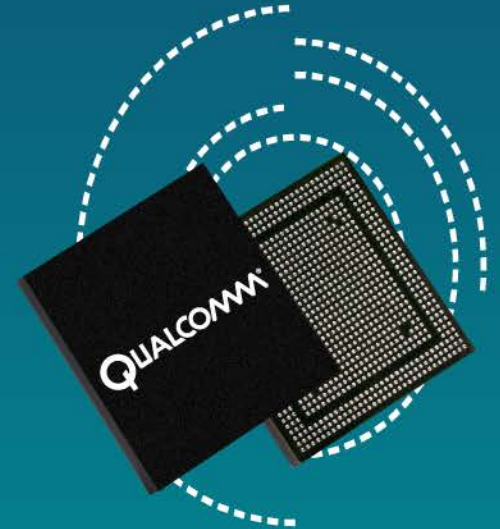


QUALCOMM®
2016-06-22 20:46:37 PDT
martin.xu@zhntd.com



MSM8974 Power Debugging

80-NA157-68 C

Confidential and Proprietary – Qualcomm Technologies, Inc.

Confidential and Proprietary – Qualcomm Technologies, Inc.

NO PUBLIC DISCLOSURE PERMITTED: Please report postings of this document on public servers or websites to: DocCtrlAgent@qualcomm.com.

Restricted Distribution: Not to be distributed to anyone who is not an employee of either Qualcomm or its subsidiaries without the express approval of Qualcomm's Configuration Management.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc.

Qualcomm reserves the right to make changes to the product(s) or information contained herein without notice. No liability is assumed for any damages arising directly or indirectly by their use or application. The information provided in this document is provided on an "as is" basis.

This document contains confidential and proprietary information and must be shredded when discarded.

Qualcomm is a trademark of QUALCOMM Incorporated, registered in the United States and other countries. All QUALCOMM Incorporated trademarks are used with permission. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

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

© 2013-2014 Qualcomm Technologies, Inc.
All rights reserved.

Revision History

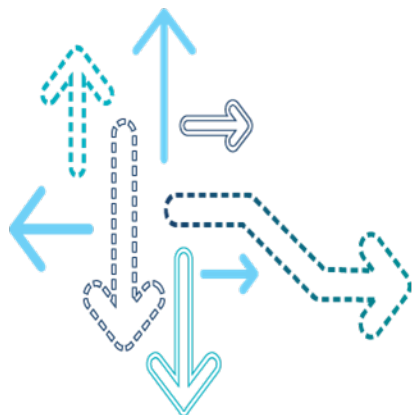
Revision	Date	Description
A	Feb 2013	Initial release
B	Aug 2013	Added more debugging details, i.e., PMIC dump, RPM RAM dump parser, and VDD CX debugging
C	Jul 2014	Added and updated debugging details, disabling the PMIC watchdog from T32 and RPM, capturing a meaningful RAM dump for sleep issue, and RPM ELF verification

Contents

- Getting Logs/Dumps
- Optimizing Sleep/Rock-Bottom Current
- Optimizing Standby Current
- Optimizing Talk Current
- Optimizing Data Current
- References
- Questions?

QUALCOMM®
2016-06-22 20:46:37 PDT
martin.xu@zhnhd.com

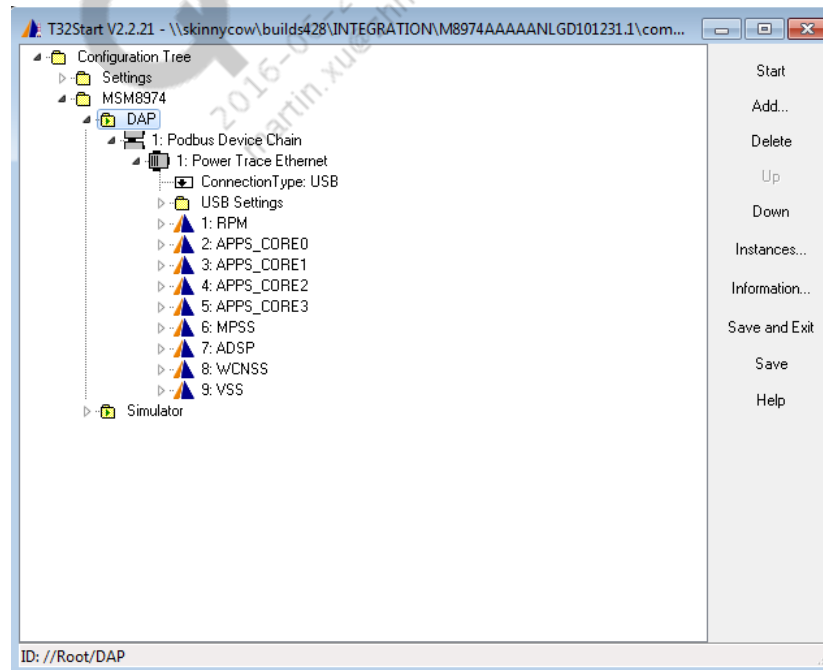
Getting Logs/Dumps



Starting T32 Session

- T32 shortcuts can be found in the metabuild at <Metabuild>\common\t32\t32Start.
- Once the t32Start window is open, select Configuration Tree→MSM8974→DAP→Podbus Device Chain→Power Trace Ethernet.
- T32 icons for RPM, Modem, Apps, etc., are located here.
- Select the processor of interest and press **Start** to bring up the Trace32 window.

