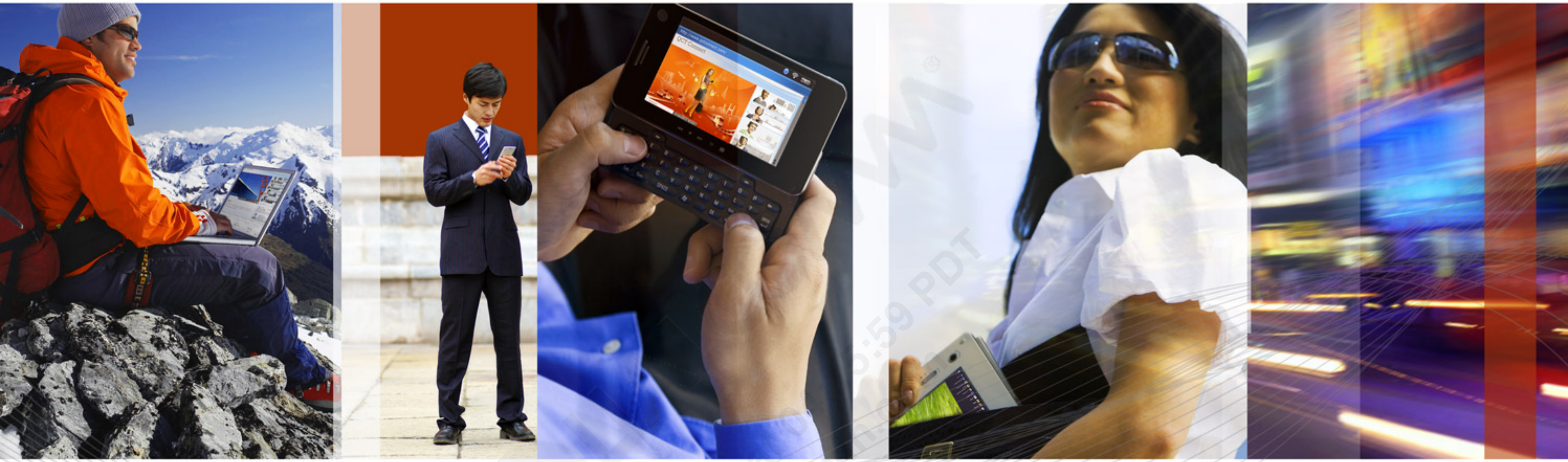


QUALCOMM CDMA Technologies

REDEFINING MOBILITY

MSM8660™ Linux Power Management Overview

80-N2190-1 B

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

Revision History

Version	Date	Description
A	Jul 2010	Initial release
B	Sep 2010	Added RPM overview diagram and information on Android™ power-management changes; updated information on SPM, runtime power management

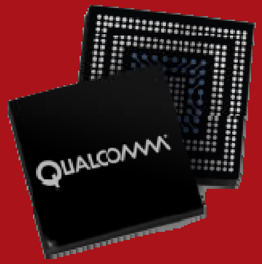
Contents

Introduction

REDEFINING MOBILITY

Introduction

- This document
  - Provides an overview of power management for the MSM8660™ ASIC
  - Provides an overview of the different Linux power modes supported
  - Discusses the Linux power management frameworks used in the MSM8660 chipset
  - Describes the new hardware blocks in the MSM8660 ASIC which help in power management, such as Resource Power Manager (RPM) and Subsystem Power Manager (SPM)
- This document does not discuss debugging of power management features; for debugging, see [Q3].
- For GPIO/input pin configuration to minimize sleep current, see [Q4].

