Power Management

Technical Design Specification

Author: Sylvester Yan
Last Updated: 04/01/2013

Executive Summary if applicable

This document introduces the traditional Android power management(including Linux PM), and explains the methods to improve the power management of a target system. As a part of a power management system, power supplies(including battery, and charger) are also involved in this document. This document guides the development engineers to achieve the power management goals. Also it helps the QA to define the test cases and test plan on power management for the target system.

Document History

Date	Author	Summary of Change	Note
04/01/2013	Sylvester Yan	Initial version	

1 Introduction

This document gives several parts of methods to improve the power management of target system. One purpose is to optimize the system configuration and behavior based on the power saving principles and mechanism to achieve long battery use time. The other purpose is on the system power itself, which improves the performance and user experience of the traditional battery and charge system, but itself does not benefit on the power life cycle.

1.1 Goals and objectives

Optimize the power consumption on running mode and sleep mode to prolong the battery life. Improve the battery and charge component to provide good performance and user experience on battery fuel gauge and battery charge management.

1.2 Statement of scope

Run-time PM optimizes the power consumption during running state, in which the target system is in active running state, but the unused sub-systems are in idle state or power off state and resume to power on and be active if requested to be used. The whole system goes to sleep state in which all the devices are powered off or in its low power mode and only wake-up related circuits are powered, so as to low down the main power consumption to be a little. The formal one improves the battery life duration of system operation, and the later optimizes the standby duration of the system.
The battery and charger are contained in the power supply management system, which provides more accurate battery fuel gauge algorithm, takes responsibility to manage the battery and charger state, and charge battery. It calculates and reports accurate battery life, and manages the charger and charging procedure.

1.3 Software context

Run-time PM locates at the Linux Kernel space, and the devices those are able to support run-time PM implement its run-time PM method according to its functional requirements in its driver module. The devices driver should put the devices on right state in which devices consume the least power in inactive or sleep mode, such as shutdown module power, go to its low power mode, and configure the I/O pins.
The battery and charger management also runs on Kernel space as module driver. The battery driver and charger drivers take responsibility for the battery fuel gauge and charging battery respectively. The battery and charger state information is obtained from local I/O or RPC from other processors.

1.4 Major constraints

As the Qualcomm chips (MSM7XXX, MSM8X25, MSM8X55, MSM8X60/MSM8X30) have independent software, they provide different resources to be used for. There may be different way to implement the same feature on different platforms. The software architectures are similar on MSM7XXX, MSM8X25, and MSM8X5, while it is totally different with MSM8960/MSM8930. PMIC chip and ADC also limit the resource that can be used.

