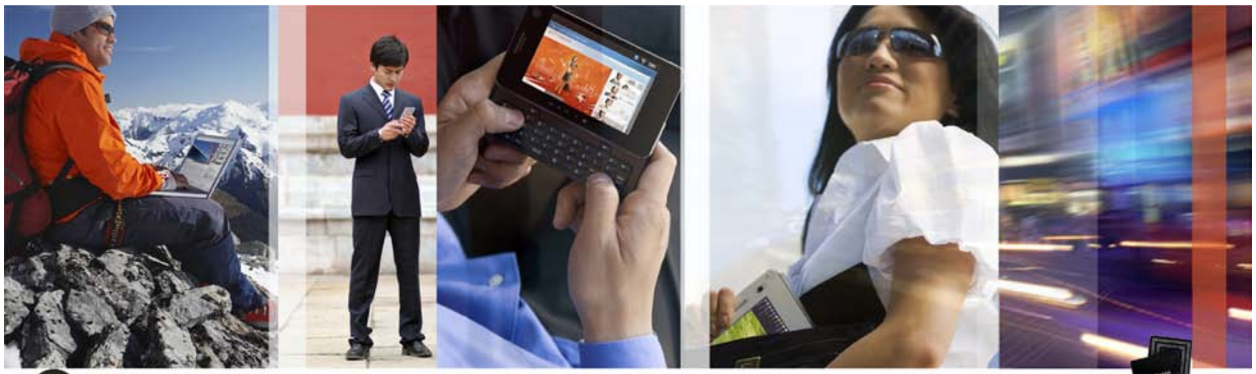


Power Consumption Optimization and Debugging Guide for MSM8960™ Devices

80-N9327-1 A

January 30, 2012

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

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

Contents

Tables

Revision history

Revision	Date	Description
A	Jan 2012	Initial release

1 Introduction

1.1 Purpose

The MSM8960™ chipset has a special ARM7™ processor called RPM to handle power resource management. This document provides details on how to optimize specific power test cases and debug issues related to the RPM, apps, and modem.

1.2 Scope

This document is intended for Qualcomm customers working with MSM8960 chipsets. This document assumes the customer has access to the RPM elf, an awareness of MSM8960 architecture, and a general understanding of MSM8960 features.

1.3 Conventions

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

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

1.4 References

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 Reference documents that are no longer applicable are deleted from this table; therefore, reference numbers may not be sequential.

Table 1-1 Reference documents and standards

Ref.	Document	
Qualcomm		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*Using RPM 1.0 Logs User Document*	80-N4713-1

1.5 Technical assistance

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

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

1.6 Acronyms

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

2 Optimizing Sleep/Rock-Bottom Current

This chapter provides details on how each of the specific aspects of the system power management may be evaluated for proper operation.

2.1 Overview of possible culprits impacting sleep current

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

- One or more major subsystems on the MSM® chipset are failing to go into their lowest power mode, and this in turn is preventing XO shutdown. On the MSM8x60, these subsystems may be:
  - Low Power Audio Subsystem (LPASS) not going into power collapse
  - Applications Subsystem (APSS) not going into power collapse
  - Modem subsystem (Q6SW-MSS) not going into sleep
  - RIVA subsystem not going into sleep
  - DSPS – Sensor subsystem not going to sleep
  - Modem subsystem (Q6FW – MSS) not going into sleep
- One or more peripheral devices (outside of the MSM chipset) are not going into their lowest power state.

If all the subsystems are good, 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 going into sleep
- An on-chip macro should have been foot-switched and was not

2.2 Test setup-related issues impacting power

In determining if test setup-related issues are impacting power, you must ensure that the setup is correct. Sleep current can be measured by placing the phone either in Airplane mode or by connecting to a callbox (WCDMA/GSM/1X mode) in Online mode.

Airplane mode

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

Online mode

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

System variables

Always ensure the variable be_gentle_to_daisy_chain is *not* set to TRUE. If this variable is set to TRUE, the system will not enter low power modes.

2.3 Ensuring APSS power collapse

2.3.1 Check RPM external logs

APSS power collapse can be confirmed by looking at 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)
- 34.521881 – rpm_status_updating (resource: "Apps L2 Cache") (in_flux: 1)
- 34.521942 – rpm_status_updating (resource: "Apps L2 Cache") (in_flux: 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, if the above messages are not available in RPM logs, it can be assumed that apps is already in power collapse, or it might not have entered in power collapse.

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

Also, QNX HLOS users need to check for *both* Core 1 and Core 0, as there is no guarantee on non-Linux targets that Core 0 will be last.

2.3.2 Check gpRPMFWMaster data structure in RPM

APSS sleep can be confirmed by checking the gpRPMFWMaster data structure on RPM. Break the Trace32 execution in RPM at the clk_regime_swfi() function to monitor this variable.

