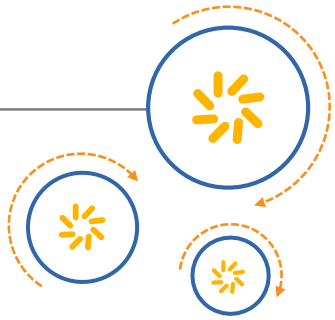
Qualcomm Technologies, Inc.



MSM8x74 Power Features and Design

80-NA157-158 B

March 3, 2015

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

Revision history

Revision	Date	Description
A	May 2013	Initial release
B	Mar 2015	Corrected figures, minor edits throughout.

Contents

Figures

Tables

1 Introduction

1.1 Purpose

This document discusses features, techniques, and hardware/software architecture details from a power perspective, and is generally applicable to all members of the MSM8x74 chipset family. The primary focus of the document is to cover the non-HLOS part of the system, as the HLOS part of the system is strongly influenced by the HLOS and OEM specifics.

This document does not assume an in-depth awareness of chipset architecture, but does assume that the reader has some level of familiarity with either this or earlier QTI chipsets.

1.2 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `#include`.

Code variables appear in angle brackets, for example, `<number>`.

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

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

1.3 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 website, register for access or send email to support.cdmatech@qti.qualcomm.com.

2 System Overview

2.1 Chip context

Figure 2-1 shows the MSM8x74 in context with the entire chipset.

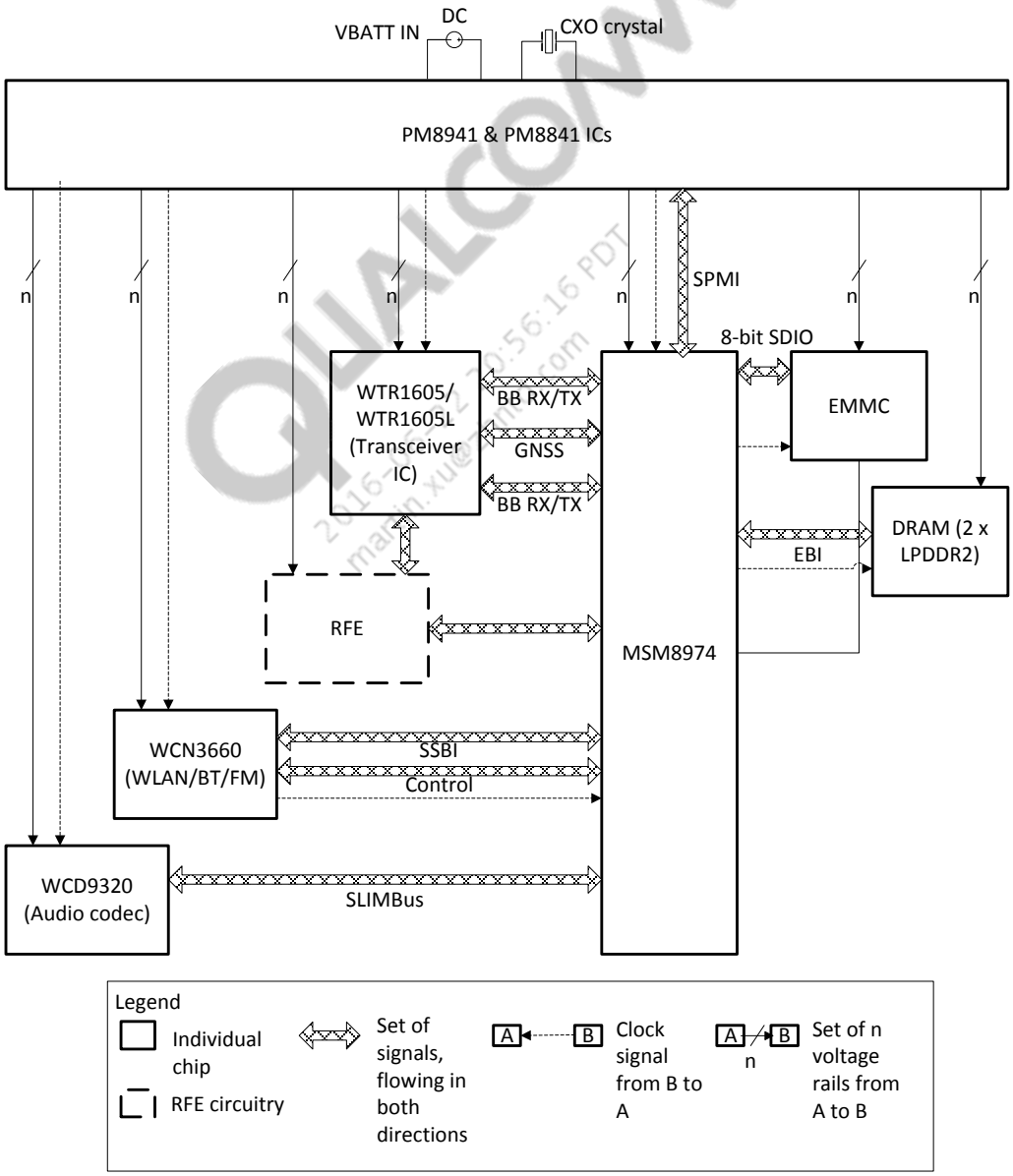


Figure 2-1 Chip context, hardware perspective

The components and connectors in Figure 2-1 are:

- MSM8x74
- Transceiver IC – This is the RF transceiver IC, WTR1605L. Based on the RF architecture chosen by the OEM, the system may utilize a single WTR1605L (circuit-switched feedback configuration) or a combination of two WTR1605Ls (simultaneous voice and data). The interface between the MSM8x74 (baseband) and transceiver IC comprises:
  - Baseband Tx to transceiver IC
  - Primary Rx (PRx) from transceiver IC
  - Diversity Rx (DRx) from transceiver IC
  - GNSS Rx (GPS Rx) from transceiver IC
  - Digital status and control signals, which include dedicated signals and an SSBI-based command interface
- PM8941 and PM8841 – These are power management ICs. The PM8841 primarily provides eight SMPSs that are used to feed key power domains on the MSM8x74, including the core domain and Krait cores. The PM8941 provides power regulation support for several other power domains on the MSM8x74, other chips in the chipset, and other parts of the circuitry, i.e., the Radio Front End (RFE). It also provides root clocks for the MSM8x74, transceiver IC, audio codec, and Wi-Fi/Bluetooth/FM ICs. The PMIC also provides other housekeeping functions, i.e., ADC, battery charging, LED drivers, etc.
- WCN3360 – This is the transceiver IC for WLAN, Bluetooth, and FM. The corresponding baseband functionality is hosted on the MSM8x74.
- DRAM – This is the PoP-type LPDDR3 connected on the External Bus Interfaces (EBI1 and EBI2) of the MSM™ dedicated for DDR. The two EBI interfaces support up to 2 GB of DDR.
- eMMC – eMMC NAND storage is connected through the SDC1 interface of the MSM.
- RFE circuitry – The RFE interface comprises power amplifiers, etc., the optional chips from QTI for PA power management (QFE1100), the converged Tx module (QFE23x0), and the RFE tuner (QFE1510).
- SPMI – This is the communication interface from the MSM to the PMIC for configuring PMIC registers to achieve voltage regulator control, XO buffer control, and control of other PMIC services, i.e., PMIC GPIOs, ADC, battery charging, etc.
- SSBI – This is the serial communication interface used by the MSM to communicate with the WCN3360 and WTRs.
- CXO – This is the 19.2 MHz clock source from the PMIC to the MSM8x74, transceiver chip, WCN3360, and WCD9320. The MSM8x74 adopts a CXO-only approach, unlike the MSM8960, which utilizes CXO for modem-related clocks and PXO for multimedia and other functions.
- Sleep clock – The sleep clock is used by the MSM in XO Shutdown (low power, sleep) mode. It is used to clock always-on domains, i.e., the MSM Power Manager (MPM), parts of the modem core, and certain timer circuits. On the MSM8x74, the sleep clock is derived from CXO, and therefore CXO circuits on the PMIC are left on during sleep. However, XO buffers, which are clock buffers on the PMIC CXO clock output paths, are still turned off during sleep.

- Power rails – This is the set of power rails from the PMICs to the MSM, transceiver IC, WCN3360, and other components onboard, powering different parts of each chip/component.

The above description specifically highlights how the clock and power are distributed from PMICs to the rest of the system. It is a simplification, with several details not included. For more detailed information about chipset options, interfaces, options in RF configurations, and other parts of the system, see *MSM8274/MSM8674/MSM8974 Chipset Design Guidelines Introduction* (80-NA437-5A) and module-specific documentation, as applicable.

2.2 MSM8x74 SoC system architecture

This section discusses the [decomposition_level:1] system_architecture and key subsystems of the MSM8x74 chip.

2.2.1 Hardware perspective

Figure 2-2 shows a simplified first level of decomposition into the MSM8x74 internal hardware structures. The goal is to set a perspective for the discussion about power in subsequent sections and to bring awareness about parts of the MSM8x74 chip that impact power directly/indirectly.

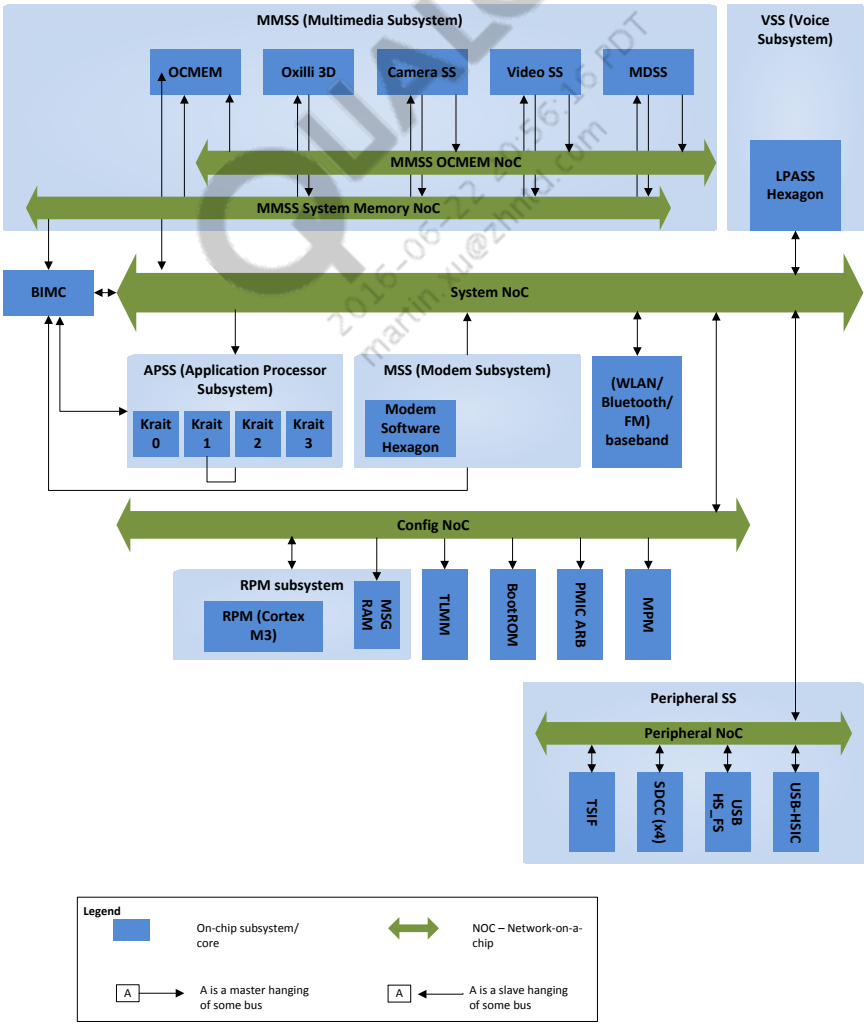


Figure 2-2 Hardware perspective, first-level decomposition

The components and connectors in Figure 2-2 are:

- Network-on-a-Chip (NoC) – NoC is the bus architecture designed to provide high performance and low latency. It is used for the main top-level interconnect (system NoC) and the configuration bus (configuration NoC), and for subsystems including the peripheral subsystem (peripherals NoC), Multimedia Subsystem (MMSS NoC), and on-chip memory (OCMEM NoC).
- MMSS – The MMSS provides hardware support and acceleration for display, camera, video encode and decode, 3D graphics, etc. It also owns an OCMEM for staging data from the DDR and reducing external DDR access. The OCMEM NoC and system memory NoC facilitate better bus bandwidths for transactions within the MMSS, without having to contend for the system NoC, when it can be avoided.
- Voice Subsystem (VSS) – Also called Low Power Audio Subsystem (LPASS), this supports audio in and out of the system and includes a dedicated 600 MHz Hexagon™ processor.
- Modem Processor Subsystem (MPSS) – This includes the modem core, GPS core, and one Hexagon processor with 600 MHz clock speed and 256 K L2 cache. The MPSS utilizes a single Hexagon processor, unlike the MSM8960 which utilized two Hexagon processors. The MPSS is connected to the rest of the SoC via a system NoC.
- Application Processor Subsystem (APSS) – The application subsystem includes four Krait processor cores with up to 2+ GHz clock speed and 2 MB L2 cache. The APSS hosts the High-Level Operating System (HLOS).
- WLAN/Bluetooth/FM subsystem – This subsystem provides baseband functionality for the WLAN, Bluetooth, and FM modules, and utilizes an ARM9 processor. The subsystem interfaces with WCN3660, the off-chip RF radio for WLAN, Bluetooth, and FM. The WLAN DAC and ADC reside on the MSM8x74. It is connected to the rest of the SoC via the system NoC.
- Peripheral subsystem – This subsystem provides hardware engines for peripheral support for USB 3.0, USB 2.0, HSIC, TSIF, four SDC interfaces, UARTS, UIM, I2C, SPI, TSSC, and PDM. The peripheral subsystem does not have a dedicated processor. Other processors in the system can configure and use its functions via the system NoC. The peripheral subsystem uses a dedicated peripheral NoC, which is connected to the system NoC.
- Resource Power Manager (RPM) subsystem – This subsystem includes a Cortex-M3 processor, unlike the MSM8960 which uses an ARM7 processor. It also includes a 16 KB message RAM, which facilitates communication between the RPM and other processors in the system. This subsystem manages systemwide resources that are shared across different execution environments, e.g., MPSS, WLAN/Bluetooth/FM, and APSS. Shared resources for the RPM's purposes comprise items such as frequencies for shared buses (NoC), shared voltage regulators, etc. Resources such as GPIOs and peripherals that might need to be shared between more than one execution environment are not managed by the RPM. The RPM subsystem connects to the rest of the SoC via a config NoC, which connects to a system NoC.
- MPM sleep Power Manager (MPM) – The MPM is the centralized MSM sleep power manager to shut down the MSM clock source for CXO and/or, optionally, put the MSM device into Retention (VDD_MIN) mode to save power. MPM hardware finite state machines control the sequence of entering Sleep states. It can assert hardware signals to isolate power domains, send SPMI commands to the PMIC to lower voltages, and shut off CXO buffers.

Upon wake-up on selected interrupt events, the MPM controls the sequence of exiting the Sleep state. It sends SPMI commands to the PMIC to make CXO clocks ready, increase voltages to required levels, etc. It does this without external support or CPU intervention.

- BIMC – This is the system memory (DDR) controller that supports dual-channel EBI system memory.
- PMIC ARB – Communication with the PMIC occurs over a single SPMI bus, which is a multidrop serial interface on which only one transaction may occur at any time. The MSM is the only SPMI bus master, yet it contains multiple processors and hardware power management blocks, e.g., the MPM, that must all talk to the PMIC. A mechanism is needed to share access to the SPMI bus between all of these resources. The role of the PMIC arbiter is to provide concurrent and autonomous PMIC access to the software and hardware entities that need to communicate with the PMIC.
- TLMM – The TLMM is a hardware block that provides a mechanism for sharing a multitude of functional and test interfaces onto a much smaller set of pins called GPIOs.
- BootROM – This is a total of 192 KB on-chip boot ROM that would typically host the apps PBL.

2.2.2 Runtime perspective

Figure 2-3 shows five different software execution environments that exist simultaneously at any point in time. The RPM is an execution environment that manages shared system resources and each of the other execution environments can communicate with the RPM to make resource requests. This transport is exposed on each execution environment via a local RPM driver.

