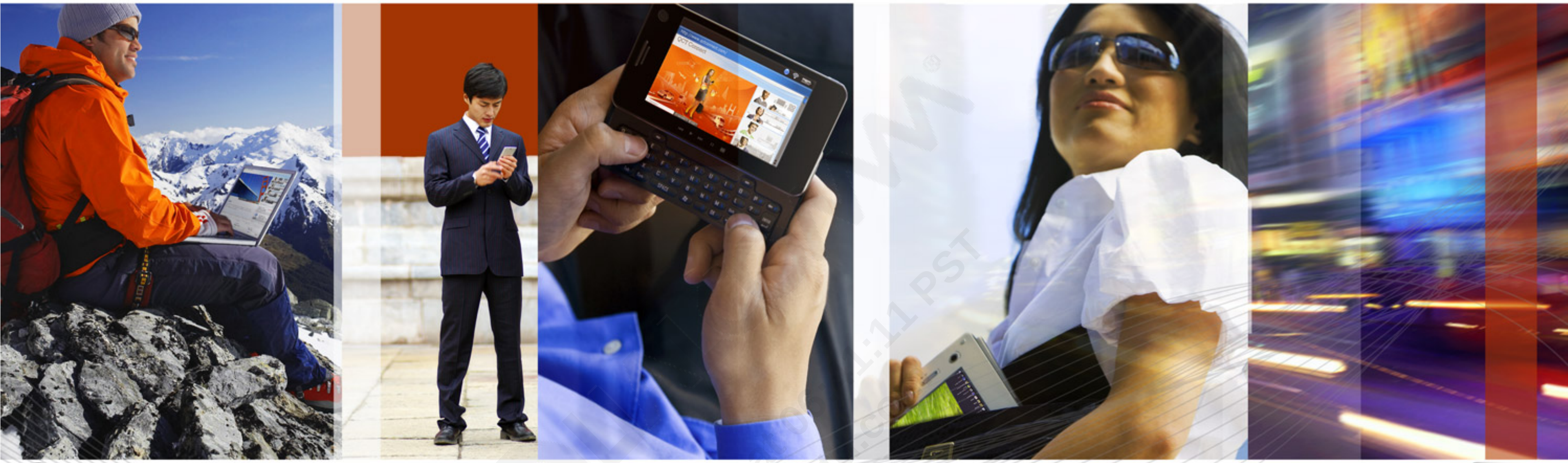


REDEFINING MOBILITY

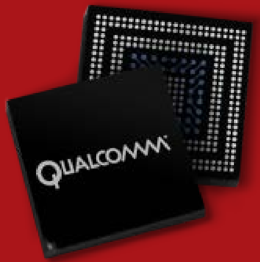
MSM8960™ Power Management and Optimization Guide