Note: Use T32 Ver Jul 2012 or later for Hexagon™ Ver 5.



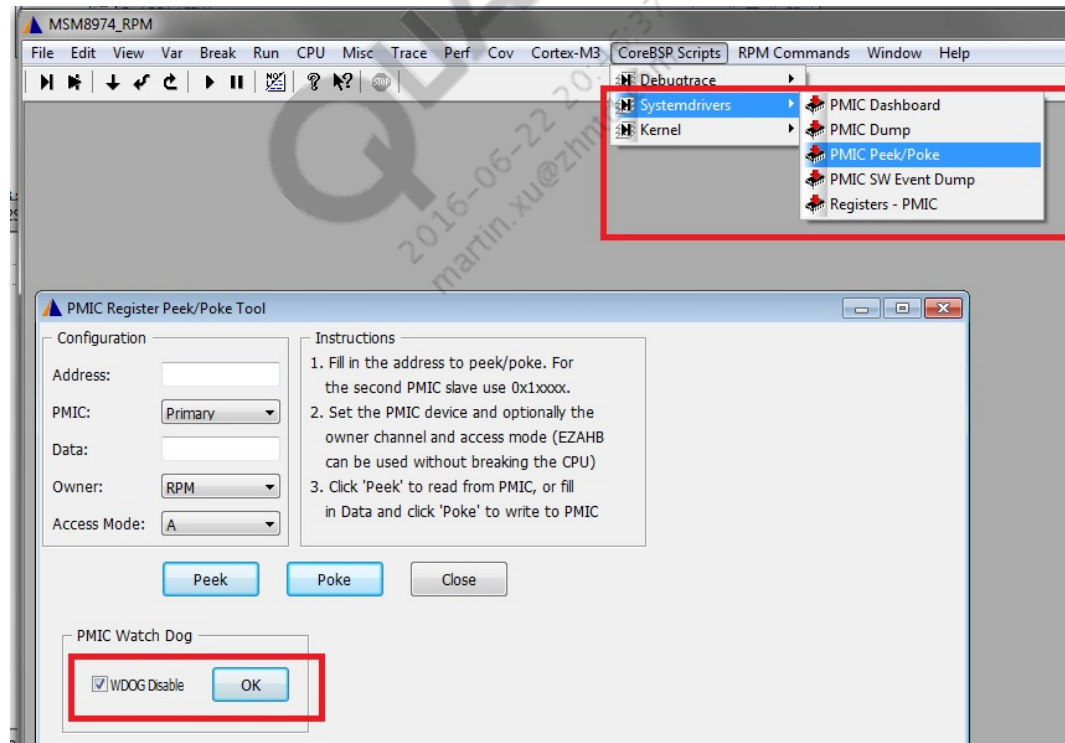
Disable the PMIC Watchdog

- PMIC watchdog was enabled in a recent software release, so to avoid the device reset when taking Clock, PMIC and GPIO dumps, it is necessary to disable the PMIC watchdog before taking the dump.
- Disable PMIC watchdog by modifying the RPM code
 1. Open the **rpm_proc\core\power\rpm\dog.c**
 2. Set the **pmic_wdog_enable = 0**.
 3. Rebuild the RPM image
 - a. `cd rpm_proc\build`
 - b. Run the **build_8974.bat**
 4. Put the device to Fastboot mode, and then run the following command to flash the rpm image to device
 - a. `fastboot flash rpm rpm_proc\build\ms\bin\AAAAANAAR\rpm.mbn`

```
#ifdef ENABLE_PMIC_WATCHDOG
volatile bool pmic_wdog_enable = 0; //set to 0 to disable the PMIC WDT.
#else
volatile bool pmic_wdog_enable = 0;
#endif
```

Disable the PMIC watchdog (cont.)

- Disable PMIC watchdog by using RPM T32
 1. Open an RPM T32 window.
 2. Type **sys.m.a** to attach T32 to the device.
 3. Type **break** to pause the device, and refer to the following screenshot to disable PMIC watchdog.