2 Architectural and component-level design

2.1 Program Structure

2.1.1 Architecture diagram

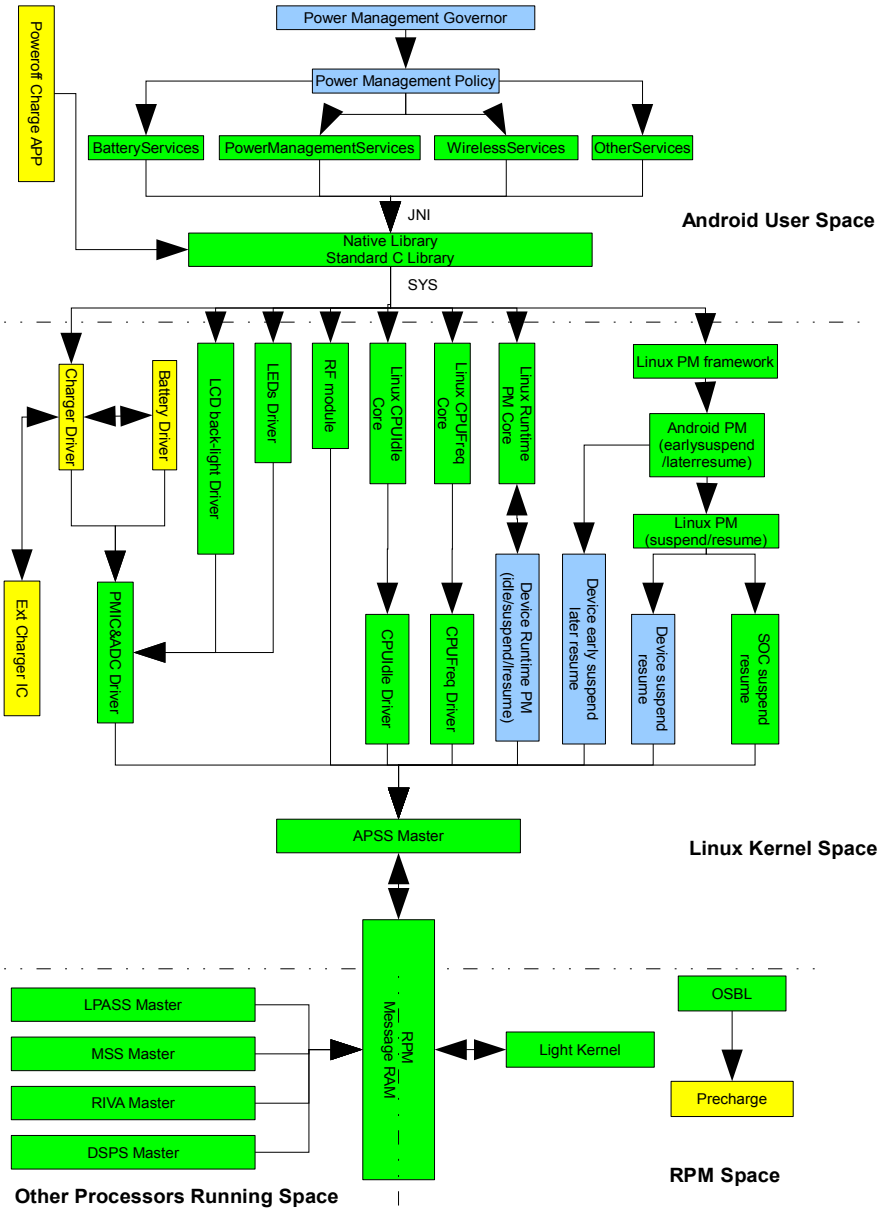

Chart 1: High Level Power Management Architecture

Yellow color represents the components for battery and charge management.
Blue color represents the components those need customization work to do for power saving optimization.
Green color represents the fundamental components from Android or Qualcomm.

2.1.2 Power Management Governor and Policy

Those components locates at the Android as an APK running in the system to help the terminal users to make a decision on system work mode for different application environment and user's preference.

2.1.2.1 Power Management Resources

The power management resources includes the configurations or settings those can impact the system running behavior directly or indirectly, whereas the system power consumption would be reduced more.

1. *LCD backlight adjustment*
2. *Screen timeout delay controlling*
3. *Dynamic wall paper installation*
4. *Sensors controlling*
5. *BT/WIFI/GPS controlling*
6. *2G/3G configuration*
7. *Periodic task controlling*

2.1.3 CPU Idle

CPUidle is a kernel power management infrastructure to support multiple idle levels differentiated by varying exit latencies and power consumption during idle.

It separates out drivers that can provide support for multiple types of idle states and policy governors that decide what idle state to use at Runtime.

The main advantage of this infrastructure is that it allows independent development of drivers and governors and allows for better CPU power management.

The reference [1] has more detailed information about the CPU idle feature.

2.1.4 CPU Freq

CPU frequency scaling (CPUFREQ) is PM method used in Running mode, which is so call dynamic power management.

The reference [2] has more detailed information about the CPU Freq feature.

2.1.4.1 CPU Frequency Scaling Governors

Performance – CPU runs at static maximum frequency.
Powersave – CPU runs at static minimum frequency.
User space – Determined by static user space program.
Ondemand – On-demand dynamic governor sets target frequency based on CPU busy/idle statistics.
Conservative – Conservative dynamic governor sets target frequency, based on CPU busy/idle statistics.

2.1.4.2 Dynamic Governor parameters

Up Threshold - CPU usage percent threshold for freq rising.
Down Threshold - CPU usage percent threshold for freq falling.
Sampling Rate - CPU usage profile sampling rate.

2.1.5 Traditional Power Management

The Linux traditional PM is initiated by user space, and it is a system wide suspend/resume, whereas any device can prevent system suspending and all devices will resume at wakeup.