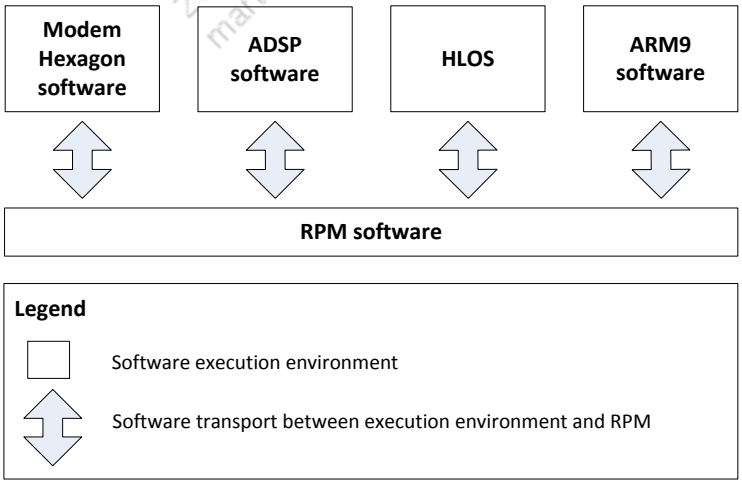


Figure 2-3 System software, first-level decomposition (runtime perspective)

2.2.3 Static perspective

Each software execution environment has its own source tree and build system:

- Modem – \modem_proc\
- ADSP – \adsp_proc\
- RPM – \rpm_proc\
- A9/WConnect – \wcnss_proc\
- HLOS – HLOS-specific

3 Process Technology and Power

3.1 Fundamental approaches to reduce power consumption

Power = Active power + static leakage power

→ Power = $\alpha C V_{DD}^2 f + V_{DD} I_{LEAK}$

Where:

- α = Factor related to effective percent of gates switching
- C = Circuit capacitance
- f = Clock frequency
- I_{LEAK} = Static leakage current

The static leakage component exists primarily due to gate and channel leakage in each transistor that is left on, but idle. This leakage current, I_{LEAK}, increases as process geometry decreases. It manifests itself in Active and Standby modes of operation and can dominate dynamic power, especially as the process technology scales down. Leakage power is critically dependent upon the operating temperature (this relationship is not represented in the equation above, and is exponential) and V_{DD}, and is not dependent on the software application that is running.

On the other hand, from the equation, the active power increases quadratically with the operating voltage V_{DD}.

Some of the fundamental approaches deployed to reduce power consumption are:

- Reduce power supply voltage V_{DD} when appropriate
- Run at a lower clock frequency when appropriate
- Disable functional units with control signals when not in use
- Disconnect parts from the power supply when not in use

Power management in mobile devices requires a comprehensive effort across the processor design, chipset architecture, and system software design.

3.2 Process technology

The MSM8x74 uses 28nm HPM process technology. HPM technology provides higher performance for mobile applications. 28nm HPM also provides lower dynamic power but has higher leakage compared to 28nm LPM used in the MSM8960 family of chipsets. Increased leakage is mitigated through several techniques discussed in this document.

4 Power-Specific Features

This chapter discusses the major power optimization features employed on the MSM8x74 chipset, with a brief operational description of their meaning.

4.1.1 Partitioned power domains

The MSM provides separated power inputs for different parts of the chip hardware; e.g., there are different power inputs for the MPSS, Krait processors, on-chip memories, etc. In tandem with this arrangement, the PMIC provides several power rails that can be individually controlled.

This provides the ability to set unique voltage levels, turn on/off, and potentially, in the case of SMPS-type regulators, switch between PFM and PWM operation (see Section 4.1.2). Many of the following power features build upon this basic capability provided by partitioned power domains. The following sections discuss which domains are power collapsible and which employ voltage scaling, and to what extent.

Table 4-1 lists the key power domains on the MSM8x74. For a detailed understanding of how PMIC SMPSs and LDOs are routed to power domains of the MSM (also called the Power Distribution Network, or PCN), see *MSM8974 Linux Android Current Consumption Data* (80-NA437-7).

Table 4-1 Key MSM8x74 power domains and PMIC rails sourcing them

Domain name	PMIC rail	Domain description
VDD_MEM	VREG_S1B (PM8841)	Power for on-chip memories
VDD_CORE	VREG_S2B (PM8841)	Power for digital core circuits, e.g., LPASS Hexagon™ processor, VFE, MDP, JPEG encode/decode, video encode/decode, Krait L2, BIMC, etc.
VDD_KRAIT (1, 2, 3, 4)	VREG_S5B, S6B, S7B, S8B (PM8841)	Power for quad-Krait applications microprocessors; Krait PDN has a unique implementation in MSM8x74 and is discussed separately
VDD_GFX	VREG_S4B (PM8841)	Power for graphics core
VDD_MODEM	VREG_S3B (PM8841)	Power for modem core and Hexagon modem processor
VDD_ALWAYS_ON	VREG_S3A (PM8941)	Always-on power domain with MPM
VDD_A1, VDD_A2	L2, L4, L6, L12, L14	Power for low- and high-voltage on-chip analog circuits, respectively

Domain name	PMIC rail	Domain description
VDD_P1, VDD_P2, .., VDD_P7	VREG_L1, VREG_L13, VREG_S3A, VREG_L1, VREG_L9, VREG_L10, VREG_L1	Power for MSM pad groups 1 through 7; each pad group powers up a set of functionally and logically grouped pads/pins, e.g., PAD group 1 is the EBI1 interface PADs on the MSM, PAD group 2 is SDC2 pads, etc.; see *MSM8274/MSM8274AB, MSM8674/MSM8674AB, MSM8974/MSM8974AB Device Specification* (80-NA437-1) for more details

In regard to split power rails, unlike a common VDD_CORE in the MSM8960, the MSM8x74 splits the CORE rail into VDD_MODEM, VDD_GFX, and VDD_CORE.

4.1.2 PMIC Auto mode

The PMIC supports two basic operating modes for its SMPS type regulators. PFM mode provides greater efficiencies at a low current draw (< 50 to 100 mA), but cannot support higher current demands. PWM mode is used to provide higher current demands and is tuned for peak efficiencies in the 200 to 400 mA range. As a result, it makes sense to put the specific SMPS in PFM mode when it is known that the SMPS load will not be high.

The PMIC also supports Auto mode for its SMPS-type regulators, such that the regulator is automatically able to switch between PFM and PWM modes based on the current loading of the regulator, without software intervention.

4.1.3 Core power collapse via GDHS/GDFS/BHS of cores

Often there are lost opportunities for saving power when several parts within the MSM are sourced with the same rail, e.g., several components of the modem core typically share one power domain/rail, i.e., VDD_MSS.

For these cases, the MSM provides special on-chip circuits called Global Distributed Head Switches (GDHS), Global Distributed Foot Switches (GDFS), or Block Head Switches (BHD) to individually disconnect these cores from the common power rail feeding them or from the common ground. When the system does not need these cores to be active, removing the power supply saves leakage power.

Table 4-2 lists examples of the cores that can be power collapsed via these mechanisms while still sharing a common voltage domain, i.e., VDD_CORE, with other cores.

Table 4-2 Example power collapse of cores/blocks

Example cores/blocks	Common domain/rail
LPASS Hexagon processor, VFE, MDP, JPEG encode/decode, video encode/decode, Krait L2, BIMC, etc.	VDD_CORE
Hexagon modem processor, proprietary modem core blocks, etc.	VDD_MODEM
Graphics processor, other graphics-specific blocks, etc.	VDD_GFX

BIMC (DDR memory controller) power collapse was not supported in earlier chipsets, i.e., the MSM8960, and is a new addition to the MSM8x74 chipset family.

4.1.4 DCVS/MP-DCVS

In computer systems, Dynamic Clock and Voltage Scaling (DCVS) is largely an umbrella term for forms of implementations of the following two-fold concept:

- Scaling of the frequencies of processors and other clock-driven digital logic, in proportion to workload demands
- Scaling of the voltage of processors and other clock-driven digital logic, in accordance with operating frequency and physical characteristics of the underlying silicon

QTI's hardware and software implementation on the MSM8x74 normally uses the term DCVS from the point of view of software running on a particular processor. It refers to software-based control of the operating frequency of a processor and the fact that it can possibly lead to scaling the processor voltage, based on the workload requirements and the QoS requirements of the processor system. The decision-making algorithm is typically implemented in the software. The DCVS software is primarily concerned with the problem of finding the optimal operating frequency at any point in time. Different processor systems choose different algorithms to this end, based on what best suits their respective case.

- DCVS/MP-DCVS for Kraits – HLOS-specific implementation
- DCVS for GPU – Also HLOS-specific
- DCVS for MSS – Standard across the board
- ADSP frequency scaling – While the ADSP supports frequency scaling, it does not employ a sophisticated DCVS; instead, it implements a light-weight static algorithm for processor frequency scaling, similar to what is employed on the earlier generation MSM8960 chipset.
- WLAN/BT/FM ARM9 frequency scaling – This ARM9 processor supports four operating frequencies just like ADSP; the workloads on ADSP are well understood and a static frequency scaling is utilized.

The change in operating frequency of a processor is typically tied to a change in the voltage fed to that processor. This, in turn, may impose a minimum operating requirement on a parent voltage domain feeding that processor's voltage domain, e.g., scaling up a Hexagon modem's frequency means scaling the voltage of the processor rail. Since this processor rail is derived from VDD_MODEM via an on-chip switch and an on-chip LDO, when this processor rail must be operated at a voltage that is higher than the voltage set for VDD_MODEM, VDD_MODEM must also be scaled up.

In addition to voltage, scaling a processor's frequency can have implications on other system resources, e.g., scaling a Hexagon modem's frequency must be accompanied by scaling of the system NoC, apps FABRIC and BIMC, and EBI1 clocks (DDR).

4.1.5 Bus scaling

Figure 2-2 shows an overview of the multi-tier bus architecture used in the MSM8x74 to connect different master and slave components during a use-case flow, i.e., system NoC, MMSS NoC, OCMEM NoC, config NoC, peripheral NoC, BIMC, etc. Based on bus traffic conditions, efficiently scaling up/down bus clocks and the associated memory interface clock with a correct voltage adjustment can have a big impact on use case power consumption.

For shared buses such as the system NoC, config NoC, etc., final frequency scaling and associated voltage scaling are owned by the bus driver on the RPM. APIs/interfaces are exposed to all other processors to enable them to make requests to the RPM. Chapter 5 provides more specific details.

The above-indicated buses are powered up from VDD_CORE; therefore, bus scaling is an important participant in voting for VDD_CORE voltage scaling.

4.1.6 Static Voltage Scaling (SVS)

In QTI's hardware and software implementation on the MSM8x74, the idea of SVS is very similar and sometimes overlaps with the idea of DCVS. However, unlike DCVS software, which deals with finding the optimal frequency of a core/processor, SVS software is concerned with the voltage scaling aspect of shared voltage domains.

The voltage domains of VDD_CORE, VDD_MEM, VDD_MODEM, and VDD_GFX support voltage scaling with discrete voltage corners (allowed levels). The rail can be safely operated at a specific voltage corner only when the cores that the rail feeds are at or below a predetermined performance (frequency) level.

It is the job of the SVS-specific software to aggregate the performance levels of cores and/or processors on a shared voltage domain and to identify the lowest voltage corner for that voltage domain at any point in time. This voltage corner may then be applied as-is, e.g., in the case of VDD_MEM, or via some combination of PVS and Adaptive Voltage Scaling (AVS) techniques (VDD_CORE, VDD_MODEM, and VDD_GFX), such that the final voltage at which the specified voltage domain will settle will be a more fine-grained value around the voltage corner, based on the silicon characteristics of that specific chip.

Table 4-3 shows the voltage domains that support SVS, processors that own SVS operation of that rail, and current proposed voltage corners.

Table 4-3 Summary of voltage domains with SVS feature

Voltage domain	Voltage corners	Owner
VDD_CORE	Four voltage corners ▪ Super-turbo – 1.0500 V ▪ Turbo – 0.9875 V ▪ Nominal – 0.9000 V ▪ SVS (low) – 0.8125 V	RPM processor
VDD_MODEM	Four voltage corners ▪ Super-turbo – 1.0500 V ▪ Turbo – 0.9875 V ▪ Nominal – 0.9000 V ▪ SVS (low) – 0.8125 V	Hexagon modem processor
VDD_GFX	Four voltage corners ▪ Super-turbo – 1.0500 V ▪ Turbo – 0.9875 V ▪ Nominal – 0.9000 V ▪ SVS (low) – 0.8125 V	RPM processor
VDD_MEM	Two voltage corners ▪ Super-turbo – 1.0500 V ▪ Turbo – 1.0500 V ▪ Nominal – 0.9500 V ▪ SVS (low) – 0.9500 V	RPM processor

NOTE: The voltages in Table 4-3 are only approximate voltage corners. Additional features (AVS, PVS) allow each chip to select a more precise voltage value around which to corner, based on its unique silicon characteristics.

For example, VDD_CORE has several different digital core blocks, including the system NoC that is on it and the Hexagon processor for the LPASS subsystem. At any point in time, some of these blocks will be in the On state and some will be in the Off state. Those that are in the On state are at specific individual performance levels (operating frequencies). If the LPASS Hexagon processor is On, the DCVS implementation on the processor selects its operating frequency, whereas the system NoC operates at a frequency selected by the shared bus driver on the RPM. However, based on the operating frequency selected by DCVS, it also issues a request to the bus driver asking that the system NoC is at a certain level. The SVS software for VDD_CORE aggregates the performance levels of the items on VDD_CORE and ascertains a common minimum voltage corner for VDD_CORE.

SVS refers to process voltage scaling (*process*, not *processor*) and is a concept that can apply to all of the digital voltage domains of the MSM that support voltage scaling.

Due to a wide variation in silicon characteristics at the 28nm process node between individual chips, as well as different regions of the same chip, it is possible that a specific voltage domain can sustain a given performance level (switching frequency) at a slightly different minimum voltage level from chip to chip.

The easiest approach to implement voltage scaling for any voltage domain in such a scenario is to determine the worst-case minimum voltage that satisfies the chips from the production line. For example, when a Krait processor is running at 384 MHz, a hypothetical worst-case minimum voltage of 0.95 V could be used across the board, even though there may also be a set of chips for which the Krait processor can sustain 384 MHz at a lower 0.85 V.

Typically, a Krait core or any other digital core that can guarantee 384 MHz at a lower voltage of 0.85 V (fast part) also has more leakage than the core that needs 0.95 V (slow part). There is a clear power incentive in running such a core at 0.85 V, rather than 0.95 V, because leakage is proportional to voltage applied. Consequently, if voltages can be applied according to the minimum needed for a digital voltage domain based on the silicon characteristics of individual chips and regions on chips, the power variance can be minimized across individual chips.

PVS is the technique of selecting a relatively finer-grained power optimal voltage level at a given performance level or voltage corner of a digital voltage domain, based on that chip's silicon characterization determined at the production line. For each digital voltage domain of the MSM that implements the PVS technique, N different bins are defined for that voltage domain of the MSM. Each bin defines a fixed performance level to voltage mapping for the discrete performance levels of that voltage domain.

Table 4-4 shows a hypothetical example of binning of a processor's voltage domain for the purpose of clarity.

Table 4-4 Hypothetical example of PVS

Perf level (MHz)	Voltages (Bin1)	Voltages (Bin2)	Voltages (Bin3)
384	0.95	0.90	0.85
594	1.00	0.95	0.90
864	1.10	1.05	1.00
1188	1.20	1.15	1.10

NOTE: In cases of SVS-based voltage domains, e.g., VDD_CORE, etc., performance levels are in terms of super-turbo, turbo, nominal, and SVS (low).

All PVS-supported voltage domains of individual MSM chips are characterized at the factory. Designated eFuses on the chip are blown to indicate the bin that is applicable to the respective voltage domain.

During operation, each time the software decides to change the performance level of a voltage domain, the appropriate voltage levels are selected in the software by finding the applicable bin with regard to the voltage domain from the factory-programmed eFuse, and then using a table similar to Table 4-4.

For the MSM8x74, the following voltage domains are currently expected to implement PVS support:

- VDD_CORE
- VDD_GFX
- VDD_MODEM
- VDD_KRAIT

4.1.7 AVS

Without software assistance at runtime, the AVS feature aims to select a finer-grained voltage value for a processor rail based on silicon characteristics, temperature, and other physical factors. The software first selects a more coarse voltage corner based on its DCVS/PVS tables for a processor voltage rail and then arms the AVS block. The AVS block automatically selects the lowest possible (more finely grained) voltage value for that voltage rail.

In the MSM8974, this feature is supported for the Krait processor when only one out of the four Kraits is running and the other three are power-collapsed.

4.1.8 XO-shutdown/CXO-shutdown

CXO is the parent clock input for most entities on the MSM and other chips in the chipset (see Section 2.1). CXO shutdown is possible only when the MSM and other chips utilizing it are unoccupied and will be unoccupied for at least a specified amount of time. During CXO shutdown, clock paths from the PMIC to the MSM for CXO are turned off, including the corresponding clock buffers on the PMIC. Clock generation of CXO is still kept On at the PMIC and is used to generate the 32 kHz sleep clock. No clock is fed to the MSM, except the 32-kHz sleep clock, which is required for the always-on parts, i.e., the MPM block. Power collapse of most parts of the MSM and the RF transceiver IC typically precedes CXO shutdown. Details of how the system is set up in the hardware and software to get into and recover from this state are revisited later in this document.

NOTE: The single CXO is a design change from the MSM8960, which used two clock sources, PXO and CXO.

4.1.9 VDD_CORE and VDD_MEM minimization

These two voltage domains must always remain On. However, beyond the voltage corners that were described for these voltage domains in the previous discussion of SVS, there is yet another voltage corner that corresponds to the retention state, which is possible when no active switching occurs on these domains. When the CXO is shut down, it is guaranteed that there is no switching activity on these states. Hence, there is a possibility to further lower VDD_CORE and VDD_MEM. While VDD_CORE can be retained at 0.5 V or 0.65 V based on silicon characterization, VDD_MEM is retained at 0.75 V.