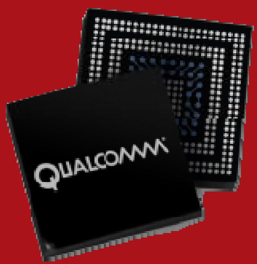


Hardware Power Enhancements

REDEFINING MOBILITY

Hardware Power Enhancements

- Dynamic clock and voltage scaling
  - Multiprocessor DCVS
  - Adaptive Voltage Scaling (AVS) for voltage level control
  - Dedicated RPM for shared resource management
  - 3 XOs and 12 PLLs for independent clock minimization
- Leakage minimization
  - Dedicated power rails for Scorpion 0/1, LPASS Hexagon™ processor, digital logic, and on-chip SRAM allow for independent power collapse and voltage minimization
  - Globally distributed foot switch control covers most major hardware blocks
- DDR optimization
  - Activity-based hardware-managed SDRAM self-refresh
  - Activity-based hardware management of DDR2 pad leakage and Low-Power operating mode

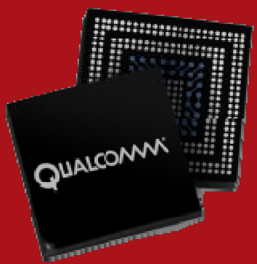


Linux Power Modes

REDEFINING MOBILITY

Linux Power Modes

- Power modes supported
  - Running
  - Sleep
    - Suspend power collapse for apps through SPM/RPM
    - Idle power collapse for apps through SPM/RPM
    - SWFI (only) – Apps execute SWFI instruction; no apps power collapse
    - Spins – Apps CPU spins; no apps power collapse



SPM

SPM

- SPM and AVS Wrapper (SAW)
- Hardware block that integrates SPM and AVS controller
  - Enables Scorpion subsystem to enter Clock Gating mode or Power Collapse mode without involving modem; apps core talks to SPM and RPM directly
  - Mode selection and configuration performed through SPM registers
  - Contains state machine that sequences clock control, clamps on/off, interacts with PMIC arbiter
    - Allows raising, lowering, and collapse of Scorpion voltage
    - Allows orderly entry into/exit from Scorpion clock gating and power collapse
    - Enables hardware-sequenced power collapse and RPM-SPM handshake
    - Signals RPM to allow RPM to put resources into low-power states
    - Creates lower and more deterministic entry and exit latencies

SPM (cont.)

- Contains an interface to the subsystem interrupt controller
  - Any pending interrupt kicks SPM state machine to restore Scorpion subsystem to Active mode
  - It can detect wakeup events via the interrupt controller
    - Wakeup events are for external peripheral wakeup signals within its subsystem
- Contains an interface to the Scorpion processor
  - Scorpion entering WFI state kicks the SPM state machine, which puts the Scorpion subsystem into Clock Gating mode or Power Collapse mode