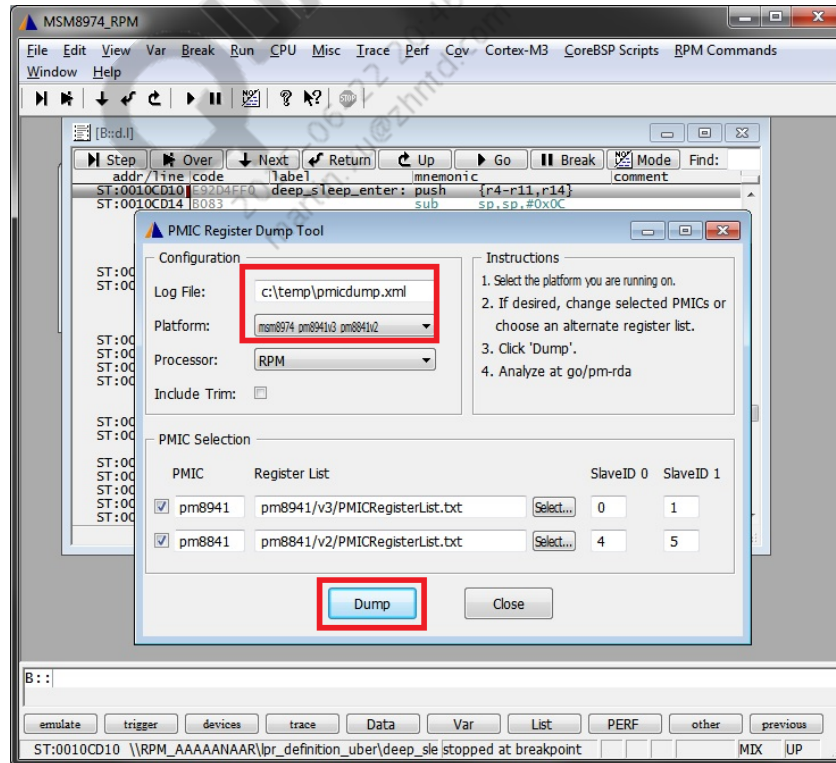


Clock Dumps

- For clock dumps:
 1. Open an RPM T32 window.
 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. Type **cd ../../rpm_proc/core/systemdrivers/clock/scripts/msm8974**.
 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.
 8. Clock dump is printed in the message area of T32.

PMIC Dumps

- For PMIC dumps:
 1. Open an RPM T32 window.
 2. Type **sys.m.a** to attach T32 to the device.
 3. Load the .elf file and set the breakpoint that needs the PMIC dumps.
 4. Once the breakpoint is hit, type **CD.DO rpm_proc\core\systemdrivers\pmic\scripts\PMICDump.cmm** in a T32 window.



PMIC Dumps (cont.)

- For PMIC dumps: (cont.)
 - By default, the raw PMIC dump file is saved in c:\temp\pmicdump.xml.
 - Use the following in a command window to convert the raw PMIC register setting to a human-readable file:
 - `cd rpm_proc\core\systemdrivers\pmic\scripts`
 - `python PMICDumpParser.py --flat=pm8941/v3/CORE_ADDRESS_FILE_CUSTOMER.FLAT --file=c:\temp\pmicdump.xml > pmicdump.txt`

RPM Debug Logs

- In MSM8974, disabling the RPM halt is not necessary for keeping the JTAG daisy-chain connection.
- Use the JTAG to modify the variable `sleep_allow_low_power_modes` to enable/disable all the RPM power modes.
 - `sleep_allow_low_power_modes = 0` disables all low power modes for RPM
 - `sleep_allow_low_power_modes = 1` enables all low power modes for RPM
- Saving RPM RAM dumps
 1. Open RPM T32 and attach by `sys.m.a`.
 2. Create the issue scenario where RPM dumps are needed.
 3. Break T32 at the desired point and run the following script to save RPM logs:
 - `do <RPM Build location>\rpm_proc\core\bsp\rpm\scripts\rpm_dump.cmm <Location to save the dumps>`

RPM Debug Logs (cont.)

- Loading RPM dumps onto the T32 simulator
 1. Open the T32 simulator and do sys.up.
 2. Run the following script:
 - do <RPM Build location>\rpm_proc\core\bsp\rpm\scripts\rpm_load_dump.cmm
<Location of logs>
 3. Load RPM.elf to start debugging.
 - d.load.elf <RPM_Build>\rpm_proc\core\bsp\rpm\build\RPM_XXXXXXXXX.elf /nocode /noclear
- RPM external logs using T32
 - While attached to the RPM and in the Break state or if the RPM RAM dumps are loaded on the T32 Simulator, run the following command in T32:
 - do <RPM_build>\rpm_proc\core\power\ulog\scripts\rpm_ulogdump.cmm <Location to save logs>
 - This places the RPM's external logs into your log directory.
 - The RPM external log requires the use of a Python parsing tool to interpret its contents. Use the following Python script on extracted logs:
 - Python \\<RPM_build_location>\rpm_proc\core\power\rpm\debug\scripts\rpm_log_bfam.py -f "RPM External Log.ulog"

RPM Debug Logs (cont.)