5 System Architecture for Power Management

This chapter discusses the hardware and software system architecture for power management.

5.1 Power distribution

The power distribution is [decomposition_level:2] system_architecture/power_distribution.

5.1.1 Hardware perspective

5.1.1.1 Power rails/domains

See *MSM8974 Linux Android Current Consumption Data* (80-NA437-7 for a current power tree (power distribution network) diagram. The diagram shows the detailed power distribution map used in QTI's reference design. This information can also be derived from looking at the reference schematic shown in *MSM8274/MSM8274AB, MSM8674/MSM8674AB, MSM8974/MSM8974AB Baseband Schematic* (80-NA437-41) and *WTR1605(L) RF (CMCC) for MSM8974/MDM9x25(M) – ET Design Example – Preliminary Reference Schematic* (80-NA437-46). Table 4-1 lists examples of key power domains on the MSM and the PMIC rails that feed them. In the power tree, more than one power domain is typically sourced from one PMIC rail and a PMIC rail can be used to subregulate one or more other PMIC rails.

5.1.1.2 Krait PDN

Design of the Krait PDN can be shown by using an example. Assume that, for a certain use case, the following voltages are required for each Krait core to achieve optimal power at the operating frequency chosen for each Krait:

- Krait 0→0.9 V
- Krait 1→0.9 V
- Krait 2→0.7 V
- Krait 3→Off

Figure 5-1 shows second-level decomposition (Krait PDN) from a hardware perspective.

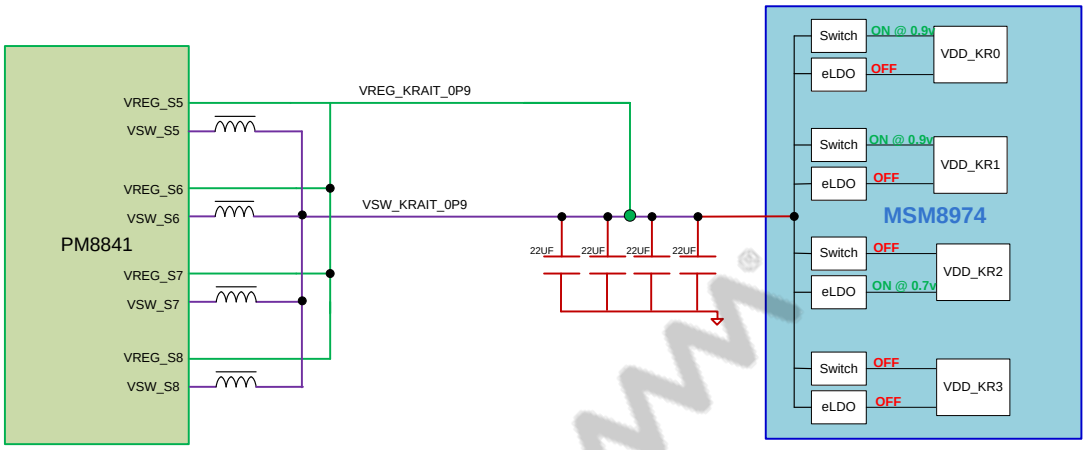


Figure 5-1 Second-level decomposition (Krait PDN) hardware perspective

Externally, the MSM provides a unified voltage domain for all four Kraits. Each Krait core has an individual on-chip LDO and switch, which allow each Krait to be operated at a different (lower) voltage or to be completely switched off. PMICs SMPS5 through SMPS8 are used to feed the unified voltage domain for four Krait cores. When at least one or more Krait cores are On, at least SMPS5 is On. Other SMPSs must be turned on based on the current requirements of all of the Kraits being operated simultaneously.

In the previous Krait configuration example, assuming that the correct combination of SMPSs is turned on and 0.9 V is supplied, Krait 0 and Krait 1 utilize the switch path and keep the switch On. Krait 2 utilizes the LDO path with LDO set to 0.7 V and Krait 3 utilizes the switch path and keeps the switch Off.

5.1.1.3 Power rail control

Figure 5-2 shows the general hardware infrastructure for enabling different cores/processors on the MSM to communicate to the PMIC for controlling voltage regulators.

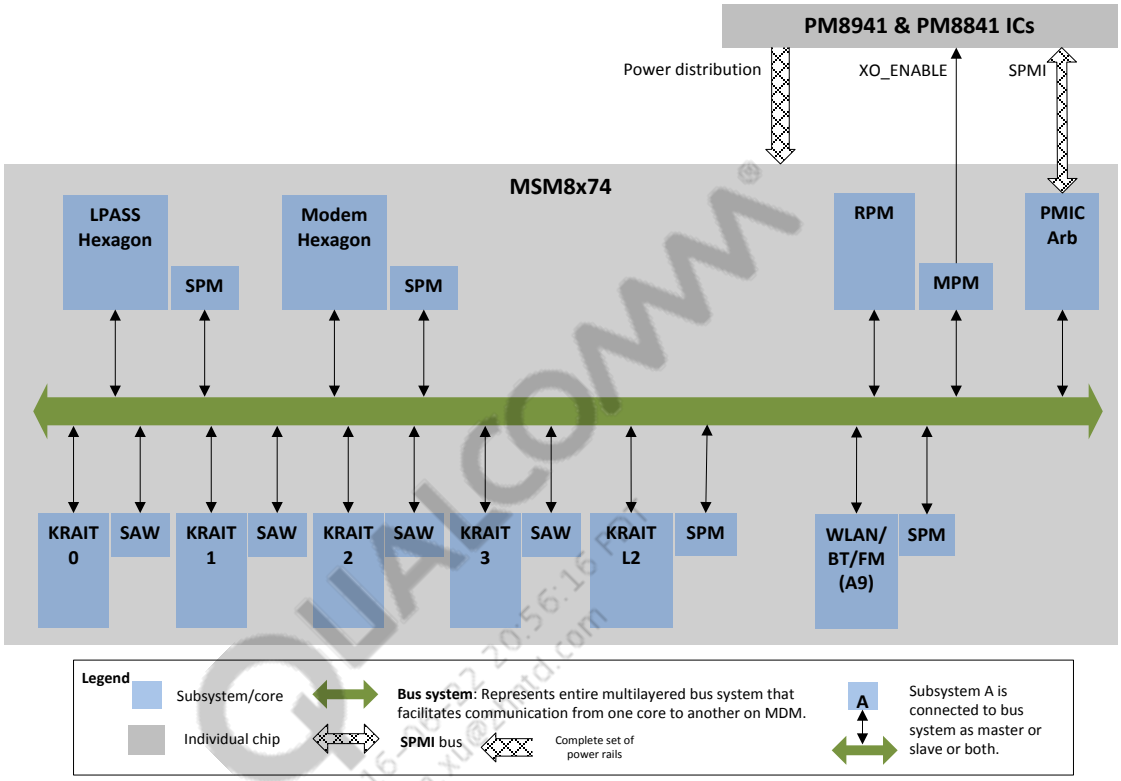


Figure 5-2 Second-level decomposition (power rail control) hardware perspective

Each hardware entity in Figure 5-2 has the following purpose:

- Krait(s), Hexagon modem, Hexagon LPASS, RPM, WLAN/Bluetooth/FM (A9) – Each of these system processors can be stakeholders in controlling the state of one or more voltage regulators in the system.
- Subsystem Power Manager (SPM) – The SPM hardware entity performs several handshaking and control tasks during power-down and power-up of its master processor. The SPM is a dedicated sleep controller for its processor and is needed because certain actions can only occur after the processor has stopped executing instructions, i.e., clamping IOs, powering down the processor rail, if applicable, or performing other complex actions to notify the rest of the system about the processor's Off state. Providing each processor with its own SPM allows more efficient power management by enabling each processor to be individually power collapsed. In the MSM8x74, the processors and the Krait L2 cache have dedicated SPMs. The four Krait processors have SPMs that are part of a larger block called SPM and AVS Wrapper (SAW), which hosts an SPM and an additional AVS module (Section 4.1.7).

  The SPM is a pure hardware block that cannot execute instructions like a processor. It is capable of controlling a VREG at a PMIC via the PMIC arbiter. It is also capable of communicating to/from the RPM, the active to power-collapsed, and vice versa, transitions of its master processor. The SPM provides a set of registers that determine SPM behavior. These registers are typically set up by the owner processor.

- SAW – The SAW is a hardware entity for each Krait processor. There is one SAW per processor core.

  AVS is a hardware block capable of making very fine-grained adjustments to the Krait operating voltage, based on the silicon characteristics at runtime. These adjustments occur around the voltage corner at which the specific voltage domain is already operating. When just one of the four Kraits is running, the HLOS can set up an initial voltage based on its operating frequency. It can further enable the AVS block, which can handshake with the PMIC to finely adjust the voltage around this selected operating point. This has the advantage of being able to operate a Krait voltage domain at the lowest possible point for an individual chip, according to its silicon characteristics.

- Krait L2 cache – The L2 cache can be active, in Memory-Retention mode, or power collapsed, i.e., no clock and no power. The L2's SPM allows set up of the correct states for the L2 cache each time the Krait processors become idle and enter Wait for Interrupt (WFI). The L2 SPM can also be programmed to send a notification signal to the RPM each time the L2 enters power collapse. This facilitates the RPM to consider more aggressive power-saving modes, i.e., XO shutdown and VDD-min.

- PMIC arbiter – The PMIC arbiter arbitrates PMIC (SPMI) commands to the PMIC from different sources on the MSM8x74. Its bus interface is appropriately set up so it can receive PMIC commands from several masters, as shown in Figure 5-2.

  - All SAWs/SPMs
  - All processors

  The PMIC arbiter serializes these commands to the PMIC on the external SPMI bus. Any entity in the system will want to communicate with the PMIC to accomplish one or more of the following from the PMIC:

  - Control one or more regulators that are shared by more than one part in the system
    - RPM is the owner of these regulators; others must request with RPM
  - Control one or more regulators that are exclusively used by the specified entity
    - That entity or its SPM own and control the regulator, e.g., VDD_MODEM is owned by software running on a Hexagon modem processor
  - Configure/control other types of PMIC functionalities, e.g., the battery charger, PMIC ADCs, PMIC GPIOs, etc.
    - The processor (software execution environment) that owns the related functionality also owns the PMIC resource and directly controls that resource.

- XO-enable – This signal can be used to signal the PMIC to turn off/on the CXO buffers in a pin-control manner.

5.2 NPA driver framework

The NPA driver framework is [decomposition_level:2] system_architecture/npa_framework.

The NPA framework manages dynamic resources, with the goals of system transparency and interface consistency. Examples of dynamic hardware resources on a Hexagon modem are the CPU clock frequency, bus bandwidths, etc. These are dynamic because the software determines, based on certain rules at runtime, the state in which each of these must be placed to achieve the desired operation. In addition to the hardware resources, there can be resources that are logical abstractions of the hardware resources that interface with the users/clients of the resources more naturally. For example, an NPA resource called CPU could be an abstraction of another NPA resource CPU clock and bus speed. The CPU could have its clients make requests in terms of MIPS requirements, implement the aggregation logic for MIPS requests from all its clients, and then make a request to the CPU clock resource in MHz and to the bus resources in terms of bandwidth.

5.2.1 /core/cpu NPA example

Examples are:

- NPA resource – The NPA resource software component represents a hardware resource or some other logical resource. To be an NPA resource, the software component should implement a specific interface dictated by the NPA framework.
- NPA client – An NPA client is a user of an NPA resource. An NPA resource may also be an NPA client.
- NPA framework – The NPA framework is a standardized interface/API that NPA resources need to export and clients need to use to be able to place resource requests. The framework effectively forms a complex graph of resources and their clients. The NPA framework also comes with an in-built logging system that maintains a history of NPA requests across the software running on that processor.
- NPA node – An NPA node is a collection of one or more resources and their dependencies. Most nodes are simple nodes, meaning that they have one resource and its dependencies. A compound node has more than one resource. This is typically useful when a set of resources needs to maintain a fixed relationship with respect to each other, but each resource has different clients and states.

5.2.2 Hardware perspective

This subsection is not applicable to this version of the document.

5.2.3 Runtime perspective

5.2.3.1 /core/cpu NPA example

Figure 5-3 uses a top-level CPU clock frequency driver (/core/cpu) as an example to show different elements of the NPA framework at runtime.

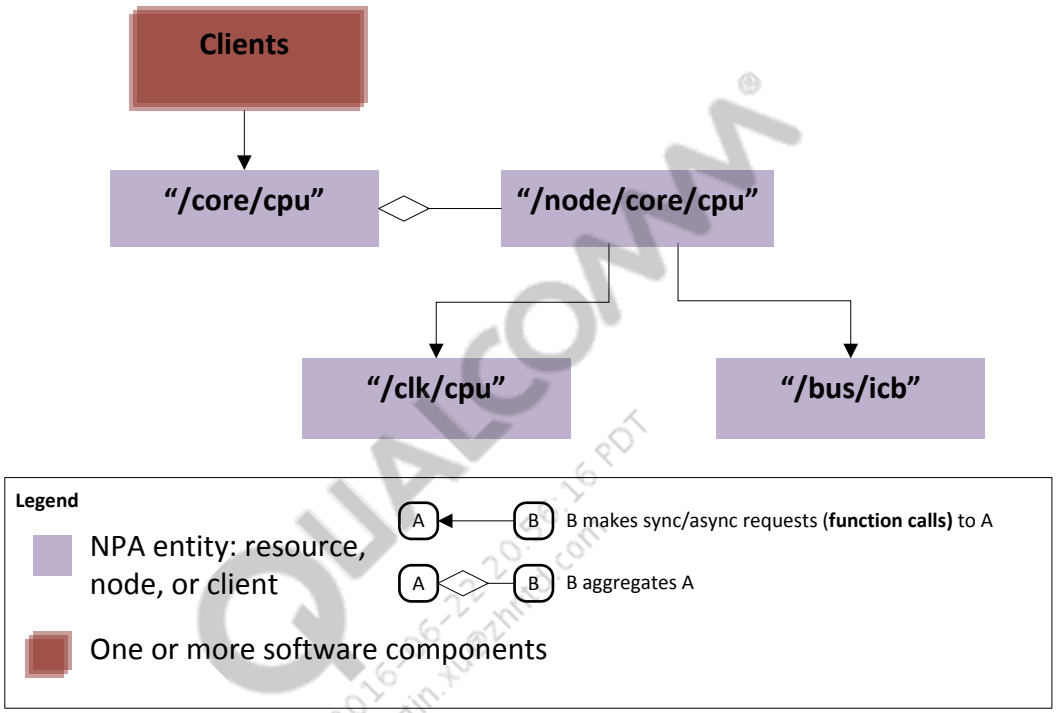


Figure 5-3 System architecture, NPA framework (runtime perspective)

- Clients – Per the example, this component represents any client in the same execution environment that could request a change in the CPU MIPS for Hexagon software; e.g., if this is in the context of a Hexagon modem, MCPM would be a client.

  To become an NPA client, the following NPA API is used:

```
npa_resource_handle cpu_hdl =
npa_create_sync_client(
"/core/cpu",        /* resource name */
     "client_name",      /* client name   */
NPA_CLIENT_REQUIRED /* work model    */
);
```

  The work model is discussed later in this chapter. A request may then be issued as follows:

```
/* Issue a work request of 100 MIPS to the CPU */
npa_issue_required_request( cpu_hdl, 100 );
```

- /core/cpu – This is the NPA resource in the example with which the clients are directly concerned. It would typically aggregate information, such as the string name of the resource, its units, the maximum allowed value, and a plug-in, i.e., an update function and supported work models.
  - Update function – This function is called with each request to the resource and aggregates all active requests to compute the new state of the resource that meets the needs of all the clients together.
  - Work model – Work model describes the nature of the request from the client, in the form of a hint, to the resource that the values requested with the current requests are known, how long they are needed, etc. An example is shown in the previous code snippet, NPA_CLIENT_REQUIRED. This is one of the more common forms of work models seen with requests. It tells the resource that the client making the request is certain about the value requested, but is not sure about how long this would be needed. The expected behavior on the part of the resource is to provide the request for as long as the client does not issue a corresponding cancel request. There are other work models, but a discussion of those models is beyond the scope of this document.
- /clk/cpu and /bus/icb – These two resources have basic elements that are similar to the /core/cpu resource. /clk/cpu represents the CPU clock and /bus/icb represents the bus arbiter on that processor/execution environment, which represents bus bandwidths that are crucial in guaranteeing the MIPS with the CPU clock.
- /node/core/cpu – This node aggregates the /core/cpu resource, a list of dependencies (not shown in Figure 5-3, which are references to resources on which this node/its member resources depend), and a driver function (also not shown in Figure 5-3). A driver function defines the dependency relationships. An example is shown further on in this chapter.

5.2.3.2 NPA and shared system resources

The NPA resource graph is defined per address space, i.e., the resource graph on the modem is distinct from the resource graph on the RPM. For all shared resources, such as CXO and shared buses, the RPM is the managing processor. Therefore, it defines a global NPA node/resource for the resource, which is then part of the RPM's NPA resource graph.