80-N5232-1 C

Revision History

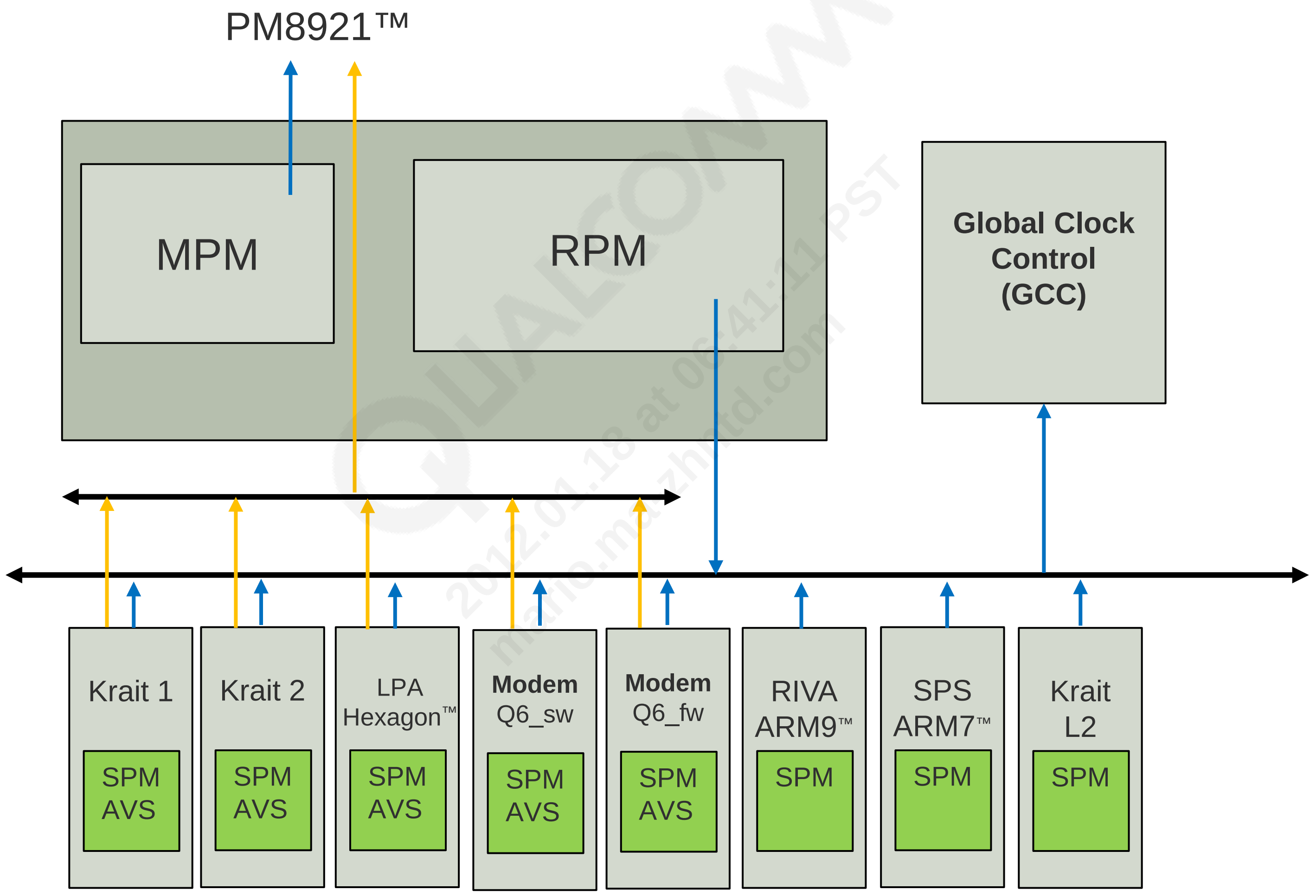
Version	Date	Description
A	Mar 2011	Initial release
B	Aug 2011	Updated power tree
C	Oct 2011	Corrected source file/API names; described CPUidle in detail

Contents



Power Management Features

MSM8960™ Hardware Features for Power Management



Note: For MSM® power management debugging, see [Q2] and [Q3].

MSM8960 Hardware Features for Power Management (cont.)

- Dynamic clock and voltage scaling
  - Subsystem Power Manager (SPM)
  - Dedicated RPM for shared resource management
  - Adaptive Voltage Scaling (AVS)
    - Not mandatory, but it could save more power
- Global Clock Control (GCC)
  - Three different oscillators – CXO (= 19.2 MHz), PXO (= 24.576 MHz), and Sleep Clock (= 32 kHz)
  - 14 PLLs
- Dual channel LPDDR2
  - Each channel and bank LPDDR2 device can enter self-refresh/deep power-down
  - Activity-based self-refresh entry/exit in HSDDRx

SPM Usage in MSM8960

- SPM v2
  - board-msm8960.c msm_spm_data[] defined initial values for SPM and call msm_spm_init(msm_spm_data, ..)
  - Enables the AVS controller and sets its frequency and upper and lower voltage limits
  - spm-v2.c and spm-devices.c will replace spm.c (MSM8x60)
  - Use MSM_SPM_REG_SAW2_AVS_CTL register
  - If CONFIG_MSM_L2_SPM was defined, it also supports L2 Low Power mode
- avs_reset_delays()
  - Called from avs_init() avs.c that writes to AVSCSR and AVSDSCR to turn on AVS measurements
  - It requests for a new voltage adjustment optimized for new clock frequency through SPM block
  - Actual effect of this function depends on kernel configuration CONFIG_MSM_CPU_AVS(avs.c) and CONFIG_MSM_AVS_HW(avs_hw.S)

RPM in MSM8960

- RPM is a hardware block required for managing shared resources to attain the lowest dynamic and static power profile.
- It operates with low latency and low power.
  - RPM communicates directly with processors and/or hardware accelerators in each subsystem to process and coordinate the shared power and resource requests.
  - Shared resources can be turned on or off and scaled on demand.
  - RPM manages power intelligently by processing data from processors, resources, applications, systems, and various monitors, such as temperature and bus.

