Power Debug Standard Operation Procedure

User Guide

80-NP058-1 A

June 4, 2014

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

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

Contents

Figures

Tables

Revision history

Revision	Date	Description
A	June 2014	Initial release

Note: There is no Rev. I, O, Q, S, X, or Z per Mil. standards.

1 Introduction

1.1 Purpose

This document is to introduce how to debug QUALCOMM B-family (8974/8x26/8x10) platform power issues.

1.2 Scope

This document is for OEM engineers who are working for power debugging and optimization. It helps to narrow down the power problem root causes and optimize the power consumption in different scenarios.

1.3 Conventions

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

Code variables appear in angle brackets, e.g., `<number>`.

Commands to be entered appear in a different font, e.g., **`copy a:*.* b:`**.

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

If you are viewing this document using a color monitor, or if you print this document to a color printer, red typeface indicates data types, blue typeface indicates attributes, and green typeface indicates system attributes.

Parameter types are indicated by arrows:

→	Designates an input parameter
←	Designates an output parameter
↔	Designates a parameter used for both input and output

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

1.4 References

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 Reference documents that are no longer applicable are deleted from this table; therefore, reference numbers may not be sequential.

Table 1-1 Reference documents and standards

Ref.	Document	
Qualcomm Technologies		
Q1	*MSM8x26 LA Power Thermal Mgmt Overview*	80-ND928-35
Q2	*MSM8x26 System Power Overview*	80-ND928-51
Q3	*MSM8x26 Clock Plan*	80-NF030-1
Q4	*Power Consumption Measurement MSM Android MDM Devices*	80-N6837-1
Q5	*MSM8974 Power Debugging*	80-NA157-68
Q6	*MSM8974 Android Power Thermal Mgmt*	80-NA157-25
Q7	*Resource Power Manager Debug Overview*	80-VP169-1
Q8	*MSM8974 Dynamic Sleep Overview*	80-NA157-48
Q9	*MPSS Debug MCPM Overview*	80-NH879-1
Q10	*MSM8974 Dynamic Sleep Overview.pdf*	80-NA157-48

1.5 Technical assistance

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

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

1.6 Acronyms

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

2 Power Problem Code

To get help from QUALCOMM for power related problems, create the case in the salesforce and select the following problem code:

- Software->BSP-Other Proc->System Power
- Software->BSP-Apps Proc->Power Mgmt
- Software->Multimedia->Power
- Software->BSP-Apps Proc->Thermal Mitigation

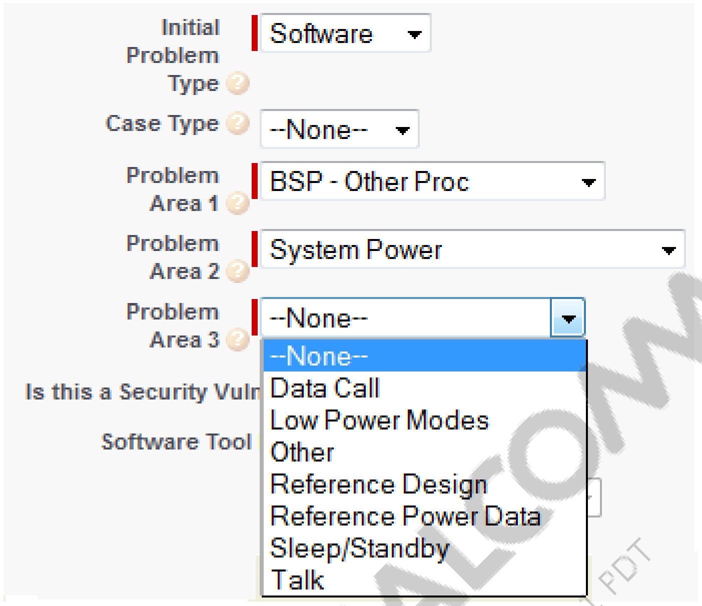
For the case, provide clear and detailed information about your problems and also provide the following items:

- Build/chipset info
- Problem reproduce environment and setup
- Problem reproduce steps
- Memory dump or `rpm/modem/kernel` log
- Current waveform and data
- Some OEM design changes to QUALCOMM reference design in HW
- Some SW changes from OEMs including UI/driver/third party APK or solutions

If more detailed information is provided for the above item, QUALCOMM engineers' response time will be less to help to fix the problems.

2.1 Software→BSP-Other Proc→System Power

This problem is mainly related to rpm and modem power features.



Initial Problem Type: Software

Case Type: --None--

Problem Area 1: BSP - Other Proc

Problem Area 2: System Power

Problem Area 3: --None--

Figure 2-1 Non-AP processors power problem code

- Data call:

  Use it for data relevant power issues, including data upload and download current problems, abnormal wakeup by QMI message.

- Lower power modes:

  Use it for the device entering into low power mode such as rpm vdd min and xo shutdown.

- Reference design:

  Use it to get reference design in QUALCOMM build.

- Reference design power data

  Use it to get Dashboard data from QUALCOMM.

- Sleep/Standby:

  Use it for the following problems:

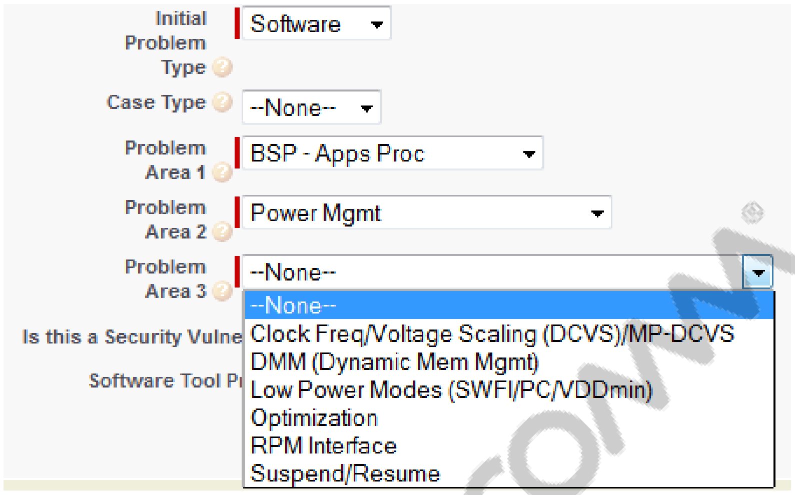
  - Modem can't sleep
  - System can't enter into standby
  - Big standby current

- Talk

  Use it for talk current problems in different networks.

2.2 Software→BSP-Apps Proc→Power Mgmt

This problem is mainly related to android power problems.



Initial Problem Type	Software
Case Type	--None--
Problem Area 1	BSP - Apps Proc
Problem Area 2	Power Mgmt
Problem Area 3	--None--

Figure 2-2 Android power problem code

- Clock Freq/Voltage Scaling (DCVS)/MP-DCVS

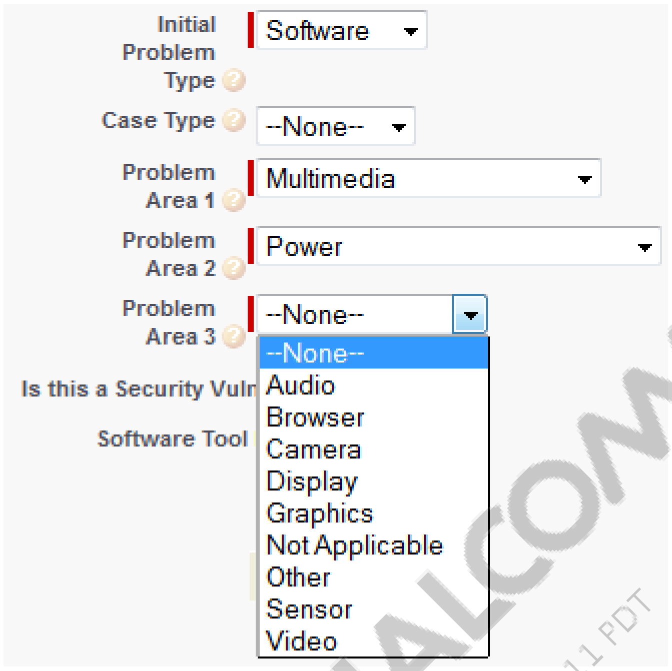
  Use it for the following problems.

  - Cpufreq
  - Mp-decision
  - Acpu clock
  - Device clock
- DMM (Dynamic Mem Mgmt)

  Mem freq current, should be rpm side.
- Low power modes (SWFI/PC/VDDmin)
  - L2 cache
  - Gdfs
  - AP C0---C3
- Optimization
  - This is for Specific user scenario current optimization such as home screen, browser, camera.audio etc.
  - Optimization relates to Operator power specs such as CMCC.
- RPM Interface

  Rpm requests related problem.
- Suspend/Resume
  - AP can't suspend.
  - Abnormal wakeup causes AP to resume from sleep.

2.3 Software→Multimedia→Power



Initial Problem Type: Software

Case Type: --None--

Problem Area 1: Multimedia

Problem Area 2: Power

Problem Area 3: --None--

Figure 2-3 MM power problem code

- Audio
  - Big mp3 playing current
  - High talk current
- Browser

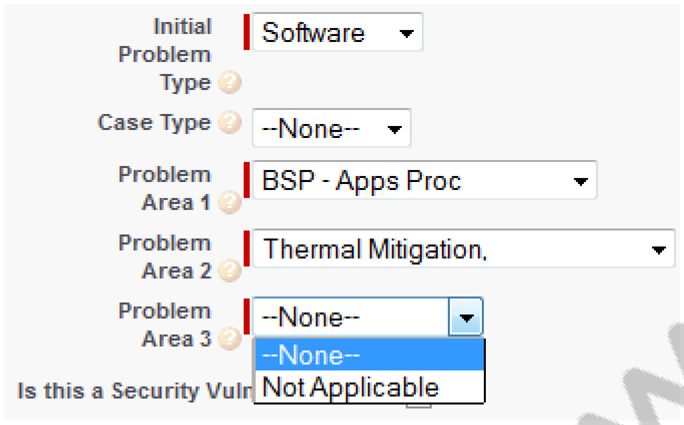
  High browser current
- Camera

  High cameral current in different camera modes and fps/resolution setting
- Display

  High current in specific scenario caused by LCD
- Graphics

  High current caused by GPU high freq

2.4 Software→BSP-Apps Proc→Thermal Mitigation



Initial Problem Type	Software
Case Type	--None--
Problem Area 1	BSP - Apps Proc
Problem Area 2	Thermal Mitigation,
Problem Area 3	--None--

Figure 2-4 Thermal problem code

This is mainly for thermal related optimization and debug:

- Thermal optimization on customer device
- Thermal config file modification and test

3 Power Debug Check Procedure

3.1 Bottom current check

- Are all processors power collapsed?
  - if yes:
    - Check current waveform to see if there is big wakeup from AP.
    - Check if RF is calibrated, QCN loaded and sim card inserted.
    - Make sure USB is unplugged during test.
    - Check rpm_stats to see if rpm ever entered into vdd min and xo/no shutdown.
    - Check gpio/clk/pmic dump in rpm side
    - Check HW design of gpio/pmic and compare it with QUALCOMM reference design.
    - Compare airplane enable/disable current.
    - For live network current ,with sim card inserted, check current waveform for paging, ask protocol engineers to help to check DRX paging waveform and protocol QXDM log.
    - Check rpm log/NPA log.
  - if no:

    Check related processors, the followings are related processors which are not in sleep.
    - APP processor:
      - If app is collapsed: check AP side kernel log with related debug mask to check enabled clock before AP power is power collapsed.
      - If app isn't power collapsed: check wakesources to see which wake sources blocks AP sleep.
    - ADSP processor: check adsp logs, may be related to audio or sensor.