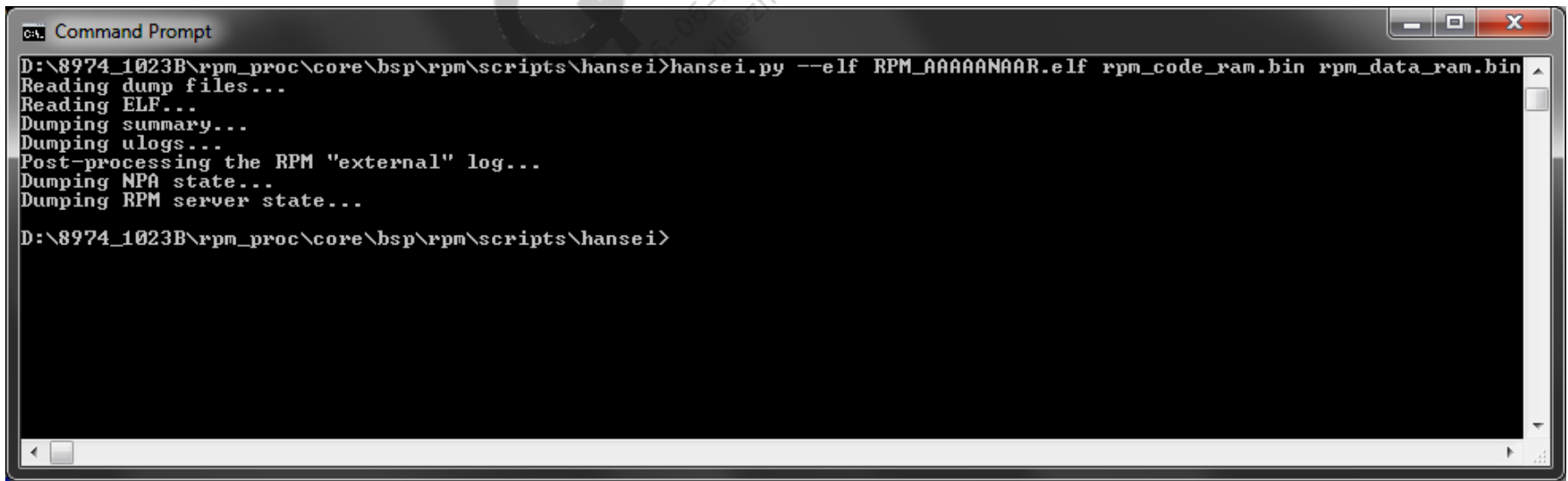
- RPM NPA logs using T32
 - While attached to the RPM and in the Break state, or if the RPM RAM dumps are loaded on the T32 simulator, run the following command in T32:
 - do <RPM_build>\rpm_proc\core\power\npa\scripts\rpm_npadump.cmm <Location to save logs>
 - This places the RPM's NPA logs into your log directory.
- Obtaining sleep count from RPM T32
 - The variable sleep_stats[0].count can be used to identify the CXO shutdown count from the RPM T32 window.
 - The variable sleep_stats[1].count can be used to identify the VDD min count from the RPM T32 window.

RPM Debug Logs (cont.)

- Checking SPM state for each master
 - Using T32, check the variable `rpm.ees[master_id].subsystem_status`; this will provide the SPM status for each master as follows:
 - SPM_AWAKE – Master is awake
 - SPM_SLEEPING – Master is sleeping
 - Various master IDs are:
 - 0 – Apps subsystem
 - 1 – Modem subsystem
 - 2 – LPASS subsystem (ADSP)
 - 3 – Pronto subsystem (Connectivity)

RPM Debug Logs (cont.)

- Hansei – An RPM RAM dump parser tool that can be found in the rpm_proc\core\bsp\rpm\scripts\hansei folder
- Usage – hansei.py [-h] --elf rpm.elf [--output path] dumpfile [dumpfile ...]
- Example – hansei.py --elf rpm.elf -o . rpm_code_ram.bin rpm_data_ram.bin rpm_msg_ram.bin



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


RPM Debug Logs (cont.)

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

Modem Debug Logs

- Modem Ulogs

- Execute the following script from Modem_Build in the modem T32 window:
 - do <Modem_Build>\modem_proc\core\power\ulog\scripts\UlogDump.cmm <Location to save logs>
- This will place the modem ulogs in the specified directory.

- Modem NPA logs

- Execute the following script from Modem_Build in the modem T32 window:
 - do <Modem_Build>\modem_proc\core\power\npa\scripts\NPADump.cmm <Location to save logs>
- This will place the modem NPA logs in the specified directory.

Modem Debug Logs (cont.)

- Getting modem RAM dumps from T32
 1. Open RPM T32 and attach by sys.m.a.
 2. Open MPSS T32 and attach by sys.m.a.
 3. Create the issue scenario where modem dumps are needed.
 4. Break MPSS T32 and RPM T32 at the desired point, and in the RPM T32 window, run the following script:
 - do <Metabuild location>\common\tools\cmm\common\std_savelogs.cmm <Location to save the dumps>
 5. This will give a menu from which the MPSS option can be selected to obtain the modem RAM dumps.
- Loading modem RAM dumps
 - Open the Q6 simulator and run the command sys.up.
 - To load modem RAM dumps to the simulator, use the following command:
 - d.load.binary MPSS_SW_log.bin 0X8400000
 - To start debugging, load modem elf using the following command:
 - d.load.elf <ModemBuildLocation>\modem_proc\build\ms\M8974AXXXXXXXXXXXXXX.elf /nocde /noclear