RPM in MSM8960 (cont.)

- RPM allows independent control of subsystems without any given subsystem being active.
  - For instance, the apps processor can go to sleep and wake up independently without impacting the modem processor.
  - This significantly reduces the request delays, interprocessor coupling, and allows significant power reductions.
  - RPM communicates with each subsystem's hardware modules through the dedicated 16 KB message RAM, which consists of 32 partitions of 512 bytes each.
  - Separate security permissions shall be enforced for each of the 32 512-byte partitions in the message RAM.
  - Each message RAM partition or a resource group shall be protected by an XPU configured as an RPU.

RPM in MSM8960 (cont.)

- Key API functions
  - int msm_rpm_get_status() – Gets the RPM's status through message RAM
  - int msm_rpm_set() – Issues a resource request to RPM to set resource values
  - int msm_rpm_clear() – Issues a resource request to RPM to clear resource values; once the values are cleared, the resources revert back to their default values
  - int msm_rpm_register/unregister_notification()
  - int msm_rpm_ack_interrupt() – ISR for interrupt from RPM
  - void msm_rpm_send_req_interrupt() – Notifies the RPM through SCSS_I PC register; triggers the interrupt in RPM
- rpm-resources.c
  - To use msm_rpm_set() and clear(), need to know exact resource ID and value
  - Wrapper functions to use msm_rpm_set() and clear() easier
    - msm_rpmrs_pxo, msm_rpmrs_l2_cache, msm_rpmrs_vdd_mem, msm_rpmrs_vdd_dig

RPM in MSM8960 (cont.)

- Related source files
  - arch/arm/mach-msm/board-msm8960.c
  - arch/arm/mach-msm/include/mach/msm_iomap-8960.h
  - arch/arm/mach-msm/io.c
  - arch/arm/mach-msm/rpm.c
  - arch/arm/mach-msm/rpm.h
  - arch/arm/mach-msm/rpm-resources.c

Linux Power Modes

- Power modes supported
  - Running
  - Suspend – Invoked by the user
    - Power collapse (with RPM)
    - Standalone power collapse (without RPM)
    - SWFI only – Apps execute SWFI instruction
  - Idle
    - Power collapse (with RPM)
    - Standalone power collapse (without RPM)
    - SWFI only – Apps execute SWFI instruction
    - Spins – Apps CPU spins, i.e., perpetual loop

Regulator and Consumer

- Regulator – Hardware perspective
  - Only PMIC LDOs could supply actual power
- Regulator – Software perspective at kernel
  - Control of PMIC LDO is the “regulator”
  - Even though there is only one hardware regulator, multiple entities could control it
  - rpm-regulator-8960.c/saw-regulator.c/pm8921-regulator.c
    Same hardware LDO registered at each regulator file
- Consumer
  - To decide which regulator will be used to control specific component power, i.e., a consumer; e.g., Krait 0 and Krait 1 core (8921_S5, S6)
    - The consumer for Krait 0 and Krait 1 registered to SPM regulator as their supplier; when regulator_get() or put() is called, spm-regulator.c will be used; refer to acpuclock-8960.c

Runtime Power Management

- System power management (traditional suspend/resume)
  - System-wide suspend and resume for all devices
  - All or nothing
  - Initiated by user space
  - Con – When the device is active, all devices should be turned on even though some devices are not used
- Runtime power management, pm_runtime
  - New power management framework is merged in Kernel 2.6.32
  - Independent power management of devices at runtime
    - Allows devices to have local suspend/resume controlled by the driver
    - Allows idle device to suspend itself
  - Ensures that one device does not prevent other devices from performing power management
    - Idle devices can enter and suspend without waiting for others

Runtime Power Management (cont.)