      If adsp isn't collapsed: check adsp logs and android side audio/sensor log for details.
    - Modem: check modem MCPM log/sleep ulog/f3 log/NPA log

      If modem is not power collapsed: check modem side MCPM log to see which modem client vote against modem in sleep.
    - WCNSS:

      If not collapsed: check wcnss log and android kernel log for Wifi/Gps/BT/NFC.

3.2 Wakeup problem debug flow

- If AP waked up
  - Check current wave form to see if there is long and high current
  - Enable kernel debugmask for resume irq to check wakeup irq in android side
  - Check /proc/interrupts to idenfify the interrupt function

Possible wakeup reasons are as following:

- SPMI interrupt

  This is the mostly wakeup from RPM side, wakeup reason can be the followings:
  - Alarm
  - Power key
  - Battery ADC interrupt
- SMD
  - Modem

    Qmi: wake up reason may relate to efs sync/network event/short messeage etc.

3.3 Can’t suspend problem

- Check rpm_stats to see if rpm ever entered into vdd_min
  - If yes: check wakeup reason, follow instructions at section 3.2.
  - If no:
    - APP processor:
      - If app is power collapsed: need to enable clk debug mask and get kernel log for checking. We can check enabled clock count from kernel log before AP is power collapsed. Ccompare enabled clock count with normal count to judge if APP power collapsed or not.
      - If app isn’t collapsed: check wakesources to see which wake sources block.
    - ADSP processor: check adsp logs, may be related to audio or sensor.

      If adsp isn’t collapsed: check adsp logs and android side audio/sensor log for details.
    - Modem: need to check modem MCPM log/sleep ulog/f3 log/NPA log

      If modem is not power collapsed: check MCPM log to see which client vote against modem sleep.
    - WCNSS:

      If not collapsed: check wcnss log and android kernel log for Wifi/Gps/BT/NFC.

3.4 Live network standby current is high

If bottom current is OK, and live network standby current is high, usually you have to check from protocol side, the followings are some items to check:

- RF calibrated or not
- QCN loaded or not
- If RF PA is good or not, create RF case to get help for RF engineer.
- Signal strength during test
- QXDM logs
- Current wave form to check the DRX paging waveform.
- Protocol related setting such as paging interval is different for different protocols.
- Neighbor cells during test
- Call box current data to compare

4 Power Debug Skills

4.1 RPM

4.1.1 How to check if RPM enters into Vdd_min

4.1.1.1 From RPM log

The following red message shows the count of entering into Vdd_min and to judge if the device enters into Vdd_min or not. If count=0, it means the device never get into Vdd_min, if count>0, it means the device ever enter into Vdd_min.

```
0x0000000105D215ED:   Clock: gcc_dehr_clk Requested State = Disable. Actual
State = 0
0x0000000105D2162E:   Clock: gcc_bimc_clk Requested State = Disable. Actual
State = 0
0x0000000105D2164F:   Clock BIMC Collapse: DEHR done
0x0000000105D216AC:   rpm_resource_settling_complete (master: "MSS SW")
(resource type: clk2) (resource id: 0) (full name: bmic)
0x0000000105D216DC:   rpm_master_set_transition_complete (master: "MSS SW")
(deadline: 0x0000000000000000) (exceeded: no)
0x0000000105D21F36:   rpm_transition_queued (master: "MSS SW") (scheduled:
"yes") (deadline: 0x0000000106722191)
0x0000000105D21F7B:   rpm_estimate_cache_hit (estimate: 0x00008C12)
0x0000000105D2201A:   rpm_estimate_cache_hit (estimate: 0x00008C12)
0x0000000105D23EB3:   deep_sleep_enter: (mode: "VDD Minimization") (count:
205)
0x0000000105D23EDE:   Clock Processor Collapse: Enter
0x0000000105D23EFE:   Clock Processor Collapse: Done
0x0000000105D2498C:   Clock Processor Collapse: Enter
0x0000000105D24C8B:   Clock PLL: GPLL0 Status = Disable
0x0000000105D24E14:   Clock: gcc_rpm_proc_fclk Frequency = 019MHz
0x0000000105D24F83:   Clock Processor Collapse: Done
0x0000000105D250A2:   deep_sleep_enter_complete: (mode: "VDD Minimization")
```

4.1.1.2 From kernel rpm statics

Use the following command to check rpm lower power mode count:

```
cat /sys/kernel/debug/rpm_stats
```

The result is as below, if the count for vmin is **0**, it means the device never enters into vdd min. non-zero means the device ever enters into vdd_min.
RPM Mode: xosd

```
count:0
time in last mode(msec):0
time since last mode(sec):794
actual last sleep(msec):0
RPM Mode:vmin
count:11
time in last mode(msec):0
time since last mode(sec):359
actual last sleep(msec):110000
```

4.1.2 Power collapsed cores check

From Jtag or memory dump

On B family platform, the previous debug variable gpRPMFWMaster is replaced by rpm/rpm.ees. It provides most part of the debug information which is provided by gpRPMFWMaster in A family platform.

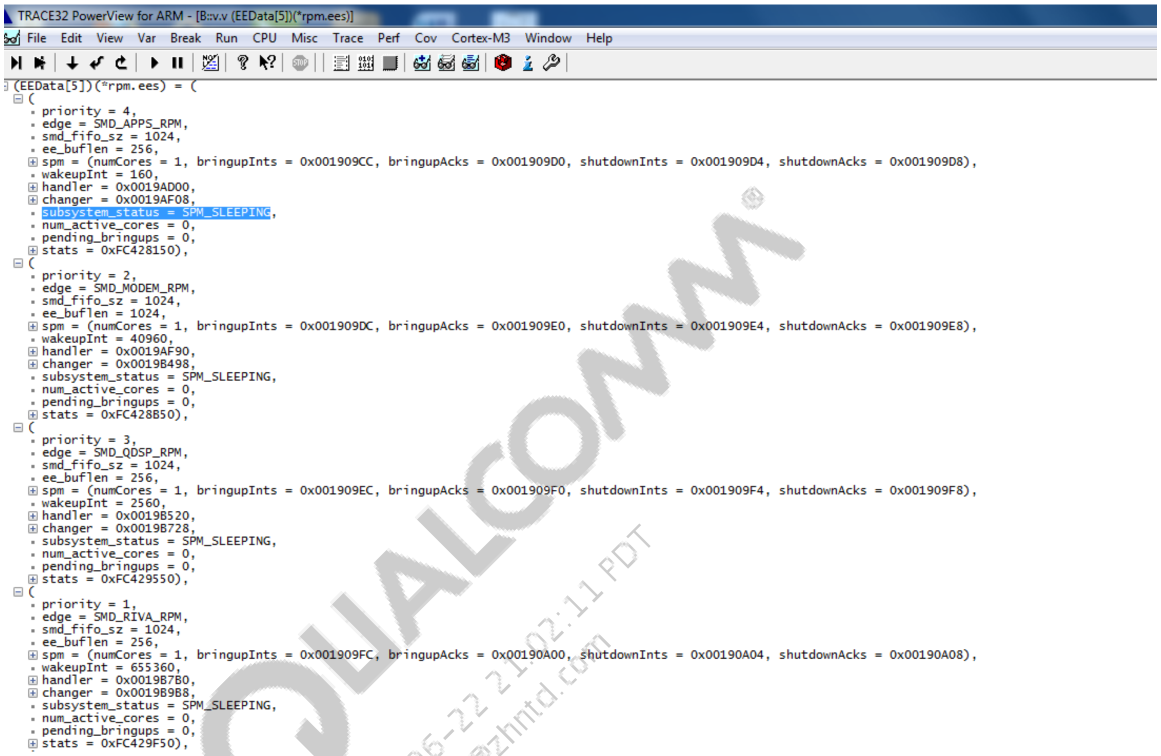
After rpm dump/elf loaded into T32 simulator, OEM can check it by typing the following commands in T32 command line.

```
v.v rpm
v.v (EEData[5])(*rpm.ees)
```

The following variables are important to check each subsystem.

- priority: indicate a specific subsystem’s priority when RPM is dealing multi request from different subsystems
- subsystem_status: indicate the subsystem is in active/sleep status.

- num_active_cores: indicate how much cores are active inside a specific subsystem, if the subsystem is active. Usually this is meaningful for APP subsystem.


TRACE32 PowerView for ARM - [B::v.v (EEData[5])(*rpm.ees)]
File Edit View Var Break Run CPU Misc Trace Perf Cov Cortex-M3 Window Help

```
(EEData[5])(*rpm.ees) = (
  (
    priority = 4,
    edge = SMD_APPS_RPM,
    smd_fifo_sz = 1024,
    ee_buflen = 256,
    spm = (numCores = 1, bringupInts = 0x001909CC, bringupAcks = 0x001909D0, shutdownInts = 0x001909D4, shutdownAcks = 0x001909D8),
    wakeupInt = 160,
    handler = 0x0019AD00,
    changer = 0x0019AF08,
    subsystem_status = SPM_SLEEPING,
    num_active_cores = 0,
    pending_bringups = 0,
    stats = 0xFC428150),
  (
    priority = 2,
    edge = SMD_MODEM_RPM,
    smd_fifo_sz = 1024,
    ee_buflen = 1024,
    spm = (numCores = 1, bringupInts = 0x001909DC, bringupAcks = 0x001909E0, shutdownInts = 0x001909E4, shutdownAcks = 0x001909E8),
    wakeupInt = 40960,
    handler = 0x0019AF90,
    changer = 0x0019B498,
    subsystem_status = SPM_SLEEPING,
    num_active_cores = 0,
    pending_bringups = 0,
    stats = 0xFC428B50),
  (
    priority = 3,
    edge = SMD_QDSP_RPM,
    smd_fifo_sz = 1024,
    ee_buflen = 256,
    spm = (numCores = 1, bringupInts = 0x001909EC, bringupAcks = 0x001909F0, shutdownInts = 0x001909F4, shutdownAcks = 0x001909F8),
    wakeupInt = 2560,
    handler = 0x0019B520,
    changer = 0x0019B728,
    subsystem_status = SPM_SLEEPING,
    num_active_cores = 0,
    pending_bringups = 0,
    stats = 0xFC429550),
  (
    priority = 1,
    edge = SMD_RIVA_RPM,
    smd_fifo_sz = 1024,
    ee_buflen = 256,
    spm = (numCores = 1, bringupInts = 0x001909FC, bringupAcks = 0x00190A00, shutdownInts = 0x00190A04, shutdownAcks = 0x00190A08),
    wakeupInt = 655360,
    handler = 0x0019B7B0,
    changer = 0x0019B9B8,
    subsystem_status = SPM_SLEEPING,
    num_active_cores = 0,
    pending_bringups = 0,
    stats = 0xFC429F50),
```

Figure 4-1 EEData check

If we have memory dump, we have two methods to check EEData.