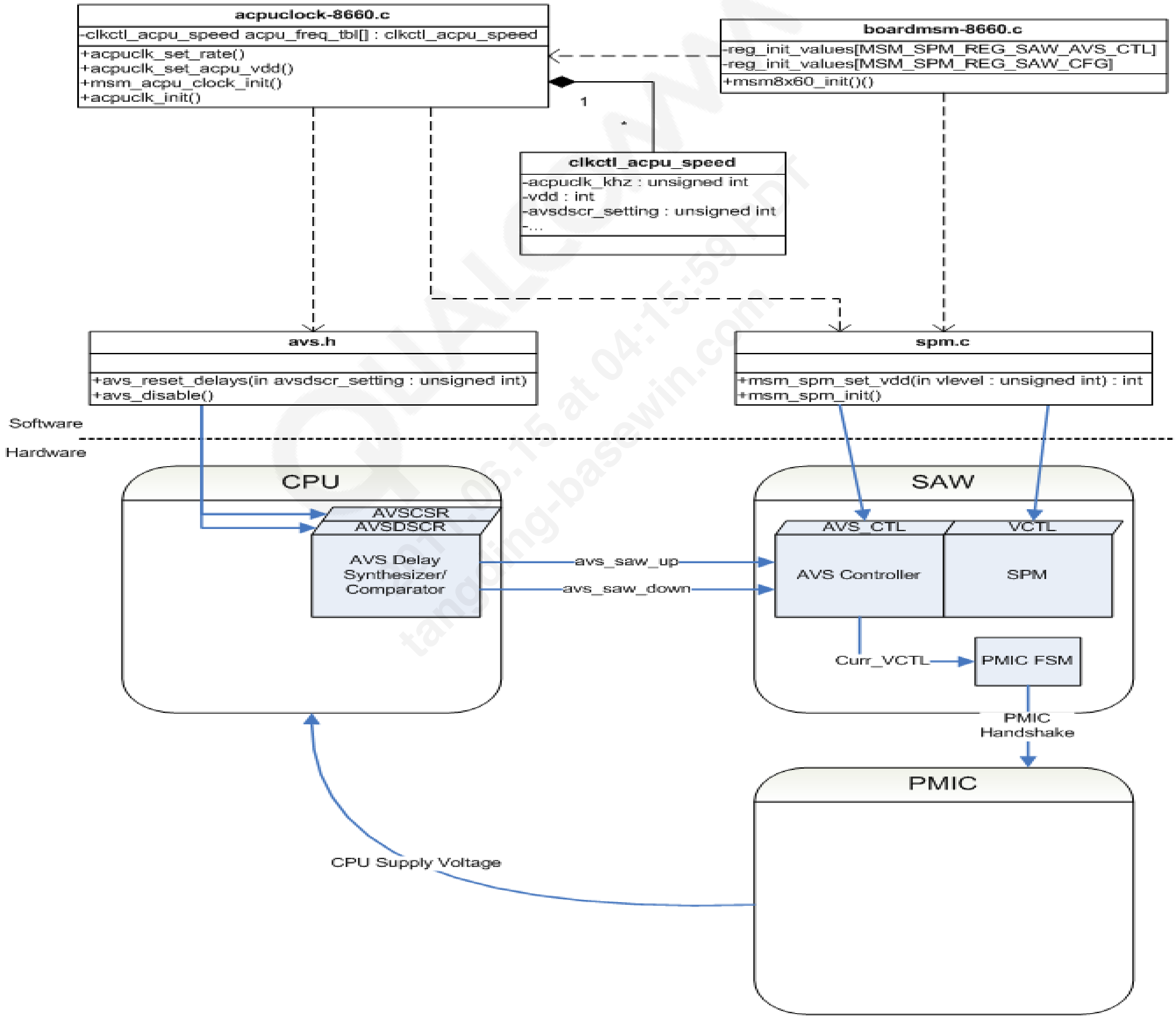
SPM (cont.)

- AVS controller is part of power management architecture
- Each core has its own AVS hardware; no software synchronization is necessary
- For specified CPU frequency, AVS seeks to operate CPU at its minimum safe voltage
- CPU hardware continually evaluates whether AVS voltage can be lowered or must be raised
- AVS controller periodically samples and responds to avs_saw_up() and avs_saw_down(), setting new CPU voltages as needed through PMIC FSM
- In steady state, there is no software interaction with hardware

SPM Usage in MSM8660

- Initial value for SAW_AVS_CTL in boardmsm-8660.c
  - Passed to spm.c msm_spm_init() and written to SAW_AVS_CTL register
  - Enables AVS controller and sets its frequency and upper and lower voltage limits
- msm_acpu_clock_init() sets initial voltage with write to SAW_VCTL
- avs_reset_delays(avsdscr) added to acpu_clock_init() writes to AVSCSR and AVSDSCR to turn on AVS measurements

SPM Usage in MSM8660 (cont.)