- With LDM
  - Runtime power management is not just used for putting a device into a Low Power state; also used to indicate device activity to its parent in LDM (usually platform or bus to which the device is registered)
  - At a minimum, drivers should report any activity state that the parent device (MSM bus driver) can enter its Low Power state
  - Depends on hierarchy in bus architecture

CPUidle

- CPUidle is a kernel power management infrastructure
- CPUs today support multiple idle levels differentiated by varying exit latencies and power consumption during idle
- CPUidle allows management of these different idle CPUs in an efficient manner
- CPUidle driver handles architecture or platform-dependent part of CPU idle states
- Initializes cpuidle_device structure for each CPU device and registers with cpuidle using cpuidle_register_device
- Supports two governors
  - Ladder and menu (default)

CPUidle (cont.)

- Legacy (MSM7x27, MSM7x30) pm2.c

swapper → cpu_idle() → pm_idle()

→ arch_idle()   pm2.c

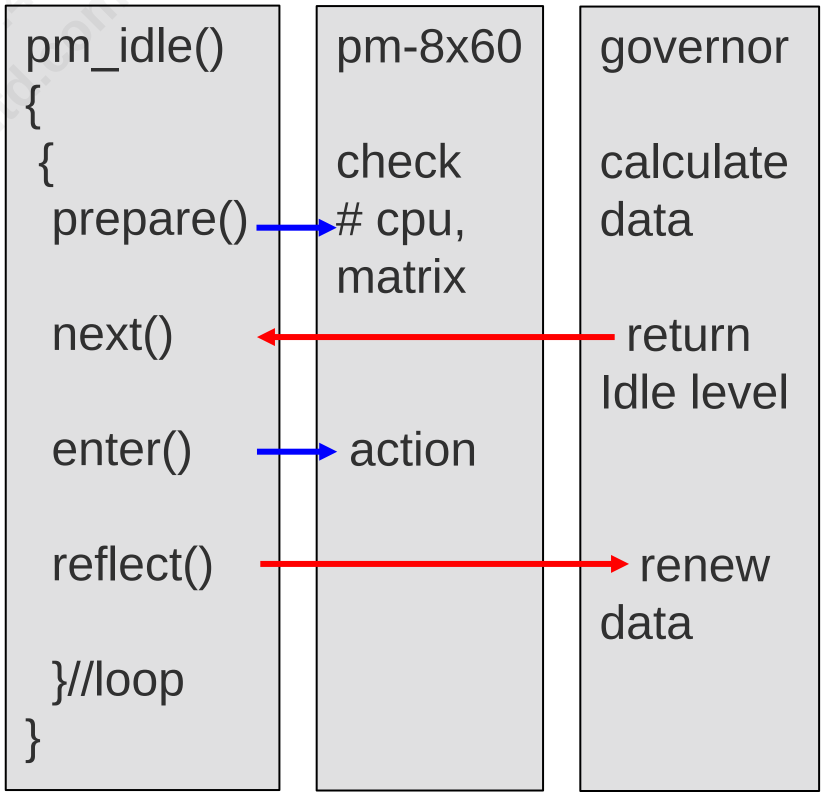
Check mode
Calculate next timer, sleep limits

Is power-collapse possible?
No → Is swfi possible?
No → for(;;)

- CPUidle (MSM8x60, MSM8960) pm-8x60.c

swapper → cpu_idle() → pm_idle()

CPUidle core override pm_idle()



CPUidle – MSM8960 Example

- arch/arm/mach-msm/board-msm8960.c

```
static struct msm_cpuidle_state msm_cstates[] __initdata = {
      {0, 0, "C0", "WFI",
              MSM_PM_SLEEP_MODE_WAIT_FOR_INTERRUPT},

      {0, 1, "C1", "STANDALONE_POWER_COLLAPSE",
              MSM_PM_SLEEP_MODE_POWER_COLLAPSE_STANDALONE},

      {0, 2, "C2", "POWER_COLLAPSE",
              MSM_PM_SLEEP_MODE_POWER_COLLAPSE},

      {1, 0, "C0", "WFI",
              MSM_PM_SLEEP_MODE_WAIT_FOR_INTERRUPT},

      {1, 1, "C1", "STANDALONE_POWER_COLLAPSE",
              MSM_PM_SLEEP_MODE_POWER_COLLAPSE_STANDALONE},
};
```

CPUidle – MSM8960 Example (cont.)