Check gpRPMFWMaster[0], 0→APPS information

- If apps is in Sleep mode, selected_set = DAL_RPM_CONFIG_SET_SLEEP = 1 = 0x1.
- If apps is still operating in Active mode, selected_set = DAL_RPM_CONFIG_SET_PRIMARY = DAL_RPM_CONFIG_SET_ACTIVE_0 = 0x0.

If the apps is *not* in its sleep set and was expected to be, see Section 2.3.1 to determine the cause.

2.4 Ensuring modem sleep

2.4.1 Check RPM external logs

Modem sleep can be confirmed by looking at 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 in 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.

2.4.2 Check the gpRPMFWMaster data structure in RPM

Modem sleep can also be confirmed by checking the gpRPMFWMaster data structure on RPM. Break the Trace32 execution in RPM at the clk_regime_swfi() function to monitor this variable.

Check gpRPMFWMaster[1], 1→Modem information

- If the modem is in Sleep mode, selected_set = DAL_RPM_CONFIG_SET_SLEEP = 1 = 0x1.
- If the modem is still operating in Active mode, selected_set = DAL_RPM_CONFIG_SET_PRIMARY = DAL_RPM_CONFIG_SET_ACTIVE_0 = 0x0.

2.5 Ensuring LPASS power collapse

2.5.1 Check RPM external logs

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

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

2.5.2 Check the gpRPMFWMaster data structure in RPM

LPASS power collapse can be confirmed by checking the gpRPMFWMaster data structure on RPM. Break the Trace32 execution in RPM at the clk_regime_swfi() function to monitor this variable.

Check gpRPMFWMaster[2], 2→LPASS information

- If the LPASS is in Sleep mode, selected_set = DAL_RPM_CONFIG_SET_SLEEP = 1 = 0x1.
- If the LPASS is still operating in Active mode, selected_set = DAL_RPM_CONFIG_SET_PRIMARY = DAL_RPM_CONFIG_SET_ACTIVE_0 = 0x0.

2.6 Ensuring overall system entering low power modes

The various low power modes that the system can enter are:

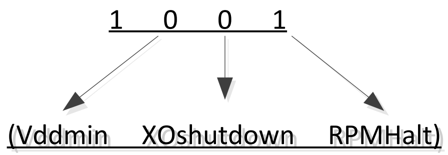
- RPM Halt
- XO shutdown
- VDD min

Before checking why the system is not entering the low power modes, ensure the following variable is set properly.

- Check the be_gentle_to_daisy_chain variable. If this is set to TRUE, the system will not enter low power modes.

Check the RPM NPA logs for the following messages to ensure that the system is entering low power modes:

- Check the Sleep/LPR Active state from RPM NPA logs. Its Active state must be 7. Otherwise, the system is not entering its respective Power Save mode, e.g., if Sleep/LPR Active state is 1: 001: (Vddmin XOshutdown RPMHalt). In this example, RPM Halt happens, but XO shutdown and VDD min do not happen. The following is the NPA log snippet:



```
: npa_dump
: npa_resource (name: "/sleep/uber") (handle: 0x3A24C) (units: on/off)
(resource max: 15) (active max: 15) (active state 0)  (active headroom: -
15) (request state: 0)
:         npa_client (name: sleep) (handle: 0x3C870) (resource: 0x3A24C)
(type: NPA_CLIENT_REQUIRED 0x40) (request: 0)
:         end npa_resource (handle: 0x3A24C)
: npa_resource (name: "/sleep/lpr") (handle: 0x3A294) (units: bitmask)
(resource max: 4294967295) (active max: 4294967295) (active state 7)
(active headroom: 8) (request state: 7)
:         npa_client (name: /node/sleep/uber) (handle: 0x3C828) (resource:
0x3A294) (type: NPA_CLIENT_REQUIRED 0x40) (request: 7)
:         end npa_resource (handle: 0x3A294)
```

2.6.1 Checking RPM Halt

Check the Sleep/LPR Active state from RPM NPA logs. The Lower Significant bit represents RPM Halt, e.g., if Sleep/LPR Active state is 1: 001: (Vddmin XOshutdown RPMHalt). In this example, RPM Halt happens, but XO shutdown and VDD min do not happen.

Before its own SWFI, RPM programs the MPM for CXO/PXO off. This is done through a call to mpm_xo_disable(), which internally calls HAL_mpm_DisableXO().

2.6.2 Checking XO shutdown

Check the Sleep/LPR Active state from RPM NPA logs. The lower significant bit represents RPM Halt, e.g., if Sleep/LPR Active state is 3, 011 = (Vddmin XOshutdown RPMHalt). In this example, RPM Halt and XO shutdown happen, but not VDD min.

2.6.3 Checking VDD min

2.6.3.1 Check Sleep/LPR state from NPA logs

Check the Sleep/LPR Active state from RPM NPA logs. The second bit represents RPM Halt, e.g., if Sleep/LPR Active state is 7, 111=> Vddmin XOshutdown RPMHalt. In this example, the system is able to enter all the low power modes.

2.6.3.2 Check Sleep/Uber state from NPA logs

To check Sleep/Uber state from NPA logs, check if VDD min is working with the expected CXO, PXO minimum voltages.

