse/suspend_enabled
echo 0 >
/sys/module/msm_pm/modes/cpu1/standalone_power_collapse/suspend_enabled
echo 0 >
/sys/module/msm_pm/modes/cpu2/standalone_power_collapse/suspend_enabled
echo 0 >
/sys/module/msm_pm/modes/cpu3/standalone_power_collapse/suspend_enabled
```

- For Idle:

```
echo 0 >
/sys/module/msm_pm/modes/cpu0/standalone_power_collapse/idle_enabled
echo 0 >
/sys/module/msm_pm/modes/cpu1/standalone_power_collapse/idle_enabled
echo 0 >
/sys/module/msm_pm/modes/cpu2/standalone_power_collapse/idle_enabled
echo 0 >
/sys/module/msm_pm/modes/cpu3/standalone_power_collapse/idle_enabled
```

- Enable power collapse (also known as power collapse with RPM notification)?
  - For Suspend:

```
echo 1 > /sys/module/msm_pm/modes/cpu0/power_collapse/suspend_enabled
echo 1 > /sys/module/msm_pm/modes/cpu1/power_collapse/suspend_enabled
echo 1 > /sys/module/msm_pm/modes/cpu2/power_collapse/suspend_enabled
echo 1 > /sys/module/msm_pm/modes/cpu3/power_collapse/suspend_enabled
```

  - For Idle:

```
echo 1 > /sys/module/msm_pm/modes/cpu0/power_collapse/idle_enabled
echo 1 > /sys/module/msm_pm/modes/cpu1/power_collapse/idle_enabled
echo 1 > /sys/module/msm_pm/modes/cpu2/power_collapse/idle_enabled
echo 1 > /sys/module/msm_pm/modes/cpu3/power_collapse/idle_enabled
```

- Disable power collapse (a.k.a. power collapse with RPM notification)?
  - For Suspend:

```
echo 0 > /sys/module/msm_pm/modes/cpu0/power_collapse/suspend_enabled
echo 0 > /sys/module/msm_pm/modes/cpu1/power_collapse/suspend_enabled
echo 0 > /sys/module/msm_pm/modes/cpu2/power_collapse/suspend_enabled
echo 0 > /sys/module/msm_pm/modes/cpu3/power_collapse/suspend_enabled
```

  - For Idle:

```
echo 0 > /sys/module/msm_pm/modes/cpu0/power_collapse/idle_enabled
echo 0 > /sys/module/msm_pm/modes/cpu1/power_collapse/idle_enabled
echo 0 > /sys/module/msm_pm/modes/cpu2/power_collapse/idle_enabled
echo 0 > /sys/module/msm_pm/modes/cpu3/power_collapse/idle_enabled
```

- Enable the Low Power mode for L2 cache

```
echo 4 > /sys/module/lpm_levels/enable_low_power/l2
```

- Disable the Low Power mode for L2 cache

```
echo 0 > /sys/module/lpm_levels/enable_low_power/l2
```

- Disable the VDD_min to 0.75 V so that the apps always votes for at least 1.0 V

```
cd /sys/kernel/debug/regulator/8916_s1_corner
/* Issue one of these to specify a voltage corner range: */
echo “4 7” > voltage /* This enforces SVS minimum */
echo “5 7” > voltage /* This enforces NOMINAL minimum */
echo “7 7” > voltage /* This enforces SUPER TURBO minimum */
```

7 Power Features Verification

7.1 To check XO shutdown and VDD minimization

7.1.1 To verify XO shutdown

The following is the step to verify from hardware:

Verify that the CXO_EN pin is low.

For steps to verify from software, see section 2.1.8

7.1.2 To verify VDD minimization

The following are the steps to verify from hardware:

1. Verify that CX rail voltage is at retention level.
2. Verify that MX rail voltage is at retention level.

Table 7-1 lists the retention levels.

Table 7-1 Retention levels

Rail	Retention voltages
CX	0.5 V or 0.65 V
MX	0.75 V

For steps to verify from software, refer to section 2.1.8

7.2 MPDecision

The following are the steps to verify:

1. Boot up the Android in SMP mode.
2. Start the mpdecision deamon (can start by invoking “mpdecision &” at shell).
3. cat /sys/devices/system/cpu/online before and after starting mpdecision you should see CPU(s) getting offlined.

7.3 CPU hotplug

The following are the steps to verify hotplug:

1. Put the CPU 1/2/3 offline/online.
2. Stop mpdecision.
3. “echo 0 > /sys/devices/system/cpu/cpuX/online” → to put the CPU offline
4. “echo 1 > /sys/devices/system/cpu/cpuX/online” → to put the CPU online
5. With the current configuration after hotplug, the respective CPU will go to WFI state.

7.4 GDSC status

To check the GDSC status of specific modules, the below registers can be checked:

```
The 32nd bit is the power on status
BIMC:
&HWIO_GCC_BIMC_GDSCR_ADDR=0x61831004
VENUS:
&HWIO_GCC_VENUS0_GDSCR_ADDR=0x6184c018
MDSS:
&HWIO_GCC_MDSS_GDSCR_ADDR=0x6184d078
JPEG:
&HWIO_GCC_CAMSS_JPEG_GDSCR_ADDR=0x6185701c
VFE:
&HWIO_GCC_CAMSS_VFE_GDSCR_ADDR=0x61858034
OXILI:
&HWIO_GCC_OXILI_GDSCR_ADDR=0x6185901c
```

7.5 To check voltage corners for CX and MX

Check the below variable to get the current active corner and the supported voltage corners:

- v.v railway.rail_state[0].corner_uvs gives the voltages of each corner for MX
- v.v railway.rail_state[1].corner_uvs gives the voltage of each corner for CX

Snippet:

```
railway.rail_state[0].corner_uvs = (
0,
650000,
1050000,
1050000,
1150000,
1287500,
1287500,
1287500,
1287500)
```

Also, the v.v railway.rail_state[0].current_active gives the active corner along with the voltage level for MX rail, and the railway.rail_state[1].current_active gives similar information for CX rail.

Snippet:

```
railway.rail_state[0].current_active_=_(
mode = RAILWAY_SUPER_TURBO = 7 = 0x7,
microvolts = 1287500 = 0x0013A54C)
```

The railway_config.c has the information about the CX and MX voltage corners for different levels.

- With RBCPR disabled, the default_uvs values are copied to variable corner_uvs
- With RBCPR enabled, the corner_uvs will be updated based on CPR recommendations

7.6 Supported retention voltages for CX and MX

The sleep_target_config.c has the information about the CX and MX supported retention voltages.

```
// retention programmed in uV ( 600000uV = 0.6V )
static const uint32 vddcx_pvs_retention_data[8] =
{
/* 000 */ 650000,
/* 001 */ 500000,
/* 010 */ 650000,
/* 011 */ 650000,
/* 100 */ 650000,
/* 101 */ 650000,
/* 110 */ 650000,
/* 111 */ 650000
};

// retention programmed in uV ( 600000uV = 0.6V )
static const uint32 vddmx_pvs_retention_data[4] =
{
/* 00 */ 750000,
/* 01 */ 650000,
/* 10 */ 750000,
/* 11 */ 750000,
};
```

7.7 Check CX RBCPR enablement and type

The enum below shows the types of RBCPR that can be enabled for a rail.

```
typedef enum
{
RBCPR_DISABLED,
RBCPR_ENABLED_OPEN_LOOP,
RBCPR_ENABLED_CLOSED_LOOP,
} rbcpr_enablement_type;
Use the below variable to check RBCPR enablement and type for CX
v.v (*((*(rbcpr_bsp_data.rails)).bsp_data)).rbcpr_enablement =
RBCPR_ENABLED_CLOSED_LOOP = 0x2,
```

7.8 Check APSS RBCPR enablement

To check APSS RBCPR enablement, look for “**qcom,cpr-enable”** line in the arch/arm/boot/dts/qcom/msm8916-regulator.dtsi

Snippet:

```
apc_vreg_corner: regulator@b018000 {
compatible = “qcom,cpr-regulator”;
reg = <0xb018000 0x1000>, <0xb011064 4>, <0x58000 0x1000>;
reg-names = “rbcpr”, “rbcpr_clk”, “efuse_addr”;
interrupts = <0 15 0>;
regulator-name = “apc_corner”;
regulator-min-microvolt = <1>;
regulator-max-microvolt = <7>;
        .
        .
        .
.
qcom,speed-bin-fuse-sel = <1 34 3 0>;
qcom,cpr-speed-bin-max-corners =
<0 0 2 4 7>;
qcom,cpr-quot-adjust-scaling-factor-max = <650>;
qcom,cpr-enable;
   };
```

7.9 To check modem DCVS

To confirm modem DCVS enablement, check the below file in EFS explorer. The default DCVS algorithm is qdsp classic algorithm.

```
/nv/item_files/Core_Cpu/CoreAll/Startup/Algorithm.txt = qdsp_classic
```