Power Control System Architecture

Document number	DEN0050
Document quality	EAC
Document version	D
Document confidentiality	Non-confidential

Power Control System Architecture

Release information

Date	Version	Changes
2023/Feb/06	D	• Release 2.1 Non-Confidential
2017/Sep/15	C	• Release 2.0
2015/Dec/15	B	• Release 1.0
2015/Jul/15	A	• Beta

Arm Non-Confidential Document Licence ("Licence")

This Licence is a legal agreement between you and Arm Limited ("**Arm**") for the use of Arm's intellectual property (including, without limitation, any copyright) embodied in the document accompanying this Licence ("**Document**"). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this Licence. By using or copying the Document you indicate that you agree to be bound by the terms of this Licence.

"**Subsidiary**" means any company the majority of whose voting shares is now or hereafter owner or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries ("Licensee") is subject to the terms of this Licence between you and Arm.

Subject to the terms and conditions of this Licence, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide licence to:

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;

(ii) manufacture and have manufactured products which have been created under the licence granted in (i) above; and

(iii) sell, supply and distribute products which have been created under the licence granted in (i) above.

Licensee hereby agrees that the licences granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

THE DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENCE, TO THE FULLEST EXTENT PETMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENCE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE'S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENCE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This Licence shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this Licence then Arm may terminate this Licence immediately upon giving written notice to Licensee. Licensee may terminate this Licence at any time. Upon termination of this Licence by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this Licence, all terms shall survive except for the licence grants.

Any breach of this Licence by a Subsidiary shall entitle Arm to terminate this Licence as if you were the party in breach. Any termination of this Licence shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

This Licence may be translated into other languages for convenience, and Licensee agrees that if there is any conflict between the English version of this Licence and any translation, the terms of the English version of this Licence shall prevail.

LES-PRE-21585 version 4.0

Contents

Power Control System Architecture

Power Control System Architecture ii
Release information ii
Arm Non-Confidential Document Licence (“Licence”) iii

Preface

About this Specification ix
Intended Audience ix
Using this Specification x
Conventions xi
Typographical conventions xi
Numbers xi
Pseudocode descriptions xi
Assembler syntax descriptions xi
Additional reading xii
Feedback xiii
Feedback on this book xiii

Chapter 1 Background

Chapter 2 Introduction

2.1 Scope and Limitations 16

Chapter 3 Overview

3.1 Power Control Challenges 18
3.2 System Control Processor 19
3.2.1 Services 20
3.2.2 Trusted Operation 20
3.3 Power Management Software 21
3.3.1 Core Power Management 22
3.3.2 Device Power Management 23
3.3.3 System Control Processor Firmware 24
3.3.4 Power Management Software Stack Examples 25
3.4 Power Control Framework 27

Chapter 4 System Partitioning

4.1 Voltage Domains 29
4.1.1 System Logic 29
4.1.2 Always-On Logic 30
4.1.3 Processor Clusters 30
4.1.4 Graphics Processor 30
4.1.5 Other Functions 30
4.1.6 SoC Partitioning Examples 31
4.2 Power Domains 32
4.2.1 Power Modes 32
4.2.2 Power Domain Choices 33
4.2.3 System Logic 34
4.2.4 Always-On Domain 35
4.2.5 Processor Clusters 37
4.2.6 CoreSight Logic 39

4.2.7 Graphics Processor . . . 40
4.2.8 Display Processor . . . 42
4.2.9 Other Functions . . . 42
4.2.10 Power Domain Hierarchy Requirements . . . 43
4.2.11 SoC Partitioning Example . . . 46

Chapter 5 Power States
5.1 Power States and Power Modes . . . 48
5.1.1 Power States . . . 48
5.1.2 Power Modes . . . 48
5.1.3 Distinction of Power States from Power Modes . . . 48
5.2 Power State Hierarchy . . . 49
5.2.1 Core Power States . . . 50
5.2.2 Cluster Power States . . . 52
5.2.3 Device Power States . . . 54
5.2.4 SoC Power States . . . 55
5.3 Coordination by System Control Processor . . . 57
5.3.1 SoC Power States . . . 57
5.3.2 Cluster Power States . . . 57

Chapter 6 Power Control Framework
6.1 Power Control Framework Overview . . . 59
6.1.1 Power Control Framework Low-Power Interfaces . . . 59
6.1.2 Power Control Framework Infrastructure Components . . . 60
6.2 Low-Power Interfaces . . . 61
6.2.1 Q-Channel . . . 61
6.2.2 P-Channel . . . 61
6.2.3 AXI LPI . . . 61
6.3 Power Modes . . . 62
6.4 System Control Processor . . . 64
6.4.1 SCP Components . . . 65
6.5 Power Management Infrastructure Components . . . 69
6.5.1 Power Policy Unit . . . 69
6.5.2 Clock Controller . . . 73
6.5.3 Low Power Distributor . . . 74
6.5.4 Low Power Combiner . . . 74
6.5.5 P-Channel to Q-Channel Convertor . . . 74

Chapter 7 System Power Control Integration
7.1 Clock Control Integration . . . 76
7.1.1 Clock Gating Levels . . . 76
7.1.2 High-Level Clock Gating Methodology . . . 79
7.1.3 High-Level Clock Domain Selection . . . 80
7.1.4 Clock Gating Control Integration . . . 82
7.2 Power Control Integration . . . 89
7.2.1 Power Domain Wake Events . . . 89
7.2.2 Hardware Abstraction with Power Policy Units . . . 89
7.2.3 Distributed PPUs . . . 91
7.2.4 System of Systems . . . 92
7.2.5 Component Integration Layer . . . 93
7.2.6 Voltage and Power-Gated Domain Clock Gating . . . 94
7.2.7 Voltage and Power Domain Boundaries . . . 96
7.2.8 Access Control . . . 101
7.2.9 Isolation and Reset Control Considerations . . . 103
7.3 Reset Control Integration . . . 105

7.3.1 Reset Signals . . . 105
7.3.2 Reset Hierarchy . . . 107
7.3.3 Reset Management . . . 109

Chapter 8 Component Design Considerations
8.1 General Low-Power Interface Guidelines . . . 112
8.1.1 Low-Power Interface Implementation . . . 112
8.1.2 Output Management . . . 117
8.1.3 Interface Management . . . 118
8.2 Component High-Level Clock Gating . . . 126
8.2.1 Q-Channel Implementation . . . 126
8.2.2 Clock Availability during Clock Control Q-Channel Quiescence . . . 127
8.2.3 QACTIVE Behavior . . . 127
8.2.4 Q-Channel Handshake Behavior . . . 130
8.2.5 Implementation Example: AXI Responder Interface . . . 131
8.2.6 Unused Clock Control Q-Channels . . . 132
8.2.7 Clock Control Q-Channel Naming Guidelines . . . 133
8.3 Component Power Control . . . 134
8.3.1 Low-Power Interface Selection . . . 134
8.3.2 General Power Control Low-Power Interface Implementation . . . 136
8.3.3 Power State Availability . . . 137
8.3.4 Power Control Q-Channel Guidelines . . . 138
8.3.5 Power Control P-Channel Guidelines . . . 142
8.3.6 Power Control Low-Power Interface Naming Guidelines . . . 149

Part A Glossary

Preface

About this Specification

This specification describes an approach to the Power Control System Architecture of SoC based on Arm components. It defines version 2.1 of the *Power Control System Architecture* (PCSA).

This version contains significant updates due to the evolution of Arm components since Version 1.0. System integration and component design guideline content is also updated.

Intended Audience

There are two intended audiences for this specification:

- System-on-Chip (SoC) architects and designers designing power managed SoCs based on Arm components.
- Component designers incorporating Arm low-power interfaces for clock and power control with the aim of compatibility to the system integration principles outlined in this specification.

Using this Specification

This specification is organized into the following chapters:

- **Chapter 1 Background**

  Read this for an introduction to this specification.

- **Chapter 2 Introduction**

  Read this for an introduction to the *Power Control System Architecture* (PCSA).

- **Chapter 3 Overview**

  Read this for an overviews of concepts used in PCSA.

- **Chapter 4 System Partitioning**

  Read this a description of partitioning a SoC based on Arm components into voltage and power domains.

- **Chapter 5 Power States**

  Read this for a description of how power states and power modes are defined in this specification.

- **Chapter 6 Power Control Framework**

  Read this for a description of the *Power Control Framework* (PCF) concepts and components.

- **Chapter 7 System Power Control Integration**

  Read this for information on system clock, power, and reset control integration.

- **Chapter 8 Component Design Considerations**

  Read this for a description of design considerations for components implementing Arm low-power interfaces.

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

bold

Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example http://developer.arm.com

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in a monospace font. The pseudocode language is described in the Arm Architecture Reference Manual.

Assembler syntax descriptions

This book contains numerous syntax descriptions for assembler instructions and for components of assembler instructions. These are shown in a `monospace` font.

Additional reading

This section lists publications by Arm and by third parties.

See Arm Developer (http://developer.arm.com) for access to Arm documentation.

[1] *Arm® Server Base System Architecture*. (ARM DEN 0029 F) Arm Ltd.

[2] *ARM® Architecture Reference Manual, ARMv8 for ARMv8-A architecture profile*. (ARM DDI 0487 A.j) Arm Ltd.

[3] *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition*. (ARM DDI 0406 C) Arm Ltd.

[4] *AMBA® Low Power Interface Specification, Arm Q-Channel and P-Channel Interfaces*. (ARM IHI 0068 C) Arm Ltd.

[5] *AMBA® AXI™ and ACE™ Protocol Specification*. (ARM IHI 0022 E) Arm Ltd.

[6] *ARM® Power Policy Unit Architecture Specification*. (ARM DEN 0051 E) Arm Ltd.

[7] *Trusted Board Boot Requirements - CLIENT*. (ARM DEN 0006 D) Arm Ltd.

[8] *Arm® CoreLink PCK-600 Power Control Kit Technical Reference Manual*. (101150_0003_00_en) Arm Ltd.

[9] *ARM® Debug Interface Architecture Specification, ADIv6.0*. (ARM IHI 0074) Arm Ltd.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an e-mail to errata@arm.com. Give:

- The title (Power Control System Architecture).
- The number (DEN0050 D).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

Chapter 1
Background

Minimizing system power is a key requirement in a broad range of products from embedded microcontrollers, through mobile application processors, to servers. Reduction in system power has many benefits including reduced cost of ownership, increased battery life, and better management of thermal limits.

To be successful in these markets, SoC designers face increasingly complex power and thermal management challenges.

From a technical standpoint, challenges arise from both the coordination of many components and their successful integration. From a commercial perspective, these challenges must be solved within increasing time to market pressure.

While high value can be placed on system specific differentiation, the availability of standardized methods to facilitate integration and coordination of the power management for SoC components, especially those from third party suppliers, is not contrary to that goal.

A framework approach to power control, using standard interfaces and infrastructure, can provide a simplified, less error prone path to a more power efficient system with an improved time to market. This increases the time and resource available for differentiating activities.

Chapter 2
Introduction

This document describes an approach for the *Power Control System Architecture* (PCSA) of SoCs based on Arm components. It addresses challenges related to both coordination and integration of the system components.

The primary aim is to describe a standard framework for system power control integration that enables a simplified, faster time-to-market path to comprehensive power management.

The framework is required, because the provision of intrinsically power efficient components alone is insufficient. The components must participate in coordinated system level clock and power management, and the integration of these components must be achieved in a timely fashion.

Guidance is also given for component designers implementing Arm low-power interfaces and seeking compatibility with the system integration principles described in this specification.

2.1 Scope and Limitations

The descriptions in this specification are primarily at a logical implementation level. Physical implementation details are beyond the scope of the document. In many cases the selection of a described approach, or method, is independent of other choices and can be taken in isolation.

Arm component integration examples are intended to be broadly applicable to other similar Arm components. However, there can be specific considerations for each component.

NOTE: The information in this document is intended to supplement the documentation that is provided with Arm components. Exact integration requirements depend on the revisions of the components included in your system. You must follow the instructions included with all Arm components in your system, and not rely solely on the information provided in this document.

Chapter 3
Overview

This chapter gives an overview of concepts used in PCSA in the following sections:

- 3.1 *Power Control Challenges*
- 3.2 *System Control Processor*
- 3.3 *Power Management Software*
- 3.4 *Power Control Framework*

3.1 Power Control Challenges

Figure 3.1 shows a simplified SoC example. The example is illustrated in terms of high-level functions.

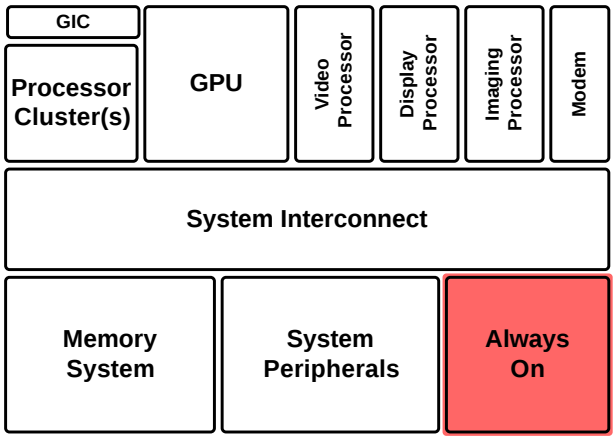


Figure 3.1: Example SoC system

The example is mobile centric but can be used to demonstrate power management challenges at a high-level in any SoC configuration.

In addition to the primary functions that the example shows, including processors, communications functionality and common system functions, there is an always-on area. This represents the power controller function that remains active in SoC sleep states.

There are coordination challenges related to this power control function, as the complexity of power and thermal management increases. These challenges arise because there are many elements to manage including clock and voltage supplies, power regions, sensor inputs, events and so on.

While it is possible to implement the power controller functions in purely fixed function hardware, there are significant disadvantages to that approach. Fixed function hardware support must be managed in a highly directed manner under the control of the OS power management software (OSPM). An implication is that application processor (AP) cores might be forced to remain active, or be woken, to perform low level functions when no primary function is required.

A fixed function solution also has limited flexibility in terms of both platform specific adaptation and the capability to address issues through workarounds.

An alternative approach is for the power controller function to be based around a processor, such as a microcontroller. This can provide a system intelligence capability that is flexible, extensible and able to act autonomously in addition to performing directed operations. In PCSA this capability is provided by the *System Control Processor*.

Another significant challenge is the integration of power management infrastructure across the SoC. This infrastructure is pervasive and requires ensuring that all components participate in clock and power domain management.

PCSA describes an approach to power control integration using standard infrastructure components, low-power interfaces and associated methods. This approach is referred to as the *Power Control Framework* and is described in Chapter 6 *Power Control Framework*.

3.2 System Control Processor

The SCP is a processor-based capability that provides a flexible and extensible platform for provision of power management functions and services.

In a mobile system, the processor of the SCP is anticipated to be a Cortex™-M microcontroller, but other Arm profile cores might be appropriate, depending on system requirements.

Figure 3.2 shows a concept level illustration of the SCP.

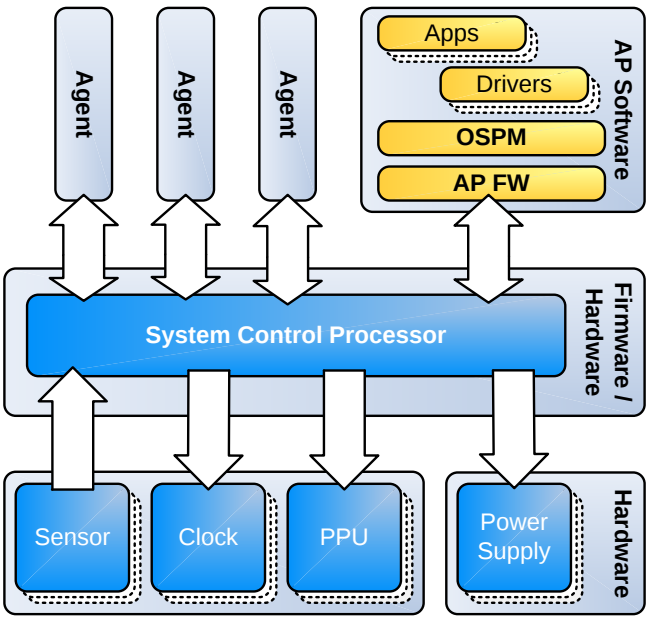


Figure 3.2: System Control Processor concept

In the upper part of Figure 3.2, the application processor (AP) software stack is shown as a requestor of SCP services. Other agents in the system might also have the capability to directly generate requests for resources that the SCP controls. Examples of such agents might be a modem subsystem in a mobile SoC or a management function in a server SoC. The SCP reconciles requests from all agents, managing the availability of shared resources and power-performance limits according to all constraints.