- S0A power rail is for PXO. S0A voltage expected during VDD min is 0.75 V.
- S1A power rail is for CXO. S1A voltage expected during VDD min is 0.55 V.

Measure both of the above voltages during sleep for confirmation that VDD min is working.

Check the Sleep/Uber Active state from RPM NPA logs. This should be 0 in Sleep. Otherwise, the respective resource is expected by some client, e.g., if Active state is 13, 1101=> mem dig pxo cxo. In this example, everything is needed by the system except PXO.

2.6.3.3 Check gpRPMFWMaster structure fields

Check gpRPMFWMaster structure fields for VDD min vote from various master processors.

- gpRPMFWMaster is an array that stores the state of each master processor:
  - gpRPMFWMaster[0] – Apps
  - gpRPMFWMaster[1] – Modem software
  - gpRPMFWMaster[2] – LPASS
  - gpRPMFWMaster[3] – RIVA
  - gpRPMFWMaster[4] – DSPS
  - gpRPMFWMaster[5] - Modem firmware

Each master processor is either in active set (primary set) or sleep set:

- selected_set = DAL_RPM_CONFIG_SET_PRIMARY = 0x0
- selected_set = DAL_RPM_CONFIG_SET_SLEEP = 0x1

MSM8660 resource-specific settings are:

- CXO = Index 5
- PXO = Index 6
- Vdd Dig (Vdd Cx, VREG S3) = Index 32 (0x20)
- Vdd Mem (Vdd Mx, VREG L24) = Index 61 (0x3D)

PMIC resources' (voltage regulators) data is stored in a buffer; the actual voltages being voted for are the buffer contents.

A useful trick for checking the regulator voltages, as the PMIC resources are not straightforward, is that their data has structure to it. You need to use Trace32 to parse this structure for you, e.g.:

```
v.v (pm_npa_vreg_smps_type*) gpRPMFWMaster[a].set[b].data[c]
```

2.6.3.4 Modem Vdd_min voting debug example

To find the voltage that the modem processor is voting for Vdd_Dig while it is asleep, check the variable in T32 using the following command:

```
v.v (pm_npa_vreg_smps_type*) gpRPMFWMaster[1].set[1].data[0x30]
```

In this example:

- gpRPMFWMaster[1] – 1 refers to modem
- set[1] – 1 refers to Sleep set
- data[0x20] – References Vdd_Dig resource

```
(pm_npa_vreg_smps_type*)gpRPMFWMaster[1].set[1].data[0x20]=0x0003BB78=
gRPMFWDataSpace[1025] -> (
    uvol = 537500 = 0x0008339C,
    pd = 1 = 0x1,
    pc = 0 = 0x0,
    pf_low = 0 = 0x0,
    ip = 1 = 0x1,
    ia = 1 = 0x1,
    fm = 0 = 0x0,
    power_mode = 1 = 0x1,
    freq = 4 = 0x4,
    freq_clk_src = 0 = 0x0,
    reserved = 0 = 0x0)
```

In this case, the modem is voting for Vdd_Dig minimization (537.5 mV).

2.6.4 Sleep current deltas/leakage – Further analysis

Further ways to analyze Sleep current delta/leakage include:

- Take the component breakdown and compare against the FFA. 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 just before sleep on the customer device to the FFA. If there are any differences in the working hardware on either side, check for any hardware changes and make sure that GPIOs are configured as needed. This will help reduce the PAD current leakage.
- Estimate design deltas from schematics. The customer device design may not be the same as the FFA design. Analyze the extra components on the customer device and approximate their power consumption if they have a breakout point to measure.

3 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. If the issue is due to high sleep current, follow the high sleep current issue debugging explained in Chapter 2.

3.1 Debugging high standby current – Awake current issue

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

3.1.1 Test setup-related issues impacting power

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

- Check the setup. Ensure the customer is testing with the callbox. Testing in a live network will obviously have higher awake current, 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.
- Check which portion of the awake current is longer compared to FFA awake. If it is XO warm-up time/RF Wakeup/RF processing/RF Sleep/System Sleep, F3 logs are useful for this analysis from the modem protocol.

3.2 Inspecting and optimizing average current during page wakeups

To inspect and optimize average current during page wakeups, compare the clock dump during awake between the customer phone and the FFA. This will give you an idea, although during awake, frequencies continue changing due to DCVS and CPU behavior.

4 Optimizing Talk Current

4.1 Test setup-related issues impacting power

To analyze test setup-related issues impacting power:

- Set NV item 00010 to the respective modem technology used (WCDMA only/GSM only/1X only). The phone is expected to work when this NV item is set to Automatic mode as well.
- Ensure Tx power is at 0 dBm per the Qualcomm standard test recommendation for voice calls.
- Turn off Rx Diversity.

4.2 Inspecting clocks

Clocks are a main cause of power consumption in voice call scenarios. Taking a clock dump during a voice call and ensuring that all the clocks are operating at recommended frequencies always helps. Another way of doing the same is to compare clock dumps during a voice call between the customer phone and the FFA and analyzing the clock differences.