RPM

REDEFINING MOBILITY

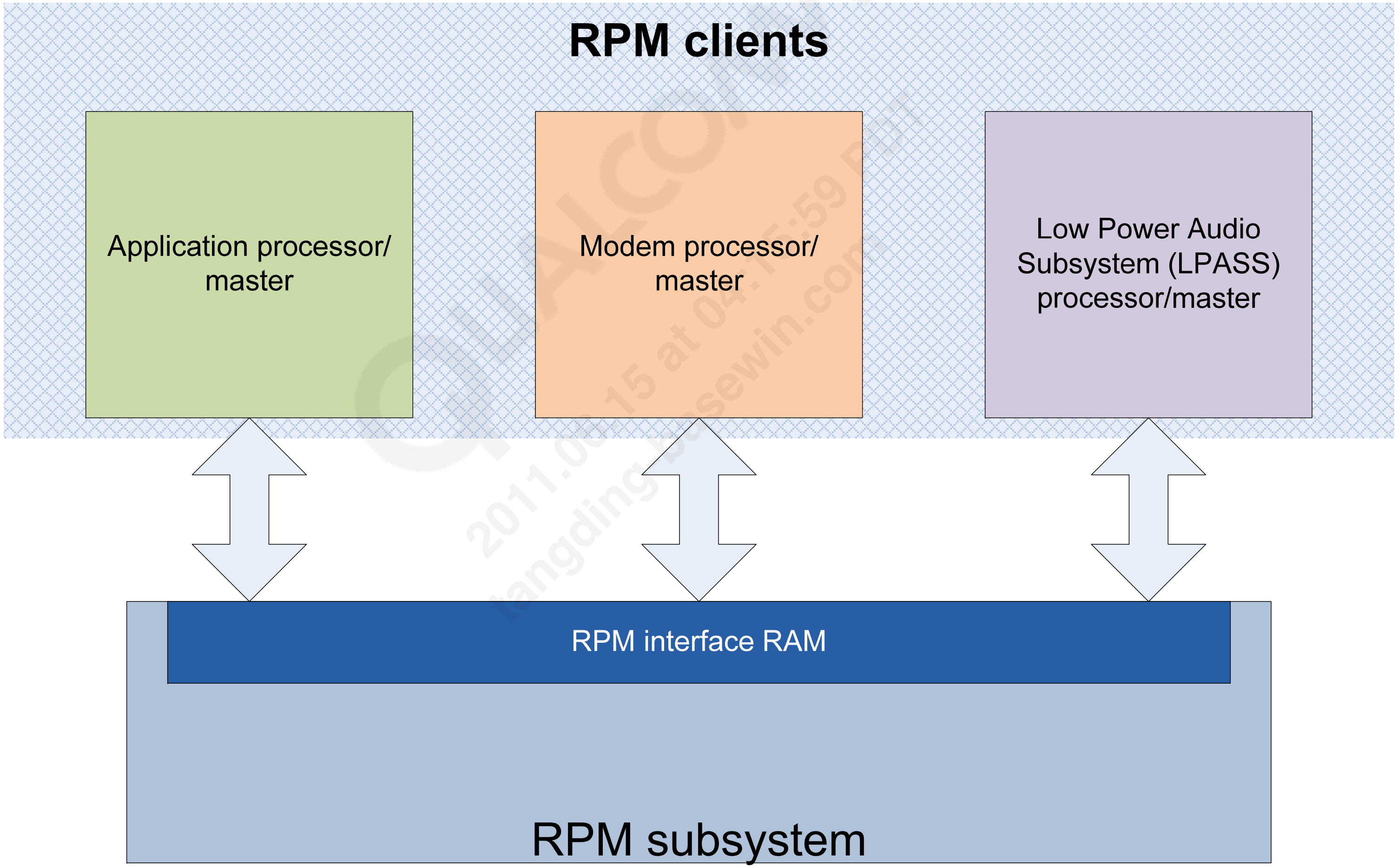
RPM

- RPM is a hardware block required for managing shared resources in order 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 shared power/resource requests.
  - Shared resources can be turned on/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.