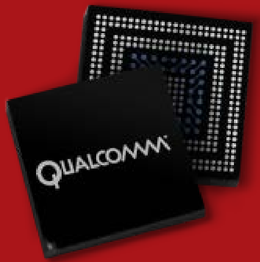
- Why is there no C2 on CPU1?
  - One master at RPM for all apps core (Krait 0 and Krait 1)
    - RPM does not have information about which Krait core uses which resource
  - C2 is for save power of shared resources

Krait 0 idle	Krait 1 idle	Notify RPM (Turn off resources)
No	No	No
Yes	No	No
No	Yes	No
Yes	Yes	Yes

  - Not necessary to notify RPM from both Krait cores
  - Krait 0 will take the role

Dynamic Memory Management (DMM)

- Make sure the DMM is enabled in the kernel; this allows LPDDR2 to enter self-refresh or deep Power-down mode when Krait enters power collapse
  - EBI1 LPDDR2 will be active (no power saving)
    - CONFIG_MSM_MEMORY_LOW_POWER_MODE_IDLE_ACTIVE
    - CONFIG_MSM_MEMORY_LOW_POWER_MODE_SUSPEND_ACTIVE
  - EBI1 LPDDR2 will be in Self-refresh mode (power saving)
    - CONFIG_MSM_MEMORY_LOW_POWER_MODE_IDLE_RETENTION
    - CONFIG_MSM_MEMORY_LOW_POWER_MODE_SUSPEND_RETENTION
  - EBI1 LPDDR2 will be in the lowest Power mode (data will be lost)
    - CONFIG_MSM_MEMORY_LOW_POWER_MODE_IDLE_DEEP_POWER_MODE
    - CONFIG_MSM_MEMORY_LOW_POWER_MODE_SUSPEND_DEEP_POWER_DOWN



Android Software Power Management Optimization

Take Advantage of Runtime Power Management

- Need to implement basic routines to register with runtime_pm core and indicate activity state; this brings your device into smart power management state whenever it is needed
  - pm_runtime_init/pm_runtime_remove()
    - Registers/unregisters pm_runtime client
    - Executed automatically from device_register() / unregister()
  - pm_runtime_enable/pm_runtime_disable()
    - Temporarily disables/enables again pm_runtime for specific driver

Take Advantage of Runtime Power Management (cont.)

- Helper function usage at device driver
  - Do not need to implement function body
  - pm_runtime_get() / pm_runtime_get_noresume()
    - Indicates to power management core that device is in use
    - Increments use count
    - Calls pm_runtime_resume() – Hardware-related, such as IRQ, regulator
  - pm_runtime_put() / pm_runtime_put_noidle()
    - Indicates to power management core that device is not in use
    - Decrements use count
    - Calls pm_runtime_idle() – Hardware-related, such as IRQ, regulator

Take Advantage of Runtime Power Management (cont.)

- Make sure these callback functions are implemented correctly
  - Implement function body
  - pm_runtime_resume()
    - When the callback completes, the power management core treats the device as active and fully functional, implying that the device can complete its I/O operations.
  - pm_runtime_idle()
    - Expected action for this callback is to check whether the device can be suspended; then queue the suspend request for that device.
    - Callback is executed by the power management core for the specified device when the device appears to be idle.
    - Return Yes or No.
  - pm_runtime_suspend()
    - If pm_runtime_idle returned Yes, pm_runtime calls pm_runtime_suspend() after checking use counter.
    - When the callback completes, the power management core treats the device as suspended, which means the device will not process data or talk to the CPU.

Monitoring CPU Frequency

- SYSFS for CPU frequency
  - /sys/devices/system/cpu/cpuX/cpufreq/time_in_state – Information on how long each CPU has been configured for each CPU frequency
  - /sys/devices/system/cpu/cpuX/cpufreq/cpuinfo_min_freq
  - /sys/devices/system/cpu/cpuX/cpufreq/cpuinfo_max_freq
  - /sys/devices/system/cpu/cpuX/cpufreq/scaling_cur_freq
  - /sys/devices/system/cpu/cpuX/cpufreq/scaling_governor

PM QoS

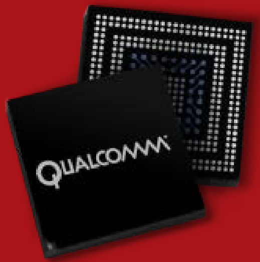
- QoS parameters (or services)
  - Related with idle
  - Compare with sleep limit
    - If QoS latency is smaller than sleep limit, idle will be executed even in power collapse
  - Used by drivers to register their interrupt latencies
  - Could be modified from user space, e.g., network daemon
  - PM_QOS CLASSES
    - PM_QOS_CPU_DMA_LATENCY
    - PM_QOS_NETWORK_LATENCY
    - PM_QOS_NETWORK_THROUGHPUT
- Related source code
  - pm_qos_params.h/pm_qos_params.c
- Give up the requirement on PM_QOS_CPU_DMA_LATENCY if configured, i.e., ensure that it is not set to a very low value; this allows apps to go into idle power collapse

PM QoS (cont.)