- Check from memory dump in Trace32 Simulator

  The method to check from memory dump is the same as checking with Jtag, the only difference is that you have to load memory dump in T32 simulator first.

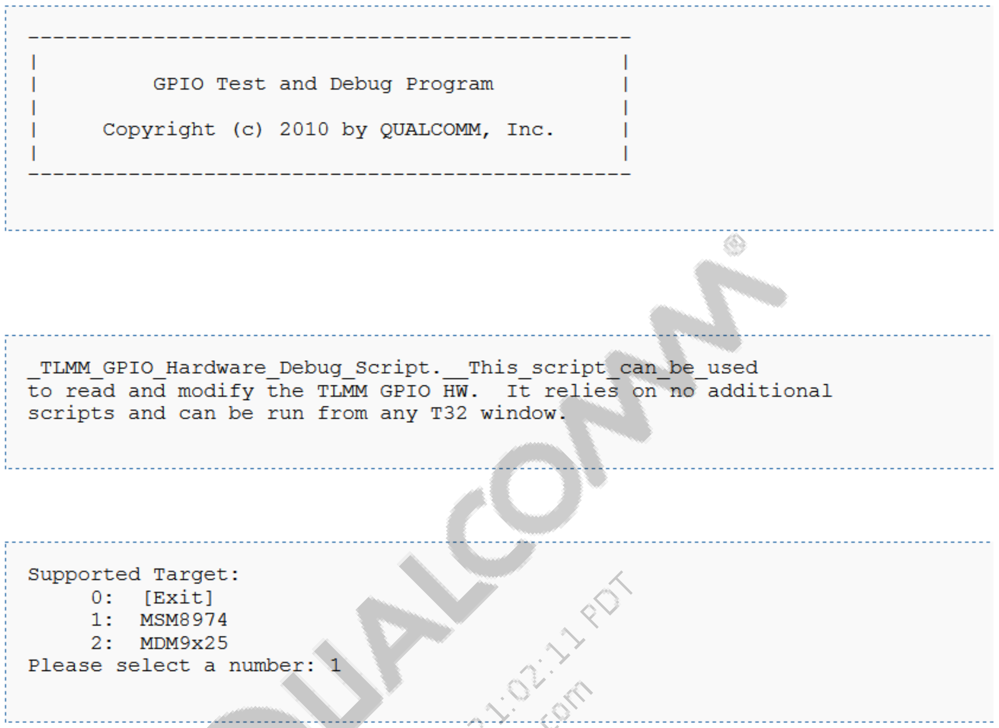
- Check from memory dump with hensei script.

4.1.3 gpio dump

Follow procedures below to get gpio dump:

1. Attach RPM T32, load rpm elf.
2. Set breakpoint at pm_npa_rpm_enter_sleep, and mpm_sw_done.
3. Put phone to sleep , it will reach the first breakpoint.
4. Disable the pmic watchdog, select `'CoreBSP Scripts' -> 'Systemdrivers' -> 'PMIC Peek/Poke'` from the T32 menu bar.
5. Go, and until code stops at the second breakpoint
6. Run `\modem_proc\core\systemdrivers\tlmm\scripts\tlmm_gpio_hw.cmm` to dump GPIO status.

NOTE: Make sure that T32 software is updated to the latest, 2013 and later.

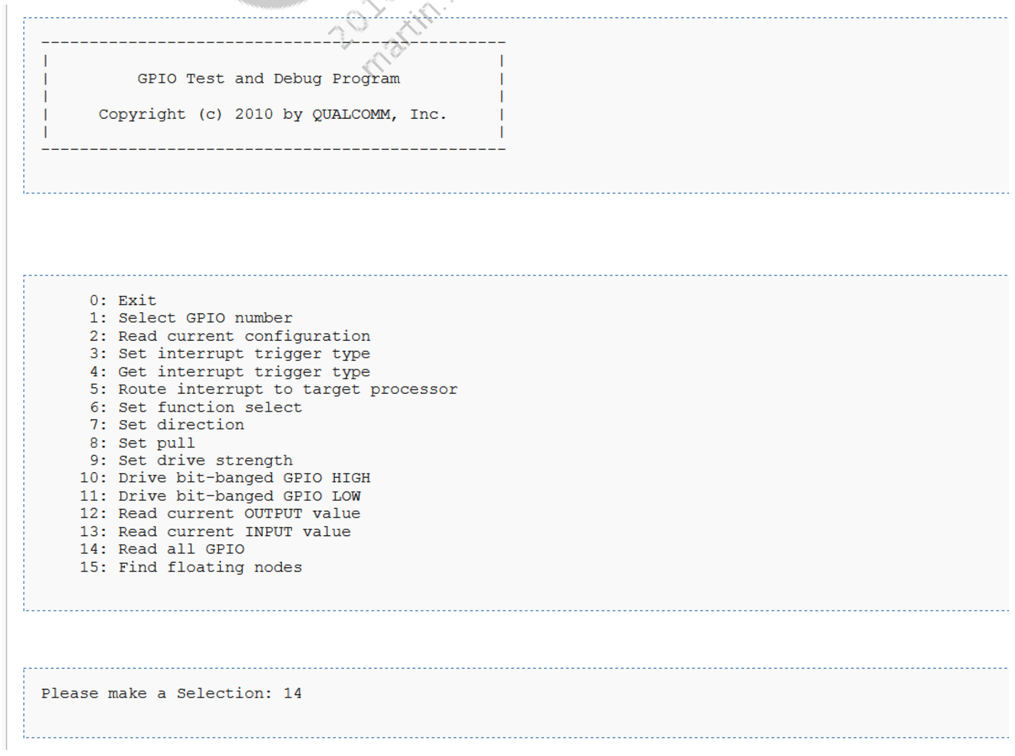


```
------------------------------------------------
|                                              |
|         GPIO Test and Debug Program          |
|                                              |
|     Copyright (c) 2010 by QUALCOMM, Inc.     |
|                                              |
------------------------------------------------
```

```
_TLMM_GPIO_Hardware_Debug_Script.__This_script_can_be_used
to read and modify the TLMM GPIO HW.  It relies on no additional
scripts and can be run from any T32 window.
```

```
Supported Target:
     0:  [Exit]
     1:  MSM8974
     2:  MDM9x25
Please select a number: 1
```

Figure 4-2 gpio dump 1



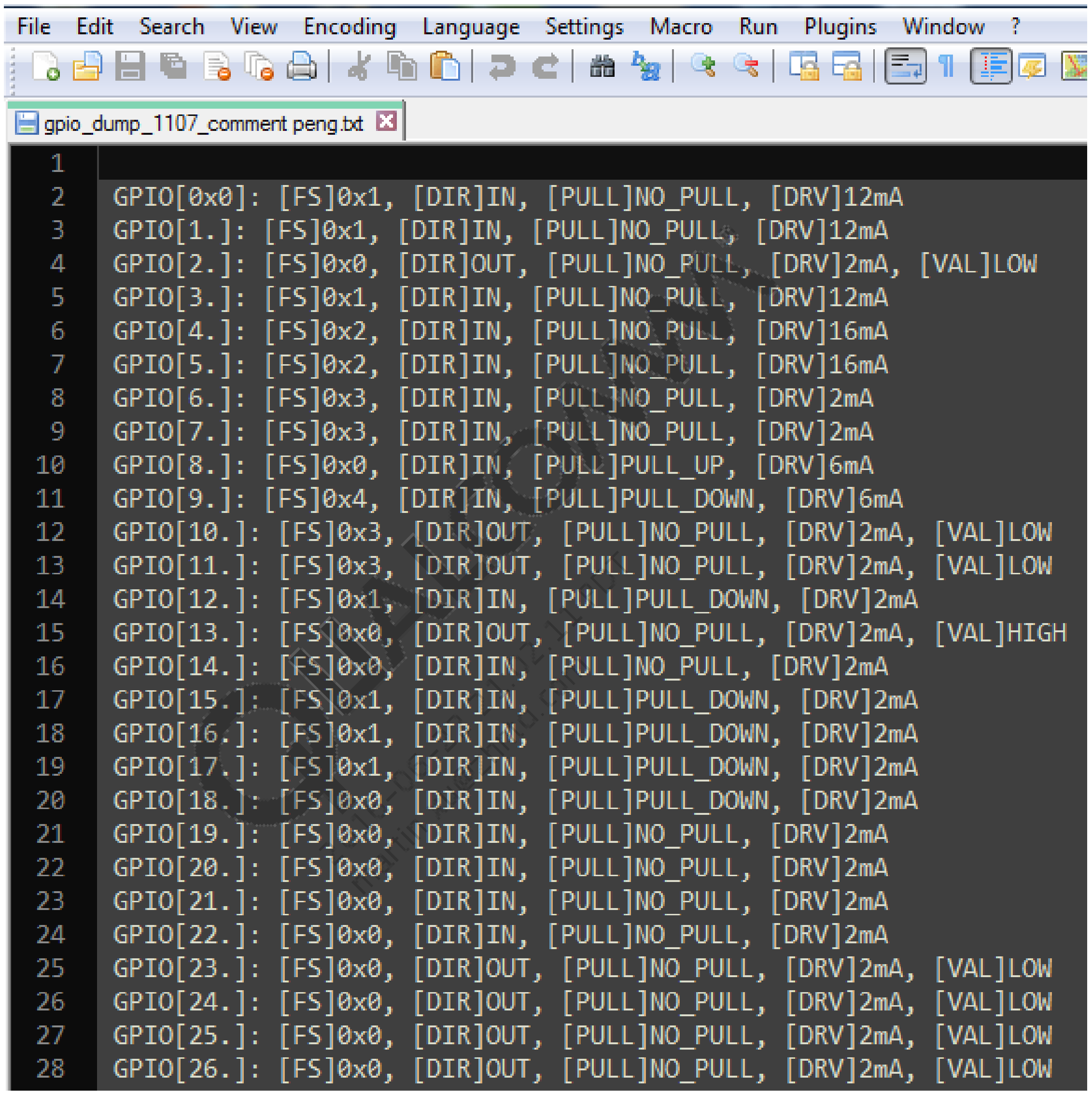
```
------------------------------------------------
|                                              |
|         GPIO Test and Debug Program          |
|                                              |
|     Copyright (c) 2010 by QUALCOMM, Inc.     |
|                                              |
------------------------------------------------
```

```
     0: Exit
     1: Select GPIO number
     2: Read current configuration
     3: Set interrupt trigger type
     4: Get interrupt trigger type
     5: Route interrupt to target processor
     6: Set function select
     7: Set direction
     8: Set pull
     9: Set drive strength
    10: Drive bit-banged GPIO HIGH
    11: Drive bit-banged GPIO LOW
    12: Read current OUTPUT value
    13: Read current INPUT value
    14: Read all GPIO
    15: Find floating nodes
```

```
Please make a Selection: 14
```