2.1.5.1 Algorithmic model

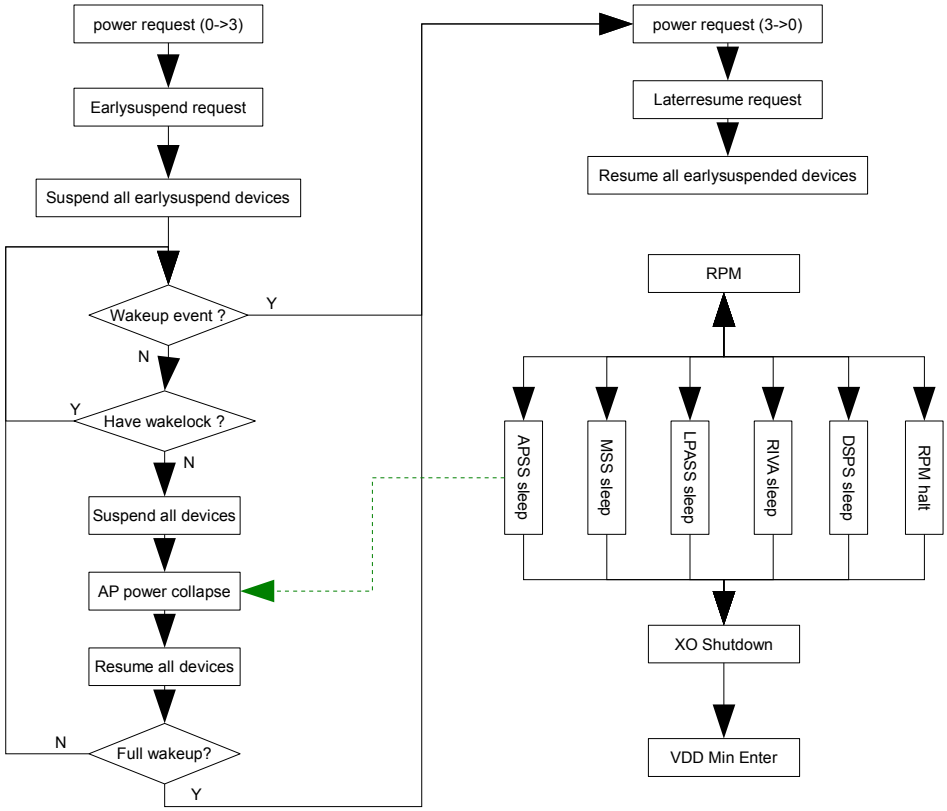

Chart 2: System Sleep Work Flow

The devices owner has to handle the device suspend contents according to the specific device HW design. As unsuitable settings of the device will cause undesirable current leakage.

The reference [4], [5], [6], [7] have more detailed information about the power consumption optimization.

2.1.5.2 Low Power Mode

RPM Halt – Only RPM in halt mode.
XO Shutdown – AP CXO buffer(switch to sleep clock) and PXO are shutdown.
VDD Min Enter – VDD voltage is minimized.

2.1.5.3 Optimize Sleep Power Consumption

Work on the following items to optimize the sleep current:

1. Make sure the major subsystems on the MSM® chipset went into sleep.

Low Power Audio Subsystem (LPASS)
Applications Subsystem (APSS)
Modem subsystem (Q6SW-MSS)
RIVA subsystem
DSPS Sensor subsystem
Modem subsystem (Q6FW – MSS)

2. XO shutdown and VDD min.
3. All peripheral devices (outside of the MSM chipset) went into their lowest power state.

4. GPIOs/MPPs are configured to their suggested low-power configuration before going into sleep

2.1.5.4 Optimize Standby Power Consumption

Check the paging awake current in the following parts:
XO warm-up
RF Wakeup
RF processing
RF Sleep
System Sleep

2.1.5.5 Optimize Talk Power Consumption

Clocks are a main cause of power consumption in voice call scenarios. Stop the unnecessary clocks running during talk time.

2.1.6 Runtime Power Management

Runtim PM is a new power management framework merged in Ver 2.6.32 as an improvement for the traditional PM. It allows to manage the devices independently at Runtime. It is controlled by the device driver and cannot prevent other devices from PM.

The reference [3] have more detailed information about the Runtime PM.