- QoS API (called from drivers)
  - pm_qos_add_request()
  - pm_qos_update_request()
- Examples are:
  - MSM7x27, MSM7x30 – SPI, I2C (384 kbps), USB_OTG
  - MSM8x60, MSM8960 – ETM, KGSL

CPUidle Monitoring

- Debugging
  - /sys/devices/system/cpu/cpu0/cpuidle (see [R1])
  - http://www.lesswatts.org/projects/powertop/
    - PowerTOP is a Linux tool that helps find programs that are misbehaving while your computer is idle
- More on CPUidle driver and framework
  - Documentation/cpuidle/
  - CPUidle subsystem (see [R2])
    - Author: Jonathan Corbet, Apr 26, 2010
    - http://lwn.net/Articles/384146/
  - Introducing cpuidle: core cpuidle infrastructure (see [R3])
    - Author: Venkatesh Pallipadi <venkatesh.pallipadi@intel.com>
    - http://lwn.net/Articles/221791/



Power Management Optimization

Sleep Current Optimization

- Ideally, no more than the sum of leakage currents of all components on the board
- Expected sleep current on customer phones is typically different from that on the Modem Test Platform (MTP) with the same chipset due to board-level differences
- At the beginning of debugging, the key is to arrive at the expected sleep current on the customer device based on MTP consumption; this helps to set the optimization goal
  - Ensure TCXO shutdown works; if not, debug it
  - Conduct breakdown measurements on the customer device
  - Ensure all GPIOs are configured to their appropriate sleep states

XO Shutdown Debugging

- RPM log example

```
142.842896: rpm_shutdown_req (master: "APSS") (core: 0)
142.842926: rpm_shutdown_ack (master: "APSS") (core: 0)
.
.
142.845093: rpm_xo_shutdown_enter (count: 403) (planned_duration:
0x00000013)
187.686523: rpm_xo_shutdown_exit
```

- NPA dump example

```
npa_resource (name: "/xo/cxo") (handle: 0x3B51C) (units: Enable)
 (resource max: 1) (active max: 1) (active state 1)
 (active headroom: 0) (request state: 1)

  npa_client (name: LPASS) (handle: 0x3BFD4) (resource: 0x3B51C)
    (type: NPA_CLIENT_REQUIRED) (request: 0)
  npa_client (name: Modem) (handle: 0x3C00C) (resource: 0x3B51C)
    (type: NPA_CLIENT_REQUIRED) (request: 1)
  npa_client (name: Scorpion 0) (handle: 0x3C044) (resource: 0x3B51C)
    (type: NPA_CLIENT_REQUIRED) (request: 0)
.
end npa_resource (handle: 0x3B51C)
```

XO Shutdown Debugging (cont.)

- Check /proc/msm_pm_stats
  - If power_collapse occurred
- RPM logs
  - Check RPM logs (see [Q2])
    - Mount debugfs
      - » adb shell mkdir /data/debug
      - » adb shell mount -t debugfs none /data/debug
    - Dump ULog to file on host computer
      - » adb shell cat /data/debug/rpm_log > C:\path_to_dump_to\dump.txt
      - » Press Ctrl-C to exit adb and stop the dump
    - The dump.txt file has hex values and can be converted to human readable for using the following python script:
      - » /modem_proc/core/power/rpm/dal/scripts/rpm_log.py
  - Recommend using TRACE32 (T32) RPM dump to get accurate RPM logs
    - USB for adb prevent XO shutdown
    - RPM logs can save logs only for few seconds

If XO Shutdown Working but Sleep Current High

- If TCXO shutdown works, the MPM block runs on the sleep clock, and clocks to the rest of the blocks are off.
- The key is to shut off as many blocks as possible and configure the rest of them to their LPM settings.
  - To determine which blocks are draining the most current, it is necessary to do a battery current breakdown by measuring current sourced by the individual PMIC voltage regulators.
  - Qualcomm hardware apps can assist in reviewing the schematic to find out the potential breakout points on the customer’s device. Breakout points on MTP are available via the MTP power tree.
  - It would be very beneficial to come up with a power tree for the customer’s device that shows additional components compared to MTP.