Figure 4-3 gpio dump 2

After previous steps, you can get a GPIO dump file which is similar as following, OEMs have to check the GPIO status/setting with those principles listed in next page


File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
gpio_dump_1107_comment peng.txt

```
GPIO[0x0]: [FS]0x1, [DIR]IN, [PULL]NO_PULL, [DRV]12mA
GPIO[1.]: [FS]0x1, [DIR]IN, [PULL]NO_PULL, [DRV]12mA
GPIO[2.]: [FS]0x0, [DIR]OUT, [PULL]NO_PULL, [DRV]2mA, [VAL]LOW
GPIO[3.]: [FS]0x1, [DIR]IN, [PULL]NO_PULL, [DRV]12mA
GPIO[4.]: [FS]0x2, [DIR]IN, [PULL]NO_PULL, [DRV]16mA
GPIO[5.]: [FS]0x2, [DIR]IN, [PULL]NO_PULL, [DRV]16mA
GPIO[6.]: [FS]0x3, [DIR]IN, [PULL]NO_PULL, [DRV]2mA
GPIO[7.]: [FS]0x3, [DIR]IN, [PULL]NO_PULL, [DRV]2mA
GPIO[8.]: [FS]0x0, [DIR]IN, [PULL]PULL_UP, [DRV]6mA
GPIO[9.]: [FS]0x4, [DIR]IN, [PULL]PULL_DOWN, [DRV]6mA
GPIO[10.]: [FS]0x3, [DIR]OUT, [PULL]NO_PULL, [DRV]2mA, [VAL]LOW
GPIO[11.]: [FS]0x3, [DIR]OUT, [PULL]NO_PULL, [DRV]2mA, [VAL]LOW
GPIO[12.]: [FS]0x1, [DIR]IN, [PULL]PULL_DOWN, [DRV]2mA
GPIO[13.]: [FS]0x0, [DIR]OUT, [PULL]NO_PULL, [DRV]2mA, [VAL]HIGH
GPIO[14.]: [FS]0x0, [DIR]IN, [PULL]NO_PULL, [DRV]2mA
GPIO[15.]: [FS]0x1, [DIR]IN, [PULL]PULL_DOWN, [DRV]2mA
GPIO[16.]: [FS]0x1, [DIR]IN, [PULL]PULL_DOWN, [DRV]2mA
GPIO[17.]: [FS]0x1, [DIR]IN, [PULL]PULL_DOWN, [DRV]2mA
GPIO[18.]: [FS]0x0, [DIR]IN, [PULL]PULL_DOWN, [DRV]2mA
GPIO[19.]: [FS]0x0, [DIR]IN, [PULL]NO_PULL, [DRV]2mA
GPIO[20.]: [FS]0x0, [DIR]IN, [PULL]NO_PULL, [DRV]2mA
GPIO[21.]: [FS]0x0, [DIR]IN, [PULL]NO_PULL, [DRV]2mA
GPIO[22.]: [FS]0x0, [DIR]IN, [PULL]NO_PULL, [DRV]2mA
GPIO[23.]: [FS]0x0, [DIR]OUT, [PULL]NO_PULL, [DRV]2mA, [VAL]LOW
GPIO[24.]: [FS]0x0, [DIR]OUT, [PULL]NO_PULL, [DRV]2mA, [VAL]LOW
GPIO[25.]: [FS]0x0, [DIR]OUT, [PULL]NO_PULL, [DRV]2mA, [VAL]LOW
GPIO[26.]: [FS]0x0, [DIR]OUT, [PULL]NO_PULL, [DRV]2mA, [VAL]LOW
```

Figure 4-4 gpio dump 3

4.1.4 Sleep GPIO config

Usually you can change QCOM initial pins usage. During this procedure, a few GPIO may get ignored and left under a none power-saving mode, this will cause current leakage.

OEM software engineers need to dump GPIO status info by jtag/script, and then check GPIO pin (with hardware colleague) one by one, to make sure all pins are configured correctly.

Usually, the initial GPIO configuration could be found and modified in `TLMMChipset.xml` in following folder:

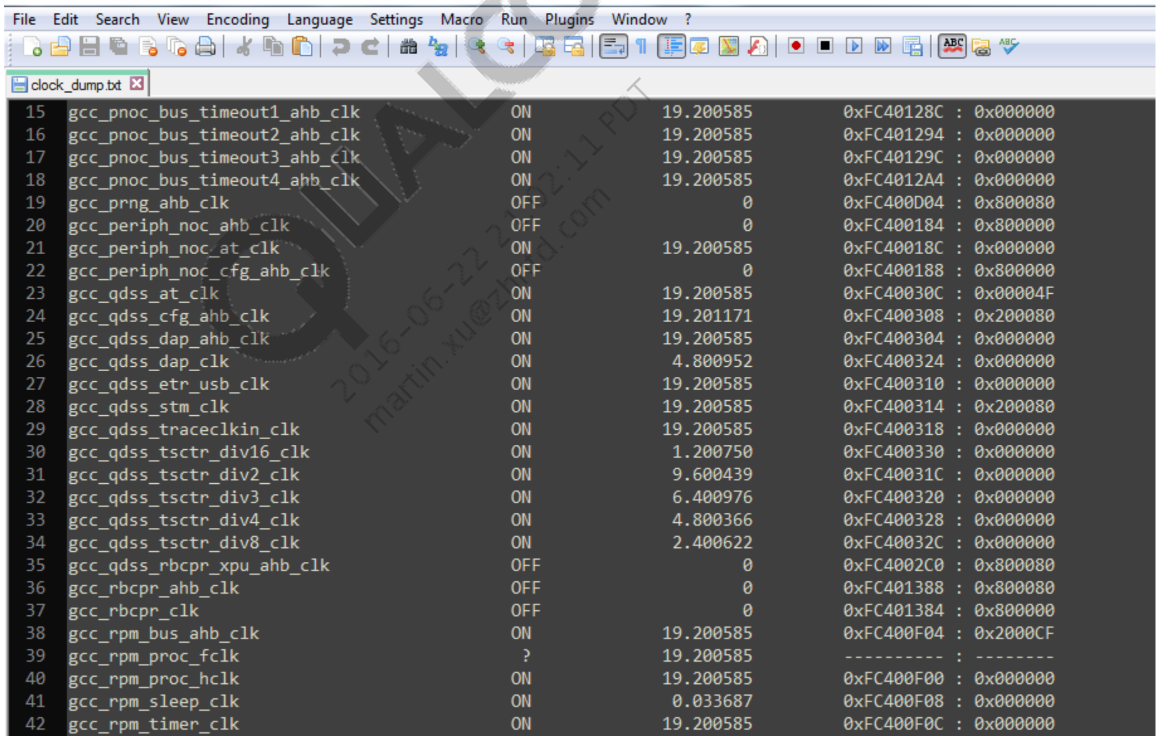
```
boot_images\core\systemdrivers\tlmm\config\msm8974
```

4.1.5 clk dump

To capture clocks dump, please follow following steps:

1. Attach RPM T32, load rpm elf.
2. Set breakpoint at pm_npa_rpm_enter_sleep, and mpm_sw_done.
3. Put phone to sleep , it will reach the first breakpoint.
4. Disable the pmic watchdog, select `'CoreBSP Scripts' -> 'Systemdrivers' -> 'PMIC Peek/Poke'` from the T32 menu bar.
5. Go , and until code stops at the second breakpoint ,
6. Run `\\boot_image\core\systemdrivers\clock\scripts\msm8974\testclock.cmm` to dump clock.

NOTE: Make sure T32 software updated to the latest, 2013 and later.


File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
clock_dump.txt

```
gcc_pnoc_bus_timeout1_ahb_clk    ON     19.200585    0xFC40128C : 0x000000
gcc_pnoc_bus_timeout2_ahb_clk    ON     19.200585    0xFC401294 : 0x000000
gcc_pnoc_bus_timeout3_ahb_clk    ON     19.200585    0xFC40129C : 0x000000
gcc_pnoc_bus_timeout4_ahb_clk    ON     19.200585    0xFC4012A4 : 0x000000
gcc_prng_ahb_clk                 OFF            0    0xFC400D04 : 0x800080
gcc_periph_noc_ahb_clk           OFF            0    0xFC400184 : 0x800000
gcc_periph_noc_at_clk            ON     19.200585    0xFC40018C : 0x000000
gcc_periph_noc_cfg_ahb_clk       OFF            0    0xFC400188 : 0x800000
gcc_qdss_at_clk                  ON     19.200585    0xFC40030C : 0x00004F
gcc_qdss_cfg_ahb_clk             ON     19.201171    0xFC400308 : 0x200080
gcc_qdss_dap_ahb_clk             ON     19.200585    0xFC400304 : 0x000000
gcc_qdss_dap_clk                 ON      4.800952    0xFC400324 : 0x000000
gcc_qdss_etr_usb_clk             ON     19.200585    0xFC400310 : 0x000000
gcc_qdss_stm_clk                 ON     19.200585    0xFC400314 : 0x200080
gcc_qdss_traceclkin_clk          ON     19.200585    0xFC400318 : 0x000000
gcc_qdss_tsctr_div16_clk         ON      1.200750    0xFC400330 : 0x000000
gcc_qdss_tsctr_div2_clk          ON      9.600439    0xFC40031C : 0x000000
gcc_qdss_tsctr_div3_clk          ON      6.400976    0xFC400320 : 0x000000
gcc_qdss_tsctr_div4_clk          ON      4.800366    0xFC400328 : 0x000000
gcc_qdss_tsctr_div8_clk          ON      2.400622    0xFC40032C : 0x000000
gcc_qdss_rbcpr_xpu_ahb_clk       OFF            0    0xFC4002C0 : 0x800080
gcc_rbcpr_ahb_clk                OFF            0    0xFC401388 : 0x800080
gcc_rbcpr_clk                    OFF            0    0xFC401384 : 0x800000
gcc_rpm_bus_ahb_clk              ON     19.200585    0xFC400F04 : 0x2000CF
gcc_rpm_proc_fclk                 ?     19.200585    ---------- : --------
gcc_rpm_proc_hclk                ON     19.200585    0xFC400F00 : 0x000000
gcc_rpm_sleep_clk                ON      0.033687    0xFC400F08 : 0x000000
gcc_rpm_timer_clk                ON     19.200585    0xFC400F0C : 0x000000
```

Figure 4-5 clock dump

Sometimes low-level clocks are prohibited from closing, due to one or a few clocks which is sourced from (or depended on) configured incorrectly before subsystems are getting into power collapse.

This will eventually affect the device getting into optimal sleep mode, and cause big bottom current.

Once OEMs get the clock dump, cross-compare against a normal clock dump, to find out the suspicious clocks. Then check the corresponding driver code which is manipulated these clocks.