The central part of the figure reflects that the SCP is a processor-based system running dedicated firmware controlling a set of hardware resources. Although not shown in the figure, the SCP has a minimum set of resources, including local private memory, timers, interrupt control, and registers for system configuration, control and status.

The lower part of the figure shows a simplified set of SCP controlled hardware resources such as clock sources, power domain gating, voltage supplies, and sensors.

The capabilities of an SCP implementation are dependent on the ability to access and control a set of resources within the SoC in addition to a required base set of functions within the SCP. SCP hardware requirements are further detailed in 6.4 *System Control Processor*.

3.2.1 Services

The SCP provides the following primary services:

- **OSPM directed operation:** The SCP performs voltage supply changes, power control actions, and clock source management under the direction of the OSPM. These services might also be used by other requesting agents.
- **Response to system events:** The SCP responds to system events with appropriate power, clock, reset, and system control actions. These actions include:
  - **Timer events:** The SCP has local timer resources that can be used for the triggering of system wakes and any periodic actions such as monitoring.
  - **Wake events:** Responding to wake requests including GIC wake requests, caused by interrupts routed to powered-off cores, and system access requests from other agents.
  - **Debug access power control:** Responding to requests from the debug access port and related controls, including power management of the debug infrastructure.
  - **Watchdog events and system recovery actions:** On a local watchdog timeout, the SCP can execute a reset and re-initialization sequence.
- **System aware functions:** The SCP can act autonomously and can perform functions such as the following:
  - The SCP can reconcile requests from OSPM and other agents for shared resources. For example, it can control the path to main memory or entry to, and exit from, SoC sleep modes without requiring AP core activity.
  - The SCP can take responsibility for monitoring sensors and measurement functions. Monitoring tasks might include process and temperature sensor data harvesting and associated actions such as operating point optimization and alarm conditions.
  - The role of the SCP in operating point selection can extend to overriding the OSPM direction, as necessary, to ensure the electrical and thermal protection of the system.
- **System initialization:** The SCP takes responsibility for power-on reset system initialization tasks, from power-on sequencing of the primary system and AP core power domains through to AP boot.

The SCP abstracts both tasks and details away from the OSPM, enabling increased use of AP core low-power states and a simplified board support package (BSP) implementation. At the same time, platform specific differentiation, fixes, and improvements can be implemented at firmware level by the silicon provider or OEM.

In a complex SoC, that is a system of systems, there can be several SCPs with distributed responsibility. In such a system, it is anticipated that there is one lead SCP which can communicate with all others. The lead SCP takes responsibility for management of all globally shared resources as well as initialization and other common tasks while others have localized responsibilities.

3.2.2 Trusted Operation

While the SCP can have wide access to the system, its resources, including its memory and peripherals, are not accessible to the rest of the system. In conjunction with an appropriate boot process the SCP can be an inherently trusted entity.

3.3 Power Management Software

Figure 3.3 shows a simplified representation of the power management software stack. The figure illustrates the relationship between the OS power management frameworks, components with capabilities to directly request actions from SCP, and their relationship to the SCP firmware.

An important aspect is that all hardware power management actions are taken by the SCP on behalf of these requestors.

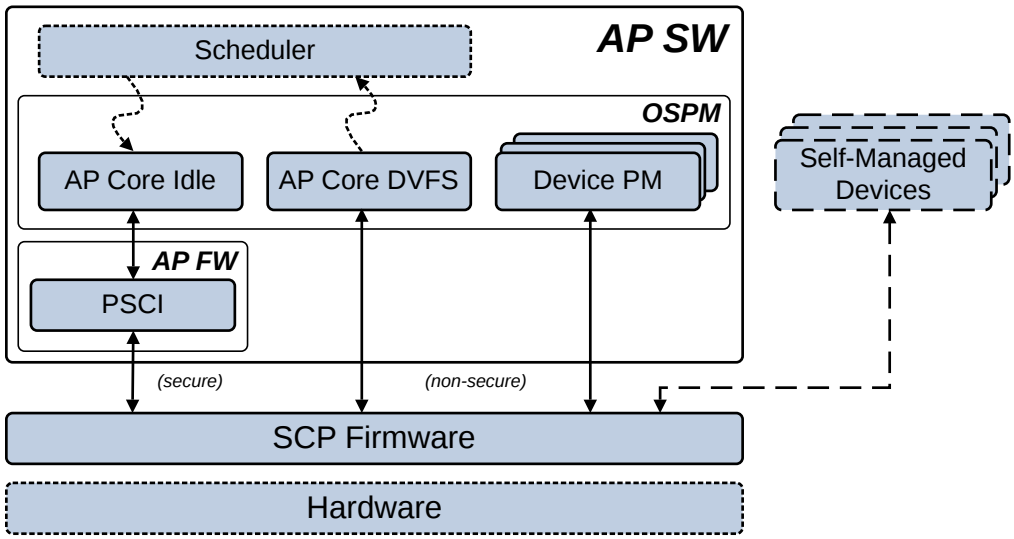


Figure 3.3: Simplified power management software stack

This simplified representation of OS power management (OSPM) can be divided into two parts:

- Core power management.
- Device power management.

3.3.1 Core Power Management

OSPM for AP cores can be broadly classified into idle management, and dynamic voltage and frequency scaling (DVFS) frameworks. As shown in Figure 3.3, these frameworks are associated with the scheduling in the OS. However, it should be noted that the association between the scheduler and the OSPM frameworks might only be a loose coupling.

Idle Management

The general principle for idle management is that when no threads are scheduled onto an AP core, the OSPM places that core into a clock gated, retention, or fully powered-off state. A powered-off core remains available to the OS for scheduling and can be woken by interrupts.

An alternative technique, commonly known as *hot-plug*, might also be implemented. In this case, AP cores are removed from the pool available to the OS for scheduling. With this technique, cores are powered-off and all interrupts and software threads are migrated to other cores. This technique can be used either in proportion to demand, or in cases where compute capacity must be limited due to power or thermal constraints.

A challenge for idle power management is that various operating systems, from various vendors, can be simultaneously executing in an Arm system. It is then necessary to have a method of collaboratively performing power control. For example, if the operating system that is managing power, running at one level of privilege, wants to enter a state that powers-on or off a core, then operating systems at other levels of privilege need to react to this request. Equally, if a core is woken from a power state by a wake-up event, it might be necessary for operating systems running at different levels of privilege to perform actions, such as restoring state. The *Power State Coordination Interface* (PSCI) specification provides an interface for this purpose.

The principle illustrated in Figure 3.3 is that the outcome of the arbitration in PSCI leads to a power state change request to an SCP software interface. The SCP firmware acts on that message and manages all hardware level details.

NOTE: Idle power states are generally selected by the OSPM. However, the AP firmware can modify this selection. For simplicity, this document only refers to the OSPM when describing idle power state decisions made in the entire AP software stack.

DVFS Management

DVFS provides a mechanism for managing the power-performance envelope. From an energy-performance perspective a wide range of OSPM policies are used to determine the required operating level. The objective is to meet performance requirements when demanded, but otherwise minimize energy consumption. Thermal management frameworks can impact the requested operating level by limiting the maximum performance allowed.

Figure 3.3 shows that DVFS requests, resulting from the interaction between the OSPM frameworks, are sent to the SCP through a software interface. The SCP then manages the detail of the hardware actions, to change voltage and frequency.

The SCP might also take a larger role in power and thermal optimization policy. This enables the path where OSPM only provides high-level performance requests and the SCP evaluates all available techniques and considerations to select an optimized performance point within the electrical and thermal constraints of the system.

3.3.2 Device Power Management

The power management of devices encompasses both device specific aspects, in drivers, and higher-level frameworks.

At a high-level, devices can be said to have two types of behavior and capability:

- **OS managed**: In this case, a device is entirely dependent on the OSPM, and any driver, to ensure the provision of system resources for its operation.
- **Self-managed:** Some devices will act as agents that make direct requests to the SCP for a subset of their power management functions. In some cases, this can be all power management functions, and such devices are described as fully self-managed. Partially self-managed devices directly request SCP for some functions but rely on OSPM for the remainder.

Most devices in the system are typically OS managed. The degree of power management required varies depending on the device.

Many basic peripherals are typically in parts of the system that are powered-on whenever the system is running. Some of these peripherals might also be clocked by default and therefore require no explicit power or clock management.

Other peripherals will require clocks to be enabled, and more complex devices can also require specific power domains to be powered-on. When a power management action is needed, the driver software typically expresses this dependency through an abstraction to the OSPM. The OSPM then requests the SCP to perform any actions to satisfy these dependencies using the SCP software interface.

As shown in Figure 3.3 self-managed devices, depending on capability, might have an independent software interface path from their own software stacks to directly request SCP actions.

3.3.3 System Control Processor Firmware

The SCP firmware is implementation specific. While an overview of anticipated capabilities is given in 3.2.1 *Services*, it is useful to define a set of minimum expected firmware services.

The definitions provided here are illustrative and are not exhaustive or limitative.

At a high-level, the SCP firmware services can be divided into two categories:

- **OSPM directed:** Services provided by a contract formed with the OSPM by an SCP software interface. These services might also be used by other requesting agents.
- **System services:** Services provided by the SCP without OSPM direction.

The SCP software interface is expected to provide, at minimum, commands to support the following:

- **Power states:** Commands to request setting of core, cluster, device and SoC power states.
- **DVFS:** Commands to request changing the operating point of a DVFS capable domain to a desired performance point.
- **Voltage supply:** Commands to request changing the level of platform power supplies outside of DVFS domains.
- **Clock supply:** Commands to request the enabling and source frequency of platform clocks outside of DVFS domains.
- **Timer:** Commands to request setting and cancelling of always-on wake-up timer events for a specific AP core. These commands are only required in a platform without AP core visible always-on timers.
- **Boot:** Commands for boot time initialization. The requirement for these commands is dependent on the boot process used.

Basic extensions to this minimal set of commands might include support for sensor management and queries for capabilities and resource states. The *ARM® System Control and Management Interface* (SCMI) is a software interface specification that provides an interface for this purpose for use by OSPM and other agents in the SoC.

A minimum set of system services that the SCP is expected to provide is as follows:

- **System initialization:** The SCP firmware must support system power-on reset initialization tasks. These include the power-on sequencing of primary system and AP core power domains through to AP core boot.
- **Events:** The SCP firmware must respond to system events. Basic system events include Generic Interrupt Controller (GIC) wake requests, local timer events, implementation defined always-on domain wake requests, debug power requests and watchdog expiration.
- **Consistency:** The SCP must ensure that there are no races between requests, for example, ensuring power domain dependencies are maintained despite the order and timing of the arrival of wake requests.

Basic extensions to this minimum set of services might include management of system time through SoC sleep states, autonomous sensor monitoring functions and further support for ensuring the consistency of the system and requested states.

3.3.4 Power Management Software Stack Examples

Mobile Systems

Figure 3.4 shows an example power management stack as might be implemented in a Linux based mobile device.

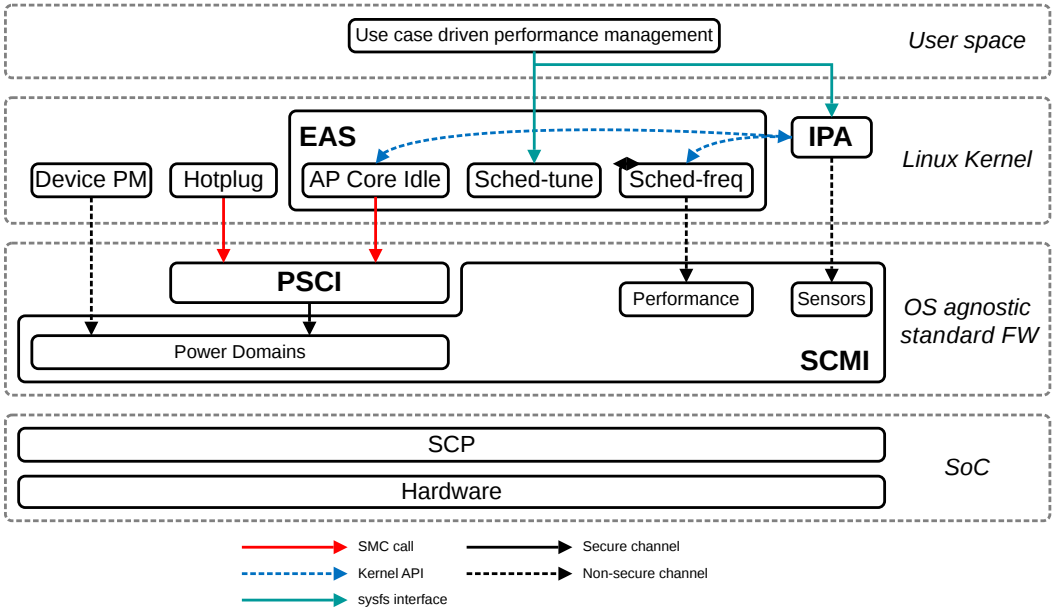


Figure 3.4: Linux mobile: example power management software stack

In the Linux kernel, *Energy Aware Scheduling* (EAS) provides the core scheduling with tight links to core idle and integrated frequency control. EAS is also linked to a thermal management solution using *Intelligent Power Allocation* (IPA). Finally, both EAS and IPA have linkages to user space performance management interfaces.

An OS agnostic firmware layer includes an implementation of PSCI as outlined in *Idle Management*. SCMI provides an interface for communication with the SCP firmware. It supports protocols for power and performance control in addition to sensors, such as those for temperature measurements, used by IPA.

Infrastructure Systems

Figure 3.5 shows an example power management stack as might be implemented in an infrastructure system, such as a server.

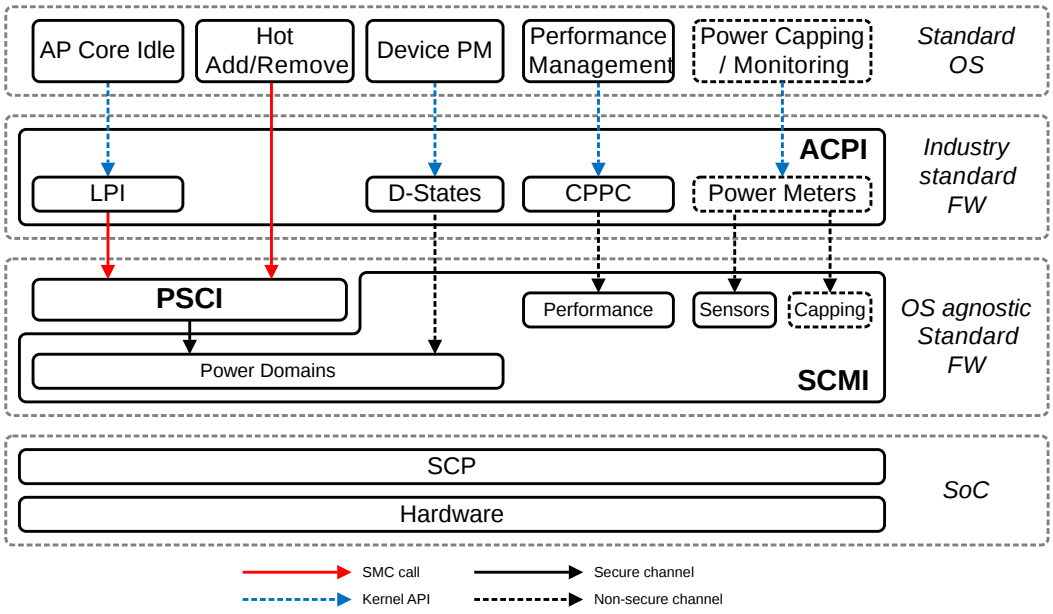


Figure 3.5: Infrastructure system: example power management software stack

In infrastructure systems, particularly servers, a desirable property of the system is for the OS to be independent of platform specific details. This enables the OS to be updated, and even changed, without modifying the platform firmware. Similarly, this enables a new hardware platform to run an unmodified OS. The *Advanced Configuration and Power Interface* (ACPI) is a standard interface, implemented in firmware, which enables such an abstraction.