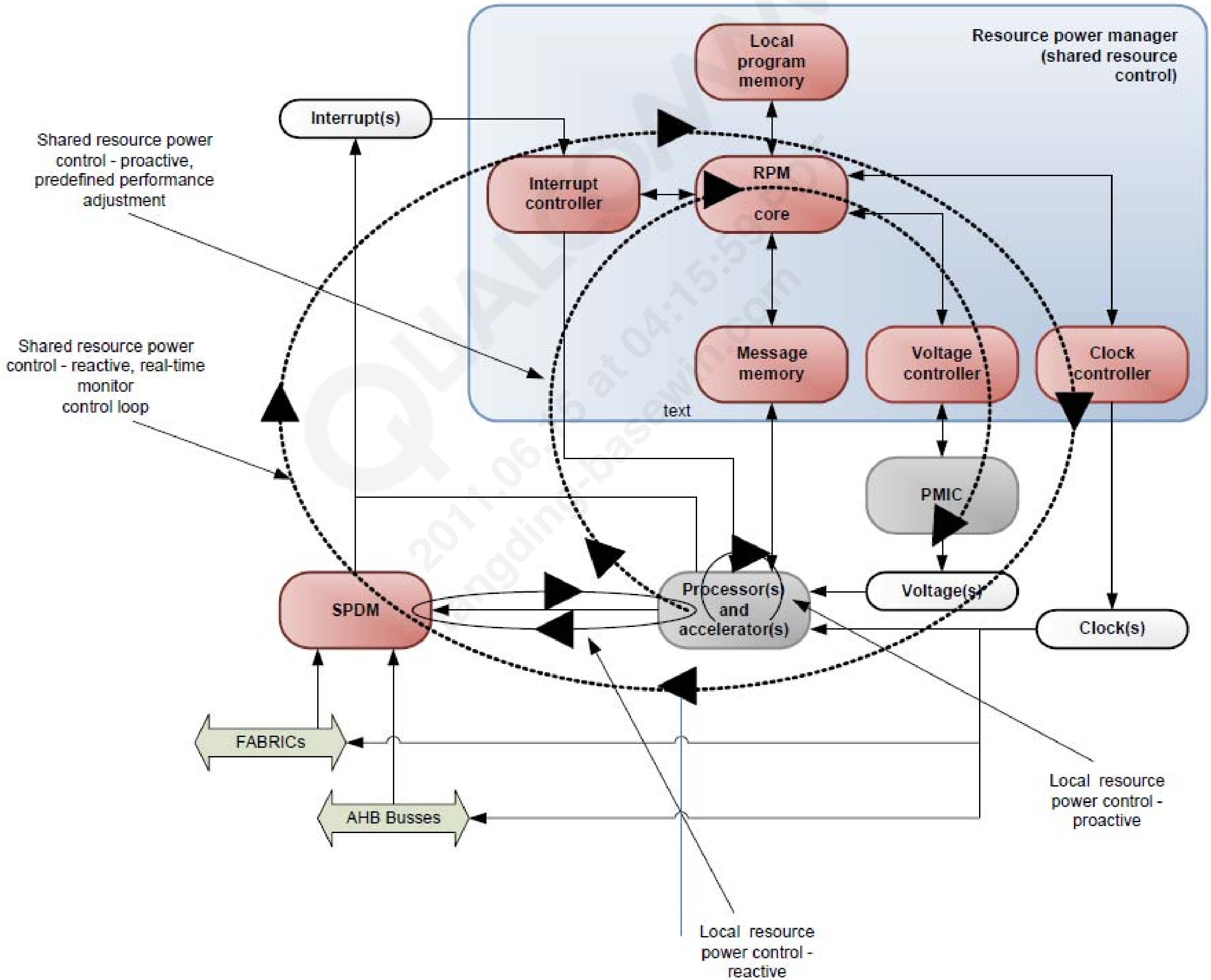
RPM (cont.)

- 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 and interprocessor coupling, and allows significant power reductions.
- For more details on RPM, see [Q2].