4.1.6 pmic dump

To capture PMIC dump:

1. Attach RPM T32, load rpm elf.
2. Set breakpoint at pm_npa_rpm_enter_sleep.
3. Put phone to sleep , it will reach the first breakpoint..
4. Disable the pmic watchdog, select '`CoreBSP Scripts`' -> '`Systemdrivers`' -> '`PMIC Peek/Poke`' from the T32 menu bar.
5. Dump pmic registers/status, select '`CoreBSP Scripts`' -> '`Systemdrivers`' -> '`PMIC Dump`' from the T32 menu bar.
6. After pmic dump is captured, check it together (with reference to pmic spec document and hardware design), to locate those ldo/power rail settings to see if they are configured correctly in sleep.

To check pmic thoroughly, pmic dump is a must, especially when you are working on small leak current.

4.1.7 RPM Dump Load/Parse

4.1.7.1 Rpm Dump Save

To debug power issue, one key step is to capture the dump which could represent the reported issue. Depend on the problem scenarios, you have following method to choose:

- PS_HOLD
  - It forces the device to reset by pulling PS_HOLD to ground NO MORE THAN 200ms. It will make the board reset into download mode, then collect the whole memory dumps file through QPST, memory debug tool.
  - Applicable to all scenarios
  - No debug info is lost during the reset, since no cache contents are lost(for rpm subsystems only).
- ERR_FETAL
  - It could add ERR_FETAL in rpm code, once get executed, it will also get rpm/whole chipset reset.
  - Applicable to all scenarios
  - No debug info is lost for all the subsystems inside the chipset.
- JTAG
  - When debugging none deep-sleeping related issue, it uses jtag to break the rpm processor and get the memory dumps manually or by script.
  - Not applicable for deep sleep (vdd_min/xo_shutdown) scenarios.
  - No debug info is lost for all the subsystems inside the chipset.

NOTE: For memory dump provided to QUALCOMM, make sure the elf and vmlinux match the memory dump.

4.1.7.2 RPM DUMP Loading

To load the RPM dumps into a T32 simulator:

1. Open a T32 simulator and do sys.up.
2. Use load.cmm or manually load as belows:

```
d.load.binary CODERAM.BIN 0xfc100000   0x100000
d.load.binary DATARAM.BIN  0xfc190000   0x190000
d.load.binary MSGRAM.BIN   0xfc428000
d.load.elf  \\elf_file_path\RPM.elf /nocode
```

Once these steps are done (with the matching elf and dump), start to parse/analyze rpm logs (rpm log/npa log) and check debug variables.

4.1.7.3 RPM DUMP Parse

You can extract ulog and NPA log with following methods:

- Extract a RPM external log in Trace32 simulator

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

- Extract an NPA log in Trace32 simulator

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

You can translate rpm log into readable format with following command:

```
python rpm_proc\core\power\rpm\debug\scripts\rpm_log.py -f “RPM External
Log.ulog” -n “NPA Log.ulog” > rpm_parsed.txt
```

NOTE: Parameter **–r**, which prints raw (hex sclk value) timestamps.

4.1.7.4 Hansei

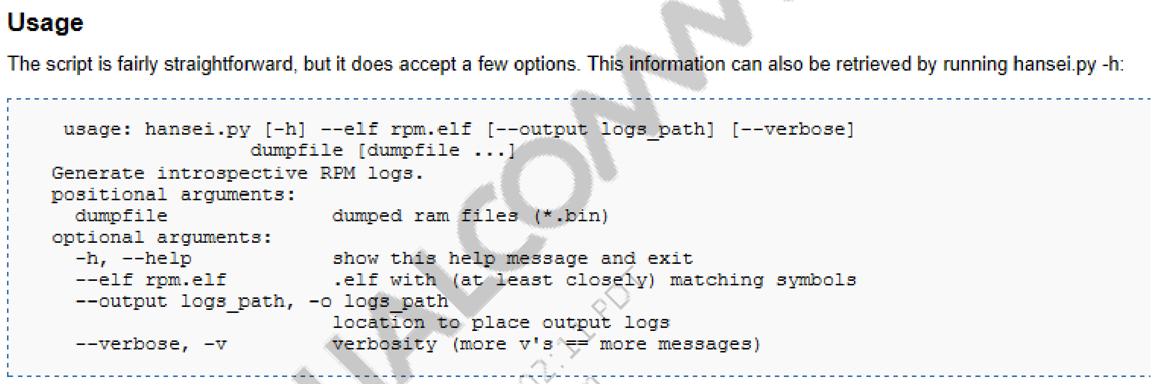
Due to Messaging/SMD redesign, you can’t gather processor sleep/active info on B-family which is usually stored in gpRPMFWMasters in A-family.

Since the info is important to debug system-level sleep issues, a new tool Hansei is introduced to parse resources/NPA info.

- Prerequisites
  - Must have a 2.7.x version of Python installed. 2.7.3 is verified OK by QCOM.
  - Must install a version of the pyelftools library that supports the ARM compiler tools.

    Such a version can be retrieved from https://bitbucket.org/pplesnar/pyelftools-pp
  - Once the pyelftools source code is extracted, install it by running command prompt, changing to the source directory, and run "`python setup.py install`“
- Location

  The hansei.py script resides on \\rpm_porc\core\bsp\rpm\scripts\hansei\hansei.py.

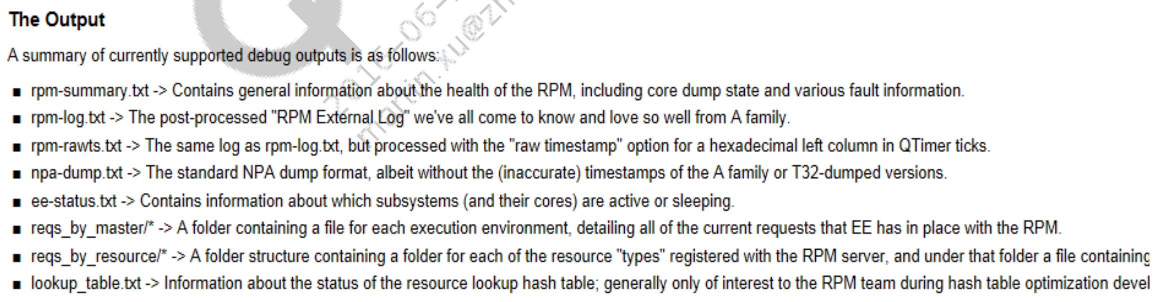


Usage

The script is fairly straightforward, but it does accept a few options. This information can also be retrieved by running hansei.py -h:

```
 usage: hansei.py [-h] --elf rpm.elf [--output logs_path] [--verbose]
                 dumpfile [dumpfile ...]
Generate introspective RPM logs.
positional arguments:
  dumpfile              dumped ram files (*.bin)
optional arguments:
  -h, --help            show this help message and exit
  --elf rpm.elf         .elf with (at least closely) matching symbols
  --output logs_path, -o logs_path
                        location to place output logs
  --verbose, -v         verbosity (more v's == more messages)
```

Figure 4-6 hansei usage



The Output

A summary of currently supported debug outputs is as follows:

- rpm-summary.txt -> Contains general information about the health of the RPM, including core dump state and various fault information.
- rpm-log.txt -> The post-processed "RPM External Log" we've all come to know and love so well from A family.
- rpm-rawts.txt -> The same log as rpm-log.txt, but processed with the "raw timestamp" option for a hexadecimal left column in QTimer ticks.
- npa-dump.txt -> The standard NPA dump format, albeit without the (inaccurate) timestamps of the A family or T32-dumped versions.
- 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 of the 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
- lookup_table.txt -> Information about the status of the resource lookup hash table; generally only of interest to the RPM team during hash table optimization devel

Figure 4-7 hansei output

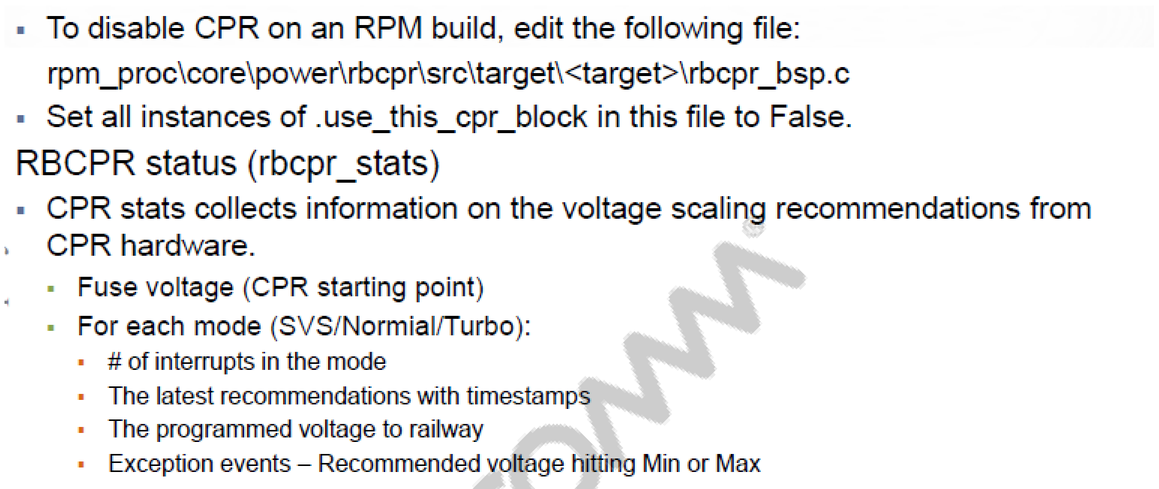
4.1.8 RPM power feature enable /disable

4.1.8.1 How to stop system from deep sleep mode

To stop the device from getting into deep sleep (xo_shutdown/vdd_min), you can hardcode variable sleep_allow_low_power_modes to FALSE in rpm code, or modify it through jtag during run-time debugging.

4.1.8.2 How to disable RBCPR

To identify if the issue is RBCPR relevant, disabling the feature is the quickest way.



- To disable CPR on an RPM build, edit the following file:
  rpm_proc\core\power\rbcpr\src\target\<target>\rbcpr_bsp.c
- Set all instances of .use_this_cpr_block in this file to False.

RBCPR status (rbcpr_stats)

- CPR stats collects information on the voltage scaling recommendations from CPR hardware.
  - Fuse voltage (CPR starting point)
  - For each mode (SVS/Normial/Turbo):
    - # of interrupts in the mode
    - The latest recommendations with timestamps
    - The programmed voltage to railway
    - Exception events – Recommended voltage hitting Min or Max

Figure 4-8 Disable RBCPR

4.1.8.3 How to disable RPM's DCVS

The running speed of RPM is adjusted dynamically, based on its current workload. To set it to a fixed seep (fast or slow), try the following method.

```
static void svs_set_mode(unsigned int mode, bool passive)
{
  unsigned int start, end;
  mode = SVS_FAST (or SVS_FAST);
  CORE_VERIFY(mode < SVS_NUM_MODES);
…….
}
```

4.1.8.4 How to disable PMIC WATCHDOG

To debug PMIC watchdog bite problem, you can disable PMIC watchdog, this will make device hang when problem reproduces, and then OEM can use JTAG to debug.