ACPI provides a set of power management services, amongst others, to a compliant OSPM implementation. In Figure 3.5 the following services are shown:

- **Low-Power Idle (LPI):** LPI, introduced into ACPI 6.0, provides a method to define the local power states for each node in a hierarchical processor topology. This affords flexibility, not offered by legacy C-states, suited to the diversity of Arm system implementations.
- **Device states:** ACPI supports a standardized device power state management abstraction.
- **Collaborative Processor Performance Control (CPPC):** CPPC provides an interface for OSPM to express a performance requirement, on an abstract scale, while the platform firmware makes a final decision on the selected frequency and voltage based on all constraints. Means are provided for OSPM to determine delivered performance.
- **Power meters:** The ACPI power meters provide means for OSPM power monitoring and for the setting of power capping limits. While, typically, these caps are determined by a management function this capability can be used in some large-scale systems, for example high-performance computing.

An OS agnostic firmware layer includes an implementation of PSCI as outlined in *Idle Management*. An implementation of SCMI spans this layer and the SCP firmware and supports protocols for power, performance control, and power monitoring using the sensor protocol.

3.4 Power Control Framework

A primary aim of PCSA is to describe a standard approach for system power control integration of SoCs. A key component of this approach is the power control framework. Figure 3.6 shows a high-level illustration of power control framework concepts.

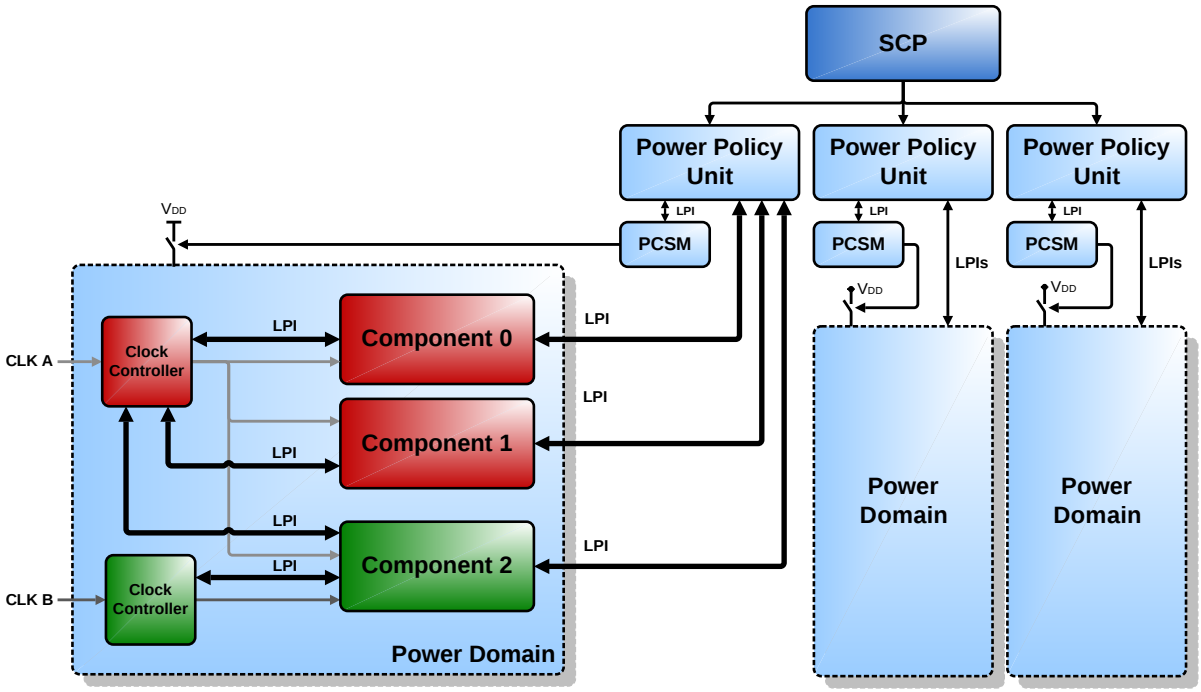


Figure 3.6: Power control framework concept

The power control framework is a collection of standard infrastructure components, interfaces, and associated methods that can be used to build the infrastructure necessary for power management of a SoC.

Standard infrastructure components include power, clock, and interfacing components.

Local interfacing between the infrastructure components and functional components use Arm Q-Channel and P-Channel low-power interfaces (LPI). Components without support for Arm LPI are managed using an integration layer adaptation approach.

For power domain control, a *power policy unit* (PPU) is defined. The PPU is fixed function hardware supporting a set of power policies programmed by the SCP through a software interface. The PPU interfaces with power domain components, using LPI as needed, to ensure safe power mode transitions.

Each PPU is paired with a power control state machine (PCSM) that abstracts the PPUs power policies to implementation technology specific power management, such as power switch and retention controls. This allows the PPU to be re-used across multiple SoC implementations and technologies without modification.

The clock controller is targeted at components supporting high-level clock gating. This approach enables the clock to be gated high in the clock tree when components are idle. High-level clock gating is supported by most Arm components.

Other infrastructure components are also available that facilitate the control and combination of LPI from controllers to devices.

For a detailed description of the power control framework, see Chapter 6 *Power Control Framework*.

Chapter 4
System Partitioning

This chapter describes partitioning of a SoC based on Arm components into voltage and power domains.

The choices described are not exhaustive and represent a subset of the possibilities. The intent is to describe the significant factors and considerations for partitioning of a SoC based on Arm components into these domains as well as critical relationships that must be maintained.

This chapter is divided into the following sections:

- 4.1 *Voltage Domains*
- 4.2 *Power Domains*

NOTE: In many SoC operation scenarios, dynamic power consumption is dominant and clocking strategy is therefore critical. This topic is addressed in 7.1 *Clock Control Integration*, from a high-level clock gating and implementation perspective. From a clock domain partitioning perspective, these considerations are implementation specific.

4.1 Voltage Domains

A voltage domain is defined here as a collection of design elements supplied by a single voltage source. The voltage supply to the domain might be scaled or removed for power or performance reasons.

In practice, a SoC can have many voltage supplies across logic domains, I/O and analog functions. In this document, the scope is limited to discussing the primary supplies of logic domains. Where secondary supply considerations exist for a logic domain, it is considered that these are physical implementation-specific details beyond the scope of this document.

While a single logic voltage supply could be used for all the SoC, this is now rarely the case except in low complexity solutions.

A primary motivation for additional voltage domains is to support DVFS for functional areas of the SoC. DVFS is a fundamental technique for both energy and performance optimization. While initially used for AP cores, it is increasingly being applied to other components of the SoC.

A second motivation can be to enable external supply switch-off, or reduction to non-functional state retention levels, to some logic areas while maintaining an operational level supply to others. This approach can be used both as complementary to and as a substitute for on-chip power gating.

From a cost perspective, the addition of voltage supplies can be significant since additional voltage regulators are required, and extra effort and complexity are required in the SoC physical implementation. A consequence of these factors is that the function size, or area, required to justify a voltage domain is significant. Therefore, the value of the addition of each voltage domain must be carefully assessed against the performance and power requirements for the design.

The following sub-sections outline options for the primary voltage domains of a system.

4.1.1 System Logic

A SoC will have some shared system logic functions typically composed of interconnect, memory system, peripherals, and other shared infrastructure.

It is convenient to consider the voltage supply for these functions as the default supply for the SoC. The exact functions contained within this voltage domain depend on the choices taken to support additional voltage domains for each function. This supply is referred to here as V_{SYS}.

SoC system logic DVFS is possible, but has challenges that must be addressed:

- Peripheral functions, such as timers and external interfaces, often have fixed frequency requirements which cannot be scaled. This can be resolved by an implementation specific combination of timing constraints, to ensure these functions can operate at the required frequency across all operating points, and resource activity limitations to voltage scaling.
- Memory system scaling presents challenges from the perspectives of DDR PHY and memory timing settings. Solutions to these problems are beyond the scope of this document.

There can also be a voltage domain partitioning of the system logic itself. An example of this could be separating the memory system from the other system logic to enable scaling of both domains independently.

4.1.2 Always-On Logic

Always-on logic is required so the SoC can be woken from sleep states.

The supply to the always-on logic of the SoC is typically the main system logic supply (V_{SYS}), described in 4.1.1 *System Logic*.

However, a second common strategy is the provision of a dedicated supply for this logic. This is one case where the size of a voltage domain might be small.

Power domain strategies associated with this choice are described in 4.2.4 *Always-On Domain*.

4.1.3 Processor Clusters

Cortex-A profile processor clusters in most markets, and invariably in mobile SoCs, will have dedicated voltage domains to enable DVFS.

In an Arm DynamIQ™ based big.LITTLE system a single cluster supports both core types. The cluster in these systems is recommended to use a voltage supply independent of the big and LITTLE core supplies. This might be the main logic supply (V_{SYS}) or as part of a scaled memory system voltage supply.

In applications where DVFS is not required, or the cost is too high, then the processor cluster is in the V_{SYS} domain.

In some applications, such as modems, Cortex-R profile processor clusters might also be given dedicated voltage domains.

4.1.4 Graphics Processor

Graphics processing performance in mobile applications has grown significantly and is anticipated to continue. GPU workloads represent throughput processing, with very high inherent parallelism, and are well suited to using DVFS to adapt the performance and energy profile of a given hardware configuration to a frame level deadline.

These properties also enable adaptation to different requirements. Cost-centric designs can implement fewer cores at higher frequency and voltage, while energy-performance-centric designs can implement more cores at lower frequency and voltage.

Therefore, a dedicated voltage domain to enable GPU DVFS is often implemented to enable these benefits.

In applications where DVFS is not required, or the cost is considered to outweigh the benefit, then the GPU cluster is in the V_{SYS} domain.

4.1.5 Other Functions

Further voltage domain partitions are less common as the cost to benefit ratio and implementation feasibility degrades.

One example might be an integrated modem, which is effectively a system within a system, as this is a function of significant size. In this case, motivation can arise from the potential for scaling, according to mode or required performance, and for independent powering when other functions have their voltage supply externally shut-off.

A second set of possibilities, for DVFS scaling reasons, could arise from other media processing functions, such as video and display subsystems, or a domain dedicated to a large accelerator such as for imaging.

As in the previous cases, all functions that do not have dedicated domains are in the V_{SYS} domain.

4.1.6 SoC Partitioning Examples

Figure 4.1 adapts the components of Figure 3.1 to provide a simplified voltage domain partitioning example of a mobile system.

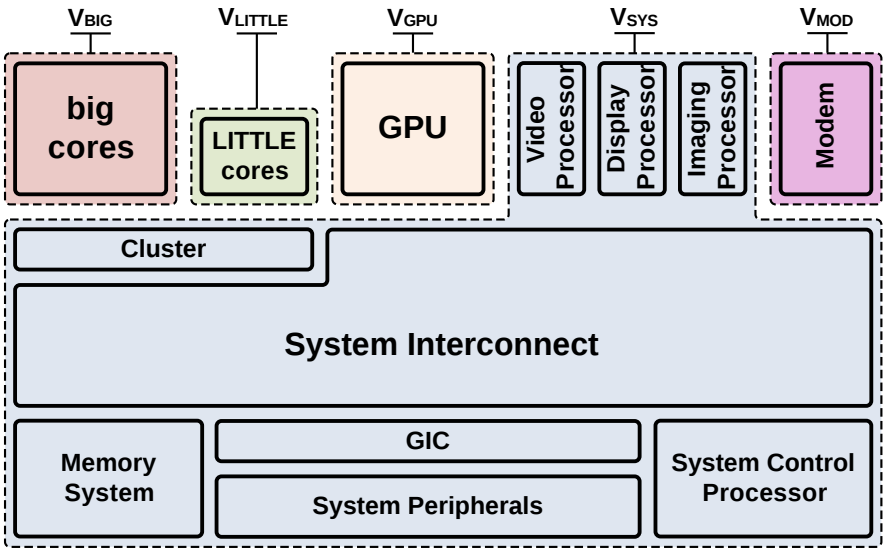


Figure 4.1: Voltage Domains: Mobile SoC example

This is a big.LITTLE system with voltage domains for independent DVFS of each of the big cores, the LITTLE cores and the GPU. The big.LITTLE implementation uses Arm DynamIQ technology and the cluster uses the system logic supply. An integrated modem also has an independent voltage domain.

A lower cost implementation might have a single voltage domain for the cores to support DVFS as a minimum requirement, it might also be limited to a single core type. Additionally, a lower cost implementation might exclude the modem and its related voltage domain.

4.2 Power Domains

A power domain is defined here as a collection of design elements, within a voltage domain, that share common power control. A voltage domain can be partitioned into one or more power domains.

Specifically, a power-gated domain is a power domain whose power can be removed by on-chip power switches. A voltage domain can then be partitioned into one or more power-gated domains as well as an always-on (non-gated) power domain.

The motivation for partitioning a voltage domain into several power domains is to facilitate techniques for static leakage power mitigation. These include modes where power is removed, with loss of context, and low leakage retention modes which keep some or all context.

The following subsections describe:

- The power modes that can be supported by a power domain.
- The choice of which power domains to implement in an Arm based SoC.
- Power domain partitioning requirements that must be respected. These include availability and power ordering relationships between vital components in the system.

4.2.1 Power Modes

This section defines the modes that a power domain can have at a hardware level. The mapping of power domain modes to a software power state view is described in Chapter 5 *Power States*.

ON

A power domain always supports an ON mode. This is the normal functional mode for the logic in the power domain. If the power domain supports DVFS, this is applied while the power domain is in this mode. This mode will normally also use clock gating and other techniques to reduce dynamic power consumption.

OFF

A power domain typically supports an OFF mode through the inclusion of on-chip power switches. The power switches are used to remove the power supply when its function is not required.

OFF power modes are destructive of context. To use this mode, mechanisms must be provided to manage any necessary context before entering OFF mode and at, or prior to, the resumption of execution after returning to ON mode. These context management mechanisms might be implemented in software or hardware.

This means there can be a significant time and energy cost to using OFF mode. However, there are many components that have negligible restore or reconfiguration requirements at power-on. Such components can then be powered-off and on without this overhead.

Retention (RET)

Retention modes offer leakage power reduction without loss of state.

There are several possible RET mode implementations and use models. RET modes can be categorized by the following criteria:

- **Entry to and exit from the mode**: This can be either hardware autonomous or software visible.
- **Physical implementation**: Full or partial state retention within the mode.

For a hardware autonomous RET mode, that is transparent to software, all state that is required to resume operations on exit from the mode must be retained by the hardware. Other state can be discarded and so, while from a functional perspective the mode provides apparent full retention, there can be partial state retention from an implementation perspective.

For a software visible RET mode an explicit software decision is taken to choose the mode, and this means that some context management might be handled by software at entry to and exit from the mode. An example of this is when only specific RAM content will be retained while all other hardware state is discarded. From an implementation perspective, these modes might require only partial retention capability.

A RET mode can be implemented using flip-flops that have a low leakage retention capability, using a secondary un-switched power supply. The primary supply for the logic cells is power-gated in the mode using on-chip switches. To gain the most power saving RAM cells that support leakage saving retention modes are also required. In the absence of this capability, RAM cells must be maintained at the operational level while a component is in RET mode.

NOTE: As previously outlined, manipulation of the external supply level can also achieve similar results for an entire voltage domain. This approach typically suits only high-level control, with increased latency, and requires the entire voltage domain to share a common retention strategy.

Since some, or even all, state is retained the time and energy cost of entering and exiting a RET mode is lower than OFF mode. However, the power consumed in this mode is higher than in OFF mode and therefore there is a trade-off in terms of opportunity to enter the mode and residency time within the mode.

Power Mode Transitions

When switching a power domain between power modes it is essential that the logic in the domain is in a safe state. Components provide various mechanisms to achieve this and these must be integrated into the power control logic.

This document describes infrastructure components for this purpose in Chapter 6 *Power Control Framework* and integration of these mechanisms in Chapter 7 *System Power Control Integration*.

4.2.2 Power Domain Choices

The fundamental aim of the power domain strategy is to minimize the powered-on area in each scenario.

From a practical perspective, implementation and control overhead constraints mean that a power-gated domain is typically large with the content representative of significant functions.