2.1.6.1 Main Runtime PM API

Suspend the device when device is ready to suspend:
pm_runtime_suspend(dev)
pm_schedule_suspend(dev, delay)
Resume the device:
pm_runtime_resume(dev)
pm_request_resume(dev)
Idle the device when device has no work to do:
pm_runtime_idle(dev)
pm_request_idle(dev)
Get the device when has work to do on the device:
pm_runtime_get()
pm_runtime_get_sync()
pm_runtime_get_noresume()
Put the device after work done on the device:
pm_runtime_put()
pm_runtime_put_sync()
pm_runtime_put_noidle()

2.1.6.2 Runtime PM API Usage in Driver

Probe:
pm_runtime_enable()
probe/configure hardware
pm_runtime_suspend()
Activity:
pm_runtime_get()
Do work
pm_runtime_put()
Completion:
pm_runtime_suspend()

2.1.7 Battery Fuel Gauge and Charge

The battery driver takes responsibility for battery fuel gauge calculation. The charger driver maintains the battery charge state machine, and provides battery and charger status information and interfaces communicating with Android.

2.1.7.1 Algorithmic model

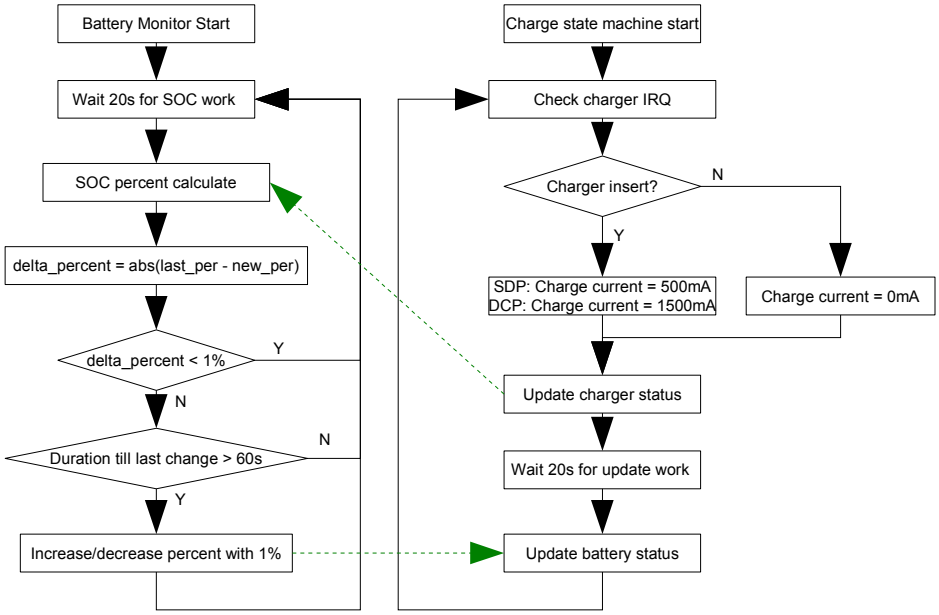

Chart 3: Battery Fuel Gauge and Charge Work Flow

The green lines represent notification/callback between battery driver and charger driver.

2.1.7.2 Battery Monitor System

BMS is an instrument used to report the remaining battery life to the end user. BMS calculates the state of charge (SoC) or remaining battery life using battery voltages, including open circuit voltage (OCV), and battery current.
The battery state of charge is reported by the BMS that supported by the Qualcomm' HW, and the detailed information about BMS can reference to the Qualcomm document [7]
There is another document [8] about the BMS that is used for early Qualcomm products(MSM7XXX, MSM8X25, MSM8X55) to improve the accuracy of the battery fuel gauge.

2.1.7.3 SOC Accuracy Improvement

Battery Profile Data Customization:
As for the target device, the battery and charger driver need to customize the battery and charger parameter according to the specific battery and charger used on the target device, so as to improve the accuracy of the battery fuel gauge.
Battery Parameters:
Capacity: mAh
Rbatt: mOhms

FCC percent_temp_ocv_lookup_table
NTC volt_temp_lookup_table
max_voltage
cool_temp_threshold
warm_temp_threshold
max_batt_chg_current
chg_terminal_current

Battery Remaining Capacity Linearly Process:

As the SOC reported by the BMS may have more than 1% changes comparing with old SOC, if that is the case, the actual SOC value report to the end user can be changed with a step by timing to achieve the value of BMS reported, but not do it at one time. That make the end user feel good as it acts like an ideal fuel gauge.

2.1.8 Battery Precharge

Precharge is used to support depleted battery charge. When battery voltage is lower than 3.4V, the system will boot to precharge mode when boot from USB charger. And boot will continue till the battery voltage reaches above 3.4V.