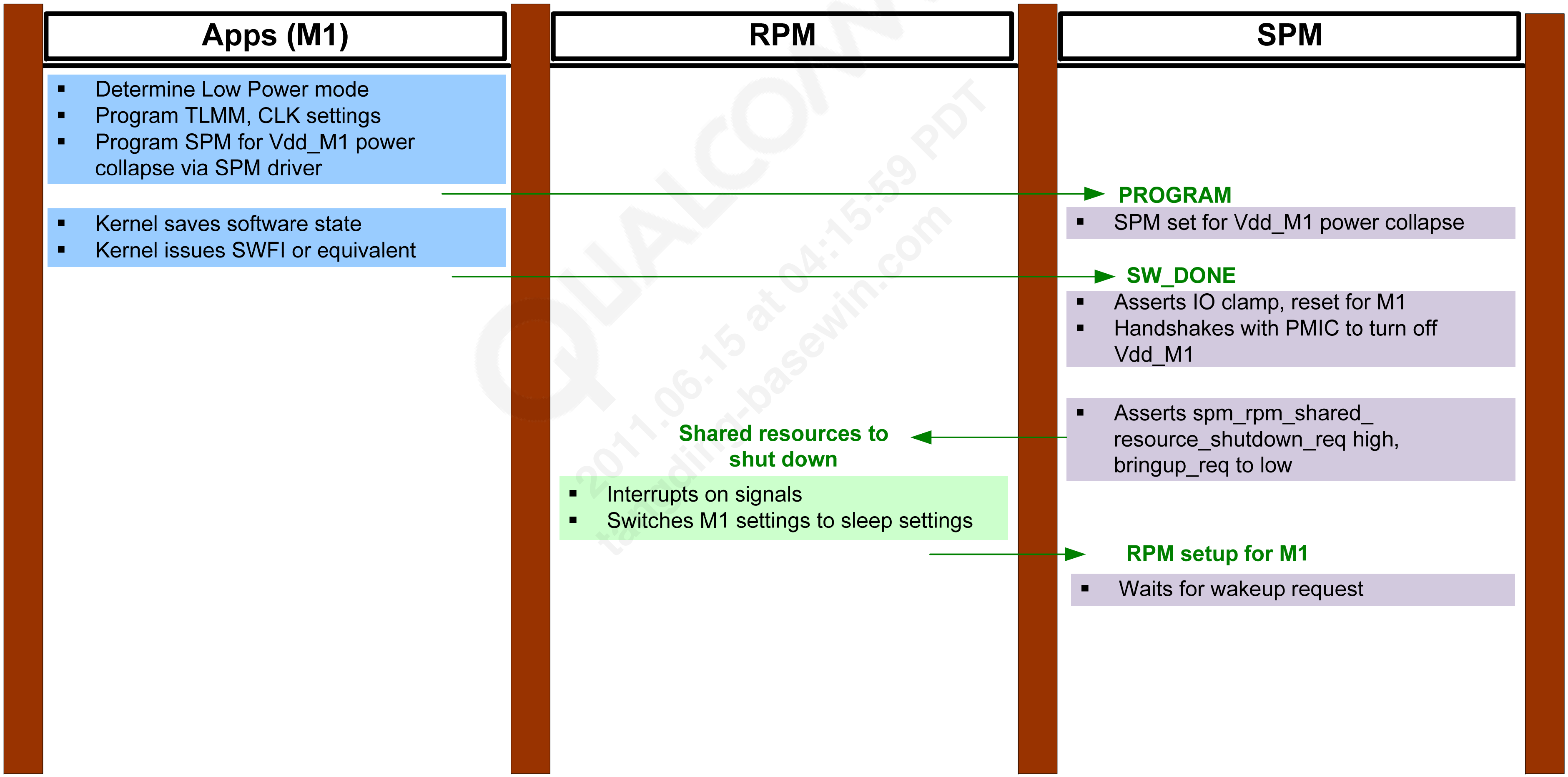
RPM Interface



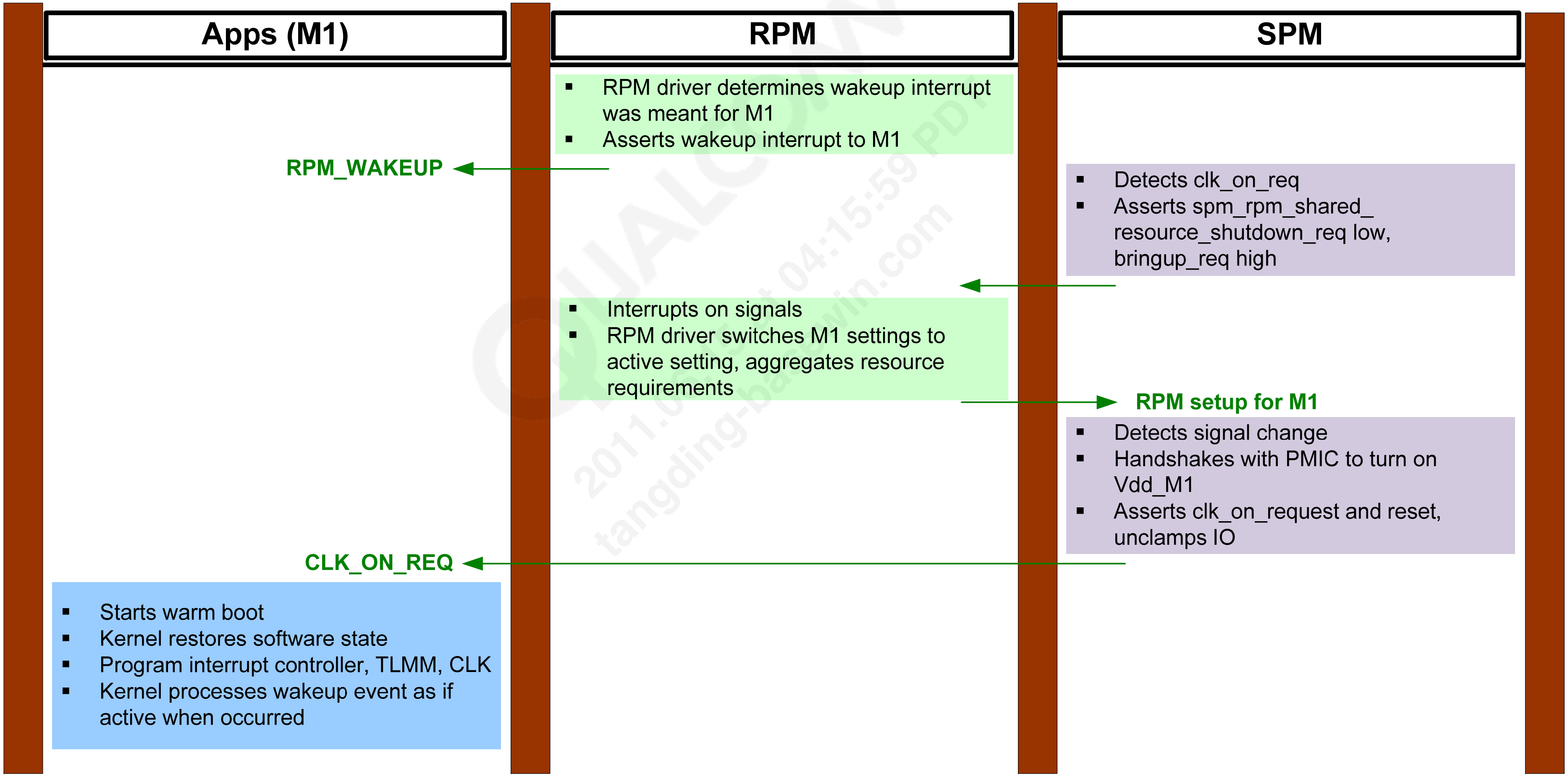
RPM Overview Diagram



Enter Power Collapse Flow



Exit Power Collapse Flow



Runtime Power Management

REDEFINING MOBILITY

Runtime Power Management

- System power management (traditional suspend/resume)
  - System-wide suspend/resume (all devices suspend/resume together)
  - Initiated by userspace
  - Con – Any device can prevent system suspend
- Runtime power management
  - New power management framework merged in Ver. 2.6.32
  - Independent power management of devices at runtime
    - Allows devices to have local suspend/resume controlled by driver
    - Allows idle device to suspend itself
  - Ensures that one device does not prevent other devices from performing power management
    - Idle devices can enter suspend without waiting for others
- Runtime power management kernel documentation available at http://lxr.linux.no/#linux+v2.6.32/Documentation/power/runtime_pm.txt (see [R1])
- Runtime power management presentation available at http://elinux.org/images/0/08/ELC-2010-Hilman-Runtime-PM.pdf (see [R2])

Runtime Power Management (cont.)