When choosing power domains, it is important to consider end use cases to ensure the low-power modes of the chosen domains can be used effectively. Although there can be many detailed use cases, they can normally be reduced to broader classes of usage. From analysis, the opportunity to power-off functions that are not required for significant periods of time can be determined.

Also, the power modes supported by a power domain need to be mapped to one of an explicit set of software power states, or an autonomous mechanism supported by a component. Where the power state control is required to be managed by software the viability of using an available framework or device driver should also be assessed.

4.2.3 System Logic

Many of the common system logic functions such as interconnect, memory system, peripherals, and other shared infrastructure will be required during all activity outside of SoC sleep states. A typical SoC gathers these functions into a single power-gated domain.

In a large scale SoC this can represent a significant area. The SoC architecture could be organized to partition this area into further power domains. However, challenges arise from the pervasive nature of common paths through interconnects and because many peripheral functions are too small for power domain implementation. Moreover, attempts to map peripheral functions into larger groups aligned to broad usage classes might prove intractable.

However, where interconnect and peripheral functionality is specific to function blocks that have dedicated power domains this should be integrated into those power domains instead of the system domain.

4.2.4 Always-On Domain

The always-on power domain scope depends on the system specific sensitivity to leakage in SoC sleep states. In cases of high sensitivity, the always-on domain might be limited to minimal wake logic. The wider scope of the SCP core subsystem can be placed into a power-gated domain that is powered-off in the deepest SoC sleep states.

NOTE: When considering the sensitivity to always-on logic area the standby current consumed by power switches, I/O ring circuits, and analog functions that remain powered is recommended to be part of that evaluation.

The minimal wake logic scope includes:

- System wake-up timer and system counter.
- Debug Access Port (DAP) logic.
- Any required external wake event detection.
- Any required detection for wake events from independent on-chip subsystems.
- Power-on reset and initialization logic. If the SCP is in a separate power-gated domain, then the always-on domain must include power-on control for the SCP domain which is responsive to all the above conditions.

Where sensitivity to always-on area and power is lower, the always-on domain can contain the entire SCP core subsystem. As a minimum, the SCP core subsystem must support initialization of the system through to AP boot, as well as transitions to and from SoC sleep states.

Always-on power domain strategies are then influenced by:

- Minimal wake logic requirement.
- System logic voltage and power domain strategy.

A corresponding set of options is illustrated in Figure 4.2:

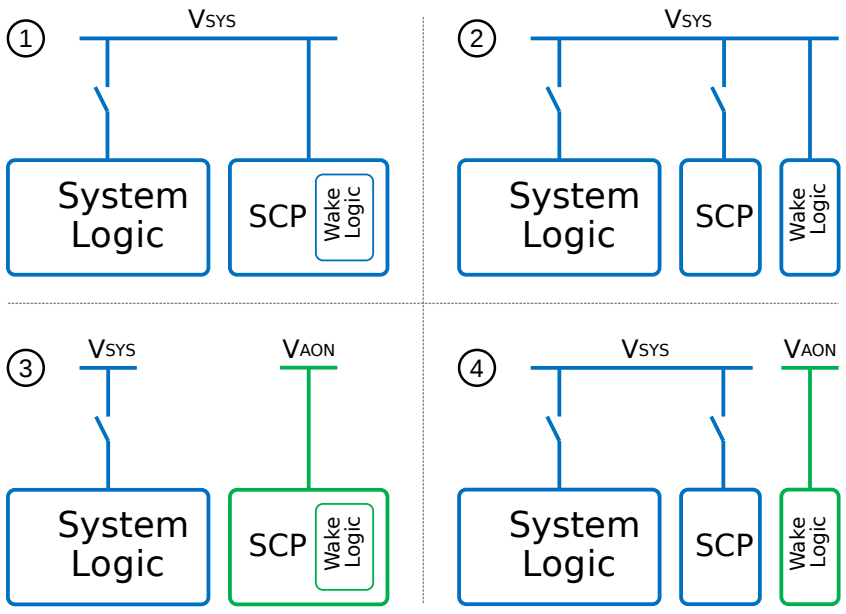


Figure 4.2: Always-on logic power domain strategies

Taking each of these options in turn:

1. In this case, the system logic power domain has on-chip power switches supplied from V_{SYS}. The sensitivity to leakage of the always-on area is not enough to necessitate a dedicated wake logic power domain and the SCP power domain is an un-gated power domain supplied from V_{SYS}.

   If there are no conflicting constraints, this arrangement is recommended due to its simplicity and lack of dependence on additional supplies at platform level.

2. The sensitivity to leakage of the always-on area in this case is enough to necessitate a dedicated wake logic power domain and this is arranged as an un-gated power domain supplied from V_{SYS}. The system logic and the SCP power domains have on-chip power switches supplied from V_{SYS}. These two power-gated domains might be merged into one if this meets the system requirements.

3. The entire SCP functionality is an un-gated power domain supplied from a separate always-on supply V_{AON}. This arrangement can allow V_{SYS} to be removed externally with the possibility that the system logic domain on-chip power switches are obviated.

4. This option is, in practice, strategy 3 in combination with the power domain options of strategy 2.

4.2.5 Processor Clusters

Cortex-A and Cortex-R profile processor clusters typically support a wide range of power domains and power modes within those domains. Most multiprocessor products support both per-core and cluster logic power domains. Some multiprocessor products also support a cluster level debug power domain.

Table 4.1 shows an example using Cortex-A profile products:

Table 4.1: Cortex-A profile power domain and modes example

Power Domain	Supported		Power Modes
	Cortex-A75	Cortex-A55	
Each core	Yes	Yes	ON, RET[a], OFF
Advanced SIMD/FP pipeline in each core	No	Yes[b]	ON, RET[a], OFF
Cluster LLC RAMs[d]	Yes	Yes	ON, RET[c], OFF
Cluster logic	Yes	Yes	ON, OFF
Debug domain	No	No	N/A

[a] Retention mode autonomously managed transparently to software
[b] On / Off modes follow core domain. Retention mode autonomously managed.
[c] Autonomous retention transparent to software at run time and explicit directed retention at power-off.
[d] Overall On / Off modes follow cluster logic domain. Additionally, Partial L3 cache power-off is supported when cluster logic is On.

Core Power Domains

The requirement for a processor core to be available to the system is highly dynamic even within a use case.

OSPM frameworks have a correspondent idle management capability which can select between the software visible power states. Policy frameworks also exist for removing a core from the pool available to the OS and this can be used, for example, in cases where compute capacity must be limited due to power or thermal constraints.

Additionally, where supported by the processor, autonomous software-transparent retention modes can be implemented which reduce the leakage opportunistically during periods where a core is halted.

In both energy and power constrained use cases, particularly at high temperatures, core leakage power can be significant and so the capability to manage this leakage is highly desirable.

Arm strongly recommends the implementation of per-core power domains in all applications that are sensitive to leakage power.

Cluster Domain and Shared Cache RAMs

Multiprocessor products also typically support a top-level cluster logic power domain which is controlled together with the shared cache RAM for on and off mode transitions. In addition:

- When the cluster logic is on, and remains operational, partial cache RAM power-off and autonomous RAM retention modes might be supported.
- When the cluster logic is off, shared cache and SCU RAM retention modes might be supported.

Since the cluster availability requirement is also dynamic, it is strongly recommended to support power gating of this area in all applications that are sensitive to leakage power.

The cluster domain would incorporate any integration logic, such as the appropriate portion of any domain bridges, for functional and CoreSight™ use, and CoreSight infrastructure dedicated to the cluster.

Debug Domain

If a debug power domain is supported by the cluster, this contains the logic required to satisfy the architectural requirements for debug through core power off.

If this is implemented as an independent power-gated domain, then only this domain needs be powered during a debug connection when no cores are powered.

Since this domain is small, it is commonly merged into the cluster logic power domain.

In the case where the debug through core power off logic is part of the cluster power domain the cluster must be powered during a debug connection.

In DynamIQ-based cores this debug logic is placed in a separate module called the DebugBlock that can be placed within or in a separate power domain to the cluster.

Processor Cluster Partitioning Examples

Figure 4.3 shows an example logical view for a MP4 processor cluster using a single voltage domain for all cores and the cluster. Per-core, cluster and cluster level shared cache RAM power gating are supported.

The cluster and shared cache RAM power-gated domains share common control for on and off mode transitions. The cache RAM can have additional control for retention support but must always be available for access when the cluster is powered-on. Per-core power-gated domains are also implemented with the option of support for dynamic retention.

The structure of the figure reflects the hierarchical power-on ordering requirements for the processor subsystem from the voltage supply (V_{AP}), available first, then power domain dependencies flowing downwards. Power-off ordering is the reverse sequence.

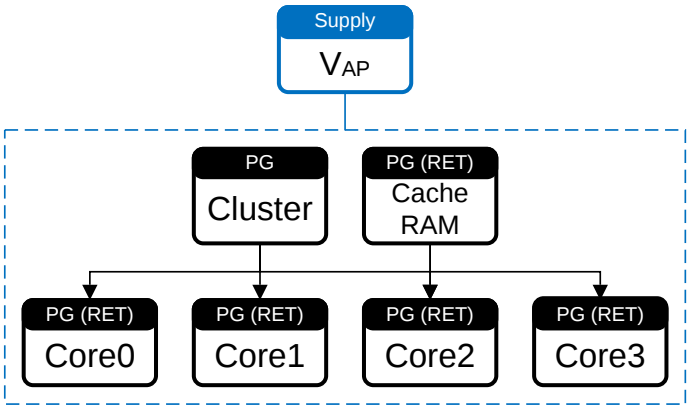


Figure 4.3: Processor power domains example 1 – logical view

For clarity Figure 4.4 shows that in the physical construction all on-chip power switches are implemented in parallel and that the logical ordering is by control sequencing. In further examples this representation will be omitted.

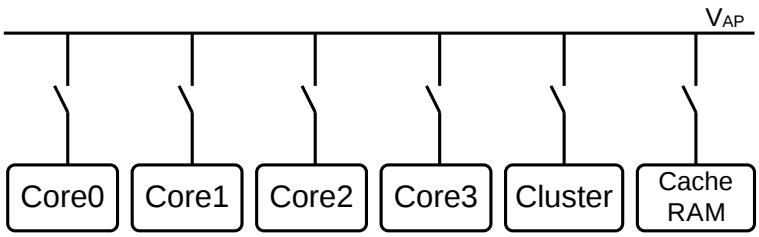


Figure 4.4: Processor power domains example 1 – physical view

Figure 4.5 shows an example logical view of a DynamIQ big.LITTLE cluster. Each of the big and LITTLE core groups have independent voltage supplies and the cluster uses the system logic voltage supply (V_{SYS}).

The cluster and shared cache RAM power-gated domains share common control for power mode transitions. The shared cache RAM can have additional control for retention and partial power-down. However, at a minimum the snoop filter RAM must be available for access when the cluster is powered-on. Per-core power-gated domains are also implemented with the option of support for dynamic retention.

The structure of the figure reflects the hierarchical power-on ordering requirements for the processor subsystem from the voltage supplies, available first, then power domain dependencies flowing downwards. Power-off ordering is the reverse sequence.

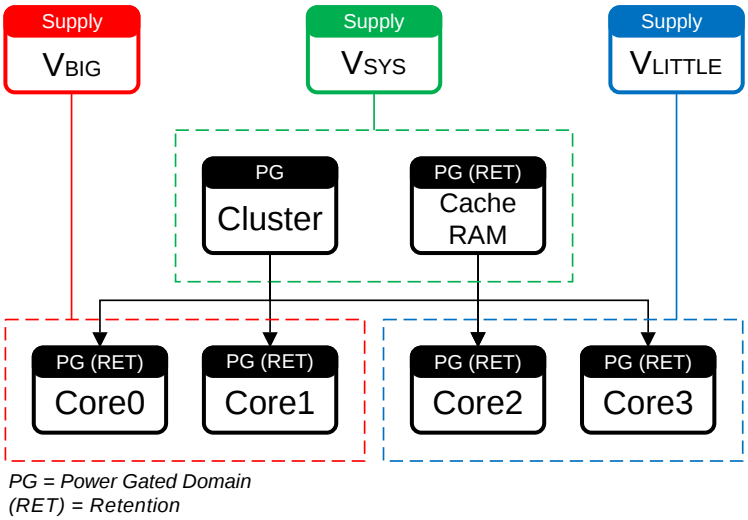


Figure 4.5: Processor power domains example 2 – logical view

4.2.6 CoreSight Logic

Shared CoreSight infrastructure, except for Debug Access Ports which must be always-on, can have a dedicated power-gated domain. This is anticipated to be within the system logic voltage domain.

If a dedicated power domain is not implemented this logic is in the system logic power domain.

4.2.7 Graphics Processor

The descriptions in this section are intended to be generally applicable to Arm Mali™ GPU products. Specific product documentation must always be consulted.

Arm Mali GPU products have an integrated power management capability. They contain a relative always-on domain that must be powered before the GPU driver software can schedule any work. This domain contains power-gated domain control support for each of the GPU cores and their top-level core group logic. This function, known as the job manager (JM), dynamically powers-on resources as work is scheduled and powers-off resources when work is completed.

From a high-level system power control and OSPM perspective, only the availability of the job manager power domain needs to be considered.

Job Manager Power Domain

If the GPU does not have a dedicated voltage domain, the job manager can either be merged into a system power domain or implemented as a dedicated power-gated domain.

If the GPU has a dedicated voltage domain, this logic must then be placed in a power domain within that voltage domain.

When the job manager is implemented as a power domain within the GPU voltage domain it can be aggregated with any other top-level integration logic such as the appropriate voltage domain portion of DVFS bridges and any GPU dedicated interconnect functions.

This power domain can be implemented as a power-gated domain or be implemented as an un-gated power domain reliant on external switch-off of the GPU voltage supply in SoC sleep states.

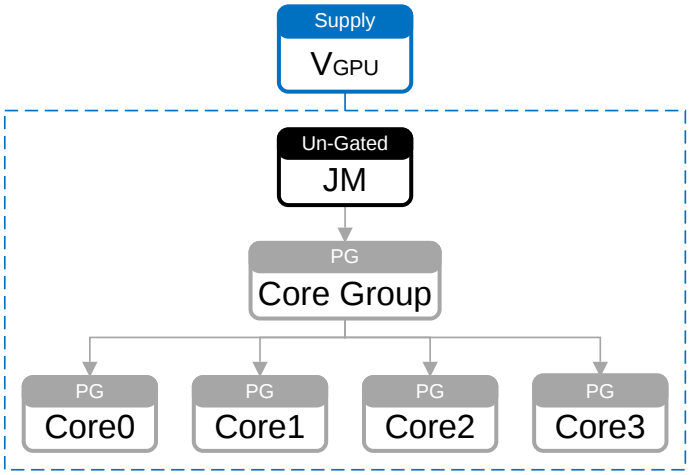
Core and Core Group Power Domains

The implementation of power-gated domains for the GPU cores and top-level core group logic is recommended since these are significant sized function blocks that are used dynamically. Either individual core-group and per-core power-gated domains or a single merged power domain can be implemented.

GPU Partitioning Examples

Figure 4.6 shows an example logical view for a MP4 GPU. The structure of the figure reflects the hierarchical power ordering requirements not the physical implementation.

In this example, a dedicated voltage domain (V_{GPU}) is provided to support DVFS. The job manager domain is an un-gated power domain in V_{GPU} and is available whenever the supply is powered. Core-group and per-core power-gated domains are supported. The lighter color shading reflects that these domains are managed by the job manager and are not system visible.



PG = Power Gated Domain

Figure 4.6: GPU power domains example 1 – logical view

Figure 4.7 shows a second example logical view for a smaller MP2 GPU. In this case DVFS is not supported and the GPU power domains are children of the system logic voltage domain (V_{SYS}).

In this example, the job manager has been merged into the system logic domain containing shared resources (SYSTOP). When SYSTOP is powered the job-manager is available. Core-group and per-core power gating are supported. The lighter color shading again reflects that these domains are managed by the job manager and are not system visible.