The Hexagon modem and other RPM masters are viewed as clients in the RPM's NPA resource graph for the shared resources. RPM masters typically have a local resource/node within their NPA resource graph, i.e., /bus/icb in Figure 5-4. The local resource/node aggregates local requests for that resource and passes them on to the systemwide resource/node on the RPM; e.g., the Hexagon software would have a local \xo\cxo NPA resource that aggregates requests for CXO from clients on the Hexagon modem. This resource would send this aggregated request to the corresponding systemwide \XO\CXO resource/node on the RPM.

Figure 5-4 shows a system architecture, NPA framework (runtime perspective) example NPA request.

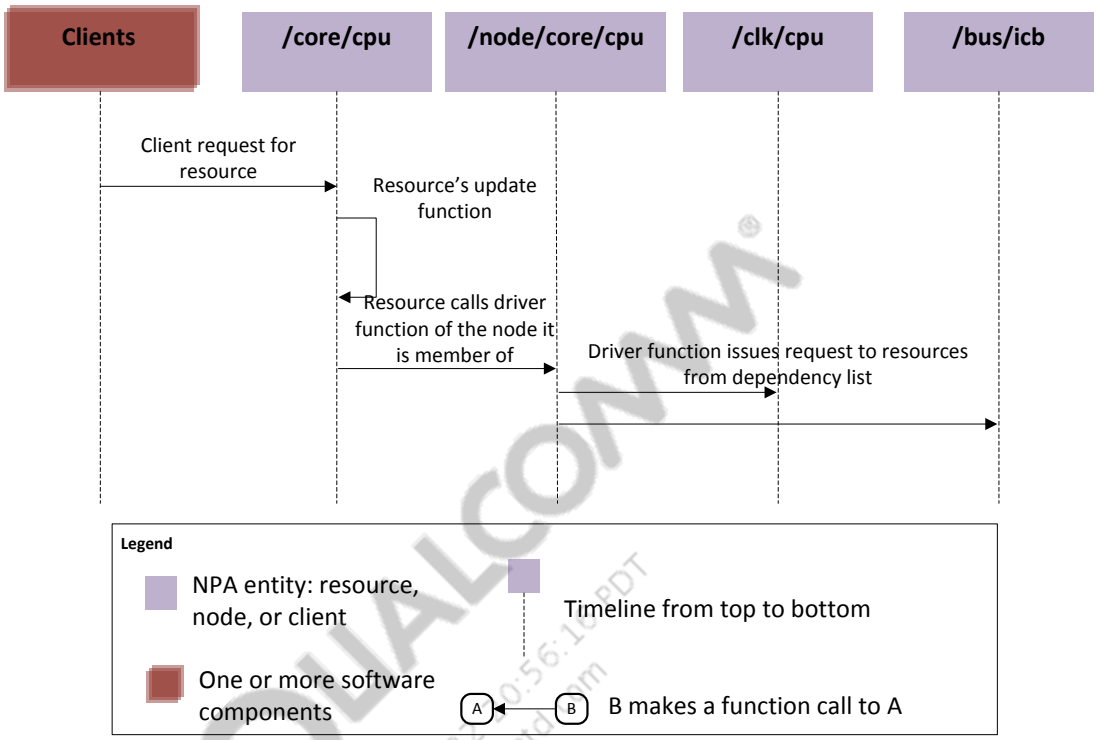


Figure 5-4 System architecture, NPA framework (runtime perspective) – Example NPA request

5.2.4 Static perspective

Source files implementing the framework are located within the modem and RPM workspaces under \core\power\npa\.

5.3 RPM

The RPM is [decomposition_level:2] system_architecture/rpm.

RPM is a dedicated processor that manages the shared system resources. This section describes how the hardware facilitates the RPM to achieve its goals and how the RPM software is designed.

5.3.1 Hardware perspective

5.3.1.1 Shared resources and RPM

This section describes the RPM, the types of interactions that occur between the RPM and the masters (using the Hexagon modem as an example), and the hardware infrastructure that exists to facilitate these interactions.

Common important shared resources on the MSM8x74 that are needed by more than one processor/core and are, therefore, ultimately owned by the RPM are:

- All system-level/shared voltage domains. If a voltage domain is shared between two or more execution environments, it is managed by the RPM. Table 5-1 shows key voltage domains and their owner processors. In doing so, the following decision-making is involved at the RPM side:
  - Aggregating voltage corner requests from all stakeholders (processors/subsystems)
  - Arriving at a finer-grained initial voltage value to be applied using PVS tables, if applicable
  - Making finer-grained adjustments based on hardware feedback, i.e., RPM-assisted AVS for VDD_CORE, VDD_GFX
- CXO buffers on the PMIC, the top-most node of the MSM clock tree
- Shared system buses, i.e., their clocks/capacity:
  - AFAB
  - System NoC
  - Config NoC
  - Peripheral NoC
  - Multimedia NoCs
- BIMC (DDR memory controller) and EBI1 clocks for DDR

Table 5-1 Key power domains on MSM8x74 and their owners

Domain name	Owner	Domain description
VDD_MEM	RPM	Power for all on-chip memories. Apps, modem, RPM, LPASS, etc., may need to have this domain in the On state, while others do not need it; hence, aggregation must occur at the RPM.
VDD_CORE	RPM	Power for digital core circuits, e.g., LPASS Hexagon processor, VFE, MDP, JPEG encode/decode, video encode/decode, Krait L2, BIMC, etc.
VDD_KRAIT (1, 2, 3, 4)	Krait/HLOS	Power for quad-Krait apps microprocessors. The HLOS is responsible for Krait voltage domain states. Hence, control is not hosted at the RPM. The Krait PDN has a unique implementation in the MSM8x74 and is discussed separately.
VDD_GFX	Krait/HLOS	Power for the graphics core. Since the software execution environment for GFX is part of the HLOS and no other subsystems on the SoC deal with GFX, GFX is also managed by Kraits.
VDD_MODEM	Modem	Power for the modem core and the Hexagon modem processor
VDD_ALWAYS_ON	Boot	Always-on power domain that includes MPM; set up during boot and then stays as-is
VDD_A1, VDD_A2	Modem	Power for low- and high-voltage on-chip analog circuits pertinent to the modem

Domain name	Owner	Domain description
VDD_P1, VDD_P2, .., VDD_P7	Boot	Power for MSM pad groups 1 through 7. Each pad group powers up a set of functionally and logically grouped pads/pins, e.g., PAD group 1 is the EBI1 interface PAD on the MSM, PAD group 2 is the SDC2 pad, etc. See *MSM8274/MSM8274AB, MSM8674/MSM8674AB, MSM8974/MSM8974AB Device Specification* (80-NA437-1) for more details.

NOTE: One way to get a sense of the shared resources controlled by the RPM on a software version is to look at the NPA dump taken from the RPM. It lists resources that are controlled by the RPM and implemented/modeled as NPA resources on the RPM.

The RPM's job is to effectively manage these shared resources. RPM masters directly control their individually owned resources. They send requests to the RPM when they need to change the state (increase/decrease/ on/off, etc.) of a shared resource. The RPM consolidates these requests from the RPM masters and determines the final state of each shared resource.

5.3.1.2 RPM interactions

Figure 5-5 shows relevant RPM interactions with the rest of the system. It shows one RPM master, the Hexagon modem processor. A similar configuration of hardware exists between other RPM masters and the RPM. Other RPM masters on MSM8x74 are:

- Krait processors and Krait L2 (together identified as one RPM master)
- Hexagon modem
- LPASS Hexagon (ADSP)
- WLAN/BT/FM A9

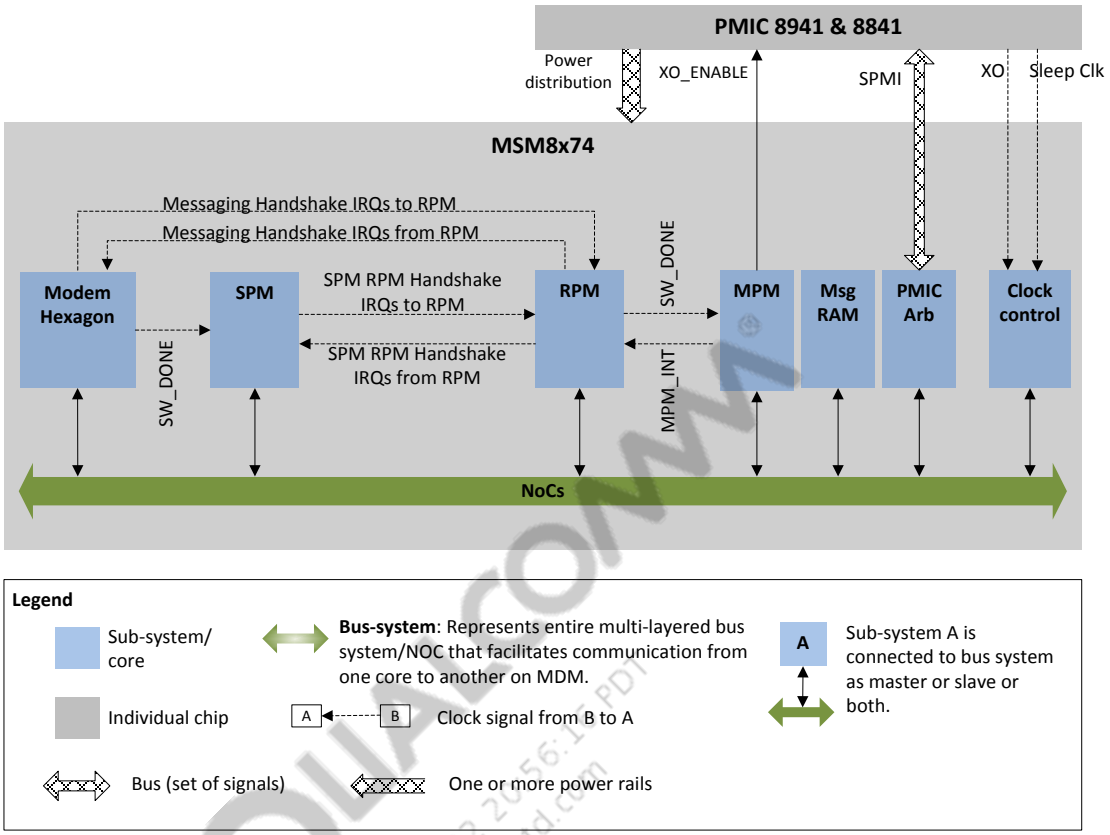


Figure 5-5 Hardware perspective, second-level decomposition (RPM interactions with rest of system)

The following is a brief description of the components and connectors in Figure 5-5:

- NoCs – This discussion does not delve into the details of exact bus structures and represents the entire multilayered bus system on the MSM8x74 as an abstraction with this component. The key idea is that the following data paths are available:
  - Data read/write path for RPM and masters to/from the message RAM via these bus interconnects
  - Data write paths from all processors, SPMs, and MPM to PMIC arbiters
- Hexagon modem – This is the example master processor for discussion purposes. It is capable of reading/writing to other subsystems/cores on the chip using the bus system. It can also be configured to assert an SW_DONE signal, with the execution of special instructions, etc. This is used to signal the SPM, which typically performs power collapse and sleep procedures beyond the point in time when the Hexagon modem processor is not capable of execution.
- SPM – This is the SPM associated with the Hexagon modem processor. All RPM masters have an associated SPM. In general, any SPM is designed to perform tasks, such as the following for a master entering sleep:
  - Assert signal lines that cause the master IOs to be clamped
  - Power collapse (via the on-chip switch or at the PMIC) the master processor, including its L1 cache, as applicable on a case-by-case basis
  - Handshake with the PMIC to turn off/lower the corresponding VDD, if applicable

- Handshake with the RPM to indicate that the master has been shut down so that the RPM can review the possibility of shutting down shared resources that it controls; basically, it interrupts the RPM, which then makes sleep set/active set transitions; concepts of active/sleep sets are discussed later with the RPM
- Wait for the wake-up request from the RPM

The actions performed by the SPM depend on what is supported for the respective master and what the master has asked for via configuring its SPM. In some cases, the master processor may configure the SPM behavior differently in different sleep cycles, based on what type of low power states make the most sense in the given conditions. All non-HLOS processors use a software component called dynamic sleep, which is designed to access the conditions and select the correct combination of sleep modes for its processor. Dynamic sleep is discussed in detail later in this document.

- RPM – Here, the RPM is shown as being capable of reading/writing to other subsystems/ cores on the system over the bus system in Figure 5-5. It can also assert a SW_DONE signal when an SWFI or equivalent instruction is issued by the software that is executing on the RPM. It uses a dedicated set of IRQs and predetermined space in a message RAM to communicate with each RPM master in the system. It also communicates with each master's SPM using dedicated IRQs when the master is entering or exiting sleep/shutdown. When the RPM is not processing requests, it places itself into SWFI or an equivalent state. This is the state in which the RPM typically spends the larger part of its time.
- MPM – The MPM is on the always-on domain and acts as the system sleep controller. It implements a state machine in the hardware, which is triggered when an SW_DONE signal is asserted by the RPM. The MPM state machine coordinates the last steps of chip-wide sleep after the RPM has stopped executing. This typically includes setting up the PMIC to disable CXO clock input to the MSM and minimizing VDD_CORE and VDD_MEM.
- Message (Msg) RAM – This is 16 KB SRAM. Msg RAM is dedicated to facilitating a shared memory-based communication between the RPM and its masters.

  Message-passing masters, e.g., Krait CPUs, audio/sensor processors, modem processors, etc., can provide change requests and desired levels of performance of shared resources via a message API and can then trigger the RPM with an interrupt of varying priority levels.

  Nonmessaging masters, e.g., the Krait L2, can provide resource change notifications via interrupts to the RPM. Controlling masters, e.g., the apps processor, would have a preconfigured messaging API for the RPM to use upon receiving such an interrupt from a nonmessaging master.

  Software protocol implementation is based on QTI SMD Lite architecture, with modifications to adapt it to the unique needs of the RPM. Each master processor utilizes an RPM driver within its execution environment to send messages to the RPM. Masters use these messages to make resource requests to the RPM.
- Clock control – Clock control is shown for the purpose of completeness and is relevant to the CXO shutdown discussion later in this document. The RPM that owns the CXO resource must delegate turnoff of the CXO to the MPM by programming the MPM. When the RPM decides to trigger CXO shutdown, it programs the MPM and asserts an SW_DONE signal to the MPM, which then deasserts the XO_ENABLE signal to the PMIC to turn off the CXO buffers at the PMIC.
- PMIC arbitrator (arb) – The PMIC arb arbitrates SPMI commands to the PMIC from different sources on the MSM8x74.

5.3.1.3 RPM and process monitor core

The process monitor core works in conjunction with the RPM and provides an AVS-like feature for the shared rails VDD_CORE and VDD_GFX. The process monitor core has the capability to compute voltage compensation values for process and temperature variations, using a special network of sensors across the SoC. These compensations are very fine adjustments, typically on the order of 5 to 12.5 mV.

The RPM configures the process-monitoring hardware and then requests voltage adjustment suggestions around the operating voltage corner selected by the RPM for VDD_CORE and VDD_GFX. When the computation results are ready, the process-monitoring block interrupts the RPM, which can then read these values from the process-monitoring block and make a decision about whether to apply these adjustments based on several other constraints, i.e., that VDD_MEM must always be maintained at a voltage higher than VDD_CORE.

This only applies to active case voltage corners, not to the retention case of VDD minimization.

Figure 5-6 shows the on-chip and off-chip hardware entities involved in achieving this fine-grained voltage compensation.