What to Look for in Sleep Current Breakdown Measurements

- In general, see if all the breakdown currents are the same as MTP for hardware components that are mapped exactly in the same manner as MTP.
- For other additional components (per the corresponding datasheet), check if the current consumption is the same as the estimated value.
  - If not, see if that component is configured for LPM by configuring its registers appropriately and/or GPIOs associated with it.
- Check if all the power features are working correctly.
  - Check the voltage on MSMC; if it is ~0.775 V, VDD minimization is working correctly.
  - Check if the RF regulators are at ~0 V; this ensures RF power collapse is working correctly.
- For further assistance on analyzing the breakdown, upload the breakdown measurements via a case to Qualcomm CDMA Technologies.

Pad Current Optimization

- GPIOs must be configured to draw the lowest possible current during sleep; this involves:
  - Appropriate GPIO configuration – Configure parameters such as direction (I/O), pull (up/down), and drive strength appropriately
  - Avoid high current drain situations such as:
    - Conflict, i.e., MSM and peripheral try to set GPIO at different states; this leads to direct path from VDD to GND
    - Floating, e.g., leaves the input GPIO pin without a definite state (high or low); in this state, there might be a high leakage current
  - Driving the appropriate GPIO output state – For those GPIOs configured as outputs, set the appropriate sleep state (high or low)
- For hardware guidelines on configuring GPIOs, see [Q4]

Pad Current Optimization (cont.)

- At MSM7x27, MSM7x30, QSD8x50
  - Check PRIMARY_CONFIGS/SLEEP_CONFIGS at modem side
- From MSM8x60/MSM8960
- Linux GPIO
  - GPIOMUX_ACTIVE/GPIOMUX_SUSPENDED – Reference count base
    - gpio_get() – Increment ref count; when 0→1, ACTIVE config will be set
    - gpio_put() – Decrement ref count; when 1→0, SUSPENDED config will be set
  - MSM8660 – gpiomux-8x60.c msm_gpiomux_install()
  - MSM8960 – There is *no* gpiomux-8960.c board-msm8960.c msm_gpiomux_install()
- Modem GPIO
  - Set RPM T32 break at xo_shutdown_enter(); contact Qualcomm CDMA Technologies
  - ./core/systemdrivers/tlmm/t32/tlmm_gpio_8960.cmm
    - Prints configuration register value for all GPIO
  - Refer to the solution #15542

Power Goal

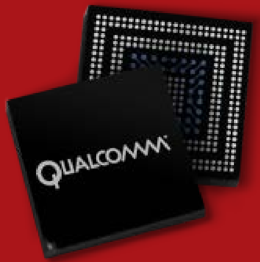
- Feature Complete (FC) 1020 (Oct, 2011)

QCT CDP	Current (mA)
Sleep	1.6
WCDMA Standby	2.4
GSM Standby	2.5
LTE Standby	2.6
MP3 playback	18.0
BT Standby	3.1
WLAN Standby	3.2
Accel Sensor Background Processing (1 Hz)	1.9

- QCT-released power number on QCT CDP for FC/Commercial Sample (CS)
  - See [Q5] for power number details

References

Ref.	Document	
Qualcomm		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*Presentation: MSM8660™ Linux Power Management Debugging*	80-VR629-2
Q3	*Presentation: MSM8660™ Linux Power Management Details*	80-VR652-2
Q4	*Configuration of Input Pins During Device Sleep*	80-VN499-7
Q5	*MSM8960/MSM8270/MSM8X60A ASIC and CDP8960 Current Consumption Data*	80-N1622-11
Resources		
R1	*PowerTOP* *Debugging – /sys/devices/system/cpu/cpu0/cpuidle*	http://www.lesswatts.org/projects/powertop/
R2	*The cpuidle subsystem (Apr 26, 2010)*	http://lwn.net/Articles/384146/
R3	*Introducing cpuidle: core cpuidle infrastructure*	http://lwn.net/Articles/221791/



Questions?

https://support.cdmatech.com