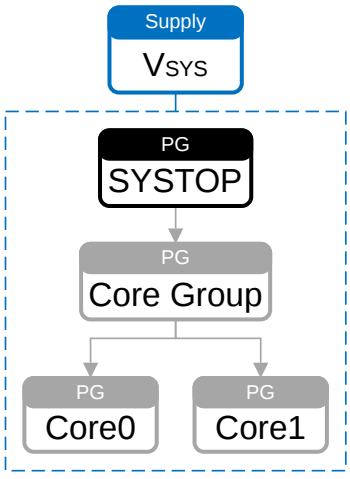


Figure 4.7: GPU power domains example 2 – logical view

4.2.8 Display Processor

The Arm Mali display processors supports multiple power-gated domains. For details consult specific product documentation.

In all cases, the supported power-gated domains are externally managed.

4.2.9 Other Functions

For the remaining SoC functions the choice to implement power-gated domains is typically made as outlined earlier, where a function or set of functions is used only in specific scenarios. Further partitioning within a function, into processing cores for example, should be considered based on the utility and capability for dynamic dimensioning of the system.

Typically, a significant function size is required to justify the saving of additional domains. However, some special cases might be found to justify small domains. One example is an offload processor that can operate independently, decoding audio for example, while the system logic is unavailable.

In a mobile SoC additional power-gated domains would be expected amongst media functions such imaging processors, and other complex functions such as high speed I/O interfaces. High speed I/O interfaces are one example that might also have requirements for ‘relative always-on’ wake domains to support suspend and wake-on-LAN like features.

4.2.10 Power Domain Hierarchy Requirements

Several fundamental relationships between critical resources must be maintained in an Arm based SoC to provide a functional system that can run a standard OS. This section provides high-level considerations for power domain hierarchies to maintain these relationships.

The *Server Base System Architecture (SBSA)* [1] describes requirements that relate to power domain hierarchy in terms of:

- System time provision compliant with the Generic Timer specification in the *ARM Architecture Reference Manual ARMv8* [2] and *ARM Architecture Reference Manual, ARMv7-A* [3].
- Semantics for two wake-up methods:
  - Interrupts from the Generic Interrupt Controller (GIC)
  - Always-on domain wake events
- System memory availability

The SBSA also describes power state semantics consistent with these requirements. Power states are covered in Chapter 5 *Power States*.

Figure 4.8 shows a power domain hierarchy similar that shown in *SBSA* which conforms to these requirements. Other examples that conform to the requirements, that are not simply subsets, are possible.

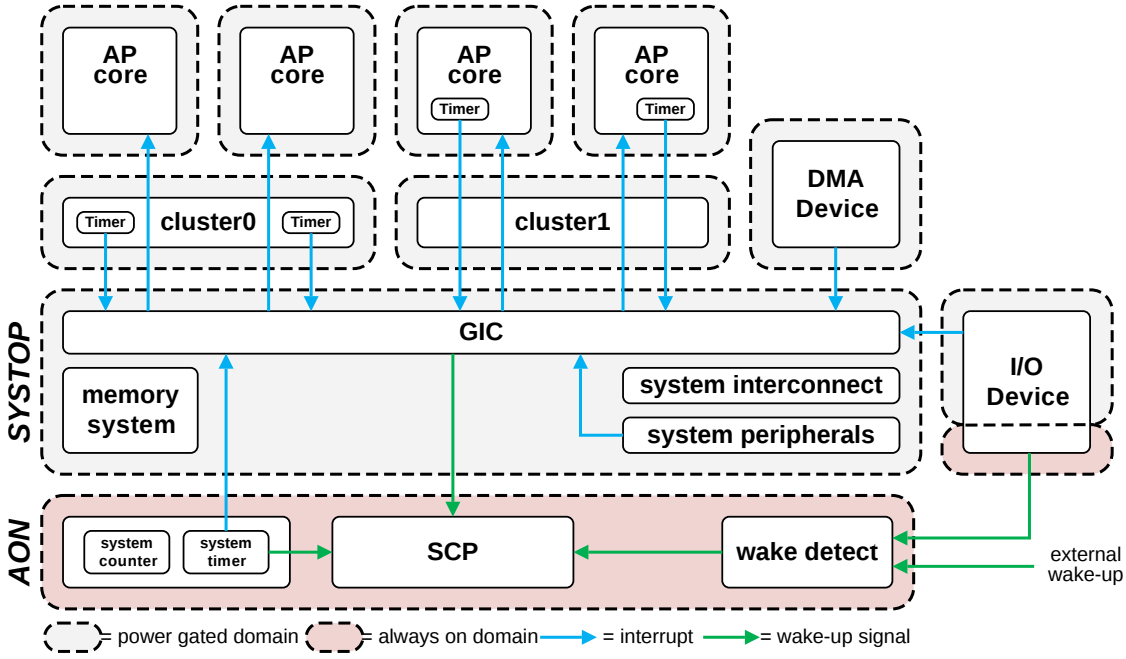


Figure 4.8: Example power domain hierarchy

In Figure 4.8 there are the following power domains:

- An always-on power domain (AON) containing the System Control Processor, a generic timer subsystem and wake detection logic.
- A system logic domain (SYSTOP) containing the GIC and shared system logic functions including the access path to system memory.
- Two processor clusters with per-core and cluster power-gated domains.

- A DMA device that is *OS managed.* It is dependent on the availability of the GIC and system memory when it is powered.
- An I/O device with *self-managed* capabilities that has a power-gated domain and an always-on domain. The always-on domain can generate wake events to express resource requests to the SCP.

The following sections describe the requirements arising from the *SBSA* [1] as relevant to power domain partitioning.

Timer Subsystem

The SoC must include a generic timer subsystem in an always-on power domain. This enables the following requirements to be satisfied:

- The system counter is required to provide a consistent incrementing view of time for the entire time the SoC is powered-on.
- The system counter count value must be visible as an input for all AP core private timers whenever those timers are on.

If the SoC supports sleep states where the GIC is powered-off, it is a requirement to be able to produce an always-on power domain wake event on expiry of an always-on wake-up timer event. The SCP firmware must support waking an AP core on this event. The interrupt is also sent to the GIC, at power-on, pending availability of the AP core.

Always-on power domain events that are also GIC SPI interrupts must be level sensitive, or regenerated accordingly, to ensure the interrupt is observed by the GIC after it is powered-on.

Generic Interrupt Controller (GIC)

In GICv2 implementations all GIC logic is within a single power domain.

In GICv3 implementations, the GIC CPU interface must be incorporated in the AP core power domain while the associated redistributor, distributor, and any interrupt translation service (ITS) support can be in other power domains.

A redistributor might be implemented in the same, or a relatively always-on power domain to the AP core it is associated with.

The GIC distributor must have a relatively always-on relationship to all AP cores. However, the distributor is not required to be always-on and can be powered-off in SoC sleep states. In the example of

Figure 4.8 the GIC is a single power domain implementation placed in the system logic domain (SYSTOP).

When the GIC distributor is powered-off an AP core can only be woken by always-on domain wake events.

Core Timers

The local AP core timers are an important source of interrupts. However, the core timers might power-off, or be reset, with the core.

If AP core timers can be powered-off, a wake-up timer must be available as an interrupt source to the GIC to allow a wake-up signal to be sent to the SCP to power-on the core.

Unless all the local AP core timers are always-on, level 0 of SBSA only requires a system-specific timer for this purpose. However, it is recommended that systems comply with at least level 1 of SBSA and, unless all local AP core timers are always-on, implement a memory mapped generic timer mapped in non-secure address space for this purpose.

In large scale systems, timer scalability might be addressed with AP core timers described as always-on in firmware tables. AP firmware abstracts the always-on functionality according to the available timer hardware resources. These resources might range, according to the scale of the system, from a single shared secure memory mapped always-on timer, to a memory mapped always-on timer for each core.

Always-On Domain Wake Events

Always-on domain wake events are fundamental to the support of SoC sleep states where only the always-on logic is powered.

A requirement has already been stated that the system must support an always-on power domain wake event on expiry of an always-on wake-up timer.

Other always-on domain wake events are implementation defined. Common examples as illustrated in Figure 4.8:

- **SoC external events:** Off-chip events from sources such as cable detection, power buttons or wireless subsystems.
- **On-chip events from devices with self-managed capability:** Complex subsystems such as high speed I/O devices or integrated modems with their own always-on domain support.

SCP firmware must wake the required system resources in response to the supported events.

System Memory

Whenever a component that can initiate memory transactions is powered-on, the system memory must be available.

However, it should also be noted that system MMUs and the GICv3, required by higher *SBSA* levels, make use of tables in memory. Therefore, when systems containing these components are in power states where at least one SMMU or the GIC is on, system memory must be available.

The definition of *available* in this case is that the memory will respond to requests without requiring intervention from software running on AP cores. Therefore, it is possible for the memory system to be powered-off when agents that require its availability are active, provided those requests trigger a mechanism that allows the transaction to be completed transparently.

In the example of Figure 4.8 the path to system memory is in the same power domain (SYSTOP) as the GIC and therefore the system memory will be available when any AP core is powered-on.

However, components, such as the I/O device in the example of Figure 4.8, with the capability to generate always-on domain wake events, might be powered when the SYSTOP domain is off. If such a component generated a request when SYSTOP was powered-off a hardware component would be required to generate a wake event to the SCP and stall the transaction until system memory was made available. The requirements for such a component are detailed in 7.2.8 *Access Control*.

Power Ordering Requirements

The power ordering requirements of the example shown in Figure 4.8 can be summarised simply as:

- The always-on domain is available whenever the SoC is powered-on.
- The SYSTOP domain is relatively always-on to any processor cluster.
- Each processor cluster is relatively always-on to its cores.
- The SYSTOP domain is relatively always-on to OS managed devices. This class of device is represented in Figure 4.8 by the DMA device.
- The I/O device in Figure 4.8 has self-managed capabilities. Its power-gated domain being powered-on is only dependent on the prior availability of its always-on domain logic.

A relatively always-on relationship is typically preserved by powering on the domain lower in the hierarchy before its dependent domains and powering off the domain only after dependent domains have been powered-off. For example: SYSTOP must be powered-on before any processor cluster is powered-on and the reverse for power-off.

However, simultaneous powering of dependent domains is possible provided management of reset release and low-power interface controls respects the logical ordering required.

4.2.11 SoC Partitioning Example

Figure 4.9 shows a high-end mobile SoC example as a summary for voltage and power domain partitioning.

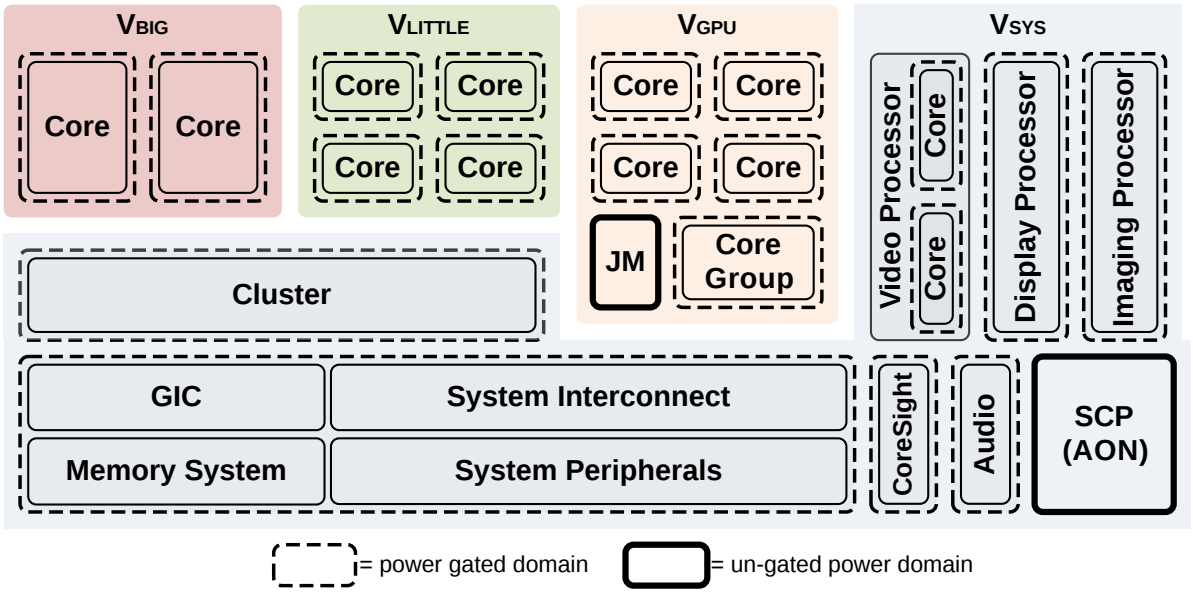


Figure 4.9: SoC voltage and power domain partitioning example

Figure 4.9 is simplified in terms of the range of functions illustrated but is representative in terms of choices that might be made.

The imaging processor and audio power domains shown do not represent Arm components. These are included as examples of components likely to have dedicated power-gated domains.

Although not shown, the core scheduler logic of the video processor is assumed to be contained within the power domain shown to contain the GIC, memory system, system interconnect and system peripherals.

Chapter 5
Power States

This chapter defines power states and power modes and the relationship between them. It also describes the definition and use of a power state hierarchy.

This chapter contains the following sections:

- 5.1 *Power States and Power Modes*
- 5.2 *Power State Hierarchy*
- 5.3 *Coordination by System Control Processor*

5.1 Power States and Power Modes

5.1.1 Power States

A power state represents a software visible abstraction of available hardware power modes.

A power state is defined according to wake capability, loss of context and power consumption. The selection of a power state by software is typically made using an idle residency prediction. This prediction is used to make a state selection based on target residences, derived from energy break even times, for each available state. Wake latency requirements, which must not be violated, might then limit the choice to a shallower state.

The power state selected by software sets constraints on which power modes can be selected.

5.1.2 Power Modes

A power mode represents the hardware power saving capabilities of a SoC function.

Although differentiated on hardware techniques power modes can be classified, similarly to power states, according to wake capability, loss of context and power consumption. Also, similarly, selection of a specific power mode can be according to residency targets and wake latency requirements.

However, not all hardware power modes are software visible. While use of these modes might be enabled or disabled in software the transitions to and from them can be hardware autonomous.

Power modes are selected within the constraints of the current power state. This means that a shallower power mode, with higher capabilities than the power state constraints require, can be selected, but a deeper power mode must not be selected.

NOTE: Power modes related to leakage saving techniques were defined in 4.2.1 *Power Modes*. A component within a power domain can also have power modes that are related to dynamic power reduction that might be visible as power states to software.

5.1.3 Distinction of Power States from Power Modes

It is important to make a clear distinction between the power modes supported by the hardware and the power state view of software. The primary reason for this distinction is that there is no direct mapping between these two views.

In summary:

- Not all hardware power modes are software visible.
- Power modes are differentiated on hardware techniques whereas the considerations for defining a software power state are wake capability and loss of context.
- A single power state can map to one or more power modes, with equivalent context and wake properties, where power the modes are chosen autonomously by the power control system. Autonomous modes must preserve the properties of the selected power state without a perceived impact to latency on exit from the state.
- The power mode selected can be shallower than the power state constraints allow, but not deeper.

5.2 Power State Hierarchy

Power states can be organized as a hierarchy of power state tables. Each power state table describes the power states available at its level of hierarchy.

From a system level power control perspective, this hierarchy of power states is convenient as it enables all legal combinations of power states and power modes to be represented with minimal redundancy.

Although there can be any number of levels in the power state table hierarchy, this document describes three levels of hierarchy. These levels are necessary for effective power management and are described in the following sections:

- 5.2.1 *Core Power States*
- 5.2.2 *Cluster Power States*
- 5.2.3 *Device Power States*
- 5.2.4 *SoC Power States*

Figure 5.1 shows an example power state hierarchy for a two-cluster system. Device power state tables are not described in the illustration but, depending on the level of OS management, some devices might also have a declared power state hierarchy.