4.3 Component breakdown

Take a component breakdown and compare it against the FFA. Based on the power rail differences, focus on respective power rail optimization. If it is core, a current clock analysis would be helpful.

5 Further RPM Debug Details and Log Procedures (RPM and Modem)

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

5.1 Clock dumps

For clock dumps:

1. Open ARM7_dynamic T32 from the T32_dynamic folder in modem build location:

   ```
   ..\..\modem_proc\tools\t32\t32_dynamic
   ```

2. Type **sys.m.a** to attach T32 to the FFA.
3. Re-create the scenario for whatever test case needs the clock dumps.
4. **cd** ..\..\modem_proc\core\systemdrivers\hal\clk\chipset\msm8960\tools
5. Break the T32 by clicking **Pause** on the top or by pressing **F8.**
6. Type **do testclock.cmm**.
7. Type **all** in the window.

Clock dump is printed in the message area of T32.

5.2 RPM debug

RPM publishes a small log into a very 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, using a set of IDs and a variable number of parameters per message.

5.2.1 RPM Halt and the JTAG daisy chain

Legacy ARM® cores (ARM7, ARM11™) resynchronize the JTAG RTCK signal to their respective processor clocks. The result is that halting the RPM clock, e.g., during sleep, causes a no-RTCK condition on the JTAG daisy-chain. Therefore, if reliable JTAG is required for any master, the RPM cannot halt, as the RPM is on the daisy-chain. A few methods of working around this are built into the system:

- RPM source code has a file, sleep_os.c, in which there is a variable named be_gentle_to_the_daisy_chain that disables all forms of RPM sleep, allowing unimpeded JTAG usage.

  Setting the be_gentle_to_daisy_chain disables all low power modes. However, this setting is not sustained across resets. Therefore, although it is the most reliable method, it is not the most convenient. This variable exists only on the RPM side. This can be checked only via ARM7 JTAG using v.v be_gentle_to_daisy_chain.

- From the command shell on the PC, connect to the device using ADB, enter ADB shell and issue the following command:

  ```
  echo 0 > /sys/module/rpm_resources/enable_low_power/rpm_cpu
  ```

  Basically, what this does is to instruct the Linux sleep software to send a message to the RPM that will then have the RPM set the be_gentle_to_daisy_chain variable.

- Create a /nv/items_files/sleep/core0/sleep_config.ini file.
  - This file must be placed at above location (using EFS Explorer).
  - This file disables XO shutdown and VDD min. Contents of the file are:

    ```
    [rpm.handshake]
    disable=1
    ```

  This again sends a message at init time to the RPM to set the be_gentle_to_daisy_chain variable. This is a read at init time file so to make this change effective, the system must be reset.

- Disabling the watchdog helps to break RPM easily.

5.2.2 Finding RPM build version used from build

Before debugging with the RPM in T32, the respective RPM symbols must be loaded. This can be accomplished by determining the RPM version the customer is using in the build. To check this information:

1. Open ARM7 and attach to the running target by **sys.m.a**.
2. Enter the command:

```
d.in 0x00108008 /LONG
   Example: 0x01010004 –> 0x01.0x01.0x0004 -> RPM.01.01.04
```

3. Load symbols from the 39th RPM reference build:

```
   d.load.elf   ..\...\RPM<version no.>\build\rpm\8960\build\RPM.elf
/nocode /noclear
```

5.2.3 Saving RPM RAM dumps

RPM dumps can be saved in the following manner and sent for further analysis.

Method 1

1. Create the issue scenario where RPM dumps are needed.
2. Open ARM7 T32 and attach by **sys.m.a**.
3. Break T32 and do the following for saving the RPM memory dump:

```
d.save.binary C:\Temp\CODERAM.bin 0x20000++0x23FFF
d.save.binary C:\Temp\MSGRAM.bin 0x104000++0x5FFF
```

Method 2

1. Create the issue scenario where RPM dumps are needed.
2. Open ARM7 T32 and attach by **sys.m.a**.
3. Break T32 and

```
do <Metabuildl location>\ common\tools\cmm\common\std_savelogs.cmm
```

This will give you a menu and you can select RPM RAM dumps option. The script prompts for the location at which to save the logs.

Using Method 2 is better, as it also saves the RPM register dumps along with a script to reload the registers in the specified location.

5.2.4 Loading RPM dumps onto T32 and extracting logs

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

1. Open the T32 simulator and do **sys.up**.
2. Go to the dumps directory and load:

```
d.load.binary MSGRAM.bin 0x108000
d.load.binary CODERAM.bin 0x20000
```

3. Find out which RPM build version is being used and load the respective RPM ELF (see Section 5.2.2).

```
d.load.elf    RPM.00.00.xx\build\rpm\8960\build\RPM.elf    /nocode
```

5.2.5 RPM External log and RPM NPA log – Using Trace32 (pre-RPM.00.00.71 version)

To get RPM logs for the RPM versions prior to 71:

1. While attached to the RPM and in the Break state, run the following command in T32:

```
do \\<RPM build location>\core\power\ulog\scripts\ULogDump.cmm <path to
your directory>
```

   This places the RPM’s NPA and external logs into your log directory. The NPA log is a plain text log, similar to the NPA logs present on other processors and platforms.