GPIO Dumps

- All GPIOs in the hardware use the following script, which can be run from the RPM T32 window, to read the current configuration of all GPIOs in the hardware:
 - do <Modem_Build>\modem_proc\core\systemdrivers\tlmm\scripts\tlmm_gpio_hw.cmm
- For default sleep configuration stored in software, use the following script to read the default sleep configuration stored in the software, which is read out from TLMM.xml:
 - do <Modem_Build>\modem_proc\core\systemdrivers\tlmm\scripts\tlmm_sleep_configs.cmm

Getting F3 Logs

- F3 trace logs
 - F3 logs are useful to analyze the Awake Current issues and explain modem behavior.
 - F3 trace logs with QXDM Professional™ (QXDM Pro) by connecting USB
 - As the USB is connected to the system for QXDM Pro, the system cannot enter sleep, hence:
 - Connect the USB, Open QXDM Pro, and press **F3**.
 - F3 logs can be seen in QXDM Pro.
- F3 trace logs with T32 (no need for USB connection)
 1. Set NV items as follows:
 - Set 1892 to 0x5.
 - Set 1895 to 0xFF.
 - Set 1962 to 0x1.
 2. 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.

Collecting Valid RAM Dump for Debugging Sleep Issue

- Normally QXDM Pro or an ADB command can be used to trigger the reset to get the RAM dump for debugging a sleep issue, but the USB connection keeps the particular subsystem awake, so it is useful to have a method to trigger the reset on RPM side to let the device enter the Download mode for a certain duration.
- Adding following code in **blue** in **rpm_proc\core\bsp\rpm\src\main.c** causes the RPM trigger the ERR_FATAL after 5 minutes ($0x15752A000 = 19.2\text{MHz} * 60\text{sec} * 5\text{mins}$)

```
@@ -51,6 +51,8 @@  
#include "timetick.h"  
#include "version.h"  
#include "image_layout.h"  
+#include "Err.h"
```

```
#if (defined(MPQ8092_STUBS))  
void pm_init(void){}  
@@ -299,6 +301,11 @@
```

Collecting Valid RAM Dump for Debugging Sleep Issue (cont.)

```
int main(void)
{
    unsigned i;
+
+ uint64_t delay_time;
+ /* 5 minutes */
+ delay_time = time_service_now() + 0x15752A000;
+
#ifdef INIT_PRNG_NO_TZ
    extern uint32 g_prng_config_enable;

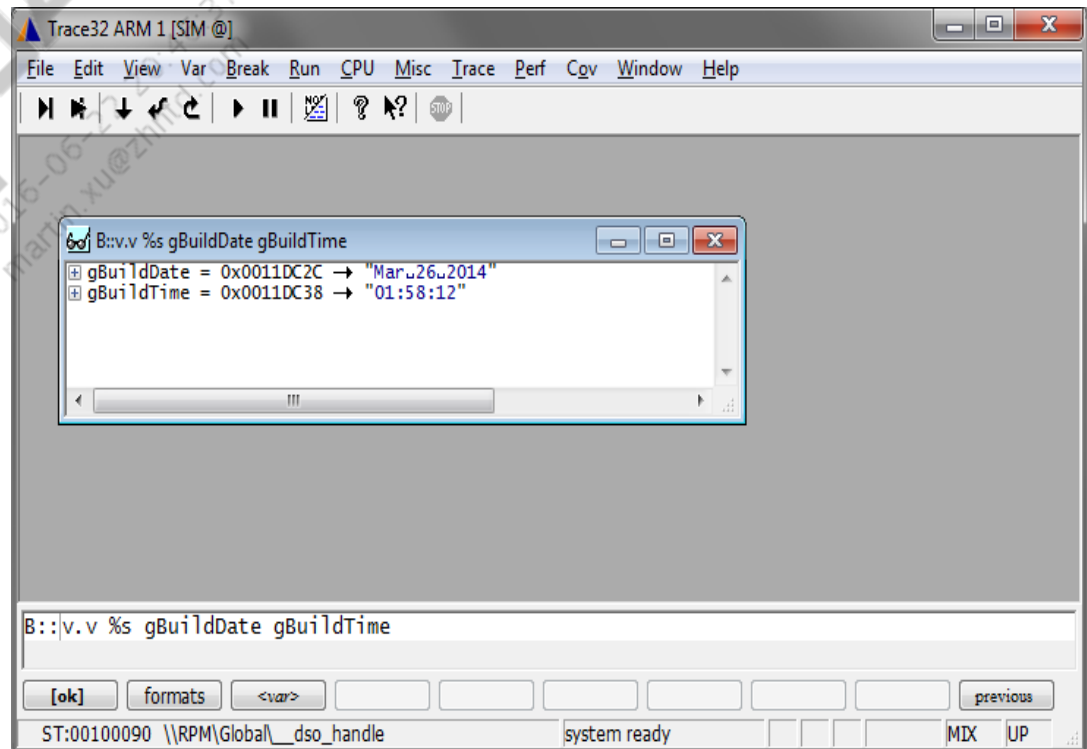
@@ -356,6 +363,10 @@
    sched_run();
    assert(!intlock_nest_level);

+ if(delay_time < time_service_now()) {
+ ERR_FATAL("PM Debugging",0, 0, 0);
+ }
+
    if(npa_num_pending_events() > 0)
    {
        npa_process_event();
    }
}
```