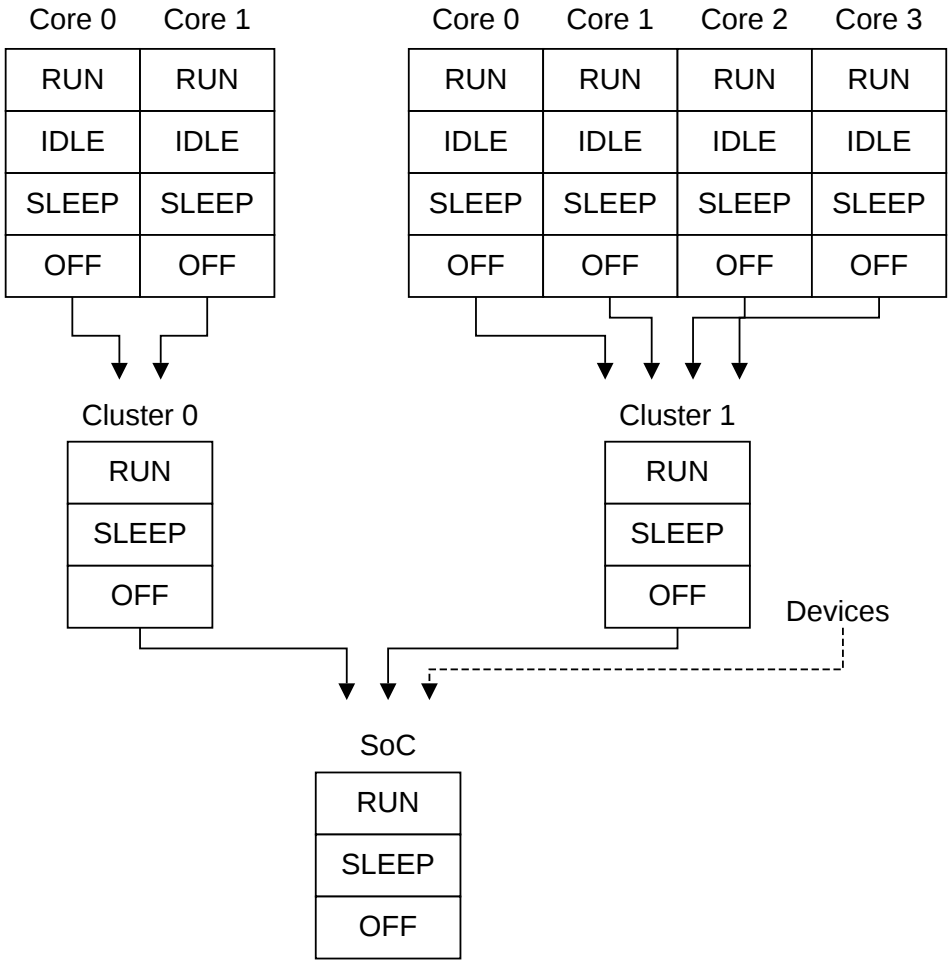


Figure 5.1: Power state hierarchy example

5.2.1 Core Power States

The *Server Base System Architecture* [1] defines the following power state semantics for AP cores.

RUN

Core is powered-on and running code.

IDLE_STANDBY

The core is in WFI state. Full context is retained and no software state saving or restoration is required. Execution automatically resumes after any interrupt or external debug request. Debug registers are externally accessible.

IDLE_RETENTION

The core is in WFI state. Full context is retained and no software state saving, or restoration is required. Execution automatically resumes after any interrupt or external debug request. Debug registers are not externally accessible.

SLEEP

The core is powered-off, but hardware will wake the core autonomously, for example on receiving a wake-up interrupt from the GIC. No context is retained so state must be explicitly saved. A woken core starts at the reset vector, and then hardware specific software will restore state.

OFF

The core is powered-off and is not required to be woken by interrupts. The only way to wake the core is by explicitly requesting it to be powered-on by the power controller, for example from system software running on another core, or an external source such as a power-on reset. This state can be used when the system software explicitly decides to remove the core from active service, giving the hardware opportunity for more aggressive power saving. No core context is retained.

NOTE: The SLEEP and OFF power states might use the same power modes but are semantically different because of the ability of the core in the SLEEP state to wake on receiving an interrupt.

Not all core power states will always be available. Their availability depends on the power modes that are supported and implemented.

Relationship to GIC Architecture

The Arm GICv2 and GICv3 architectures define wake request mechanisms that are used to route interrupt requests for powered-off cores to the power controller.

The GIC wake request mechanisms are architected to support software managed SLEEP and OFF states which do not retain context and re-start through a reset. In this case interrupt forwarding to the core is disabled and the wake request mechanism is routed to the power controller.

By comparison, the IDLE_STANDBY and IDLE_RETENTION states, that maintain the core context, resume automatically on an interrupt. In this case interrupt forwarding to the core remains enabled during the state and the wake request mechanism is not used. In these states, while there may be a software choice to enter the state, no other software management is required.

Implementing core support for IDLE_STANDBY and IDLE_RETENTION states requires that logic for wake sources is outside of the core power domain and remains active during these states. Wake source logic includes the private timer as well as the detection of interrupts, events, snoop accesses and debug requests.

Example Mapping of Core Power States to Modes

Table 5.1 uses the supported features of Cortex-A55 to provide an example mapping of AP core power states to the available core power modes:

Table 5.1: Cortex-A55: mapping AP core power states to modes

Power State	Power Mode		Note
	Core	Advanced SIMD / FP	
RUN	ON	ON	WFI or WFE
		RET	WFI or WFE, Advanced SIMD/FP retention
IDLE_STANDBY	ON	ON	WFI or WFE
		RET	WFI or WFE, Advanced SIMD/FP retention
	RET	RET	WFI or WFE, core retention
SLEEP	OFF	OFF	Off, can be woken by interrupts
OFF			Off, cannot be woken by interrupts

NOTE: Cortex-A55 does not implement support for the IDLE_RETENTION state semantic as the core will respond to external access of debug registers in WFI or WFE.

The grouping of the supported power states to power modes is concerned with the properties of loss of context and wake properties:

- The RUN power state represents any mode where the core is running. There are no context or wake considerations. The HW can autonomously, transparent to software, enter and exit the Advanced-SIMD/FP pipeline from a retention mode. The hardware implementation is intended to ensure there is no perceived latency impact on execution time.
- The IDLE_STANDBY and IDLE_RETENTION states represent any mode where the core is in WFI or WFE with no loss of context. Execution can resume directly from any wake event. The retention modes are autonomous, and the hardware implementation is intended to ensure there is no latency impact perceived by software because of this autonomous power mode selection.
- The SLEEP and OFF states represent the modes where the core has lost context. These states are differentiated by their wake properties. Wake latency from these states is significantly higher.

5.2.2 Cluster Power States

This document defines the following AP cluster power states.

RUN

The cluster is powered-on and can support any core moving to any power state.

SLEEP_RETENTION

The cluster is powered-off, but able to wake on receiving a wake capable interrupt. At least one core in the cluster is in SLEEP, while other cores are in SLEEP or OFF.

The cluster shared cache content is retained.

Before a core in the cluster can move to a higher power state, the cluster must first move to RUN.

SLEEP

The cluster is powered-off, but able to wake on receiving a wake capable interrupt. At least one core in the cluster is in SLEEP, while other cores are in SLEEP or OFF.

The cluster shared cache content is not retained.

Before a core in the cluster can move to a higher power state, the cluster must first move to RUN.

OFF

The cluster is powered-off and will not wake on receiving an interrupt. All cores in the cluster are in OFF.

The cluster shared cache content is not retained.

Before a core in the cluster can move to a higher power state, the cluster must first be moved to an appropriate higher power state.

NOTE: SLEEP and OFF power states might use the same hardware power modes but are semantically different at the system level because of the ability of the cluster in SLEEP to wake when a core in the cluster receives a wake capable interrupt.

Example mapping of Cluster Power States to Modes

Table 5.2 uses the supported features of the DynamIQ Shared Unit (DSU) processor cluster to provide an example mapping of cluster power states to available power modes.

The example highlights the value of both the grouping of modes into power states and the hierarchical approach where many core modes and states can be collapsed into few cluster level power states.

Table 5.2: DynamIQ Shared Unit: cluster power states mapping to modes

Cluster Power State	Core Power State	Power Mode		Note
		DSU	L3 Cache RAMs	
RUN	Any	ON	ON	Full L3 cache.
			RET	Dynamic L3 RAM retention.
			OFF	Partial L3 cache power-off.
SLEEP_RETENTION	All Cores in SLEEP or OFF, with at least one in SLEEP	OFF	RET	Cluster off. Cores can wake from interrupts. Static L3 retention.
SLEEP	All Cores in SLEEP or OFF, with at least one in SLEEP	OFF	OFF	Cluster off. Cores can wake from interrupts.
OFF	All Cores in OFF	OFF	OFF	Cluster off. Cores in this cluster cannot wake from interrupts.

5.2.3 Device Power States

The OSPM will have a mechanism to represent device dependencies that prevent the SoC from entering a sleep state until those dependencies are resolved. This dependency management places devices at the same level in power management hierarchy as processor clusters.

This document defines the following simple device power states.

RUN

Device is powered-on and can perform its operations. Driver specific actions might however be needed to enable clocks and other capabilities.

OFF

Device is powered-off. The only way to wake the device is by explicitly requesting it to be powered-on. Typically, the driver software will express this dependency through an abstraction to the OSPM. The OSPM can request the SCP to perform any required actions.

5.2.4 SoC Power States

The SoC power states used in this document are for illustration purposes only. The power states for the SoC might have additional levels of hierarchy and will consider the power states of components not covered in this section.

This document defines the following SoC power states.

RUN

The SoC system is available and can support any processor cluster moving to any power state. Devices can also move to any power state.

At least one AP core in the system is in RUN or SLEEP.

The GIC is on and system memory is available.

SLEEP

The SoC system is unavailable and can be powered-off, but always-on domain wake capabilities including the system counter and wake-up timer remain powered-on. The SoC can self-wake using the wake-up timer. It can also wake in response to any system specific always-on domain wake event.

At least one of the processor clusters must be in SLEEP, and remaining clusters must be in SLEEP or OFF. As the GIC is powered-off, the only interrupts able to wake processors are always-on domain wake events.

OS managed devices are OFF.

Before a processor cluster or OS managed device can move to a higher power state, the system must first move to RUN.

Self-managed devices, depending on their capabilities, can be in any power state. However, if system logic resources are required an always-on domain wake event must be used to request these services. There must be interlocks in place to guarantee safe behavior until the system has moved to RUN.

System memory is not available. Any required context is either migrated to off-chip memory or retained on-chip. Any external DRAM holding system context must be retained, typically by placing the devices into a self-refresh mode prior to entering the SLEEP state.

SLEEP states can be of varying depth according to power saving and increased entry and exit latency. The OSPM selects the SLEEP state depth according to its latency requirements and any target residency prediction.

SLEEP states of varying depth are suggested to be named with an incrementing numeric suffix corresponding to increasing wake latency, for example **SLEEP0**, **SLEEP1** and so on.

DEEPSLEEP

The SoC is powered-off, including the system counter and wake-up timer. The system is unable to self-wake. It can only wake in response to an external event, such as a power-on reset.

External DRAM memory is held in a retention state by implementation specific means.

OFF

The SoC is powered-off, including the system counter and wake-up timer. The system is unable to self-wake. It can only wake in response to an external event, such as a power-on reset.

External DRAM memory is not retained. Hibernation of state to a non-volatile memory might be used but this is beyond the scope of this document.

Power State Mapping

Table 5.3 shows an example of how these power states might map to power domains in a system.

Table 5.3: SoC power states

SoC Power State	Processor Cluster States	OS Managed Device States	Power Mode		System Memory	Note
			GIC	Wakeup timers, system counter		
RUN	Any	Any	ON	ON	Available	AP cores able to run, can wake up on any enabled interrupt. OS managed devices operable.
SLEEP	All clusters in SLEEP or OFF, with at least one in SLEEP	OFF	OFF	ON	Not available. Context is migrated or retained. DRAM self-refresh.	Only interrupts from wake-up timer or other IMPLEMENTATION DEFINED system events including from self-managed devices, can cause a wake-up.
DEEPSLEEP	All clusters in OFF	OFF	OFF	OFF	Not available. DRAM self-refresh.	External wake-up only, for example, power-on reset
OFF	All clusters in OFF	OFF	OFF	OFF	OFF	External wake-up only, for example, power-on reset

5.3 Coordination by System Control Processor

Not all power state requests will be acted on explicitly. In addition to HW autonomous power mode selection, the SCP firmware can reconcile OSPM power state requests along with other requests or constraints, such as from self-managed devices, in the process of power mode selection. In this case, the power mode selection is coordinated by the SCP.

The following sections detail SCP coordination of SoC and cluster power states and related considerations for managing these states.

5.3.1 SoC Power States

While the OSPM SoC SLEEP state selection is determined according to only its requirements, the SCP can select the power mode by reconciling constraints from other agents.

For example, in a system where there are devices with self-managed capabilities the SCP can reconcile requests from these devices and the OSPM SoC SLEEP state constraints to manage the availability of common resources, such as the path to system memory.

To avoid creating a dependency that forces AP cores to be active during all self-managed device activity, the SCP instead controls the entry to and exit from power modes during SLEEP states. Based on this reconciliation of constraints, the SCP can also determine the depth of the power modes selected.

For example, SoC SLEEP state depths might align to different clocking modes. With a higher wake latency constraint, the SCP might be switched to a low frequency clock, allowing a higher speed clock to be turned off. At a lower wake latency constraint, this mode would not be used.

SoC SLEEP state management tasks, depending on the depth of the mode, include voltage supply management, power control, clock supply, preservation of system time through clock changes, system configuration save and restore, and arbitrating access to shared system resources. Some of these actions, configuring the memory system for example, mean there are considerations in terms of the access SCP must have to the system to be able to complete them.

The SoC OFF state is managed by the SCP, but the decision is anticipated to be taken directly by the OSPM and might be done in contradiction to requests from self-managed devices. This OSPM decision is expected to override any other requests.

The SCP might also autonomously place the system, or sub-components, into an OFF state as a protective measure in an alarm condition such as thermal runaway.

5.3.2 Cluster Power States

While the OSPM might provide mechanisms to enable cluster power state selection, it might be better to implement the coordination in the SCP since it has a more recent view of the system conditions. To do this requires the SCP to track AP core power state constraints to determine when a cluster power mode transition can, or must, occur.

In preparation for a cluster power-off the SCP might enforce a period where no new cores can become active in the cluster. This removes the risk of a core powering-on and creating a race condition in the cluster power-off sequence, but depends on both hardware support and firmware design.

To be able to manage cluster power mode transitions, the SCP must be able to control any required shared cache flushing process and control the presence of the cluster in system coherency. Some, but not all, Cortex-A profile processors support an externally initiated shared cache flush which is hardware controlled and does not require a core to be active.

SCP control of the clusters presence in system coherency might require access to control registers in the coherent interconnect.

Chapter 6
Power Control Framework

The power control framework is a collection of standard infrastructure components, interfaces, and associated methods which can be used to build the infrastructure necessary for power management of a SoC.

This chapter describes the primary framework components and low-power interfaces and is divided into the following sections:

- 6.1 *Power Control Framework Overview*
- 6.2 *Low-Power Interfaces*
- 6.3 *Power Modes*
- 6.4 *System Control Processor*
- 6.5 *Power Management Infrastructure Components*

6.1 Power Control Framework Overview

Figure 6.1 shows a high-level illustration of the power control framework concept, including the primary components and how they are interfaced.

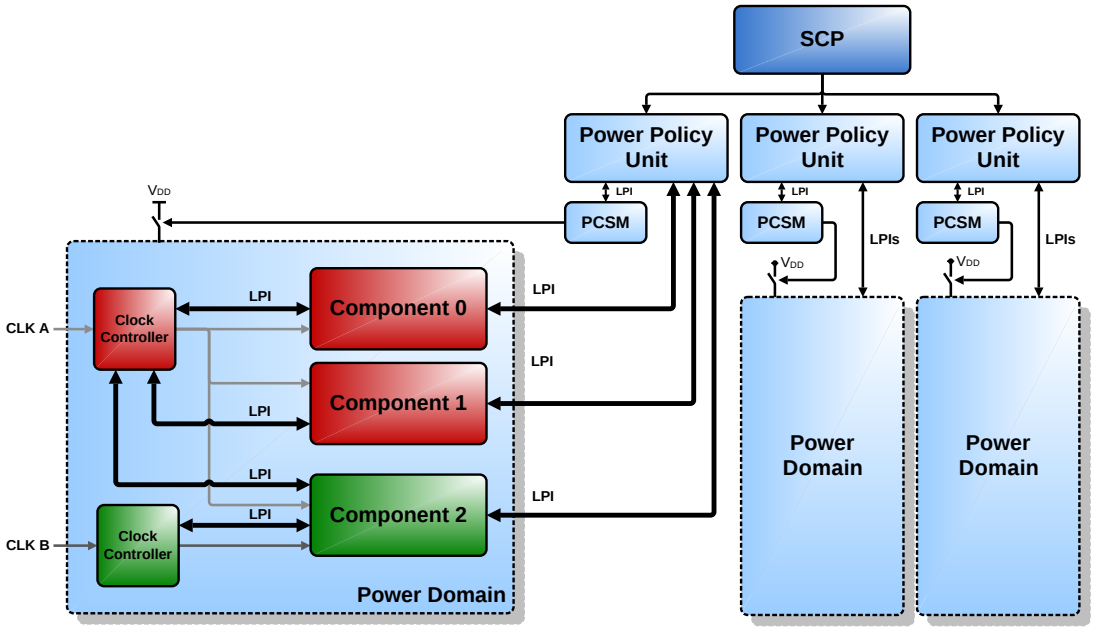


Figure 6.1: Power control framework overview

6.1.1 Power Control Framework Low-Power Interfaces

The Low-Power Interfaces (LPI) described here are:

- 6.2.1 *Q-Channel*
- 6.2.2 *P-Channel*
- 6.2.3 *AXI LPI*
  - This interface is deprecated and is discussed only for use with legacy components.