2. The RPM external log requires the use of a parsing tool to interpret. If you need to read it, from your log directory, open Cygwin and issue the following command:

```
python \\<RPM build location>\core\power\rpm\dal\scripts\rpm_log.py -f
"RPM External Log.ulog” > parsed_output.txt
```

3. rpm_log.py –h provides a list of supported options.

5.2.6 RPM external log and RPM NPA log – Using T32 (RPM.00.00.71 and later)

Here is the procedure to get RPM logs for the RPM versions 71 and later. The instructions are very similar to the instruction for RPM versions prior to 71. As of RPM 71, the NPA log and the RPM external log have been combined. This means the dump procedure is slightly more complex, but a benefit is that more NPA log entries appear than in the pre-71 baselines, the messages are properly interleaved, and a new high-resolution timestamp is available that can be used for better profiling.

To get RPM logs for the RPM versions 71 and later:

1. While attached to the RPM and in the Break state, run the following commands in T32:

```
do \\<RPM build location>\core\power\ulog\scripts\ULogDump.cmm <path to
your directory>

do \\<RPM build location>\core\power\npa\scripts\NPADump.cmm < path to
your directory>
```

2. Run the following script for postprocessing:

```
python rpm_log.py -f "RPM External Log.ulog" -n "NPA Log.ulog"  >
parsed_output.txt
```

5.2.7 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.

Note that taking RPM logs using adb can interfere with time response/performance of RPM toward system-wise requests.

5.3 Checking how many times system is able to enter in different low power modes (statistics)

To obtain the count of how many times the system is able to enter different low power modes, enter the following:

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

rpm_stats provides this information.

5.4 Modem NPA logs

Execute the following script from Modem_Build to obtain the modem NPA logs:

```
<Modem_Build>\modem_proc\core\power\npa\scripts\NPADump.cmm
```

5.5 RPM NPA log analysis

See [Q2] for more information on RPM and NPA log analysis.

5.6 GPIO dumps

5.6.1 All GPIOs in the hardware

1. Use the following script to read the current configuration of all GPIOs in the hardware:

   ```
   <Modem_Build>\core\systemdrivers\tlmm\t32\tlmm_gpio_8x60.cmm
   ```

2. Select Option 13: Read All GPIO Configurations.

5.6.2 Default sleep configuration stored in software

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

   ```
   <Modem_Build>\core\systemdrivers\tlmm\t32\tlmm_sleep_configs.cmm
   ```

2. Select Option 1: Read All GPIO Configurations.

5.7 F3 trace logs

F3 logs are useful to analyze the Awake Current issues and explain modem behavior.

5.7.1 F3 trace logs with QXDM Professional® (by connecting USB)

As the USB is connected to the system for QXDM Pro, the system cannot enter sleep, hence;

1. Connect the USB, Open QXDM Pro, and press **F3**.
2. F3 logs can be seen in QXDM Pro.

5.7.2 F3 trace logs with T32 (no need of USB connection)

1. NV items to be set items are as follows:
   a. 1892 should be set to 0x5
   b. 1895 should be set to 0xFF
   c. 1962 should be set to 0x1
2. After setting the above, remove the USB and power cycle.
3. Attach the JTAG, reproduce the scenario, press any key on the phone and break the trace.
4. Execute the following script:

```
do <Modem_Buildr>\build\ms\recover_f3.cmm” in JTAG/TRACE window.
```

6 Debugging Apps Processor

6.1 RPM regulator logs

Mount the debugfs in ADB shell.

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

- In adb shell, type **dmseg** to see the dmseg's. Dmessages are a snapshot of the kernel messages.
- In adb shell, type **cat /proc/kmsg** to see kmsg's. Kmessages are kernel messages.
- The only difference between dmseg and kmsg is that the former shows just the requests for the current state whereas a kmsg is a persistent log.
- The RPM regulator driver contains a module parameter, debug_mask, which can be used to conditionally print regulator requests sent to the RPM. The lower four bits of this mask have the following meaning:
  - Bit 0 – Print requests that are actually sent to the RPM
  - Bit 1 – Show all votes made through the rpm_vreg_set_voltage API or made through the regulator framework for regulators with sleep_selectable != 0 (even if the votes did not result in an RPM request)
  - Bit 2 – Print duplicate requests made by consumers that were not actually sent to the RPM

6.1.1 Enabling RPM regulator messages

The following echo command can be used to print regulator requests sent to RPM during dmseg or kmsg:

```
echo 1 > /sys/module/rpm_regulator/parameters/debug_mask
```

6.1.2 Example of RPM regulator messages

The following are example kmsg log snippets showing requests for regulators.