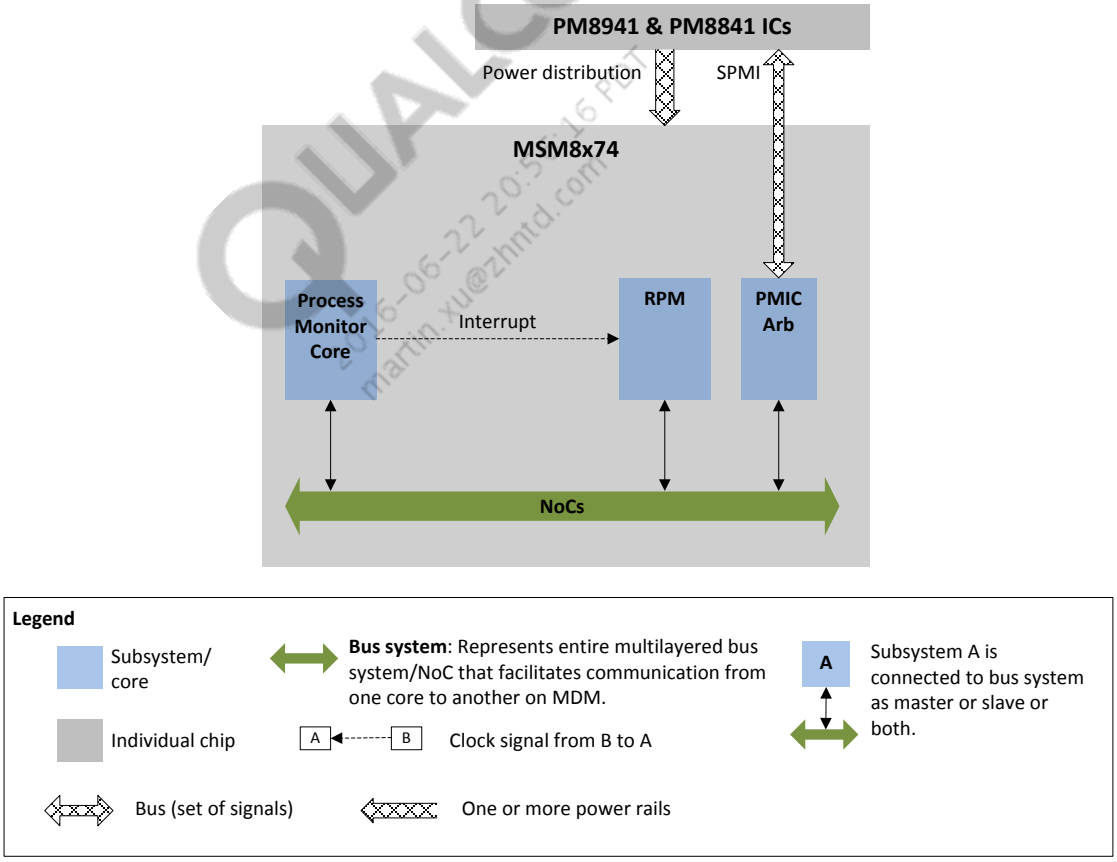


Figure 5-6 Hardware perspective, second-level decomposition (RPM and process monitor core)

The components and connectors are:

- Process monitor core – This hardware block can be requested to compute a voltage compensation value for VDD_CORE and/or VDD_GFX around the current operating voltage corner, via a set of register writes. Computation results are dumped in registers and an interrupt is generated.
- RPM – This processor hosts the driver software to initialize and exercise the process monitor core. It interprets the suggested values and makes a decision about whether to apply the suggested values based on several other criteria that may influence the final minimum voltage value that must prevail. For example, if a voltage rail client requested an absolute voltage at the rail, rather than a voltage corner, then that absolute voltage or higher must prevail at the rail and process monitor compensations must be disregarded.
- PMIC arb – This is the gateway to access the PMIC on the SPMI interface.
- NoCs – In this case, the NoC is the configuration NoC of Figure 2-2.
- Interrupt – This represents the capability to use the process monitor core in an interrupt-driven way.
- PM8941 and PM8841 ICs – If the RPM software decides to apply a voltage compensation, it must update a PMIC with the new voltage settings for VDD_CORE and/or VDD_GFX.
- SPMI – The PMIC register writes for voltage adjustment occur over this interface.
- Power distribution – These represent the voltage rails, i.e., VDD_CORE and VDD_GFX, that are the subject of discussion in this feature.

5.3.2 Runtime perspective

Figure 5-7 shows the RPM software first-level decomposition in a runtime perspective.

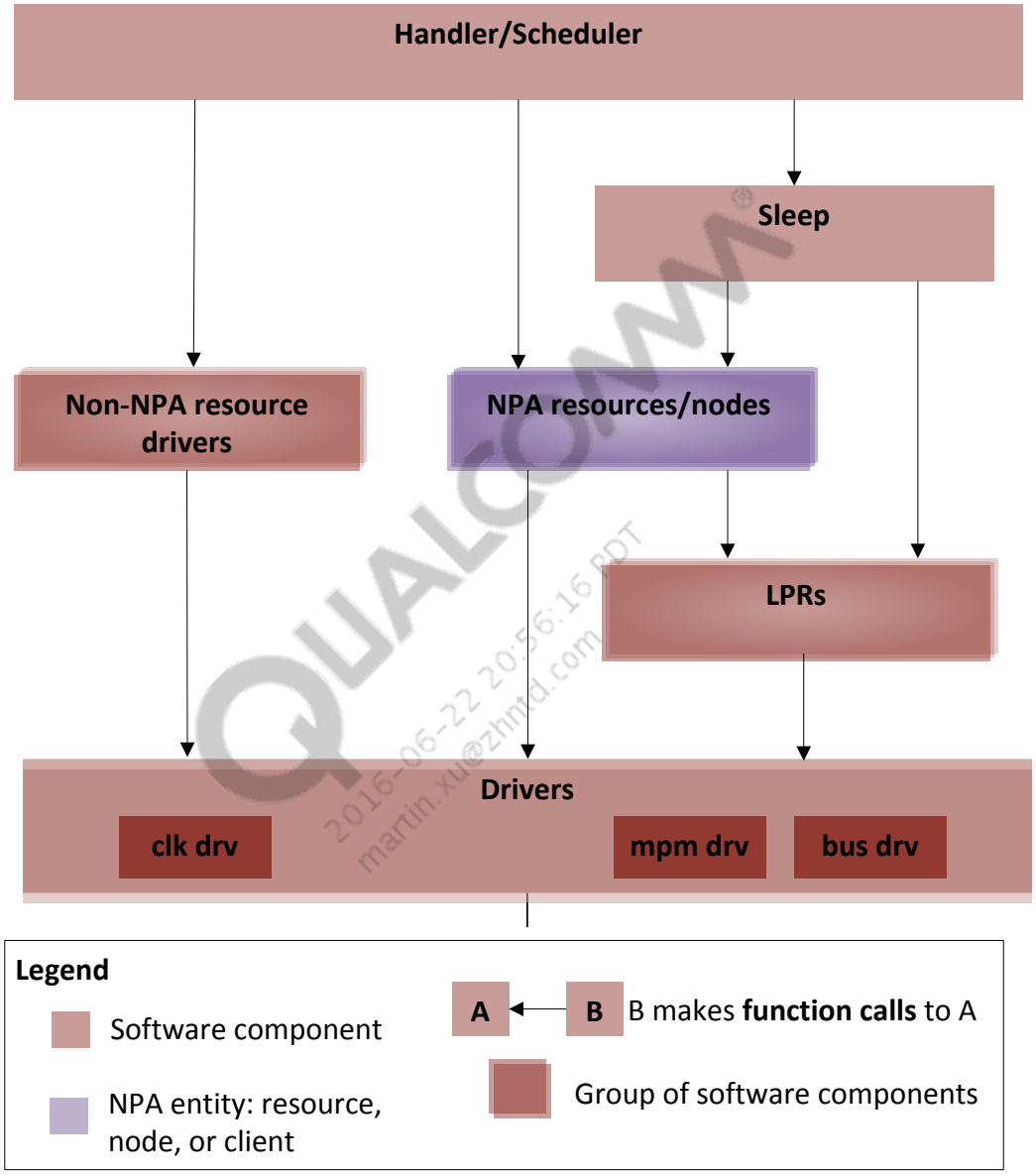


Figure 5-7 RPM software first-level decomposition (runtime perspective)

Key components in the RPM software are:

- Handler/scheduler – This is an infinitely executing function that schedules processing of incoming messages and, when there are no messages to schedule, schedules the execution of sleep. RPM firmware is designed as a cooperative kernel; further details are outside the scope of this document.
- Sleep/active sets (not shown in the diagram) – All resource requests from a master to the RPM must fall under one of two sets: active set requests or sleep set requests. All requests from a master that must be provided while the master is active are requested as part of the active set. Requests that must be provided while the master is in sleep (power-collapsed with RPM notification) are requested as part of the sleep set. In other words, any request from a master must be qualified as being for the sleep set or the active set.

  The RPM always maintains a sense of whether a master is in the Active state or Sleep state, and invokes a corresponding set for that master. This approach makes transitions into and out of sleep very fast, by obviating the need for the master to reconfigure/rerequest new configurations each time while entering and exiting sleep.

  The RPM services the active set requests as and when they arrive, because when the request comes from a master it means the master is active. When the master prepares to sleep, it updates its sleep set configuration, if needed. Eventually, a master indicates it is entering sleep via an SPM-RPM interrupt. The RPM then invokes a sleep set configuration of that master.

  In the MSM8960 family of chipsets, the active set and sleep set configurations for masters were implemented as a set of centralized data structures.

  In the MSM8x74 family of chipsets, a resource-specific driver on the RPM maintains a record of what each master requested for active and sleep sets, as far as that resource is concerned. When the RPM realizes that a master transitioned from the Active state to the Sleep state, or vice-versa, it notifies the registered resource drivers (which master and what state). The drivers compute the new global state of the resource they control and apply that state. This typically leads to drivers adjusting shared resources such as clocks, voltage rails, etc.

  When it wakes up a master from sleep, the RPM wakes up the master with its active set configuration. Before the RPM handshakes with the SPM to wake up the master, it applies an active set for that master. This triggers the driver chain to recompute the new state for the affected resources and apply those state changes. When this is complete, the master is brought out of sleep.

  There is an exception to this with regard to the modem processor, which utilizes the concept of a next active set. This extra set definition facilitates a faster transition in and out of sleep for the modem for those cases where the modem was in a different active set configuration just before going into sleep, and will want to wake-up to a different active set configuration, which it knows well in advance.
- Sleep – The RPM implements a lightweight version of dynamic sleep, similar to what is implemented for software running on a Hexagon modem and LPASS processors. Sleep implementation on the RPM is revisited in more detail later in this document.

- LPRs – These are a sleep-specific concept and expand as low power resources. All resources that cannot enter their Low Power mode until sleep executes on that execution environment fall under this category. CXO is an example of this. Even though all software clients of the /xo/cxo node may have voted off /xo/cxo (meaning they do not need CXO any more), the /xo/cxo resource is not actually turned off, because it can only be turned off when the RPM processor has stopped executing all instructions. This, in turn, occurs when sleep executes. The LPR blocks in Figure 5-7 represent the collection of data and functions that implement the details of how each low power resource is handled. For example, an LPR for a CXO resource would implement functions that can be called to program the MPM to put off CXO after an RPM processor halts. It would also provide other housekeeping functions specific to low-level management of CXO.
- NPA resources/nodes – Several resources on the RPM are modeled as NPA resources, e.g., the xo/cxo node that represents the CXO buffer. Different software entities on the RPM, which can also be proxies of their counterparts on other processors, can become clients of this xo/cxo NPA node. These clients can then make requests to the xo/cxo node indicating whether they need the CXO. In addition to nodes such as /xo/cxo, a couple of logical nodes, i.e., /sleep/latency and /sleep/wake-up nodes, are described in the discussion of dynamic sleep.
- Non-NPA resources – This group of software components represents resource drivers that are not implemented as NPA resources.
- Drivers – At the bottom of the stack are drivers that know how to change clocks, program the MPM, scale bus, etc.

Function calls from the handler service into sleep and resources are handled as follows. The handler service invokes the NPA or non-NPA type relevant resource when there is a message/ request from a master to update the resource state for the active or sleep set. It also invokes the resources when there is a transition from the active set to the sleep set for a particular master, and vice versa. The handler service invokes sleep when there are no more messages/requests to process. This is the state in which the RPM software is expected to spend most of its time.

Function calls from resources to LPRs represent the control flow that enables or disables and LPR, e.g., if the /xo/cxo resource node was to be voted-off (no outstanding client requests asking for the CXO to be On), this would enable the LPR corresponding to CXO.

Function calls from sleep to LPR represent the fact that, when sleep executes, it queries each LPR to determine if that LPR is enabled or disabled. Sleep only considers entering, i.e., putting into Low Power mode, enabled LPRs.

Function calls from resources and LPRs to low-level drivers represent the fact that APIs/services of these drivers are used to access the specific hardware in question.

Figure 5-8 shows a hypothetical example of how a request from the Hexagon modem processor to raise the SYSNOC performance level goes to the RPM. It does not show all of the drivers involved. There is an NPA node on the Hexagon modem that aggregates all requests for SYSNOC local at the modem and then sends these requests to the RPM, which has a SYSNOC NPA node that aggregates all requests for SYSNOC from different processors on the SoC.

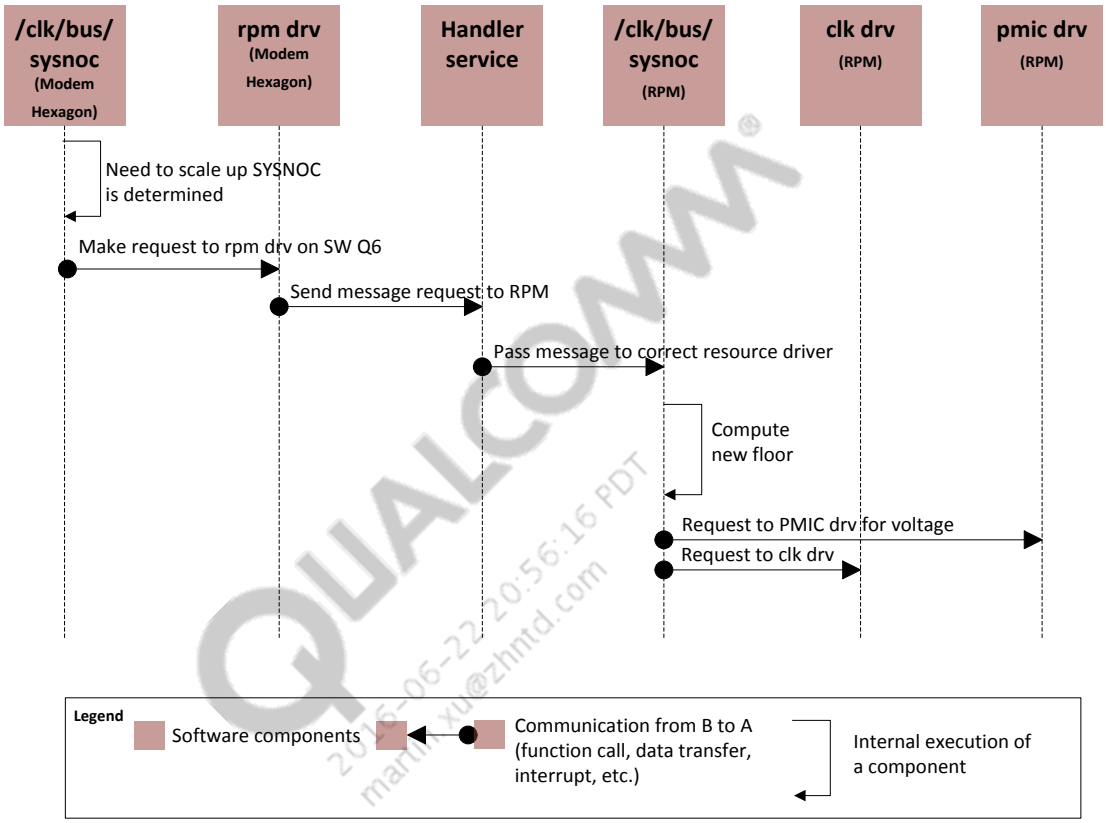


Figure 5-8 RPM software first-level decomposition (runtime perspective) – Shared resource (bus) update example

5.3.3 Static perspective

Table 5-2 shows the static perspective.

Table 5-2 Static perspective

Subsystem	Source codes
Sleep subsystem	▪ rpm_proc/core/power/sleep/inc/sleep_v.h ▪ rpm_proc/core/power/sleep
RPM main/message processing loop	rpm_proc/core/bsp/rpm/src/main.c
CLOCK DRV	▪ rpm_proc/core/api/systemdrivers/clock.h ▪ rpm_proc/core/systemdrivers/clock
MPM DRV	▪ rpm_proc/core/api/power/mpm_utils.h – MPM public utils API, e.g., reading the MPM TIMETICK counter ▪ rpm_proc/core/power/mpm/inc/mpm.h – MPM driver interface for use by sleep and other power-specific software ▪ rpm_proc/core/power/mpm – Larger MPM driver implementation
PMIC DRV	rpm_proc/core/systemdrivers/pmic
BSP/HWIO	rpm_proc/core/bsp
NPA framework	rpm_proc/core/power/npa

5.4 RPM deep dive

5.4.1 RPM sleep – Sleep subsystem interfaces

[decomposition_level:3] system_architecture/rpm /sleep

RPM sleep provides decision-making logic for selection of one of the three Low Power modes when the RPM becomes idle.

- RPM halt (essentially a local Low Power mode)
  - The RPM processor is put in a Halt state and any interrupt brings it out of this state. This mode has minimal overhead and power savings.
- XO shutdown (systemwide Low Power mode)
  - Systemwide Sleep mode, where all processors and subsystems are nonfunctional and the root of the clock tree (XO buffers at the PMIC) is turned off. This mode has significant power savings and time overhead to enter and recover from.
- VDD minimization (systemwide Low Power mode)
  - This mode is a super set of XO Shutdown mode and involves lowering of the VDD_CX and VDD_MX voltage rails to retention levels after executing XO-shutdown, therefore saving more power at the cost of added time to enter and exit the mode.

This section models the entire sleep-specific software as one sleep subsystem. It shows its interface/interactions with the rest of the RPM software components at runtime and the source code organization for the respective runtime components.

5.4.1.1 Hardware perspective

This subsection is not applicable to this version of the document.

5.4.1.2 Runtime perspective

Figure 5-9 shows the runtime perspective [decomposition_level:3] system_architecture/ rpm/sleep.