OEM can disable PMIC watchdog with following methods:

- Modify variable pmic_wdog_enable to **0** in rpm code.
- In runtime debugging, you can disable pmic watchdogby selecting '`CoreBSP Scripts`' -> '`Systemdrivers`' -> '`PMIC Peek/Poke`' from the T32 menu bar.

4.1.8.5 How to disable DDR_TRACING

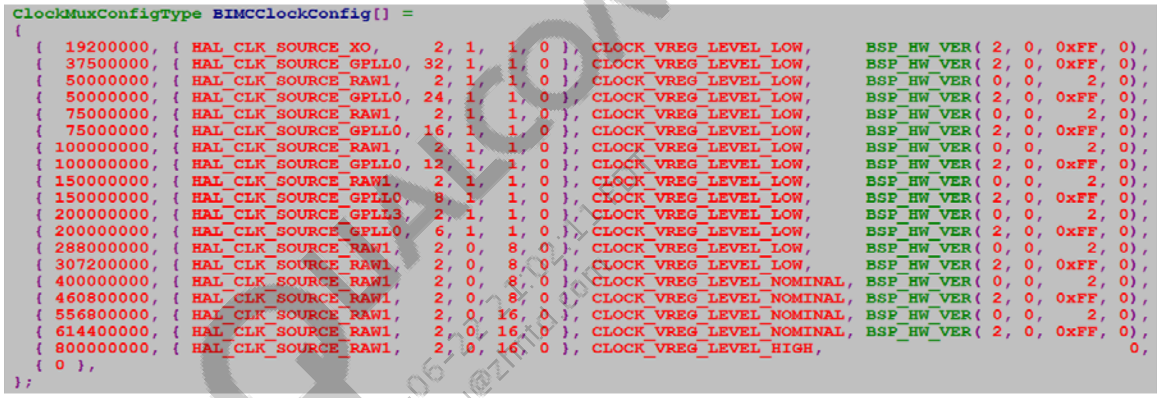
OEM could modify the Sconscript under rpm_proc\core\power\sleep\build, and comment out the following lines:

```
if env['MSM_ID'] == '8974':
env.Append(CPPDEFINES = 'MSM8974_DDR_TRACING')
```

or just comment out the codes defined under micro MSM8974_DDR_TRACING, in rpm code.

4.1.8.6 How to lower DDR running frequency

To force DDR running at lower freq, comment out those high freq setting inside the following array (in rpm code).



```
ClockMuxConfigType BIMCClockConfig[] =
{
  {  19200000, { HAL_CLK_SOURCE_XO,     2, 1,  1, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 2, 0, 0xFF, 0),
  {  37500000, { HAL_CLK_SOURCE_GPLL0, 32, 1,  1, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 2, 0, 0xFF, 0),
  {  50000000, { HAL_CLK_SOURCE_RAW1,   2, 1,  1, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 0, 0,    2, 0),
  {  50000000, { HAL_CLK_SOURCE_GPLL0, 24, 1,  1, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 2, 0, 0xFF, 0),
  {  75000000, { HAL_CLK_SOURCE_RAW1,   2, 1,  1, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 0, 0,    2, 0),
  {  75000000, { HAL_CLK_SOURCE_GPLL0, 16, 1,  1, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 2, 0, 0xFF, 0),
  { 100000000, { HAL_CLK_SOURCE_RAW1,   2, 1,  1, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 0, 0,    2, 0),
  { 100000000, { HAL_CLK_SOURCE_GPLL0, 12, 1,  1, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 2, 0, 0xFF, 0),
  { 150000000, { HAL_CLK_SOURCE_RAW1,   2, 1,  1, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 0, 0,    2, 0),
  { 150000000, { HAL_CLK_SOURCE_GPLL0,  8, 1,  1, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 2, 0, 0xFF, 0),
  { 200000000, { HAL_CLK_SOURCE_GPLL3,  2, 1,  1, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 0, 0,    2, 0),
  { 200000000, { HAL_CLK_SOURCE_GPLL0,  6, 1,  1, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 2, 0, 0xFF, 0),
  { 288000000, { HAL_CLK_SOURCE_RAW1,   2, 0,  8, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 0, 0,    2, 0),
  { 307200000, { HAL_CLK_SOURCE_RAW1,   2, 0,  8, 0 }, CLOCK_VREG_LEVEL_LOW,     BSP_HW_VER( 2, 0, 0xFF, 0),
  { 400000000, { HAL_CLK_SOURCE_RAW1,   2, 0,  8, 0 }, CLOCK_VREG_LEVEL_NOMINAL, BSP_HW_VER( 0, 0,    2, 0),
  { 460800000, { HAL_CLK_SOURCE_RAW1,   2, 0,  8, 0 }, CLOCK_VREG_LEVEL_NOMINAL, BSP_HW_VER( 2, 0, 0xFF, 0),
  { 556800000, { HAL_CLK_SOURCE_RAW1,   2, 0, 16, 0 }, CLOCK_VREG_LEVEL_NOMINAL, BSP_HW_VER( 0, 0,    2, 0),
  { 614400000, { HAL_CLK_SOURCE_RAW1,   2, 0, 16, 0 }, CLOCK_VREG_LEVEL_NOMINAL, BSP_HW_VER( 2, 0, 0xFF, 0),
  { 800000000, { HAL_CLK_SOURCE_RAW1,   2, 0, 16, 0 }, CLOCK_VREG_LEVEL_HIGH,                          0,
  { 0 },
};
```

Figure 4-9 DDR frequency table

4.2 Modem debug

For modem debug, check F3 log/Ulog log/NPA log, the followings provide methods on how to check these logs.

4.2.1 F3 log

To recover SW F3 log from Trace32 simulator, use the following command :

```
do <MPSS_build>\modem_proc\core\debugtools\err\cmm\recover_f3.cmm
```

The F3 log will be extracted at `C:\Temp\f3log.txt`

To recover FW F3 log from Trace32 simulator, use the following command :

```
do <MPSS_build>\modem_proc\modem\fw\scripts\fw_dump.cmm c:\Temp\fw_dump.xml
```

On command window, run the command below:

```
perl <MPSS_build>\modem_proc\modem\fw\scripts\fw_dump_parse.pl
c:\Temp\fw_dump.xml > c:\Temp\fw_dump.txt
```

OEM can search for keyword "MCPM" in logs to get MCPM logs, if no MCPM log, it means there is no MCPM operations logged to TRACE BUFFER in F3 time period.

The following F3 message indicates MCPM is not IDLE:

```
00:00:32.405:                      l1_drx.c:2050     DRX: MCPM not idle
```

The following message indicates WCDMA sleep/wakeup is OK and the last MCPM configuration is sleep:

```
  12:00:54.468   861    mcpm_drv.c:Matched MCPMDRV cfg API type : 4 0 0…//4
mean wcdma data in
MCPMDRV_CFG_APIType(MAPI_WCDMA_DATA_RC0dc_RC1dc_RC2dc_RC3dc_DLhs21_ULhs11_B
W5m).
 12:00:54.485   861    mcpm_drv.c:Matched MCPMDRV cfg API type : 1 0 0…//1
mean wcdma data no longer requested.
 12:00:55.100   861    mcpm_drv.c:Matched MCPMDRV cfg API type : 4 0 0…
 12:00:55.115   861    mcpm_drv.c:Matched MCPMDRV cfg API type : 1 0 0…
```

4.2.2 Ulog

To get the sleep logs from Trace32 simulator, use the following command:

```
<MPSS_build>\modem_proc\core\power\ulog\scripts\UlogDump.cmm
\\path\to\write\the\logs
```

To merge logs together, use the following command:

```
<MPSS_build>\modem_proc\core\power\sleep\scripts\ULogMerge.py
```

You can use this script to merge any ulogs together. i.e, you can merge a sleep log with the NPA log or any other sleep log.

There are 6 types of logs for Dynamic sleep:

- Sleep WarningError.ulog

  This log shows any warnings or errors that have occurred during sleep.
- Sleep Statistics.ulog

It contains number of times each Low Power Mode (LPM) has been entered and how long (in sclks) the processor stayed in that LPM.

- Sleep Info.ulog

  This is the main log that should be used for debugging. It has the most concise and helpful information.

- Sleep Profiling.ulog

  This log contains messages that are used to measure delays and timing in the system.

- Sleep Debug.ulog

  This log contains extra information and details, and it is used in addition to the info log if necessary.

- Sleep Requests.ulog

  This log contains messages that are logged while enabling or disabling any LPM.

The sleep dump is used to capture a snapshot of the sleep subsystem. It can be used to obtain the following information:

- All sleep logs in independent and merged format
- Name of the solver that's currently in use
- LPRs & LPRMs that are registered
- Enter latency, exit latency and power savings for each LPRM
- Whether each LPRM is enabled or disabled
- List of synthesized LPRMs and their component modes
- Number of times each LPRM has been entered

After the script execution finishes, there will be a file called `mergedOuput.ulog` in the output directory.

A sample sleep info Ulong is as below.

```
……
241.426360:   Entering modes (hard deadline: 0x114f923d2) (backoff
deadline: 0x114f8dad1) (backoff: 0x4901) (sleep duration: 0xae7a01)//modem
sleep entering.
241.426447:   Adjusted NPA scheduler deadline (old NPA deadline:
0x114f92484) (new NPA deadline: 0x11502b2ad) (timer deadline: 0x11502c1eb)
(offset: 0x98e29)
241.426584:   Sleep set sent (wakeup time requested: 0x1150268fa)
241.426645:   Program QTMR (match tick: 0x11502667c)
242.055382:   Exiting modes
242.055557:   Wakeup (reason: Scheduled) (interrupts pending: 242) (backoff
deadline: 0x1150268fa) (sleep exit deadline: 0x11502b1fb)     //it waked up
by interrupt 242.
```

```
242.055610:   Master Wakeup (Actual: 0x11502758b) (Expected: 0x1150268fa)
(Error: 3217)//waked up earliar then scheduled by interrupt.
```

4.2.3 NPA log

Ulogs show requests for NPA nodes in the past, NPA resource logs show current request status from npa clients.

1. Use the following command, OEM can recover modem Ulogs from memory dump in Trace32 simulator :

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

2. Use Extract modem NPA log: in Trace32 simulator with the following command:

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

   Scripts will place the modem logs in the specified directory. If need Ulogs in longer period, it needs to increase the NPA Log Size with T32 or change code directly.

3. After booting, but before running, set the variable **npa_config_data.log_buffer_size = 131072**.
4. Run as normal. The NPA log will be allocated at 128k instead of 64k. Here can choose any size that is a power of 2.

The log buffer is heap-allocated, so there's a limit to how large the buffer can be without causing system problems and the heap usage can differs from target to target.