2.1.8.1 Algorithmic model

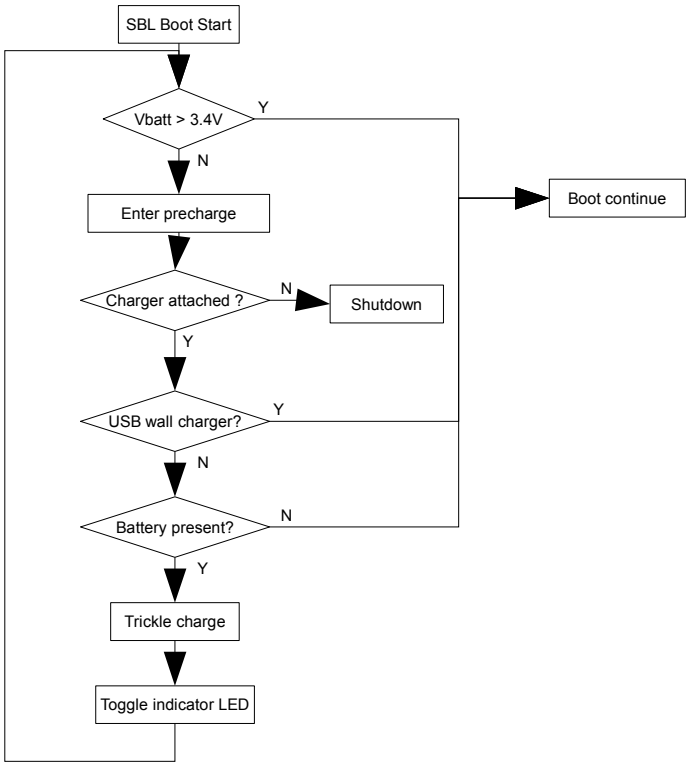

Chart 4: Battery Precharge Work Flow

The precharge runs at SBL context, which is in the early of system boot.
When battery voltage below 3.2V, the precharge will enter trickle charge mode with 200mA charge current. When it goes up to 3.2V, charge current will be 350mA ~ 500mA according to the USB enumeration. Finally, it will continue to boot when voltage reaches above 3.4V. At the precharge phase, device will not response any user operation, but only can be observed that it is in f006 USB mode.

USB wall charger will not be blocked at the precharge phase, as it is assumed that it has enough power to support system boot without battery.
If there is no valid battery detected, it will boot up powered by charger. If that is the case, the boot up may fail for over load of charger capability. However, it helps the factory test phase without battery for the main board.
The device will shutdown as long as the charger is removed.

2.1.9 Battery Poweroff Charge

When the system boots up to APP bootloader, it checks the power up reason. It will set boot argument for pause boot on charging if it is booting up from charger. Then the system enters Poweroff Charge mode, and displays battery charging animation.

2.1.9.1 Algorithmic model

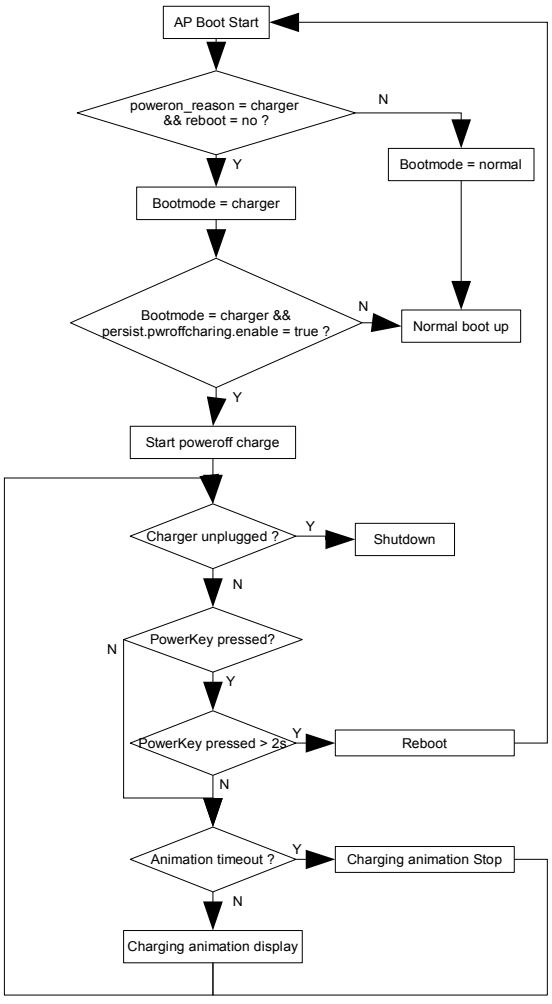

Chart 5: Poweroff Charge Work Flow

Here the so called Poweroff Charge is not the exact meaning of power off. Actually, the device is just in state that only some of critical services start up, and the charge task is running at front end.