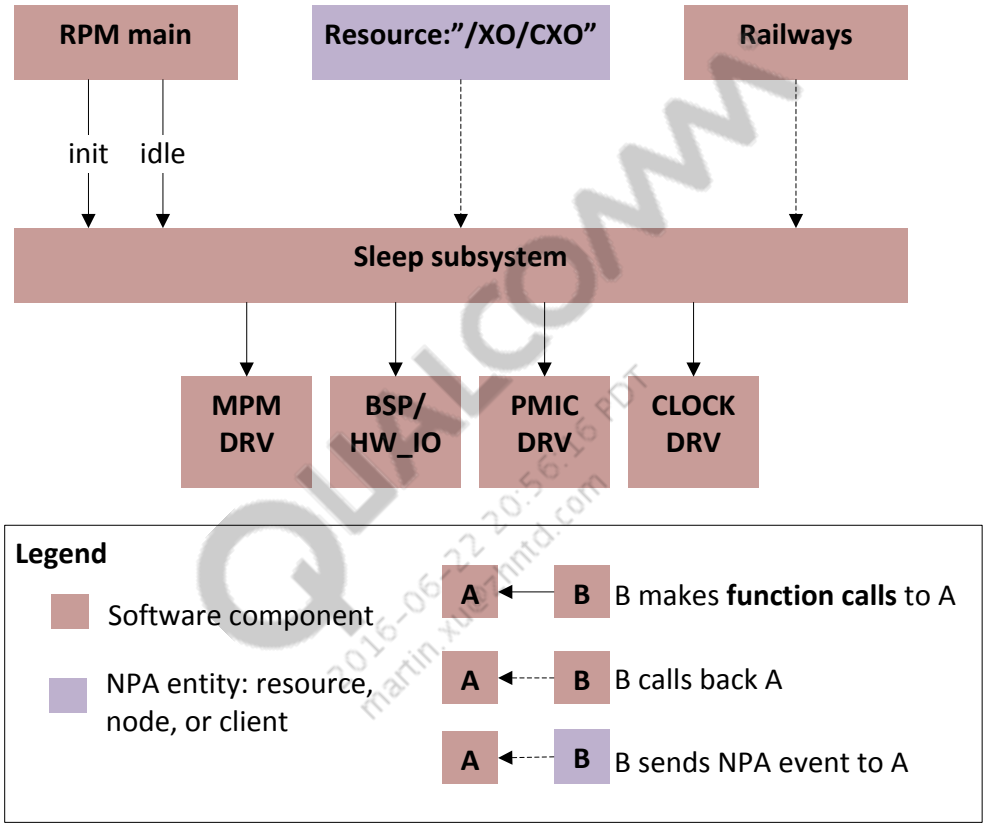


Figure 5-9 Runtime perspective [decomposition_level:3] system_architecture/ rpm/sleep

The RPM main calls two sleep APIs, the init API (sleep_init) at the time of initialization and the idle API (sleep_perform_lpm), which is called when the RPM main scheduling loop determines that it is idle.

During the init stage, the sleep subsystem registers with the /XO/CXO NPA resource for a state change NPA event. Next, the /XO/CXO NPA resource calls back into the sleep subsystem with state change information each time a state change occurs for /XO/CXO. The sleep subsystem uses this event notification to set up (enable/disable) the XO shutdown and VDD min LPRMs.

During the init stage, the sleep subsystem also registers a callback with railways (subsystem), which calls back into the sleep subsystem each time a VDD-CX/MX operating voltage level is changed. At the time of entering XO shutdown or VDD min, sleep uses this information to set up the system to wake up to the last used VDD-CX/MX operating voltage levels.

The sleep subsystem calls one or more low-level driver APIs (MPM, PMIC, CLOCK, HW_IO) to enter/exit the XO shutdown/VDD min or RPM Halt modes.

5.4.1.3 Static perspective

Table 5-3 shows the static perspective.

Table 5-3 Static perspective

Subsystem	Source codes
Sleep subsystem	▪ rpm_proc/core/power/sleep/inc/sleep_v.h ▪ rpm_proc/core/power/sleep
RPM main	rpm_proc/core/bsp/rpm/src/main.c
CLOCK DRV	▪ rpm_proc/core/api/systemdrivers/clock.h ▪ rpm_proc/core/systemdrivers/clock
Railways	▪ rpm_proc/core/api/railway.h ▪ rpm_proc/core/power/railway
MPM DRV	▪ rpm_proc/core/api/power/mpm_utils.h – MPM public utils API, such as reading MPM TIMETICK counter ▪ rpm_proc/core/power/mpm/inc/mpm.h – MPM driver interface for use by sleep and other power-specific software ▪ rpm_proc/core/power/mpm – Larger MPM driver implementation
PMIC DRV	rpm_proc/core/systemdrivers/pmic/target/msm8x74_pm8941_pm8841/rpm/npa
BSP/HWIO	rpm_proc/core/bsp
Resource:"/xo/cxo"	rpm_proc/core/systemdrivers/clock/hw/msm8x74/ClockRPMNPA.c

5.4.2 RPM sleep – Sleep subsystem internals

[decomposition_level:4] system_architecture/rpm /sleep/sleep_components

This section breaks down the sleep subsystem into its key internal runtime components and provides source code organization information.

The decision-making logic that determines what Low Power mode to eventually enter during sleep is a two-level process. At the first level, LPRMs are enabled/disabled based on runtime input from other parts of the software. At the second level, the sleep subsystem computes the most optimal combination of Low Power modes that should be entered, based on the time cost and power savings of each mode and the knowledge of when the system needs to reawake.

5.4.3 Hardware perspective

This section is not applicable to this version of the document.

5.4.4 Runtime perspective

Figure 2-2 shows a runtime view of the key components that form the sleep subsystem and how these internal components deliver the external interfaces of the sleep subsystem.

The sleep subsystem on the RPM models two LPRs, rpmsleep and deep_sleep_common. The rpmsleep LPR models three mutually exclusive LPRMs, rpm_halt, xo_shutdown, and vdd_min. Each of these three corresponds to a unique combination of hardware states for low power when the software is in sleep. Deep_sleep_common is a facilitator LPR that helps reuse common code on a memory-constrained RPM with just one LPRM, also named deep_sleep_common. This LPR implements common steps that are applicable when entering and exiting xo_shutdown or vdd_min modes. Consequently, each time rpmsleep.xo_shutdown mode is entered, deep_sleep_common.deep_sleep_common is also entered to execute the common portion of the steps.

In the runtime representation, LPRMs are captured as a collection of structures that aggregate functions and other LRPM data and functions, such as enter and exit, that may be used to enter/exit the mode, as well as other functions that return time/power costs of entering and exiting that mode. LPR is represented as a set of structures/functions that aggregate LPRMs.

The /sleep/uber NPA node aggregates all of the RPM LPRs and, therefore, LPRMs. Any runtime components that wish to enable/disable an LPRM must become clients of this NPA node and must issue appropriate NPA requests to this node. In the existing system, the sleep monitor (described next) is the only client of this NPA node. It issues requests to enable/disable xo_shutdown and vdd_min on behalf of the /XO/CXO NPA node. The rpm_halt LPRM and deep_sleep_common are always enabled.

The sleep monitor is a collection of structures and functions at runtime that tracks two key events in the RPM, any change in the state of the /XO/CXO NPA node state (needed/not needed) and any change in the value of Active mode VDD_CX/VDD_MX voltage levels.

The CXO status is used to enable or disable the xo_shutdown and vdd_min LPRMs by issuing appropriate NPA requests to the /sleep/uber NPA node. Therefore, in this context, the sleep monitor serves as a proxy for the /XO/CXO NPA node and makes requests to /sleep/uber on its behalf.

The sleep monitor also tracks active mode VDD_CX/VDD_MX voltage levels, so that each time it brings the system out of sleep, it brings it back to the same voltage levels for VDD_CX/VDD_MX.

At init time, the sleep monitor registers for the NPA event with a /XO/CXO resource and for a post-voltage change callback with the railways subsystem. It also registers as an NPA client of the NPA node /sleep/uber. For each /XO/CXO state change event, the sleep monitor issues an NPA request to /sleep/uber node.

The NPA node /sleep/lpr acts as an NPA dependency node for /sleep/uber node, which represents the LPRs/LPRMs. Briefly, this design implies that even though certain LPRMs are enabled at an arbitrary time as far as /sleep/uber is concerned, those LPRMs have a dependency on /sleep/lpr, which ultimately dictates when (if at all) those LPRMs are entered. This dependency relationship allows a more elegant decoupling of /sleep/uber/ and its LPRMs from the other two key components of the sleep framework, sleep solver and sleep_perform_lpm, which do not have direct knowledge of available and enabled LPRs/LPRMs and indirectly refer them via /sleep/lpr node. (At init time, an NPA node such as /sleep/uber that aggregates one or more LPRs must register its LPRs with /sleep/lpr node, which internally tracks the registered LPRs. LPRs not registered will not be known to /sleep/lpr and, therefore, will not be evaluated by sleep solver/sleep_perform_lpm.) The detailed semantics of NPA dependencies are outside the scope of this discussion.

Sleep solver is a collection of structures and functions that collectively provide use of an available solver-algorithm, which takes as input the list of available enabled LPRMs, wake-up time, and latency constraints, and computes an optimal LPRM combination that should be selected for the given sleep cycle.

The sleep_perform_lpm function is called in the main context of RPM when the RPM scheduler determines there is an idle condition. This function maintains a client handle to /sleep/lpr node and uses it to find the list of enabled LPRMs. It fetches wake-up and latency information from yet another two NPA nodes (not described in this discussion) and then calls upon sleep solver with these as inputs. Sleep solver returns an optimal LPRM combination (list of pointers to LPRMs) and then enters functions of all the LPRMs that are called. Following this, the software sleeps. This is followed by wake-up, when exit functions for the same set of LPRMs are called.

Figure 5-10 shows runtime perspective [decomposition_level:4] system_architecture/rpm/sleep/sleep_components.

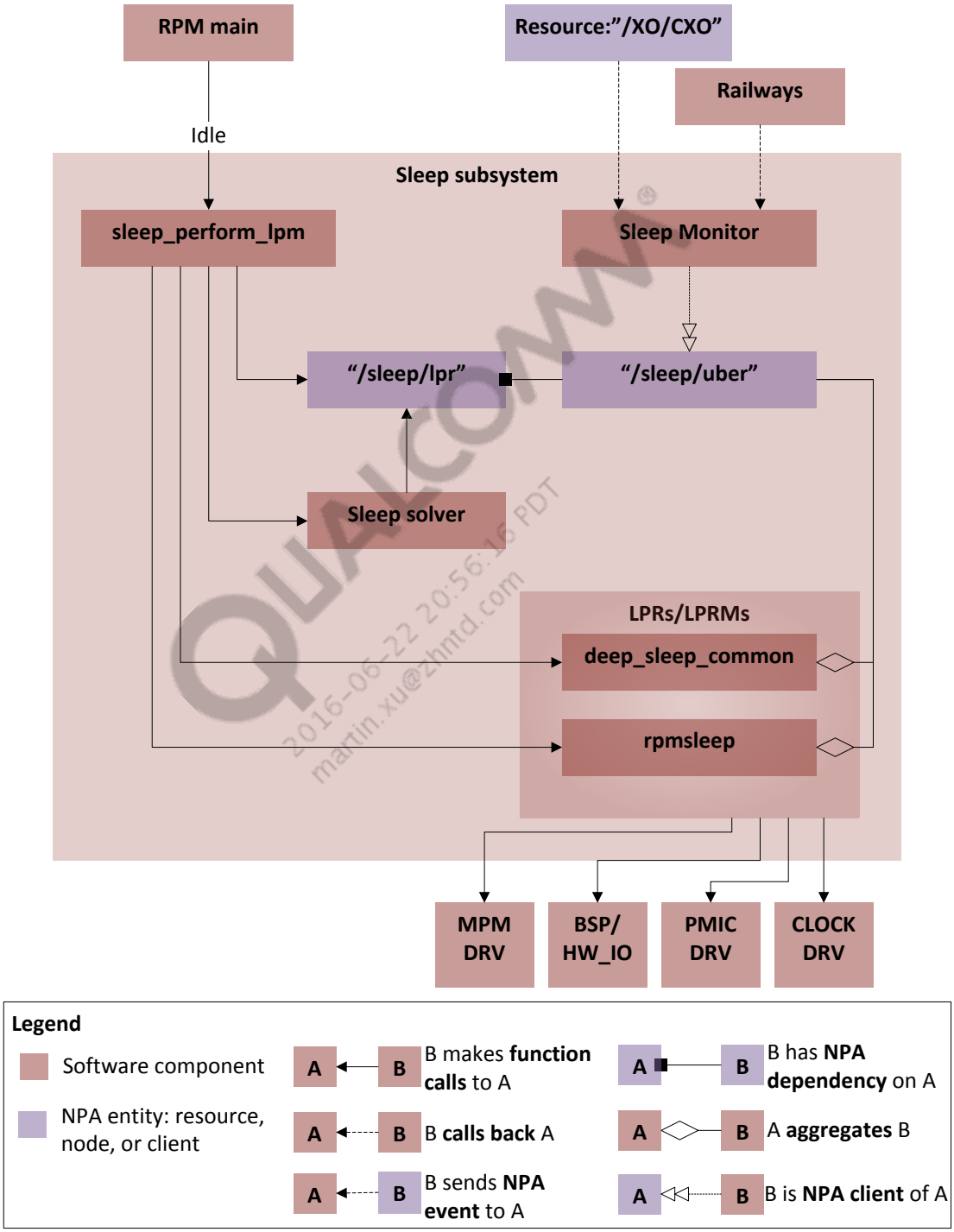


Figure 5-10 Runtime perspective [decomposition_level:4] system_architecture/rpm/sleep/sleep_components

5.4.4.1 Static perspective

Table 5-4 Static perspective

Subsystem	Source codes
sleep_perform_lpm	▪ rpm_proc/core/power/sleep/inc/sleep_v.h ▪ rpm_proc/core/power/sleep/src/os/rpm/sleep_os.c
Sleep Monitor	▪ rpm_proc/core/power/sleep/src /asic/rpm/sleep_target.c
"/sleep/uber"	▪ rpm_proc/core/power/sleep/src /asic/rpm/node_definition_uber.c
"/sleep/lpr"	▪ rpm_proc/core/api/power/sleep_lpr.h ▪ rpm_proc/core/power/sleep/src/npa_nodes/sleep_lpr.c, NPA resource/node implementation, including driver functions, etc.
Sleep solver	
LPRs/LPRMs	▪ rpm_proc/core/power/sleep/src /asic/rpm/lpr_definition_uber.c, actual definitions of the enter/exit functions for LPRMs ▪ rpm_proc/core/power/sleep/src/lpr/SleepLPR_lookup_table.c ▪ rpm_proc/core/power/sleep/src/lpr/LPR_SLEEP_MODES.c

5.5 Modem power

[decomposition_level:2] system_architecture/modem

5.5.1 Hardware perspective

This section presents a high-level hardware perspective of the modem subsystem from the point of view of power.

Figure 5-11 shows the modem subsystem, second-level decomposition (hardware perspective).

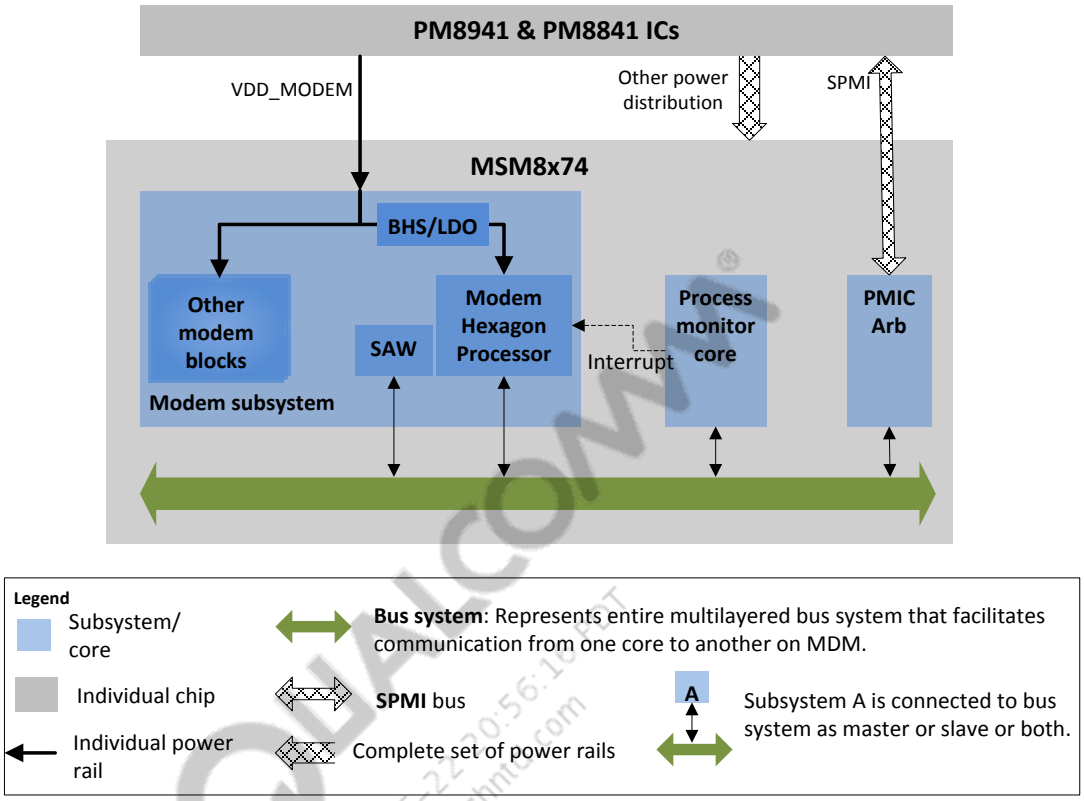


Figure 5-11 Modem subsystem, second-level decomposition (hardware perspective)

Some of the hardware features shown in Figure 5-11 are:

- VDD_MODEM – The dedicated power rail feeding all cores/blocks in the modem subsystem, including the Hexagon modem processor. One dedicated regulator from the PMIC feeds VDD_MODEM on the MSM.
- BHS/LDO for modem processor – This block provides a power-collapsing modem processor, even when another block in the modem subsystem needs VDD_MSS to be On. The LDO part of this block allows scaling of the modem processor to a lower voltage than the combined VDD_MODEM operating voltage, if runtime conditions permit.
- Process monitor core – Plays the same role for VDD_MODEM in conjunction with the modem processor as it does for VDD_CORE and VDD_GRAPHICS in conjunction with the RPM. See Section 5.3.1.3 for more details.