RPM ELF and RAM Dump Match Verification

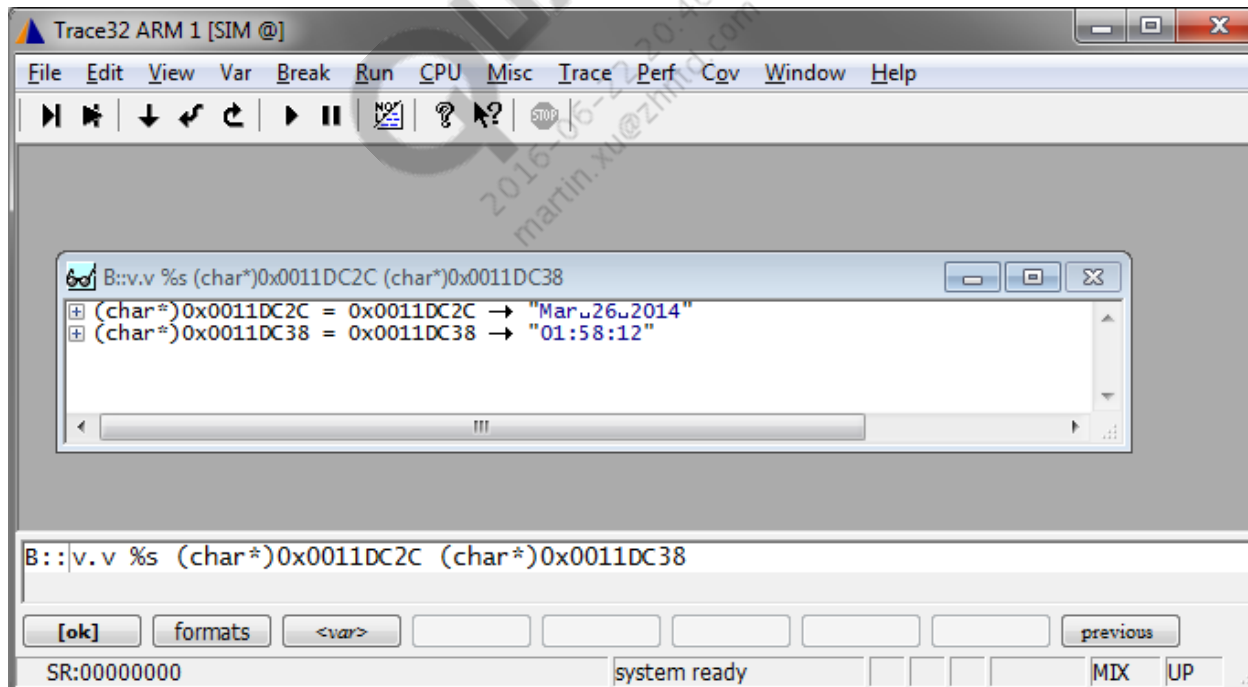
- To save the back and forth time due to a mismatching RPM ELF file, it is better to use the following instruction to verify whether the RPM ELF file and the RPM RAM dump are a match or not.
- RAM ELF file build date check:
 - Open a RPM T32-simulator windows and then use the following command to check the build date and time.

- sys.u
- D.LOAD RPM.elf
- v.v %s gBuildDate gBuildTime



RPM ELF and RAM Dump Match Verification (cont.)

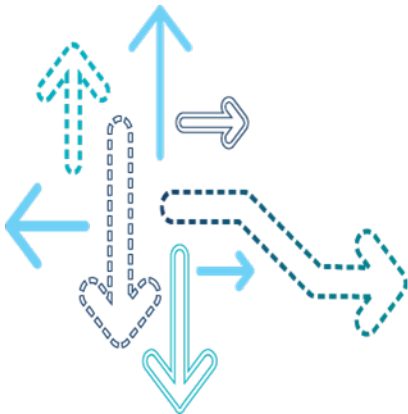
- RPM RAM dump build date check:
 - Open a RPM T32 simulator windows and then use the following command to check the build date and time.
 - sys.u
 - d.load.BINARY codeRAM.bin 0x100000
 - v.v %s (char*)0x0011DC2C (char*)0x0011DC38



Powertop Logs

1. From the command prompt, type the following commands to push the powertop binary:
 - `adb root`
 - `adb push <Apps Build Location>\android\out\target\product\msm8974\system\bin\powertop /data/`
 - `adb shell "chmod 777 /data/powertop"`
2. Reproduce the desired scenario and from the command prompt, type the following adb commands to get powertop logs:
 - `adb shell`
 - `sleep 5 && /data/powertop -d -t 60 > /data/powertop.log &`
3. Disconnect the USB within 5 sec after running this command.
4. Plug the USB back in within 70 to 75 sec and type the following adb commands to get the powertop data:
 - `adb root`
 - `adb shell`
 - `cat /data/powertop.log`

Optimizing Sleep/Rock-Bottom Current



Usual Suspects for Higher Sleep Current

- One of the following major subsystems is not voting for sleep:
 - Low Power Audio Subsystem (LPASS)
 - Applications Subsystem (APSS)
 - Modem subsystem
 - Pronto subsystem
- One or more peripheral devices (outside of the MSM chipset) are not going into their lowest power state.
- GPIOs are 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.
- The system did not enter VDD minimization.