In AP bootloader phase, there is boot argument set to indicate entering poweroff charge mode: *bootmode=charger,* or *bootmode=normal* otherwise. Only when the device power up reason is USB or USB-WALL and current boot is not a reboot. Before init task to decide load the power off charge task after checking *bootmode=charger*, it will check the system property *"persist.pwroffcharging.enable"* that is used to switch the power off charge functionality. Finally, both of conditions are satisfied, poweroff charge task runs.

The poweroff charge task will response some of user operations, such as power key press, or long press, and unplug charger. One time press/release will resume the charge animation from sleep, and long time press (> 2 seconds) will trigger a device reboot. Charger removal will lead the device to shutdown.

The charge animation will timeout to sleep with 3 display cycles.

3 Restrictions, limitations, and constraints

The BMS highly depends on the Qualcomm HW/SW support on MSM8960/MSM8930. For MSM7X27A/MSM8X25/MSM8X55 platform, the BMS has HW limitations which impact the SOC accuracy.

4 Testing Issues

4.1 Classes of tests

4.1.1 Dead battery charge(Precharging mode)

4.1.1.1 USB and AC charger are both able to charge battery

4.1.1.2 Device powers off when all chargers are unplugged

4.1.1.3 Boot continue when battery reaches above 3.4V

4.1.1.4 Red LED blinks(if support)

4.1.2 Poweroff charge(Poweroff charging mode)

4.1.2.1 USB and AC charger are both able to charge battery

4.1.2.2 Device powers off when all chargers are unplugged

4.1.2.3 Device is able to boot up when keep press power key for 2s

4.1.2.4 Charging animation is able to be on when press power key

4.1.2.5 System Logo disaply transits smoothly to charing animation without screen blink

4.1.2.6 Disable/Enable Poweroff charging feature via NV 6860 or property "persist.pwroffcharing.enable"

4.1.3 Sleep current consumption(Airplane mode)

4.1.3.1 Measurement sleep current with WIFI and BT off

4.1.3.2 Measurement sleep current with WIFI on

4.1.3.3 Measurement sleep current with BT on

4.1.4 Standby current consumption

4.1.4.1 Measurement standby current without SIM card

4.1.4.2 Measurement standby current using GSM card

4.1.4.3 Measurement standby current using 3G card

4.1.4.4 Measurement standby current with WIFI on

4.1.4.5 Measurement standby current with BT on.

4.1.5 Poweroff current consumption.

4.1.5.1 Measurement current after SW device power off

4.1.5.2 Measurement current after HW device power off.

4.1.6 Battery charing current

4.1.6.1 Measurement the charing current for USB-PC

4.1.6.2 Measurement the charing current for USB-WALL

4.1.6.3 Measurement the charing current for AC-DC(if support)

4.1.7 Charger Detection

4.1.7.1 USB host(SDP) charger detection

4.1.7.2 USB wall(DCP) charger detection

4.1.7.3 AC/DC charger detection

4.1.8 Battery Status Validation

4.1.8.1 Battery Temperature matches the actual temperature

4.1.8.2 Battery Status Bar matches the actual status

4.1.8.3 Battery Life Percentage matches the actual remained capacity

4.1.8.4 Low Battery triggers system auto power off.(battery life reaches 0% or battery voltage reaches power off threshold: can be programmable)

5 References

[1] *kernel/Documentation/cpuidle/*.txt*
[2] *kernel/Documentation/cpufreq/*.txt*
[3] *kernel/Documentation/power/runtime_pm.txt*
[4] *80-N9327-1_A_Power_Consumption_Optimization_Debugging_MSM8960.pdf*
[5] *80-VR652-2_C_MSM8x60_APQ8x60_Linux_Power_Management.pdf*
[6] *80-VR652-1_B_Linux_Pwr_Mgmt_Details.pdf*
[7] *80-VB073-17 BATTERY MONITORING SYSTEM (BMS) TRAINING APPLICATION NOTE.pdf*
[8] *Battery_Management_System.ppt*