The protocols for Q-Channel and P-Channel are described in detail in the *Low-Power Interface Specification, ARM Q-Channel and P-Channel Interfaces* [4].

The protocol of the AXI LPI is described in detail in the *AMBA® AXI™ and ACE™ Protocol Specification* [5]. From a power control framework perspective, this interface is deprecated and only for use with legacy components and only with the restrictions described in 6.2.3 *AXI LPI*.

6.1.2 Power Control Framework Infrastructure Components

The power control framework infrastructure components described here are:

- 6.4 *System Control Processor* (SCP)
- 6.5.1 *Power Policy Unit* (PPU)
- 6.5.2 *Clock Controller* (CLK_CTRL)
- 6.5.3 *Low Power Distributor* (LPD)
- 6.5.4 *Low Power Combiner* (LPC)
- 6.5.5 *P-Channel to Q-Channel Convertor* (P2Q)

More details about integrating these components and related requirements for other components can be found in *System Power Control Integration*.

The architecture of the *Power Policy Unit* (PPU) is described in detail in the *ARM® Power Policy Unit Architecture Specification* [6].

6.2 Low-Power Interfaces

The Low-Power Interfaces (LPI) are controller to device interfaces used to guarantee the availability, and removal, of a resource to the controlled component.

There are several LPI, each interface is complementary and should be selected as appropriate to the circumstances in which they are used.

6.2.1 Q-Channel

Q-Channel has simple run-stop semantics which are ideal for clock control and simple power control. The low-power mode entered can vary, but only according to a common configuration of both the component and the controller before the mode is requested.

Full details of the Q-Channel protocol can be found in *Low-Power Interface Specification, ARM Q-Channel and P-Channel Interfaces* [4].

6.2.2 P-Channel

P-Channel has a mode specification capability meaning that transitions to different modes can be specified on a single interface. This makes it suitable for more complex power control with multiple modes and more complex mode transitions.

Full details of the P-Channel protocol can be found in *Low-Power Interface Specification, ARM Q-Channel and P-Channel Interfaces* [4].

6.2.3 AXI LPI

AXI LPI is an interface supported by some legacy components and is not used on new components. Q-Channel is backward compatible with AXI LPI within the restrictions detailed in 6.2.3 .

Full details of the AXI LPI protocol can be found in *AMBA AXI and ACE Protocol Specification* [5].

For details of the compatibility with Q-Channel see the *Low-Power Interface Specification, ARM Q-Channel and P-Channel Interfaces* [4].

Restrictions on the use of AXI LPI

The AXI LPI specification has a denial mechanism that requires the level of the **CACTIVE** signal to be evaluated when **CSYSACK** goes LOW, at the completion of a low-power request handshake. If at this point **CACTIVE** is HIGH, the controller must maintain the supply of clock or power guaranteed by the interface.

Q-Channel is not backward compatible with the AXI LPI denial mechanism and controllers designed to the Q-Channel specification cannot be used with AXI LPI components that are dependent on this specific behavior.

Arm CoreLink™-400 components with AXI LPI are not dependent on this denial mechanism and can be used with controllers designed to the Q-Channel specification.

NOTE: While a Q-Channel controller can be used with an AXI LPI component that does not rely on the denial mechanism, a controller designed for an AXI LPI component cannot be used with a Q-Channel component.

6.3 Power Modes

Power modes cover the combinations of logic and RAM power states for a power domain and the associated clock, reset and isolation control.

The PCSA defines a range of power modes, these are listed in Table 6.1.

Table 6.1: PCSA power modes

Power Mode	Short name	Logic Mode	RAM Mode	Description
Debug Recovery Reset	DBG_RECOV	ON	ON	Warm reset application with logic and memories on. This mode is used to enable reset of a component, while retaining specific state for later debug analysis.
Warm Reset	WARM_RST	ON	ON	Warm reset application with logic and memories on.
On	ON	ON	ON	Logic on and any memory on. The component is functional.
Functional Retention	FUNC_RET	ON	RET	Logic on with memories retained. The component is still functional.
Memory Off	MEM_OFF	ON	OFF	Logic on with memory off. The component is still functional.
Full Retention	FULL_RET	RET	RET	Logic and memory in retention.
Logic Retention	LOGIC_RET	RET	OFF	Logic retention with memories off.
Emulated Memory Retention	MEM_RET_EMU	ON	ON	Logic on and memories on. This mode is used to emulate a memory retention condition without removing power.
Memory Retention	MEM_RET	OFF	RET	Logic off with memories retained.
Emulated Off	OFF_EMU	ON	ON	Logic on and memories on. This mode is used to emulate a power-off condition without removing power.
Off	OFF	OFF	OFF	Logic off and memories off.

The PCSA also defines the P-Channel **PSTATE** and **PACTIVE** bit values related to these power modes. These are listed in Table 6.2.

Table 6.2: PCSA power mode PSTATE and PACTIVE usage

Power Mode	PACTIVE bit	PSTATE [3:0]	Mode Priority
DBG_RECOV	10	0b1010	High
WARM_RST	9	0b1001	
ON	8	0b1000	
FUNC_RET	7	0b0111	
MEM_OFF	6	0b0110	
FULL_RET	5	0b0101	
LOGIC_RET	4	0b0100	
MEM_RET_EMU	3	0b0011	
MEM_RET	2	0b0010	
OFF_EMU	1	0b0001	
OFF	0	0b0000	Low

6.4 System Control Processor

The system control processor (SCP) is a processor-based capability that provides a flexible and extensible platform for provision of power management functions and services.

An overview of the SCP functions and services is given in 3.2 *System Control Processor*.

Figure 6.2 shows an example SCP hardware overview.

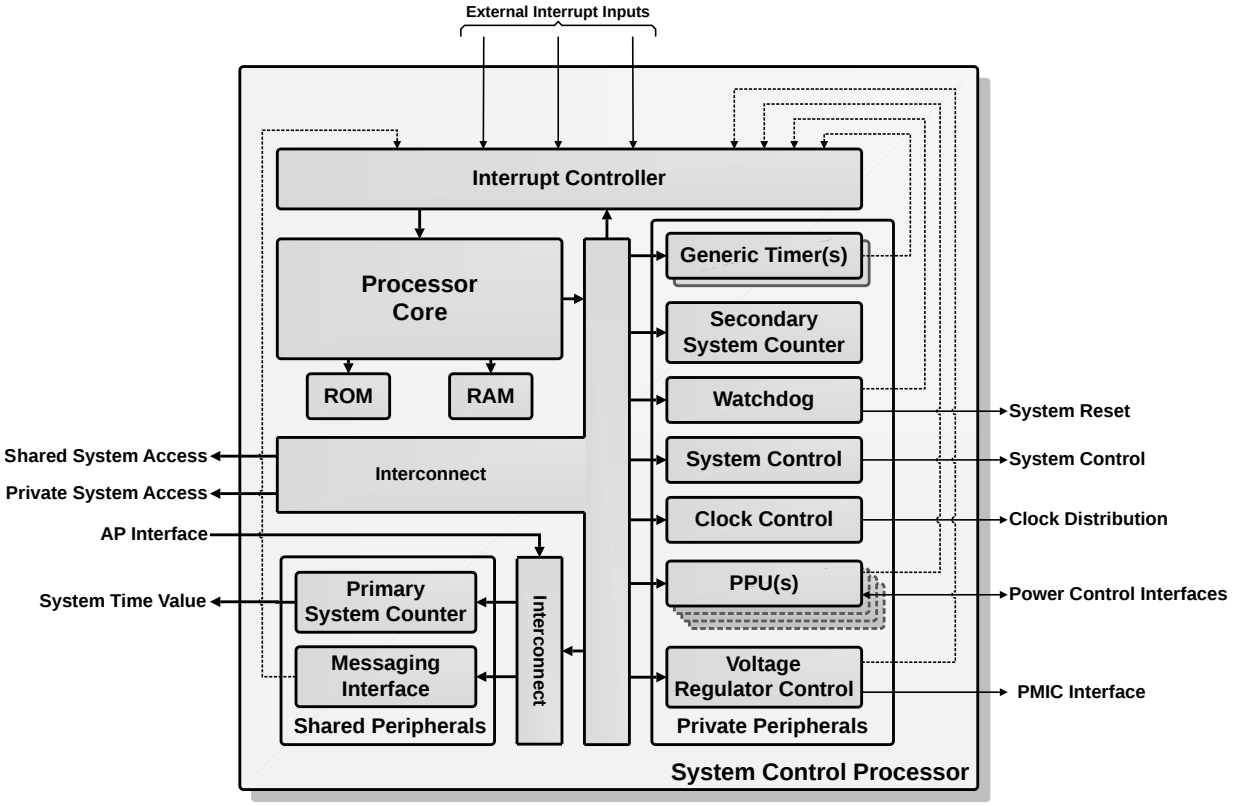


Figure 6.2: SCP example overview

NOTE: The exact structure of the SCP will depend upon the choice of processor and peripherals. This section describes a generalized structure providing the capability required to support the functions and services described in this document.

6.4.1 SCP Components

This section describes the components of the SCP subsystem.

SCP Processor Core

The processor runs firmware that takes actions based upon requests, events and scheduled tasks.

SCP Processor Core Selection

The selection of the processor used in the SCP is dependent on the requirements of the system. A list is given below of important items to consider:

- **Processing Capacity**: The SCP must have enough processing capacity to handle the tasks it is expected to run, especially any tasks with real-time dependencies or where tasks might need to be handled concurrently. Tasks that scale with AP core count, such as idle management and sensor monitoring, will typically dominate utilization. AP core idle management is typically the most latency sensitive task load and the tail of the latency distribution can be managed by the provision of enough performance to maintain an overall moderate utilization. Selection criteria include the type of processor and the frequency at which it operates.
- **Interrupt Types**: Some processors have vectored interrupts that can ease the connection, handling, and latency of interrupts, reducing or removing the need for software interaction to discover interrupt causes.
- **Interrupt Priority and Latency**: The intrinsic interrupt latency should be considered alongside other interrupt features of the core. These can include hardware interrupt priority, interrupt nesting and tail chaining between interrupt handlers, and hardware context switching between interrupt vectors. In the case of vectored interrupts, the number of interrupt inputs available should also be considered. If this is too low and an additional interrupt controller is required within the SCP this adds latency to the interrupt sequence in both firmware and hardware
- **Area and Power Consumption**: The SCP will typically be always-on. Therefore, power consumption is an important consideration. This is especially true for SLEEP states where the SoC might spend a considerable amount of time.
- **Debug and Trace**: The ability to debug and profile the firmware is critical for development. Considerations include the native debug and trace support of the SCP processor subsystem available when the SoC is in SLEEP states and the integration into the broader CoreSight SoC debug and trace system.
- **Trusted Operation**: As the SCP controls sensitive parts of the SoC, security is a concern. The SCP runs from private local memory and its firmware can be loaded through a trusted boot process allowing the SCP to be inherently trusted. However, where the SCP needs to access other parts of the system, the security of this and the components it accesses need to be considered.

For mobile systems, the SCP processor core might be an Arm Cortex-M microcontroller, for example a Cortex-M3. Other systems might consider another Arm profile core such as a Cortex-R or Cortex-A. In all cases the choice is dependent on the factors outlined above.

SCP Processor Core Memory

The processor has ROM for boot and RAM for storing firmware instructions and data. The ROM and RAM are private to the SCP.

A possible implementation is that the ROM is used at boot to bring the system to a state where a host processor can access the memory system and load, either directly or indirectly, the SCP firmware.

Trusted boot requirements for client systems are provided in the *Trusted Board Boot Requirements – CLIENT* [7] specification.

The SCP firmware code and data spaces are entirely within its private RAM. The SCP can then operate while the remainder of the SoC is off and system memory is unavailable. However, when available, the SCP can access system memory and other parts of the system as required.

System Counters and Generic Timers

The system counters provide a common time reference for the SoC. Generic timers use the counter values to produce interrupts and wake-up events. The primary system counter value is distributed to other system elements, including application processors and debug infrastructure, to provide a consistent view of time.

The primary system counter resolution requires a clock source that, due to power consumption reasons, might be turned off in SoC SLEEP states. The provision of a secondary constantly running low speed counter, typically using a real-time clock source as its input, enables the required view of time to be preserved through SoC SLEEP states. Generic timers using this counter are used to generate wake-up events in these states.

The SCP is responsible for managing the transfer of time between these system counters at entry to and exit from SLEEP states to ensure a consistent view of time is presented to the system. This can be achieved with a combination of hardware and firmware capability in the SCP.

For details on the ARMv7 Architecture System Counters and Generic Timers see the *ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition* [3].

For details on the ARMv8 Architecture System Counters and Generic Timers see the *ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile* [2].

Watchdog

The SCP watchdog provides functionality to prevent system deadlocks. If the watchdog is not written to on a regular basis by the SCP then the watchdog will produce an interrupt, and this will ultimately lead to a reset of the system.

A syndrome register must be available to inform the SCP processor of the last reset cause so it can take appropriate actions.

While the SCP provides this specific watchdog functionality, other system watchdogs might be managed by the AP software. See the *Server Base System Architecture (SBSA)* [1] for specific requirements.

Voltage Regulator Control

The SCP manages voltage supplies for functions including post-boot switch-on, switch off and DVFS voltage level changes.

The voltage supplies are typically provided by a separate power management IC. The voltage regulator control component provides the interface for this function. The protocol of the interface is implementation specific dependent on the choice of power management IC.

Clock Control

The SCP does not control run time dynamic gating of clocks at component activity level. This is managed by clock controllers with hardware autonomous Q-Channel management.

The SCP manages clock source enabling, selection, and division. Clock sources might include off chip sources, such as crystal oscillators, and on-chip sources such as PLLs. Each clock source will typically be able to be divided to produce a multitude of frequencies for different components.

These settings might be static, set up once when a component is required or powered up, or changed at the request of the component or related software, such as for AP or GPU DVFS.

For more details on dynamic clock gating see 6.5.2 *Clock Controller* and 7.1 *Clock Control Integration*.

System Control

The SCP can manage miscellaneous system control tasks. Where a specific responsiveness is not a constraint, this can often be handled by register controlled outputs and interrupt inputs.

A simple application example is managing a four-phase request acknowledge handshake with a SoC component. When hardware external to the SCP acts as the requestor, the request signal is connected to an interrupt line conditioned to generate a pulse at each edge of the request. An input status register bit can also be used to determine the level of the request. A control register bit driving an SCP output is used for the acknowledge signal. When the roles are reversed so are the corresponding signal connections.

Figure 6.3 shows an example of this application where a SoC component acts as the requestor.

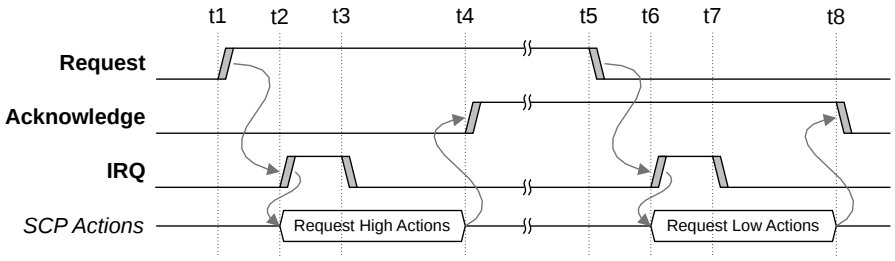


Figure 6.3: System control handshake example

The inclusion of any system control registers and allocation of interrupts for this purpose is optional and depends on system requirements.