Setup-Related Issues

- To determine whether test setup-related issues are impacting power, you must ensure that the setup is correct. Sleep current can be measured either by placing the phone in Airplane mode or by connecting to a call box (WCDMA/GSM/1X mode) in Online mode.
 - Airplane mode
 - No wake-ups are expected during Airplane mode.
 - The phone should exhibit a constant current consumption behavior throughout.
 - If there are random wake-ups 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 call box and getting repeated paging cycles as expected.

Ensuring Apps Power-Collapse

- 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)
 - 34.522064 – rpm_master_set_transition (master: "APSS") (leaving: "Active Set") (entering: "Sleep Set")
- If these messages are not in the RPM logs, either APSS is already in power-collapse or it has not entered power-collapse.
- To further verify, also check rpm.ees[0].subsystem_status in RPM T32; this will provide the subsystem status of APSS as SPM_AWAKE or SPM_SLEEPING.

Note: In rpm.ees[0].subsystem_status, 0 stands for Master 0 = Apps processor.

Ensuring Modem Power-Collapse

- Modem power-collapse can be confirmed by looking at the RPM external logs. Check for the following messages in the 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")
- If these messages are not in the RPM logs, either the modem is already in power-collapse or it has not entered power-collapse.
- To further verify, also check rpm.ees[1].subsystem_status in RPM T32; this will provide the subsystem status of the modem as SPM_AWAKE or SPM_SLEEPING.

Note: In rpm.ees[1].subsystem_status, 1 stands for Master 1 = Modem processor.

Ensuring LPASS Power-Collapse

- LPASS power-collapse can be confirmed by looking at the RPM external logs. Check for the following messages in the 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")
- If these messages are not in the RPM logs, either LPASS is already in power-collapse or it has not entered power-collapse.
- To further verify, also check rpm.ees[2].subsystem_status in RPM T32; this will provide the subsystem status of LPASS as SPM_AWAKE or SPM_SLEEPING.

Note: In rpm.ees[2].subsystem_status, 2 stands for Master 2 = LPASS subsystem.

Ensuring Pronto Power-Collapse

- Check rpm.ees[3].subsystem_status in RPM T32; this will provide the subsystem status of Pronto as SPM_AWAKE or SPM_SLEEPING.

Note: In rpm.ees[2].subsystem_status, 3 stands for Master 2 = Pronto subsystem.

Ensuring All Masters Are Power-Collapsed

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

*** APPS ***

status: SPM_AWAKE
num_active_cores: 2
pending_bringups: 0x00000000

*** MSS ***

status: SPM_SLEEPING
num_active_cores: 0
pending_bringups: 0x00000000

*** LPASS ***

status: SPM_SLEEPING
num_active_cores: 0
pending_bringups: 0x00000000

*** WCSS ***

status: SPM_SLEEPING
num_active_cores: 0
pending_bringups: 0x00000000

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 that the following variable is set properly:
 - `sleep_allow_low_power_modes` should be set to TRUE.

Ensuring Overall System Entering Low Power Modes (cont.)

- Check the RPM NPA log for the following messages to ensure that the system enters Low Power modes:
 - Check the active state of the “/sleep/uber” node from the RPM NPA logs.
The NPA log snippet is:
 - npa_resource (name: “/sleep/uber”) (handle: 0x195f20) (units: on/off) (resource max: 15) (active max: 15) (**active state: 3**) (active headroom: -12) (request state: 3)
 - npa_client (name: sleep) (handle: 0x196060) (resource: 0x195f20) (type: NPA_CLIENT_REQUIRED) (request: 1)
 - npa_client (name: vddcx) (handle: 0x196098) (resource: 0x195f20) (type: NPA_CLIENT_REQUIRED) (request: 2)
 - If the active state is 3, it means that one of the EEs is not in Power Collapse mode, and aggregation of VDD CX is not in retention voltage; in this state, the device does not enter Low Power mode.
 - If the active state is 2, it means that all of the EEs are in Power Collapse mode, but aggregation of VDD CX is not in retention voltage; in this state, the device does not enter VDD Minimization Power mode.

Ensuring Overall System Entering Low Power Modes (cont.)

- The following variable identifies which master did not request retention voltage:

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

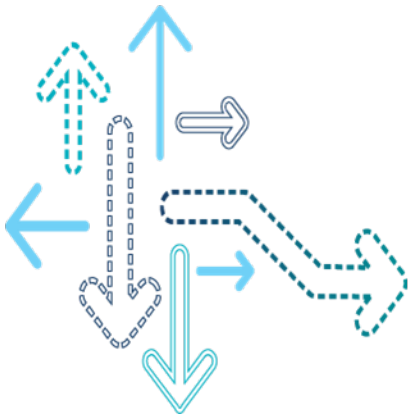
Ensuring Overall System Entering Low Power Modes (cont.)