- Runtime power management not just used for putting device into a low-power state; also used to indicate device activity to its parent in LDM (usually platform or bus to which device is registered)
- At minimum, drivers should report activity state so that parent device (MSM bus driver) can enter its low-power state

Runtime Power Management (cont.)

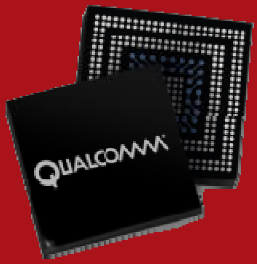
- Basic routines to register with runtime_pm core and indicate activity state
  - pm_runtime_init/pm_runtime_remove()
    - Register/unregister pm_runtime client
  - pm_runtime_get/pm_runtime_put()
    - Indicate to power management core whether device is in use
    - Increment/decrement use count
    - Call pm_runtime_resume()/pm_runtime_idle()
  - pm_runtime_enable/pm_runtime_disable()
  - pm_runtime_set_active/pm_runtime_suspended()

Runtime Power Management (cont.)

- Three new callbacks added for runtime power management usage in dev_pm_ops structure
- Called by runtime_pm state transition functions
  - pm_runtime_idle ()
  - pm_runtime_suspend()
  - pm_runtime_resume()
- These routines are used to put device into and out of a low-power state
  - Examples of actions to take include halting bus port, disabling clock or regulator, putting GPIO pin into high-impedance state, and flipping foot switch
  - Exact actions and sequence are specific to device

Runtime Power Management (cont.)

- pm_runtime_suspend()
  - When callback completes, power management core treats device as suspended, which means device will not process data or talk to CPU
- pm_runtime_resume ()
  - When callback completes, power management core treats device as active and fully functional, implying that device can complete its I/O operations
- pm_runtime_idle()
  - Expected action for this callback is to check whether device can be suspended and then queue suspend request for that device
  - Callback executed by power management core for specified device when device appears to be idle
  - Device idle is indicated to power management core by two counters
    - Device usage counter
    - Counter of device active children



CPUidle

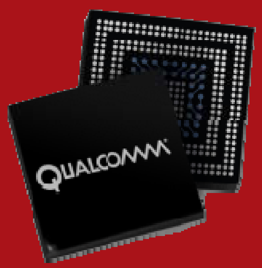
REDEFINING MOBILITY

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
- 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
  - CPUidle driver can support multiple idle states based on parameters such as varying power consumption, wakeup latency, etc.
  - Main advantage of this infrastructure is that it allows independent development of drivers and governors and allows for better CPU power management

CPUidle Driver

- CPUidle driver handles architecture or platform-dependent part of CPU idle states
- Provides idle state detection capability and can also support entry/exit into CPU idle states
- Initializes cpuidle_device structure for each CPU device and registers with cpuidle using cpuidle_register_device
- Can support dynamic changes by using cpuidle_pause_and_lock, cpuidle_disable_device and cpuidle_enable_device, and cpuidle_resume_and_unlock



Android Power Management Changes

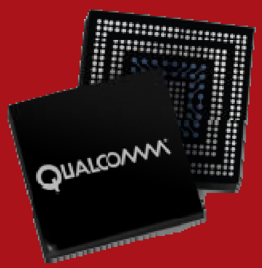
REDEFINING MOBILITY

Android Power Management Changes

- wakelocks renamed to suspend_blocker
- The timeout for the userspace wakelock API removed
- early_suspend replaced by runtime power management
- early_suspend levels are not supported

References

Ref.	Document	
Qualcomm		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*Resource Power Manager (RPM) Overview*	80-VP169-1
Q3	*Linux Power Management Debugging Guide*	80-VR629-1
Q4	*Configuration of Input Pins During Device Sleep*	80-VN499-7
Resources		
R1	*Runtime Power Management Framework for I/O Devices*	http://lxr.linux.no/#linux+v2.6.32/ Documentation/power/runtime_pm.txt
R2	*Runtime Power Management*	http://elinux.org/images/0/08/ELC-2010-Hilman-Runtime-PM.pdf



Questions?

https://support.cdmatech.com

REDEFINING MOBILITY