1. <6>[100.499526] rpm-regulator: vote received 8921_S3 : voter=1, set=A, v_voter=1100 mV, v_aggregate=1200 mV
2. <6>[100.508992] rpm-regulator: ignored duplicate request 8921_s3 : set=A; req[0]={163, 0xC00324B0}, req[1]={164, 0x00018065}

3. <6>[100.525072] rpm-regulator: vote received 8921_s3 : voter=3, set=S, v_voter=1100 mV, v_aggregate=1100 mV
4. <6>[100.533646] rpm-regulator: buffered 8921_S3 : s=S, v=1100 mV, ip= 50 mA, fm=none (0), pc= none (0), pf=none (3), pd=Y (1), ia= 50 mA, freq=12, clk=0; req[0]={165, 0xC003244C}, req[1]={166, 0x00018065}

Lines 1 and 3 were printed as a result of bit 1 being set in rpm_regulator.debug_mask. Line 2 was printed as a result of bit 2 being set in rpm_regulator.debug_mask. Line 4 was printed as a result of bit 0 being set in rpm_regulator.debug_mask.

The components of lines 1 and 3 have the following meaning:

- vote received <regulator name> – Name of the regulator
- voter – ID of the voter:
  - 0 – Regulator framework consumers (any driver that used regulator_set_voltage, regulator_enable, etc.)
  - 1 – ACPU clock 0 driver
  - 2 – ACPU clock 1 driver
  - 3 – Clock driver (or some other driver using the private rpm_vreg_set_voltage() API on a regulator other than 8058_s0 or 8058_s1)
- set – Specifies when the request will take effect:
  - A – Active, i.e., immediately
  - S – Sleep, i.e., the request will take effect after both apps processor cores have been suspended
- v_voter – Voltage requested by this voter in mV
- v_aggregate – Aggregate voltage (max) of all voters; this is what is actually sent to the RPM

The components of line 4 have the following meaning:

- (buffered|sent) <regulator name> – Name of regulator; buffered implies that the request will be sent right before suspend, sent implies that the request will be sent immediately
- s – Set, specifies when the request will take effect:
  - A – Active, i.e., immediately
  - S – Sleep, i.e., the request will take effect after both apps processor cores have been suspended
- v – Voltage requested in mV. A regulator disable request is signified by passing v = 0
- ip – Peak current required by all consumers of this regulator on the apps processor. It is aggregated in the RPM to determine if high or low power mode should be set.
- fm – Force mode; this allows an RPM master to bypass load current aggregate, it is unused by the rpm-regulator driver and should always be 0.
- pc – Pin control; specifies which pin control inputs to use. It can be none (0), or a combination of – A0, A1, D0, and D1.

- pf – Pin function; specifies which pin control function to use:
  - on/off (0) – Pin controls enable state of the regulator i.e., on or off
  - HPM/LPM (1) – Pin controls mode of the regulator i.e. HPM or LPM
  - sleep_b (2) – Regulator follows sleep_b signal to transition to LPM automatically during power collapse
  - none (3) aka manual – Do not use pin control
- pd – Pull down when disabled
- ia – Average current, currently unused by the RPM and set equal to ip in the rpm-regulator driver
- freq – Frequency, specifies the switching frequency of SMPS regulators based on enum values
- clk – Clock source, unused by rpm-regulator driver and always set to 0
- req[0] – Raw value of first DWORD of the request sent to the RPM. It is a tuple of RPM ID and value.
- req[1] – Raw value of second DWORD of the request sent to the RPM. It is a tuple of RPM ID and value.
  - In this example, the 64-bit long bit-packed struct used in the RPM to hold the request would have a value of 0x00018065C00324B0.

Requests for switches and the NCP (negative charge pump) contain one additional component:

- state – Enable state of the regulator: on (1) or off (0)
  - In the case of the NCP, the request voltage is not 0 in a disable request; instead, state is set to 0.

6.2 PM8921™ regulator

The PM8921 regulator driver contains a module parameter, debug_mask, which can be used to conditionally print regulator actions taking place. The lower four bits of this mask have the following meaning:

- Bit 0 – Print actions that resulted in a PMIC SSBI write
- Bit 1 – Print duplicate actions which did not result in PMIC SSBI writes
- Bit 2 – Print initial values after the regulator is registered with the regulator framework core
- Bit 3 – Print out address and data of PMIC SSBI writes to regulator registers

6.2.1 Enable PM8921 action printing at runtime

```
echo 1 > /sys/module/pm8921_regulator/parameters/debug_mask
```

The other techniques listed in the RPM regulator section for setting the debug_mask at compile time also apply to the PM8921 regulator. Just change rpm_regulator to pm8921_regulator in the commands.