Following sample ULOGs show NPA requests, it shows resources "**/core/cpu**“, "**/core/cpu/latency**“, "**/sleep/idle/plugin**” requested, some is complete/released afterwards.

```
0x00000001952A7E55:   npa_resource_unlock (resource: "/core/cpu")

0x00000001952A7ED1:      request complete (handle: 0x0ada5fe8) (sequence:
0x0013ff00) (request state:192) (active state:192)

0x000000019530B825:   npa_issue_required_request (handle: 0x0ade6ab8)
(client: "VADC") (request: 1) (resource: "/core/cpu/latency") //request for
node /core/cpu/latency.

0x000000019530B90F:   npa_issue_custom_request (handle: 0x0add0cc8)
(client: "/node/core/cpu/latency") (request: 1) (resource:
"/sleep/idle/plugin")

0x000000019530BA8F:      request complete (handle: 0x0add0cc8) (sequence:
0x000d0c00) (request state:1) (active state:1)

0x000000019530BB27:      request complete (handle: 0x0ade6ab8) (sequence:
0x00098400) (request state:1) (active state:1) //request complete for node
/core/cpu/latency.
```

```
0x0000000195312D7D:   npa_complete_request (handle: 0x0ade6ab8) (client: 
"VADC") (resource: "/core/cpu/latency")

0x0000000195312E09:   npa_issue_custom_request (handle: 0x0add0cc8) 
(client: "/node/core/cpu/latency") (request: 4294967295) (resource: 
"/sleep/idle/plugin")//request for node /sleep/idle/plugin.

0x0000000195312EEA:      request complete (handle: 0x0add0cc8) (sequence: 
0x000d0d00) (request state:1) (active state:1)//request complete for node 
/sleep/idle/plugin.

0x0000000195312F71:      request complete (handle: 0x0ade6ab8) (sequence: 
0x00098500) (request state:0) (active state:-1)

0x00000001953923BE:   npa_issue_internal_request (handle: 
0x0ada5fe8)(client: "CPU_Dynamics_Mips_Changed") (resource: "/core/cpu") 
//request for node /core/cpu.

0x000000019539262A:   npa_resource_lock (resource: "/core/cpu") (sequence: 
0x0013ff00)

0x0000000195392A61:   npa_resource_unlock (resource: "/core/cpu")
```

Following NPA log shows MCPM WCDMA request npa resource "/mcpm/vdd/cx“ and “//mcpm_bus/snoc”.

```
: npa_resource (name: "/mcpm/vdd/cx") (handle: 0xADFEAD8) (units: ModeID) 
(resource max: 7) (active max: 7) (active state: 3)  (active headroom: -4) 
(request state: 3)

……

 :         npa_client (name: mcpm_npa_resrc_vdd_cx_lte) (handle: 0xAE032F8) 
(resource: 0xADFEAD8) (type: NPA_CLIENT_REQUIRED) (request: 0)

 :         npa_client (name: mcpm_npa_resrc_vdd_cx_wcdma) (handle: 
0xAE03350) (resource: 0xADFEAD8) (type: NPA_CLIENT_REQUIRED) (request: 3) 
// mcpm_npa_resrc_vdd_cx_wcdma requested /mcpm/vdd/cx

 :         npa_client (name: mcpm_npa_resrc_vdd_cx_do) (handle: 0xAE033A8) 
(resource: 0xADFEAD8) (type: NPA_CLIENT_REQUIRED) (request: 0)

……

 :         end npa_resource (handle: 0xADFEAD8)

 : npa_resource (name: "/mcpm_bus/snoc") (handle: 0xADD60F8) (units: MBps) 
(resource max: 1024000000) (active max: 1024000000) (active state: 800) 
(active headroom: -1023999200) (request state: 800)

……

 :         npa_client (name: mcpm_npa_resrc_modem_bw_lte) (handle: 
0xAE22E18) (resource: 0xADD60F8) (type: NPA_CLIENT_REQUIRED) (request: 0)

 :         npa_client (name: mcpm_npa_resrc_modem_bw_wcdma) (handle: 
0xAE22E70) (resource: 0xADD60F8) (type: NPA_CLIENT_REQUIRED) (request: 800) 
// mcpm_npa_resrc_modem_bw_wcdma requested /mcpm_bus/snoc

 :         npa_client (name: mcpm_npa_resrc_modem_bw_do) (handle: 
0xAE22EC8) (resource: 0xADD60F8) (type: NPA_CLIENT_REQUIRED) (request: 0)

……

 :         end npa_resource (handle: 0xADD60F8)
```

4.2.4 Modem debug—clock debug

The following logs show cxo is requested:

```
: npa_resource (name: "/xo/cxo") (handle: 0xAB4E000) (units: on/off)
(resource max: 1) (active max: 1) (active state: 1)  (active headroom: 0)
(request state: 1)
:         npa_client (name: /clock) (handle: 0xAB553F8) (resource:
0xAB4E000) (type: NPA_CLIENT_REQUIRED) (request: 1)
:         end npa_resource (handle: 0xAB4E000)
```

You can recover modem side enabled clocks in trace32, use the command below:

```
do <MPSS_build>\
modem_proc\core\systemdrivers\clock\scripts\ClockDriver.cmm
```

- Select “1: Show enabled clocks”
- Select “2: Show enabled sources”

A sample output is as below:

```
[--- Enabled Clocks ---]
(45.) 0x4330ECD0 = -> cc_q6sw_core_clk [HAL_CLK_SOURCE_PLL6 = 0x10,
CLOCK_VRE
(61.) 0x4330ED3F = -> cc_pmem_aclk [CLOCK_VREG_LEVEL_LOW = 0x0]
(87.) 0x4330E88A = -> cc_ce1_hclk [CLOCK_VREG_LEVEL_LOW = 0x0]
(88.) 0x4330E87A = -> cc_ce1_core_clk [CLOCK_VREG_LEVEL_LOW = 0x0]
```

Hclk is for clock gating from AHB bus, which will not affect XO shutdown, here cc_ce1_core_clk can be root cause firstly.

4.2.5 MCPM clock dump

To extrace mcpm log in trace32, use the command below:

```
<MPSS_build>\modem_proc\mpower\mcpm\scripts\mcpm_trace.cmm
```

This can only be done in non-stripped build.
A sample output is as below:

```
0x0000000014F9BB41:         MCPM_START_REQ::MCPM_A2_START_REQ
Data: 0x00000000 0x00000000 0x00000000//MCPM A2 Tech is requested.
0x0000000014F9C1E1:          MCPM_TDEC_END::MCPM_1X_START_REQ
Data: 0x00000001 0x00000001 0x00000000
```

```
0x0000000014F9C2D2:    MCPM_MSS_ENABLE_REQ::MCPM_1X_START_REQ
Data: 0x00000001 0x00000001 0x00000000
0x0000000014F9C393:           MCPM_END_REQ::MCPM_A2_START_REQ
Data: 0x00000001 0x00000001 0x00000001
0x0000000014F9C3BF:         MCPM_START_REQ::MCPM_RF_START_REQ
Data: 0x00000001 0x00000001 0x00000001//MCPM RF Tech is requested.
0x0000000014F9C8AE:       MSS_CONFIG_START::MCPM_1X_START_REQ
Data: 0x00000000 0x00000000 0x00000000
0x0000000014F9C8BF:         MSS_CONFIG_END::MCPM_1X_START_REQ
Data: 0x00000000 0x00000000 0x00000000
0x0000000014F9C907:        MODEM_PUP_START::MCPM_1X_START_REQ
Data: 0x00000000 0x00000000 0x00000000
```

4.2.6 How to change the clock configuration from T32 for debugging purpose

You can change the clock configuration in Trace32 for debugging purpose and also break attmc_init in Trace32 and then change the variable mcpm_drv_config:

To change the needed clock or config for a certain mode, use the command below:

```
mcpm_drv_config = (
    [0] = (mcpm_cfg_pll = (pll_0 = PLL_OFF, pll_1 = PLL_OFF, pll_2 =
PLL_OFF), mcpm_cfg_v
    [1] = (mcpm_cfg_pll = (pll_0 = PLL_OFF, pll_1 = PLL_OFF, pll_2 =
PLL_OFF), mcpm_cfg_v
    [2] = (mcpm_cfg_pll = (pll_0 = PLL_ON, pll_1 = PLL_OFF, pll_2 =
PLL_OFF), mcpm_cfg_vo
    [3] = (mcpm_cfg_pll = (pll_0 = PLL_OFF, pll_1 = PLL_OFF, pll_2 =
PLL_OFF), mcpm_cfg_v
    [4] = (mcpm_cfg_pll = (pll_0 = PLL_ON, pll_1 = PLL_ON, pll_2 = PLL_OFF),
mcpm_cfg_vol
    [5] = (mcpm_cfg_pll = (pll_0 = PLL_ON, pll_1 = PLL_ON, pll_2 = PLL_OFF),
mcpm_cfg_vol
    [6] = (
      mcpm_cfg_pll = (pll_0 = PLL_ON, pll_1 = PLL_ON, pll_2 = PLL_OFF),
      mcpm_cfg_vol_mode = (vol_level = MCPM_NOMINAL),
      mcpm_cfg_clkreg = (
        mss_clk_bus_cfg_val = 0x9B,
        modem_clk_tdec_cfg_val = 30,
        modem_clk_cdma_hdr_cfg_val = 30,
        mod_offline_div_val = 0,
        modem_clk_gsm_cfg_val = 46,
        modem_clk_gsm_dco_cfg_val = 1938700515,
        modem_clk_vpe_cfg_val = 0),
        mcpm_cfg_blkreg = (mss_enable = 3, mtc_enable = 3, tx_enable = 1,
demback_enable =
```

4.2.7 Power feature enable /disable

4.2.7.1 How to disable Q6 power collapse

There're two methods to disable Q6 power collapse:

- Set the flag sleep_allow_low_power_modes to FALSE. in code
- Set in NV browser with the following procedure:
  a. Open NV browser from QXDM.
  b. Select '**MCPM**' from '**Category Filter**' .
  c. Select NV 67202 MCPM Configuration Source.
  d. Set '**Disable_Q6_PwrCollapse**' to **1**.