- Check if the counts of the following variables are increasing after each expected power-collapse:
 - `sleep_stats[0].count` //XO shutdown power mode counter
 - `sleep_stats[1].count` //VDD min shutdown power mode counter

Sleep Current Deltas/Leakages – 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 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 helps 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.

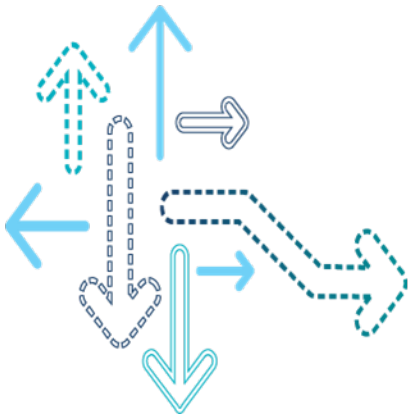
Optimizing Standby Current



Debugging High Standby Current

- Test setup-related issues impacting power
 1. Check the setup.
 - Ensure the testing is done with the call box.
 - Testing in a live network will obviously have higher awake current, so it is not recommended.
 2. Disable neighbor cells in the call box.
 3. Disable all data operations/browsing in the phone through the GUI.
 4. Ensure power levels used in the call box are as recommended. These power levels include Rx cell power, Tx cell power, AGC, etc.
 5. Ensure that all the debug logging is disabled through NV items.
 6. 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.
 7. To inspect and optimize average current during page wake-ups, compare the clock dump during awake between the target phone and the FFA. This will give you an idea, although during an awake cycle, frequencies continue changing due to DCVS and CPU behavior.

Optimizing Talk Current



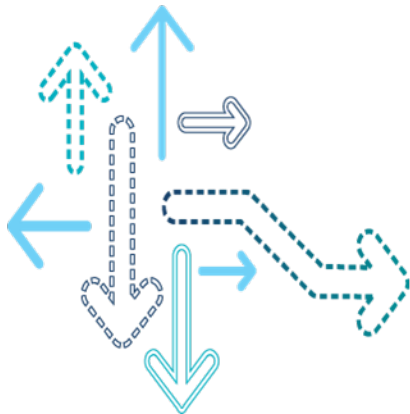
Optimizing Talk Current

- Test setup-related issues impacting power
 - To analyze test setup-related issues impacting power:
 1. Set NV item 00010 to the respective modem technology used (WCDMA only/GSM only/1X only). The phone is expected to work when this NV item is set to Automatic mode as well.
 2. Ensure Tx power is at 0 dBm per the Qualcomm Technologies, Inc. (QTI) standard test recommendation for voice calls.
 3. Turn off Rx Diversity.
 4. See [Q3] for any test setup-related issues/NV-related issues.
- 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 by comparing clock dumps during a voice call between the target phone and the FFA, and analyzing the clock differences.

Optimizing Talk Current (cont.)

- Inspecting other subsystems
 - Other subsystems, such as apps processor and Pronto, are expected to be in their low power states during talk cases.
 - Ensure that these are in their low power state.
- 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.
- Analyzing F3 logs
 - Collect F3 logs for the technology you are testing.
 - F3 logs can be used for a comparative analysis between an FFA and target device to check if there is anything different going in the protocol.

Optimizing Data Current



Optimizing Data Current

- Apart from the techniques discussed in Optimizing Talk Current, the following debug methods can be used to optimize data current as it also involves the apps processor.
 - Analyzing powertop logs
 - Get powertop logs and compare them against FFA. In powertop logs, look for:
 - Time in each of C-States (low power states)
 - Time in each frequency
 - Interrupt activity
 - These tests would give a rough estimate of why the apps processor is consuming higher current.
 - Analyzing top data
 - Check for any extra running processes using top data. Top data can be obtained using adb shell with the following command:
 - `top -m <number of processes to be displayed>`
 - For example, “`top -m 5`” will give the top five running processes.
 - These can be compared against FFA top logs and any redundant processes that can be killed should be killed.

Optimizing Data Current (cont.)

- The following variables can be compared with the FFA to ensure that they are set the same in the FFA as well as on the target device.
 - /sys/devices/system/cpu/cpufreq/ondemand/sampling_rate
 - /sys/devices/system/cpu/cpufreq/ondemand/io_is_busy
 - /sys/devices/system/cpu/cpufreq/ondemand/sampling_down_factor

References

Ref.	Document	
Qualcomm Technologies		
Q1	Application Note: Software Glossary for Customers	CL93-V3077-1
Q2	Presentation: MSM8974 Power Management Overview	80-NA157-10
Q3	Power Consumption Measurement Procedure for MSM™ (Android™-Based)/MDM Devices	80-N6837-1

QUALCOMM®
2016-06-22 20:46:37 PDT
martin.xu@zhnhd.com

Questions?

<https://support.cdmatech.com>