6.2.2 Example PM8921 regulator kmsg ouput

The following are example kmsg log snippets showing requests for regulators.

```
1. <6>[    0.315948] pm8921_vreg_masked_write: 8921_s1: write(0x009)=0x15
2. <6>[    0.316040] pm8921_vreg_show_state: enable      8921_s1  : on , v=1225000 uV,
   mode=HPM, pc= none
3. <6>[    0.361083] pm8921_vreg_show_state: initial     8921_lvs1: off, pc= none
4. <6>[    3.395446] pm8921_vreg_show_state: set mode    8921_l3  : on , v=3075000 uV,
   mode=LPM, pc= none
5. <6>[    3.404449] pm8921_vreg_show_state: set voltage 8921_l4  : on , v=1800000 uV,
   mode=HPM, pc= none
6. <6>[    6.253906] pm8921_vreg_show_state: set mode    8921_l7  : on , v=2950000 uV,
   mode=HPM, pc= none
7. <6>[    6.280242] pm8921_vreg_show_state: set mode    8921_l6  : on , v=2950000 uV,
   mode=HPM, pc= none
8. <6>[    6.288055] pm8921_vreg_show_state: set mode    8921_l7  : on , v=2950000 uV,
   mode=HPM, pc= none
9. <6>[    6.310058] pm8921_vreg_show_state: set mode    8921_l6  : on , v=2950000 uV,
   mode=HPM, pc= none
```

Line 1 was printed as a result of bit 3 being set. Lines 2, 4, 5, 6, and 7 were printed because bit 0 was set. Line 3 was printed because bit 2 was set. The duplicate actions on lines 8 and 9 were printed because bit 1 was set.

The components of the log messages are:

```
pm8921_vreg_show_state: <action> <regulator name> <enable state> [additional values]

pm8921_vreg_masked_write: <regulator name>: write(<SSBI register address>)=
<new register value>
```

Additional values are:

- v – Voltage to which the regulator is physically set, in microvolts
- mode – LPM (low power mode) or HPM (high power mode); for SMPS regulators, LPM is PFM, and HPM is PWM
- pc – Pin control inputs in use; can be none, or a combination of D1, A0, A1, and A2

6.3 Logging clocks during suspend

This is a very important tool to debug what clocks were left on when going into Power Collapse in the apps processor. This is to debug the situation when you see the apps processor power collapsing but not voting for TCXO.

6.3.1 Enabling the variable

To enable the variable:

1. Mount the debug file system.

   mount –t debugfs debug sys/kernel/debug

2. Enable the debug_suspend variable.

   echo 1 > /sys/kernel/debug/clk/debug_suspend

6.3.2 Procedure

The procedure is:

1. Make sure using the RPM logs, apps is the processor that is not voting for sleep.
2. Use the USB cable to enable the logging through adb shell
3. Remove USB and perform 4 to 5 suspend/resume cycles on the device.
4. Connect the USB cable again and get a DMESG log.

6.3.3 Example/analysis

Example analysis is:

```

<6>[  228.603607] mipi_dsi_cdp_panel_power: state : 0

<4>[  228.611541] Freezing user space processes ... (elapsed 0.02 seconds)
done.

<4>[  228.640197] Freezing remaining freezable tasks ... (elapsed 0.01
seconds) done.

<4>[  228.660247] Suspending console(s) (use no_console_suspend to debug)

<6>[  228.707275] PM: suspend of devices complete after 37.048 msecs

<6>[  228.709228] PM: late suspend of devices complete after 1.953 msecs

<4>[  228.709228] Disabling non-boot CPUs ...

<6>[  228.709686] msm_pm_enter

<6>[  228.709686] msm_pm_enter: power collapse

```

```
<6>[  228.709686] Enabled clocks:

<6>[  228.709686]     pll8_clk

<6>[  228.709686]     afab_clk

<6>[  228.709686]     afab_a_clk

<6>[  228.709686]     cfpb_clk

<6>[  228.709686]     cfpb_a_clk

<6>[  228.709686]     ebi1_clk

<6>[  228.709686]     ebi1_a_clk

<6>[  228.709686]     mmfpb_clk

<6>[  228.709686]     mmfpb_a_clk

<6>[  228.709686]     mmfpb_a_clk

<6>[  228.709686]     sfab_clk

<6>[  228.709686]     sfab_a_clk

<6>[  228.709686]     pmem_clk

<6>[  228.709686]     dma_bam_p_clk

<6>[  228.709686] Enabled clock count: 14

<6>[  228.709686] msm_mpm_irqs_detectable: cannot monitor
000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000
000,00000000,00000000,00000000,00000000,00000000

<6>[  228.709686] msm_pm_enter: return

<6>[  228.710876] PM: early resume of devices complete after 1.190 msecs

<6>[  228.711853] wakeup wake lock: event2-319

<6>[  228.825439] PM: resume of devices complete after 113.409 msecs

<4>[  228.922119] Restarting tasks ...

<6>[  228.934295] request_suspend_state: wakeup (3->0) at 228796324577
(1970-01-02 00:04:13.069415797 UTC)

<4>[  228.944183] done.
```

As shown in the logs above, after msm_pm_enter, 14 clocks were left on when entering power collapse on the apps processor.

These clocks must not be misinterpreted. By default, every time you go to power collapse some clocks are displayed in this log as on, but are later turned off by the RPM depending on the votes for TCXO shutdown.