4.2.7.2 How to disable Q6 DCVS

To disable/enable Q6 DCVS, OEM can use EFS explorer via QPST as follows:

1. Browse to /nv/item_files/CoreCpu/CoreAll/Startup/Algorithm.txt
2. Disable Q6 DCVS: change content of Algorithm.txt to "Disabled"
3. Turn on Q6 DCVS: change content of Algorithm.txt to "qdsp_classic"
4. Fix Q6 level at initial clock given by MCPM: change content of Algorithm.txt to "Requests“

4.2.7.3 Disable low power modem through EFS explorer

To disable low power modes via the file system on the modem, a sleep_config.ini file must be placed at following folder (using EFS Explorer):

```
/nv/item_files/sleep/core0/sleep_config.ini
```

The ini file must be in the format as below, for details, refer to [Q10].

```
[section]
disable=1
Where:
section = LPR or LPR.LPRM name to configure
Any LPR or LPRM that's registered with the sleep subsystem may be
configured in this way. The current ones for the 8974 modem are:
cxo.shutdown
pxo.shutdown
cpu_vdd.footswitched
```

```
        vdd_dig.min_level_0
        vdd_dig.min_level_1
        rpm.handshake --- if this is used for the section name, then all low power
        modes that handshake with the RPM will be disabled (and thus, RPM halt will
        also be disabled).
        all_lprms --- if this is used for the section name, then the sleep task
        will return without doing any modes, including SWFI.
```

4.3 Android power debug skill

4.3.1 Debug mask for wakeup

To debug wakeup problem, OEM can enable following debug mask:

```
mount –t debugfs none /sys/kernel/debug
echo 1 > /sys/kernel/debug/clk/debug_suspend
echo 1 > /sys/module/msm_show_resume_irq/parameters/debug_mask
echo 4 > /sys/module/wakelock/parameters/debug_mask
echo 1 > /sys/module/lpm_levels/parameters/debug_mask
echo 0x16 > /sys/module/smd/parameters/debug_mask
```

4.3.2 Debug for suspend fail problem

- Check wakeup_sources

  It is possible that a kernel wakelock, and not a userspace wakelock, is preventing suspend. statistics on all wakeup_sources (active or otherwise) are maintained in sys node: `/sys/kernel/debug/wakeup_sources`

  This file contains information:

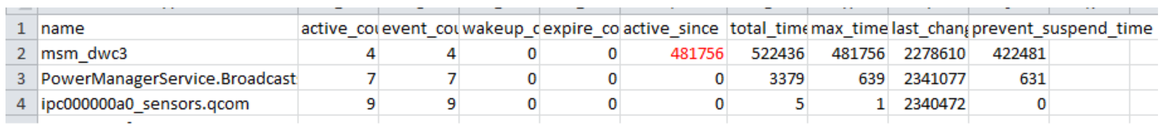
- the total amount of time a wakeup source has prevented suspend
- the amount of time a wakelock has been active since the last activation etc. The unit of time is milliseconds.

To check if a wakelock is currently preventing suspension, the active_since field has to be considered. If the value of this field is non-zero, the wakelock is then active and is preventing suspend.

Use the following command to get wakeup_sources:

```
Cat /sys/kernel/debug/wakeup_sources >/data/wakeup_sources.txt
```

After getting `wakeup_sources.txt`, you can use Excel to open it and use SPACE to delimit it, and then sort from largest to minimum according to active_since column, and then you can get an example of wakeup_sources as below:



	name	active_co	event_co	wakeup_c	expire_co	active_since	total_time	max_time	last_chang	prevent_suspend_time
1	name	active_co	event_co	wakeup_c	expire_co	active_since	total_time	max_time	last_chang	prevent_suspend_time
2	msm_dwc3	4	4	0	0	481756	522436	481756	2278610	422481
3	PowerManagerService.Broadcast	7	7	0	0	0	3379	639	2341077	631
4	ipc000000a0_sensors.qcom	9	9	0	0	0	5	1	2340472	0

Figure 4-10 Wakeup source opened in Excel

Driver msm_dwc3 active-since column is 481756>0, this means wakelock in driver msm_dwc3 prevents device to suspend, and then check driver msm_dwc3 code and log for further debugging.

The power:wakeup_source_activate and power:wakeup_source_deactivate events are written to the trace buffer any time a wakeup source is acquired or released and it can provide information on how often a wakeup source is being used by a driver.

To enable these events, you can enable following:

```
echo "power:wakeup_source_activate power:wakeup_source_deactivate" >
/sys/kernel/debug/tracing/set_event
```

Once the above done, the traces will be present in `/sys/kernel/debug/tracing/trace`.

4.3.3 Log enable for source files

To enable kernel log in specific source file, OEM can use the following commands:

```
adb shell mount -t debugfs none /sys/kernel/debug adb shell "echo 8 >
/proc/sys/kernel/printk"
adb shell "echo 'file qpnp-adc-tm.c +p' >
/sys/kernel/debug/dynamic_debug/control"
adb shell "echo 'file qpnp-adc-common.c +p' >
/sys/kernel/debug/dynamic_debug/control"
adb shell "echo 8 > /proc/sys/kernel/printk"
```

4.3.4 Log enable for specific function

You can enable log in specific function. In the following commands, log in the function qpnpint_handle_irq will be enabled.

```
adb shell "echo 'func qpnpint_handle_irq +p' >
/sys/kernel/debug/dynamic_debug/control"
```

4.3.5 powertop

Powertop is used to see cpu running statistics which can help to debug power issues.

Powertop usage is as following:

powertop --h

Usage: powertop [OPTION...]

- -d, --dump read wakeups once and print list of top offenders
- -t, --time=DOUBLE default time to gather data in seconds
- -r, --reset Reset PM stats data
- -h, --help Show this help message
- -v, --version Show version information and exit

Use the following procedures to get powertop log:

1. Plug USB
2. In adb shell, execute the following command:
   ```
   Sleep 10 && /data/powertop [-r] -d -t 30 > /data/powertop.log &
   ```
3. Wait for 10 seconds to start your test, and then unplug USB and wait for over 30 seconds
4. Plug USB.
5. Get `powertop.log` from phone with the following command:
   ```
   adb pull /data/powertop.log
   ```
   And then provide powertop.log to case for check.

4.3.6 CPU freq log

- To enable cpu freq change log

```
mount –t debugfs none /sys/kernel/debug
cd /sys/kernel/debug
echo -n 'file acpuclock-8x60.c +p' > dynamic_debug/control
echo -n 'file acpuclock-krait.c +p' > dynamic_debug/control
```

- To check cpu freq stats:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/stats
cat /sys/devices/system/cpu/cpu1/cpufreq/stats
cat /sys/devices/system/cpu/cpu2/cpufreq/stats
cat /sys/devices/system/cpu/cpu3/cpufreq/stats
```

- To lock cpu freg:

  echo the same freq to following sys mode will lock cpu freq to the setting freq.

```
/sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

```
/sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
```

- To enable/disable specific freq for ACPU

  ACPU freq table is defined in acpu_freq_tbl_* structure of specific platform.

```
arch/arm/mach-msm/acpuclock-<platform name>.c
```

For 8974, it is defined in `arch/arm/mach-msm/acpuclock-8974.c`. the first column of following table used to enable/disable freq in the row: **1**:**enable**, **0**:**disable**

```
static struct acpu_level acpu_freq_tbl_2p3g_pvs0[] __initdata = {
{ 1, {  300000, PLL_0, 0,   0 },  L2(0),  800000,  72 },
{ 0, {  345600, HFPLL, 2,  36 },  L2(1),  800000,  83 },
{ 1, {  422400, HFPLL, 2,  44 },  L2(2),  800000, 101 },
{ 0, {  499200, HFPLL, 2,  52 },  L2(2),  805000, 120 },
{ 0, {  576000, HFPLL, 1,  30 },  L2(3),  815000, 139 },
{ 1, {  652800, HFPLL, 1,  34 },  L2(3),  825000, 159 },
{ 1, {  729600, HFPLL, 1,  38 },  L2(4),  835000, 180 },
{ 0, {  806400, HFPLL, 1,  42 },  L2(4),  845000, 200 },
{ 1, {  883200, HFPLL, 1,  46 },  L2(4),  855000, 221 },
{ 1, {  960000, HFPLL, 1,  50 },  L2(9),  865000, 242 },
{ 1, { 1036800, HFPLL, 1,  54 }, L2(10),  875000, 264 },
{ 0, { 1113600, HFPLL, 1,  58 }, L2(10),  890000, 287 },
{ 1, { 1190400, HFPLL, 1,  62 }, L2(10),  900000, 308 },
…
{ 1, { 1958400, HFPLL, 1, 102 }, L2(19), 1040000, 565 },
{ 0, { 2035200, HFPLL, 1, 106 }, L2(19), 1055000, 596 },
{ 0, { 2112000, HFPLL, 1, 110 }, L2(19), 1070000, 627 },
{ 0, { 2188800, HFPLL, 1, 114 }, L2(19), 1085000, 659 },
{ 1, { 2265600, HFPLL, 1, 118 }, L2(19), 1100000, 691 },
{ 0, { 0 } }
};
```

4.3.7 Hoplug cores

Core 0 can’t be hotplugged, Core 1/2/3 can be hotplugged,

- To remove core :

```
echo 0 > /sys/devices/system/cpu/cpu1/online
echo 0 > /sys/devices/system/cpu/cpu2/online
echo 0 > /sys/devices/system/cpu/cpu3/online
```

- To add back core 1:

```
echo 1 > /sys/devices/system/cpu/cpu1/online
echo 1 > /sys/devices/system/cpu/cpu2/online
echo 1 > /sys/devices/system/cpu/cpu3/online
```

4.3.8 Scaling governor

- To check scaling governor

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

- To set new governor

```
echo <new_governor> >
/sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo ondemand > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

4.3.9 Mpdecision

Use Mpdecison daemon to start/stop/enable debug with commands below:

- Start mpdecison

  `Start mpdecision`
- Stop mpdecison

  `Stop mpdecision`
- Enable mpdecision debug

  `Start mpdecision --debug`

4.3.10 Power feature enable /disable

Following sys node can be used to enable the lower resource,

```
echo 2 > /sys/module/lpm_resources/enable_low_power/l2
echo 1 > /sys/module/lpm_resources/enable_low_power/pxo
echo 1 > /sys/module/lpm_resources/enable_low_power/vdd_dig
echo 1 > /sys/module/lpm_resources/enable_low_power/vdd_mem
echo 1 > /sys/module/pm_8x60/modes/cpu0/power_collapse/suspend_enabled
echo 1 > /sys/module/pm_8x60/modes/cpu1/power_collapse/suspend_enabled
echo 1 > /sys/module/pm_8x60/modes/cpu2/power_collapse/suspend_enabled
echo 1 > /sys/module/pm_8x60/modes/cpu3/power_collapse/suspend_enabled
echo 1 > /sys/module/pm_8x60/modes/cpu0/power_collapse/idle_enabled
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
echo 1 >
/sys/module/pm_8x60/modes/cpu2/standalone_power_collapse/suspend_enabled
echo 1 >
/sys/module/pm_8x60/modes/cpu3/standalone_power_collapse/suspend_enabled
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
echo 1 >
/sys/module/pm_8x60/modes/cpu2/standalone_power_collapse/idle_enabled
echo 1 >
/sys/module/pm_8x60/modes/cpu3/standalone_power_collapse/idle_enabled
```

echo 0 to above sys node will disable related low power mode.

4.3.11 Check system alarm

- Use the following commands to get android alarms and statistics:
  `adb dumpsys alarm >alarms.txt`
- Use the following commands to enable android debug message in logcat.
  `setprop persist.alarm.debug 1`

4.3.12 Kernel timer check

Use the following commands to check kernel timer:

```
Sys node /proc/timer_stats can be used to check kernel timer stastics,
customer can use following command  to get timer statics in specific
scenario:
echo 0 > /proc/timer_stats && sleep 10 && echo 1 > /proc/timer_stats &&
sleep 30 && cat /proc/timer_stats > /data/timer_stats &
```

OEMs need to provide file `/data/timer_stats` to salesforce case for check.