The other items in the figure were discussed in previous contexts and are included in the figure for purposes of completeness.

5.5.2 Runtime perspective

This section presents a high-level runtime view of the software running on the Hexagon modem processor.

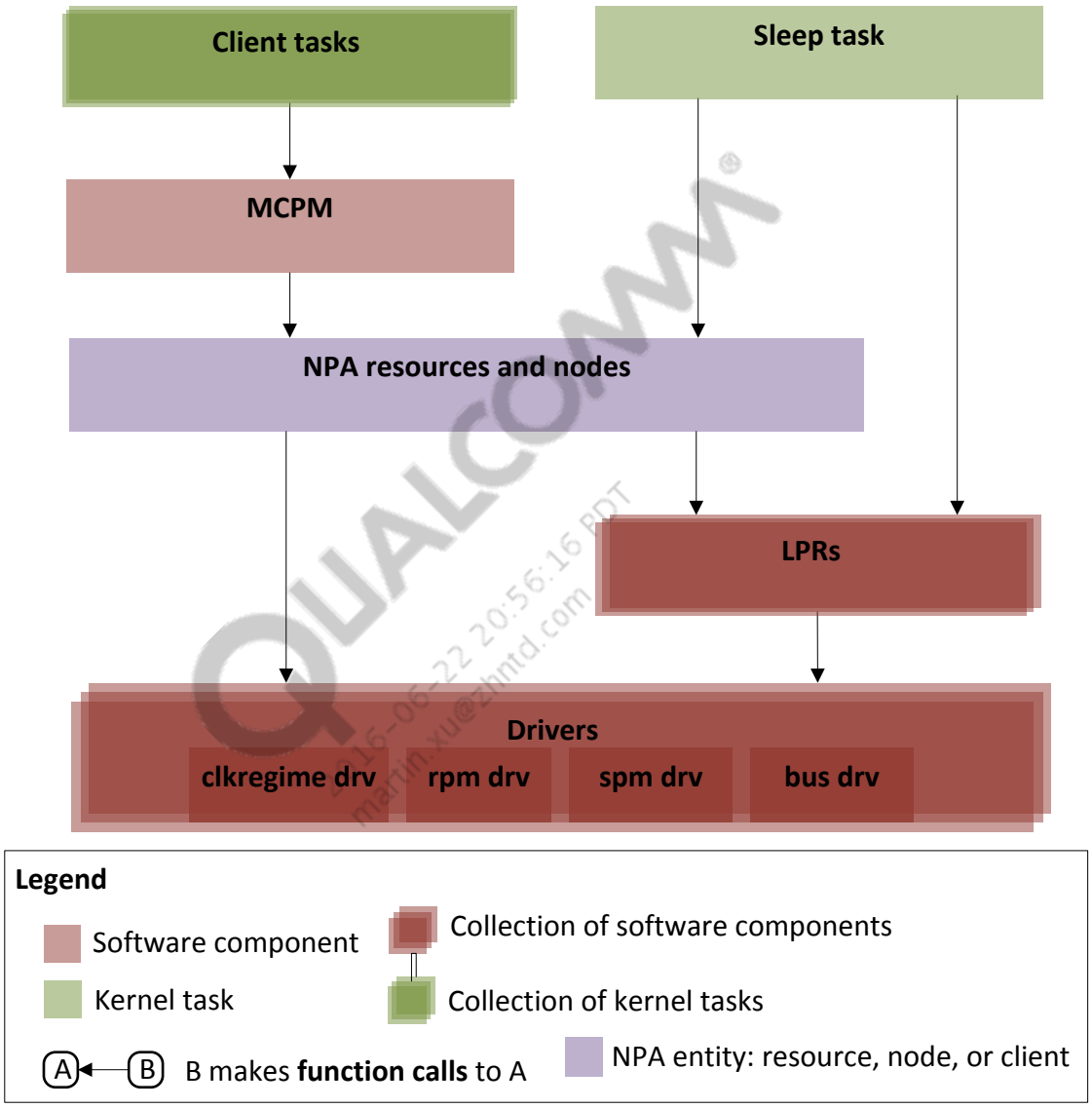


Figure 5-12 Modem software first-level decomposition (runtime perspective)

Some of the software components shown in Figure 5-12 are:

- Tasks – The Hexagon modem processor runs the QuRT™ kernel, and most of the applications/services execute a large part of their work within the context of these tasks. Figure 5-12 shows the client applications/services as tasks. This also emphasizes that calls into blocks that are lower in the stack, such as MCPM, and NPA requests run in the context of the client task.

- Node Power Architecture (NPA) resources/nodes – The NPA framework is described in Section 5.2. This block in Figure 5-12 represents all resource drivers on the modem processors that are implemented as NPA resources/nodes. These include, but are not limited to, specific buses, CPU MIPS, CXO, PXO, etc.
- LPRs – From a Hexagon modem perspective (or, for that matter, any other processor in the system), there are resources that cannot be put into the Off state or some other low power state by the software executing directly on the processor, e.g., a processor VDD rail that cannot be put into the Off state directly by the processor software. This would mean removing power while the processor is still executing. Other hardware components need to mediate this activity for a reliable shutdown. LPRs are software abstractions of such resources (the VDD rail in the above example) and their LPR states (Off state). They typically comprise data structures and functions that achieve the desired mediation from other hardware to negotiate a resource shutdown. This is discussed in more detail with concepts of dynamic sleep. LPRs will only implement those aspects of a resource that are needed to put them into a particular low power mode. There must still be an NPA resource corresponding to that LPR, which exposes that resource to the rest of the software and implements and handles dependencies with the rest of the resources/software.
- Sleep task – This is the lowest priority task that is scheduled when there is no other task to run. This task is explicitly shown here for its vital role in power management and is the focus of discussion in a dedicated Section 5.6.1. Conceptual implementation of sleep on the Hexagon modem is identical to what was discussed for RPM in Sections 5.4.1and 5.4.2.

  Sleep calls into NPA resources in Figure 5-12 represent sleep calls into two specific NPA nodes, wake-up and latency, that aggregate wake-up and latency requirements from software clients on the modem. Sleep uses these latency and wake-up values to compute what low power modes may be entered.

  Sleep also calls into LPRs when one or more of these needs to be entered/exited.
- Other tasks – In the discussion of power management, all other tasks are collapsed and details are rarely provided.
- Modem Clock and Power Manager (MCPM) – The MCPM is a centralized controller for the modem core (different from the Hexagon processor) clock speeds, modem core voltage and power collapse, modem block configuration, and Hexagon processor clock speeds. Its decisions are closely paired with the mode of operation of the modem at any given time. Clients make requests to the MCPM that are indicative of the change in modem operation, which computes a new operational state and maps it to a new modem clock and power configuration. Therefore, modem operation is broken down into modes of operation such as voice call, data call, standby, etc. For each of these modes, clock and power configuration are statically predetermined.
- Drivers – These are the low-level drivers that understand the intricate details of working with specific hardware on the MSM. Figure 5-12 shows the more commonly used drivers. In this document, additional drivers are introduced as required in discussion of a particular power feature.

5.5.3 Static perspective

This subsection is not applicable to this version of the document.

5.6 Modem power deep-dive

5.6.1 Dynamic sleep

Dynamic sleep is a data-driven approach to putting the device into the lowest possible power state, given the latency and wake-up constraints. The larger theme is that certain resources are designated as LPRs from a processor perspective, as already described. Each such LPR has one or more LPRMs that the resource can enter when the processor sleeps. When the processor becomes free, the sleep task runs and selects no LPRM or one LPRM for each LPR. In this way, the software latency and wake-up requirements are met and power consumption of the combination is also at a minimum.

5.6.1.1 Hardware perspective

This subsection is not applicable to this version of the document.

5.6.1.2 Runtime perspective

In describing the software concepts around sleep implementation, an example is provided to show how the sleep software determines the state of the Hexagon modem processor power rail once the processor has entered sleep and has stopped executing instructions (SWFI or the equivalent).

Figure 5-13 shows runtime components of the software that collaborate in this decision-making and execution.

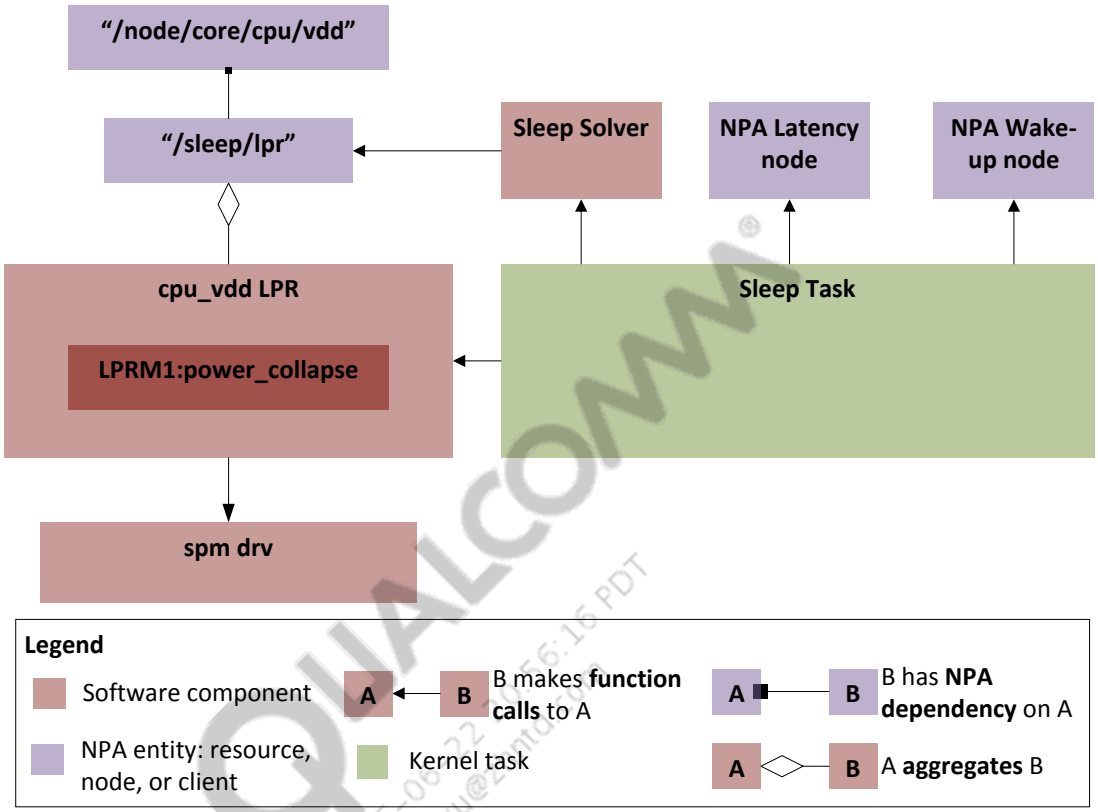


Figure 5-13 Hexagon modem software, second-level decomposition (runtime perspective) – Dynamic sleep hypothetical example

The state of the CPU voltage rail is modeled primarily via two runtime components, an NPA node and an LPR.

- /node/core/cpu/vdd (NPA resource node) – This NPA node exposes the state of the CPU VDD to the rest of the software executing on the modem processor. Assume that the CPU VDD can be either On or power-collapsed/Off. This node, therefore, models the CPU VDD as On or Off. Other software subsystems on the modem processor may register with this NPA node as clients and may be able to query its state, as well as make requests asking it to be On at any point in time. When no client is explicitly requesting the CPU VDD to be On, this node sets the state to Off. This does not mean that the CPU VDD is just turned off, it only implies that sleep may consider putting the CPU VDD to Off, when it has a chance to execute.
- cpu_vdd LPR – An LPR is a collection of one or more LPRMs. Each LPRM models the software execution of how to put a resource into that particular mode. The cpu_vdd LPR supports one LPRM, which is called power-collapsed, and provides functions and structures for putting the CPU VDD into the Off state.

- LPRM1 (power-collapsed) – This LPRM corresponds to power collapsing of the VDD to the Hexagon software processor and comprises a pair of enter/exit functions that can be used to enter and exit this mode, as well as associated data structures. It also provides overhead (latency, etc.) information pertaining to entering and exiting this mode. This information can be used to determine if it is beneficial to enter this mode. In the example, if sleep executes and /node/core/cpu/vdd is in the Off state, sleep uses this LPRM to execute turning off the CPU rail, if it decides to do so.
- /sleep/lpr (NPA resource) – This NPA node is the housekeeping node used by the sleep subsystem. All LPRs and LPRMs are registered with this NPA node. It also records the most current status of each corresponding NPA node. In the example, when /node/core/cpu/vdd changes its state, it also updates this state change with /sleep/lpr. When sleep executes, this node hosts the status regarding which LPRMs are enabled across the supported set of LPRs.
- NPA latency node – This is a logical NPA node (not relating to any physical system resource) that aggregates the latency requirements of all its clients. On the Hexagon modem, L1 clients such as GERAN, GPS, LTE, etc., provide latency requirements via this NPA node, when applicable. A latency requirement of 10 ms from a client would mean that, when an interrupt is fired, that client expects to service it no later than 10 ms.
- NPA wake-up node – The wake-up node covers two types of wake-ups – hard wake-up, which is the time until the next scheduled wake-up, and soft wake-up. For hard wake-up on the modem processor, this comes from timers or the sleep controller, which determines when the modem wakes up for the next page wake-up, etc. Soft wake-up is something the clients can hint if they know when the next wake-up interrupt is expected. This saves sleep from entering, only to exit again. LTE L1 is a soft wake-up client on the modem processor. Soft wake-ups do not cause scheduled wake-ups from sleep.
- SPM driver – When the enter functions for LPRMs are called, they typically interact with the SPM driver and set up the SPM for performing the desired action when the processor enters sleep. In the example, if sleep decides to enter a power_collapse LPRM for the processor rail, it calls the enter function provided by the power_collapsed LPRM, which makes calls into the SPM driver to set up SPM registers so that when the processor enters WFI (or the equivalent), the SPM shuts off the power to the modem processors.
- Sleep solver – Sleep solver defines the algorithm used by sleep to select a subset of the enabled LPRMs, so wake-up and latency requirements are honored and the system goes to the lowest possible power mode.
- Sleep task – This is the lowest priority QuRT kernel task that begins when the modem software becomes idle. The sleep task locks interrupts, reads latency, wake-up nodes, and all registered LPRs, and produces a list of available LPRMs. It then feeds this information to the sleep solver, which returns an ordered list of selected LPRMs. The sleep task calls the enter functions of each LPR. SWFI is the last on the list, which causes the modem processor to issue SWFI (or an equivalent). When the processor is awakened from sleep, it resumes execution where it left, sleep calls the exit functions for each LPRM in the reverse order, and releases any interrupt locks.

NOTE: The LPR/LPRM example shown in Figure 5-13 is hypothetical, in that the actual implementation implements the cpu_vdd LPR, but it has a slightly different set of LPRMs. See Table 5-5 for the specific set of LPRs/LPRMs implemented on the modem processor.

Figure 5-14 shows some of items that were previously discussed.