The advantage of this log is in case when you have verified that apps is not going to power collapse in a customer device you can get a log on the same build/target on a reference device (CDP/MTP for MSM8960, which goes to expected power collapse) to check what clocks are left on for that particular build when you enable this logging.

If any clocks, other than those on the reference device, are on in a customer device, you can be sure that those clocks are your culprits.

6.4 Enabling/disabling various modes

To enable/disable various modes:

- Enable standalone power collapse:
  - For suspend:
    - echo 1 > /sys/module/pm_8x60/modes/cpu0/standalone_power_collapse/suspend_enabled
    - echo 1 > /sys/module/pm_8x60/modes/cpu1/standalone_power_collapse/suspend_enabled
  - For idle:
    - echo 1 > /sys/module/pm_8x60/modes/cpu0/standalone_power_collapse/idle_enabled
    - echo 1 > /sys/module/pm_8x60/modes/cpu1/standalone_power_collapse/idle_enabled
- Disable standalone power collapse:
  - For suspend:
    - echo 0 > /sys/module/pm_8x60/modes/cpu0/standalone_power_collapse/suspend_enabled
    - echo 0 > /sys/module/pm_8x60/modes/cpu1/standalone_power_collapse/suspend_enabled
  - For idle:
    - echo 0 > /sys/module/pm_8x60/modes/cpu0/standalone_power_collapse/idle_enabled
    - echo 0 > /sys/module/pm_8x60/modes/cpu1/standalone_power_collapse/idle_enabled
- Enable power collapse (also known as power collapse with RPM notification)?
  - For suspend:
    - echo 1 > /sys/module/pm_8x60/modes/cpu0/power_collapse/suspend_enabled
    - echo 1 > /sys/module/pm_8x60/modes/cpu1/power_collapse/suspend_enabled
  - For idle:
    - echo 1 > /sys/module/pm_8x60/modes/cpu0/power_collapse/idle_enabled
    - echo 1 > /sys/module/pm_8x60/modes/cpu1/power_collapse/idle_enabled

- Disable power collapse (a.k.a. power collapse with RPM notification)?
  - For suspend:
    - echo 0 > /sys/module/pm_8x60/modes/cpu0/power_collapse/suspend_enabled
    - echo 0 > /sys/module/pm_8x60/modes/cpu1/power_collapse/suspend_enabled
  - For idle:
    - echo 0 > /sys/module/pm_8x60/modes/cpu0/power_collapse/idle_enabled
    - echo 0 > /sys/module/pm_8x60/modes/cpu1/power_collapse/idle_enabled
- Enable the low power mode for L2 cache
  - echo 1 > /sys/module/rpm_resources/enable_low_power/L2_cache
- Disable the low power mode for L2 cache
  - echo 0 > /sys/module/rpm_resources/enable_low_power/L2_cache
- Enable the low power mode for PXO
  - echo 1 > /sys/module/rpm_resources/enable_low_power/pxo
- Disable the low power mode for PXO
  - echo 0 > /sys/module/rpm_resources/enable_low_power/pxo
- Enable the vdd minimization to 0.75v
  - echo 1 > /sys/module/rpm_resources/enable_low_power/L2_cache
  - echo 1 > /sys/module/rpm_resources/enable_low_power/pxo
  - echo 1 > /sys/module/rpm_resources/enable_low_power/vdd_dig
  - echo 1 > /sys/module/rpm_resources/enable_low_power/vdd_mem
- Disable the vdd minimization to 0.75v so that Apps always votes for at least 1.0v
  - echo 0 > /sys/module/rpm_resources/enable_low_power/vdd_dig
  - echo 0 > /sys/module/rpm_resources/enable_low_power/vdd_mem
- Enable the vdd minimization to 0.5v?
  - echo 1 > /sys/module/rpm_resources/enable_low_power/L2_cache
  - echo 1 > /sys/module/rpm_resources/enable_low_power/pxo
  - echo 2 > /sys/module/rpm_resources/enable_low_power/vdd_dig
  - echo 2 > /sys/module/rpm_resources/enable_low_power/vdd_mem
- To check whether the various low power modes are enabled:
  - cat /sys/module/rpm_resources/enable_low_power/L2_cache
  - cat /sys/module/rpm_resources/enable_low_power/pxo
  - cat /sys/module/rpm_resources/enable_low_power/vdd_dig
  - cat /sys/module/rpm_resources/enable_low_power/vdd_mem

- Enable the low power mode for L2 cache
  - echo 1 > /sys/module/rpm_resources/enable_low_power/L2_cache
- Disable the low power mode for L2 cache
  - echo 0 > /sys/module/rpm_resources/enable_low_power/L2_cache
- Enable the low power mode for L2 cache
  - echo 1 > /sys/module/rpm_resources/enable_low_power/L2_cache
- Disable the low power mode for L2 cache
  - echo 0 > /sys/module/rpm_resources/enable_low_power/L2_cache