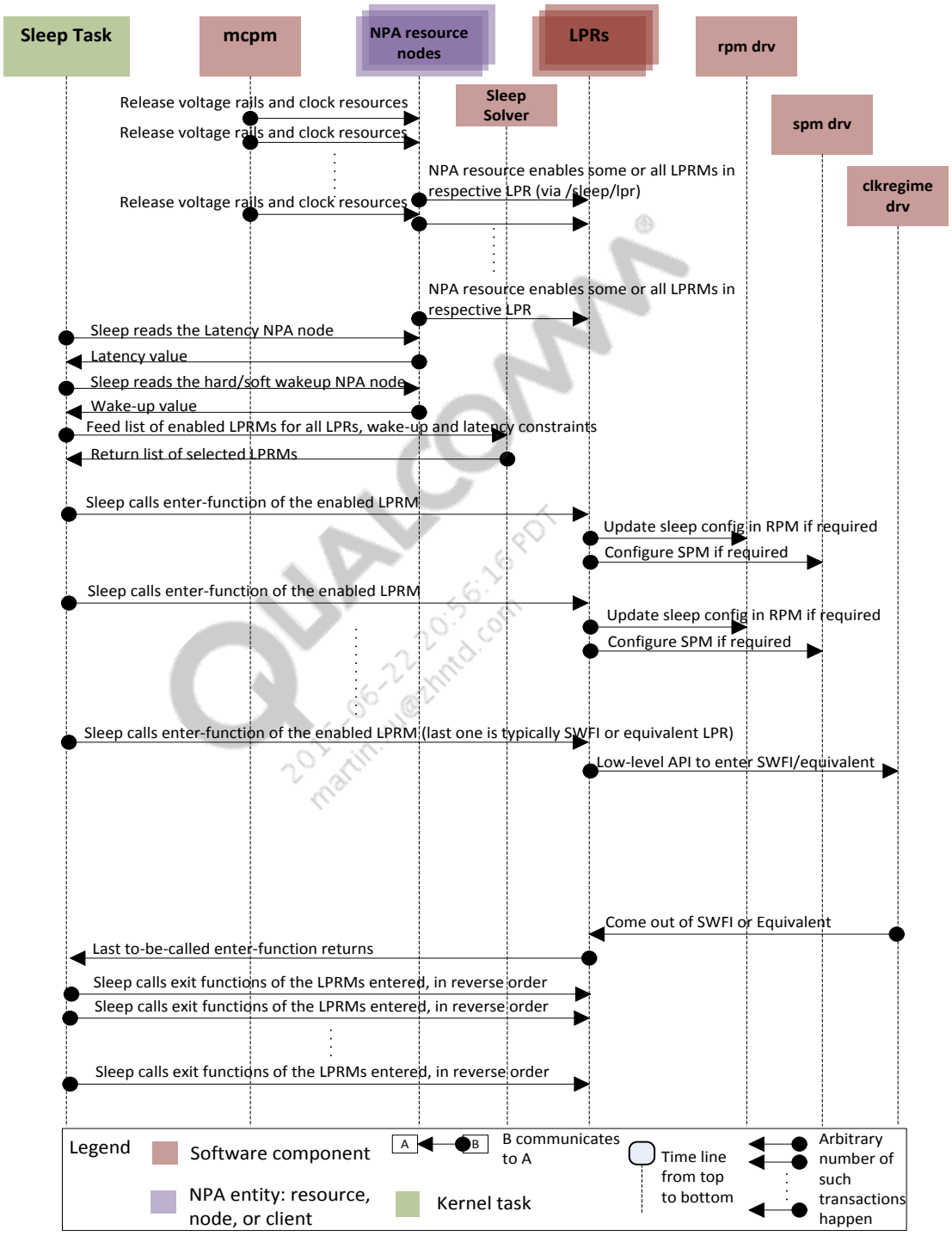


Figure 5-14 Hexagon modem software, second-level decomposition (sequence diagram) – Dynamic sleep

Table 5-5 is a list of LPRs/LPRMs currently implemented for the Hexagon modem processor. This is subject to change in the future.

Table 5-5 LPRs/LPRMs

LPR	NPA resource	LPRMs
cpu_vdd	/node/core/cpu/vdd	Power_collapse – Programs the SPM to perform power collapse of the modem processor after the Hexagon modem SWFI (or equivalent)
l2_cache		▪ ret – Programs the SPM to put the L2 cache in Retention mode ▪ no_ret – Programs the SPM to put the L2 cache in Nonretention mode
CXO	/node/xo/cxo	Shutdown – Registers an RPM request to update the modem sleep set with CXO voted off by the modem
vdd_dig	/node/rail/vdd_dig	Min – Registers an RPM request to update the modem sleep set with modem voting for entering VDD minimization (for VDD_CORE and VDD_MEM)
rpm	This helper LPR is not exposed to the rest of the software; it does not have a corresponding NPA node accessible by the rest of the software.	Sync – Sends the modem sleep set configuration to the RPM in a single transaction
npa_scheduler	Fork	Sends the modem next-awake-set configuration to the RPM

5.6.1.3 Static perspective

Sleep-related source files are located at modem_proc\core\power\sleep.

5.6.2 MCPM

The MCPM is a centralized controller for the modem core (different from the Hexagon processor) clock speeds, modem core voltage and power collapse, modem block configuration, and Hexagon processor clock speeds. Its decisions are closely paired with the mode of operation of the modem at any given time. Clients make requests to the MCPM (indicative of the change in modem operation), which computes a new operational state and maps it to a new modem clock and power configuration. Modem operation is therefore broken down into modes of operation, such as voice call, data call, standby, etc. For each mode, clock and power configuration is statically predetermined.

5.6.2.1 Hardware perspective

This subsection is not applicable to this version of the document.

5.6.2.2 Runtime perspective

Figure 5-15 shows the Hexagon software second-level decomposition for MCPM.

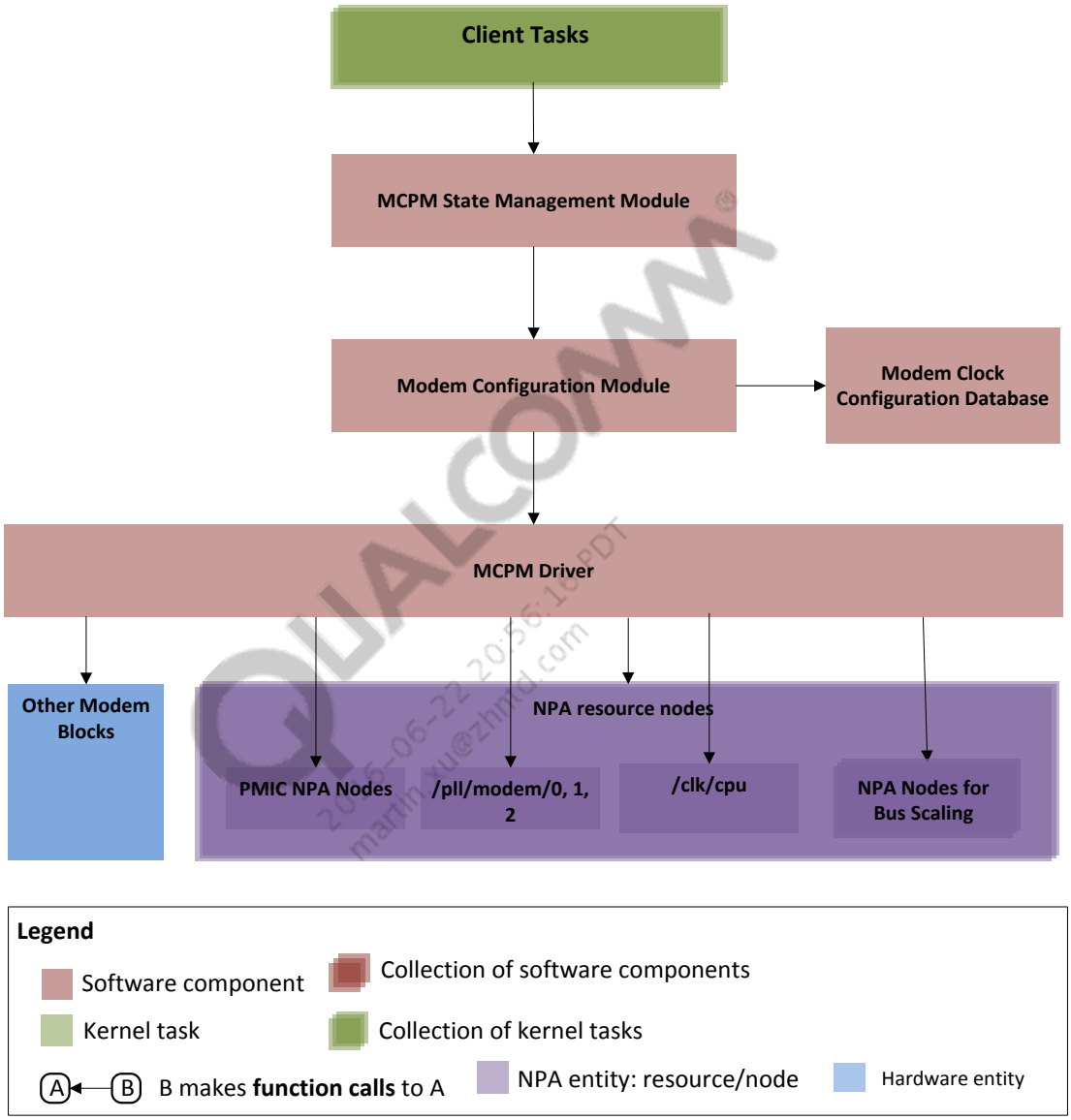


Figure 5-15 Hexagon modem software, second-level decomposition (runtime perspective) – MCPM

The following is a brief description of the components and connectors:

- Client tasks – These represent clients, such as the LTE software, GPS software, GERAN software, etc. Each client makes calls into the MCPM as and when they need to perform, e.g., LTE_WAKEUP_REQ, GPS_START_REQ, etc.
- MCPM state management module – Modem software operation is quantized into discrete states. On/Off states of the modem hardware components and clock speeds are statically determined for each state. The MCPM state management module internally maintains the current state to track the modem software operational mode. The MCPM state has information such as which technologies are active and what technology is in each operational mode, i.e., Idle, Voice, and Data. State changes are triggered with client calls, such as LTE_WAKEUP_REQ, etc.
- Modem configuration module – Based on the new mode, the difference from previous configurations is determined and these differences are applied in an appropriate order. Identifying configuration changes and determining the order of application of these changes are performed by this component. For example, the relative order of scaling voltages and changing frequencies would be different based on whether the frequency is scaled up or down. Consequently, ordering should be performed. This component then makes calls into the MCPM driver, which interfaces with low-level drivers to apply these changes.
- Modem clock configuration database – This is a database of all On/Off and clock configurations for each modem state. The configuration covers:
  - Hexagon modem processor speeds
  - PLL usage
  - Modem clock rates
  - Hardware block enable/disable
  - Modem voltage (VDD_MODEM)
- MCPM driver – The MCPM driver component programs the modem hardware registers for modem configuration and calls NPA nodes for voltage and processor/PLL speeds. It provides an API for the modem configuration module to perform:
  - Modem block enable/disable and clock control, i.e., GPS block, Rx frontend, etc.
  - Modem core power gating

  Some blocks are also capable of power collapse.

5.6.2.3 Static perspective

The MCPM implementation is shipped as a precompiled library.

5.7 Executing XO shutdown and VDD minimization

CXO is the source to the clocks supplied to different functional blocks on the MSM. XO shutdown disables the CXO clock supply from the PMIC to the MSM. It is a system-level resource controlled by the RPM. XO shutdown requires system-level resource awareness. Although it has better power-saving capability (typically corresponding to the lowest power state) than individual processor power collapse and GDFS, it has a longer transition period to go into/out of shutdown.

5.7.1 Runtime perspective

This subsection is not applicable to this version of the document.

5.7.2 Hardware perspective

In a broader sense, XO shutdown and VDD minimization occur as follows:

- All RPM masters (Kraits, modem, etc.) release clock requests/requirements that they may have issued to their local clock regime driver. This typically coincides with masters entering their lowest power mode (Power Collapse).
- During Power Collapse, they update the sleep set indicating they do not need a CXO resource (the clock regime driver does this when there are no outstanding clock requests to it) and their consent to minimize VDD_CORE and VDD_MEM voltage rails to retention levels.
- At some point in time, sleep is triggered on the RPM after collapse of the last RPM master and the RPM sets up the MPM for XO shutdown and VDD minimization.
- Sleep on the RPM issues an SWFI instruction and asserts an interrupt to the MPM.
- The MPM state machine is triggered and handshakes with the PMIC to turn off the CXO buffers and put VDD_CORE and VDD_MEM to minimization values.
- The MPM waits for a wake-up interrupt.

Hardware components involved in RPM interactions with a master have been discussed (see Figure 5-5). These are also the hardware components involved in the XO shutdown and VDD minimization procedures.

Figure 5-16 shows the XO shutdown and VDD minimization sequence with key events at the hardware level, using the Hexagon modem as the last master entering the sleep set.

The assumption is that the Hexagon modem is the last master to enter Power Collapse and other masters have already entered into their low power state (Power Collapse), along with appropriate votes for XO and VDDs.

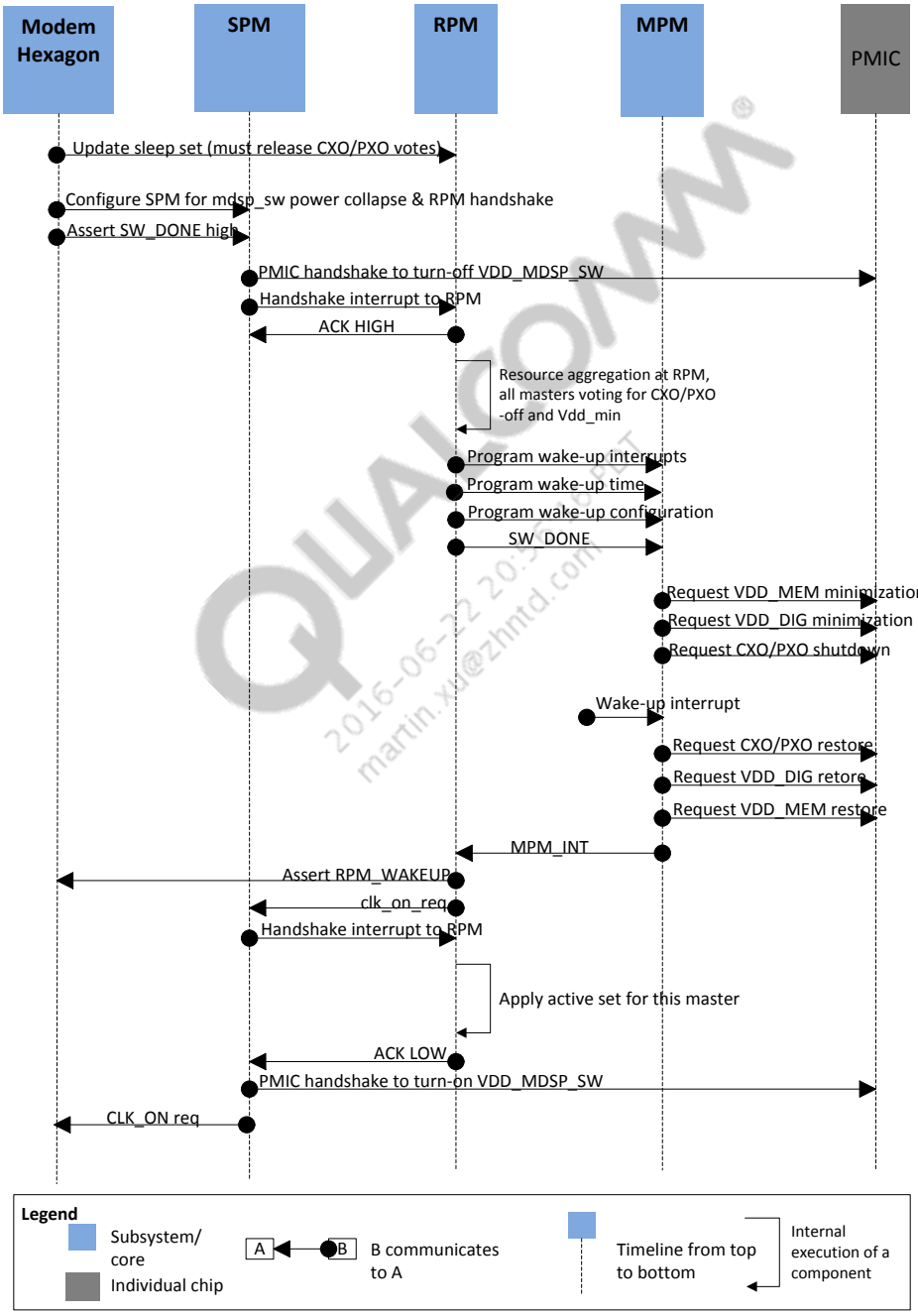


Figure 5-16 Hardware perspective, second-level decomposition (sequence diagram) – XO shutdown and VDD minimization

5.7.3 Runtime perspective

This subsection is not applicable to this version of the document.

5.7.4 Static perspective

This subsection is not applicable to this version of the do

A References

A.1 Related documents

Documents	
Qualcomm Technologies, Inc.	
MSM8274/MSM8274AB, MSM8674/MSM8674AB, MSM8974/MSM8974AB Device Specification	80-NA437-1
MSM8974 Linux Android Current Consumption Data	80-NA437-7
MSM8274/MSM8674/MSM8974 Chipset Design Guidelines – Introduction	80-NA437-5A
MSM8x74/MSM8x74AB Design Guidelines – Digital Baseband	80-NA437-5B
MSM8274/MSM8274AB, MSM8674/MSM8674AB, and MSM8974/MSM8974AB Baseband Reference Schematic	80-NA437-41
WTR1605(L) RF (CMCC) for MSM8974/MDM9x25(M) – ET Design Example – Preliminary Reference Schematic	80-NA437-46

A.2 Acronyms and terms

Acronym/term	Definition
MPM	MSM Power Manager
RFE	Radio Front End
EBI	External Bus Interface