Messaging Interface

To allow the communication of requests between the OSPM and the SCP messaging interface, through a software interface such as SCMI, hardware support is required. While this can take several forms, the solution must be one that is simple to describe generically to an OS. Schemes using shared memory mailboxes and doorbell interrupts are typical and well suited to this purpose.

The messaging interface must be usable by any AP core in the system.

A typical embodiment is a simple piece of hardware that allows either entity to send and read messages and generate interrupts to each other to indicate the availability of a message.

A typical OSPM to SCP communication method might be:

1. OSPM:
    a. Stores a message in the mailbox memory.
    b. Uses a doorbell register to generate an SCP interrupt.
2. SCP receives the interrupt then:
    a. Reads the message from the mailbox memory.
    b. Clears the doorbell interrupt.
3. SCP acts based on the message. SCP might also send a callback response, using the same operation but in the opposite direction.

In a system with self-managed devices or subsystems capable of directly requesting SCP to take actions the messaging capability would be required to be extended to allow communication between these agents and the SCP.

Power Policy Units

Power Policy Units (PPU) are specialized hardware components used to abstract low level control of power domains away from the SCP firmware. The SCP firmware makes only high-level power domain policy decisions and programs them into the PPU.

The number and location of PPUs depends upon the topology of the design. A minimum of one PPU is located within the always-on domain, to provide a first level of system wake capability.

Other PPUs can be distributed around the SoC as described in 7.2.3 *Distributed PPUs*.

See 6.5.1 *Power Policy Unit* for more details on the PPU.

See 7.2 *Power Control Integration* for more details on the integration of PPUs.

Sensor Control

The SCP is anticipated to be able to access on-chip process, voltage, and temperature sensor information either through a dedicated peripheral, or using the SoC interconnect.

Additional Peripherals

Additional peripherals can be included which are private to the SCP. Typically, these are always-on domain peripherals providing wake-up functionality.

Peripheral Access

In general, SCP peripherals are privately mapped for security reasons. Two important exceptions, mapped in both SCP and AP address spaces, are identified in the example of Figure 6.2:

- **Primary system counter:** AP software must be able to access the system counter as a requirement of the Generic Timer specification.
- **Messaging Interface:** Shared access is required to facilitate the messaging mechanism described in 6.4.1 .

For SCP peripherals that are outside of the always-on subsystem, such as distributed PPUs, the SCP might support a physically private peripheral extension port or rely on shared interconnect resources. In case of shared interconnect, the SoC integrator must consider security controls on access to these peripherals by other agents.

System Access

The SCP is anticipated to have access to the wider SoC resources, including peripherals and memory, using the shared system interconnect.

Access to SoC resources allows the SCP to perform actions as part of power control sequences. This could include configuration of components and save and restore functions. For example, the configuration of interconnect and memory controller components.

Limiting this access, except where necessary, is not recommended as it restricts the tasks that the SCP can perform.

6.5 Power Management Infrastructure Components

6.5.1 Power Policy Unit

This section gives an overview of the Power Policy Unit (PPU). Complete details of the PPU can be found in the *ARM Power Policy Unit Architecture Specification* [6].

The PPU is a standard component for abstracting software-controlled power domain policy down to low-level hardware control signaling. In a typical arrangement, one PPU is used to control each power-gated domain.

The SCP firmware can program the power policy of a PPU. This policy can be either a static power mode, or a range of modes that the PPU can transition between dynamically. This dynamic behavior is based on activity indicators from component LPIs without the need for further SCP programming. This enables hardware autonomous modes, such as dynamic retention, that can be entered and exited transparently to software. This provides responsive power control enabling components to be in the lowest power state possible, while maintaining functionality, with only policy level control from the SCP.

Figure 6.4 shows the PPU interfaces.

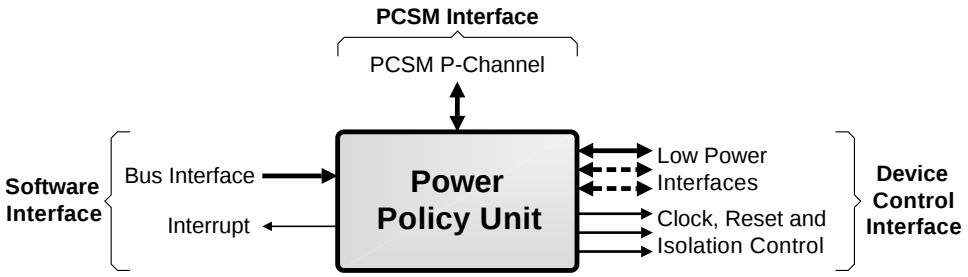


Figure 6.4: Power Policy Unit interfaces

The PPU interfaces are:

- **Software Interface**: A bus interface for programming, for example AMBA APB, and an interrupt that is used by the SCP for PPU configuration and policy control.
- **Power Control State Machine (PCSM) Interface**: An LPI to communicate power state changes to the PCSM that controls implementation and technology specific aspects of power control such as power switch and memory retention control.
- **Device Control Interface**: Low-level control for components within the power domain. It includes:
  - One or more LPI, depending on the needs of the components in the power domain.
  - Device controls, including clock enables, resets, and isolation controls.

Figure 6.5 shows how these interfaces are connected.

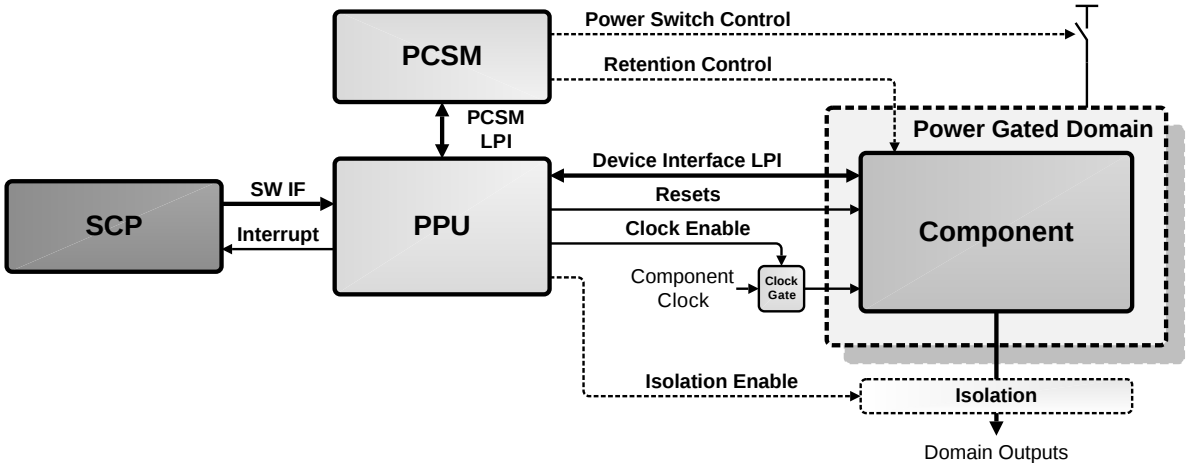


Figure 6.5: PPU integration example

A power-gated domain can contain more than one component which means there can be multiple LPIs. Dependent on the clock and reset domains of the components in the power domain there can also be multiple resets and clock enables.

The dotted lines show control signal connections which are not normally present in the RTL but will be added as part of a synthesis flow using UPF or similar means.

For more details on PPU integration see 7.2 *Power Control Integration*.

Power Control State Machine (PCSM)

Low level power control details such as power switch control or control signals for logic or RAM retention can be technology and cell library specific. To avoid modification of the core PPU function it interfaces to an implementation dependent power control state machine (PCSM) that controls these elements. This allows the PPU to be a generic and re-usable standard component.

The power control state machine is controlled from the PPU with a P-Channel LPI interface. The PCSM converts the P-Channel power mode requests to implementation dependent controls.

Reset Control

The PPU provides resets for the power domain. This ensures that the relevant resets are applied to maintain the correct component state when entering and exiting power modes.

The PPU has multiple reset outputs that are used in different power modes dependent on the reset action required.

For example, there are separate resets for retention and non-retention components. When a domain is in a retention mode, the retention registers must not be reset, since the retained state would be lost. However, non-retention registers do need to be reset.

There can also be differences between warm and power-on resets. On a warm reset some state might be required to be preserved, such as for debug or RAS purposes.

Clock Control

The PPU provides clock gating controls for the power domain. This ensures that clock inputs can be gated as required to maintain safe and correct behavior when entering and exiting power modes.

The PPU provides multiple clock enables for use in different power modes. For example, when the off state of a power domain is emulated for debug purposes. In this case, certain clocks need to remain enabled to allow debug access to the component.

NOTE: The PPU controls the clocks to manage power mode requirements, not activity based high-level clock gating. High-level clock gating is managed by a clock controller, for more details see 6.5.2 *Clock Controller* and 7.1 *Clock Control Integration*.

Isolation Control

The PPU provides isolation cell controls for the power domain. These controls are used to ensure that no floating values are propagated when the domain is powered-off.

The PPU provides multiple isolation controls for use in different power modes. For example, when the off state of a domain is emulated for debug purposes. In such a case, certain isolation cells might not be enabled to allow debug access to a component while the remaining isolation cells are enabled corresponding to the functional behavior of the domain.

PPU Policy Support

The PPU supports two power domain mode groups, power modes and operating modes.

Power Modes

The PPU supports all the PCSA defined power modes as specified in Table 6.1.

Not all PPUs are required to support all modes, so power mode support is design time configurable.

Operating Modes

Operating modes represent configurations of the standard power modes, or the power domain in general. The meaning of each operating mode is specific to one or more components within the domain.

Operating mode transitions occur when in the ON power mode, however the operating mode can maintain context in certain other power modes, for example, power modes with retention.

Some examples of uses for operating modes are:

- To enable multiple RAM configurations:
  - For example, resizing caches while a component is active, to save leakage power by powering off some RAM instances.
- Thread management within multi-thread processor cores.
  - Ensures correct thread management as interrupts for a logically powered-off thread are only available to the power control infrastructure outside of the processor.
- Configuration and access control management, for example, to enable save/restore operations.

Operating modes and the expected use models are specified in the *ARM Power Policy Unit Architecture Specification* [6].

Emulated Power Modes

To enable components to be debugged through power-off the PPU supports emulated power modes.

When a power mode is emulated the PPU completes all the normal control sequences except for the communication with the PCSM. This means, for example, that power switches are not turned off.

Some parts of the design can emulate power-off by asserting the appropriate resets causing a loss of state and functionality. Other parts of the design are required for debug access, or contain debug state, so remain functional with resets de-asserted. The PPU supports different resets to provide this capability.

6.5.2 Clock Controller

The clock controller is used to provide high-level clock gating for components in a clock domain that have either Q-Channel LPI clock gating support, or AXI LPI clock gating support according to the restrictions outlined in 6.2.3 .

NOTE: The use of AXI LPI is deprecated by the PCSA and must only be used for interfacing with legacy components.

Figure 6.6 shows the clock controller interfaces.

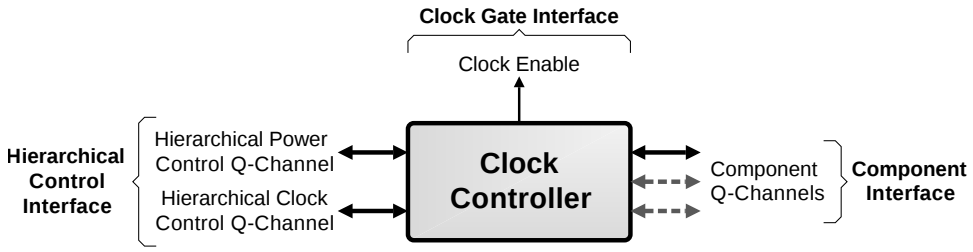


Figure 6.6: Clock controller interfaces

The clock controller interfaces are:

- **Clock Gate Interface**: This is a clock enable signal to control a clock gate.
- **Component Interface**: Consisting of one or more Q-Channel interfaces, depending on the needs of the domain.
- **Hierarchical Control Interface**: This consists of a power control and clock control hierarchical Q-Channels to allow control over the clock controller from higher level components.

The clock controller combines clock control LPIs from multiple components to manage a single clock domain. It uses the LPIs to ensure all components are in a quiescent state before the clock is gated. It also ensures the clock is running again before any component leaves the quiescent state.

The clock controller allows LPIs to be controlled asynchronously so that the synchronous clock enable from the clock controller can be connected to a clock gate at the root of the clock tree. This high-level clock gating can result in near zero dynamic power in idle scenarios.

This high-level clock gating control does not exclude any clock gating from being implemented inside components at a finer granularity.

NOTE: The clock controller does not include the clock gate but provides an enable that must be used synchronously.

For more details on the clock controller function see the *ARM CoreLink PCK-600 Power Control Kit Technical Reference Manual* [8].

For more details of clock controller integration see 7.1 *Clock Control Integration*.

6.5.3 Low Power Distributor

A PPU or Clock Controller are typically required to communicate with many components within a clock or power domain.

A Low Power Distributor (LPD) distributes a single Q-Channel or P-Channel to multiple channels.

It can either send all output channel requests simultaneously or sequence them one after another.

The LPD for both Q-Channel and P-Channel protocols is available as part of the CoreLink PCK-600 Power Control Kit. For more information see the *ARM CoreLink PCK-600 Power Control Kit Technical Reference Manual* [8].

6.5.4 Low Power Combiner

Some components, such as protocol bridges between power domains, are required to be controlled from the controllers of multiple power domains. To achieve this a Low Power Combiner (LPC) can be used to control a Q-Channel component from multiple Q-Channel controllers.

The LPC requests component Q-Channel quiescence when any controller Q-Channel becomes quiescent. It requests exit of component Q-Channel quiescence when all controller Q-Channels are non-quiescent. This allows, for example, a power domain bridge to be made quiescent when any of its associated power domains are entering a low power mode.

The LPC is available as part of the CoreLink PCK-600 Power Control Kit. For more information see the *ARM CoreLink PCK-600 Power Control Kit Technical Reference Manual* [8].

6.5.5 P-Channel to Q-Channel Convertor

A P-Channel controller can also be required to control Q-Channel components within a power domain.

The P-Channel to Q-Channel Convertor (P2Q) converts a P-Channel request to a Q-Channel request. It is configurable how the power modes map to either Q-Channel quiescent or running states.

The P2Q is available as part of the CoreLink PCK-600 Power Control Kit. For more information see the *ARM CoreLink PCK-600 Power Control Kit Technical Reference Manual* [8].

Chapter 7
System Power Control Integration

This chapter describes the system integration of power management features in the following sections:

- 7.1 *Clock Control Integration*.
- 7.2 *Power Control Integration*.
- 7.3 *Reset Control Integration*.

7.1 Clock Control Integration

This section describes:

- Levels of clock gating.
- Clock gate placement to achieve maximum effect.
- Integration approaches to achieve effective and efficient clock gating implementation.

A clock tree is built of clock buffers that propagate the clock over the physical distance between the clock source (a clock input or a PLL) and the clock endpoints (registers or RAMs). Additional buffers are added on branches of the tree to balance the clock arrival time at each synchronous endpoint. This helps to achieve timing closure by allowing the maximum time for logic propagation between synchronous elements.

Clock switching in the clock tree buffers consumes dynamic power regardless of whether any useful work is being done at the endpoint. Therefore, to make a power efficient system it is required to gate the maximum amount of the clock tree possible in addition to the endpoints.

Without high-level clock gating, clock tree power dominates dynamic power consumption in idle scenarios.

7.1.1 Clock Gating Levels

There can be multiple levels of clock gating within a system. This specification uses the following classification:

- **Low-Level**: Clock gates inserted automatically by synthesis tools.
- **Mid-Level**: Instantiated clock gating, typically synchronously controlled, within components.
- **High-Level**: Instantiated gating of entire clock domains.

These clock gating levels are all complementary and should be implemented regardless of the presence of other levels within the structure. Each level has benefits with different levels of power saving and temporal granularity.

Figure 7.1 provides an illustration of these clock gating levels.

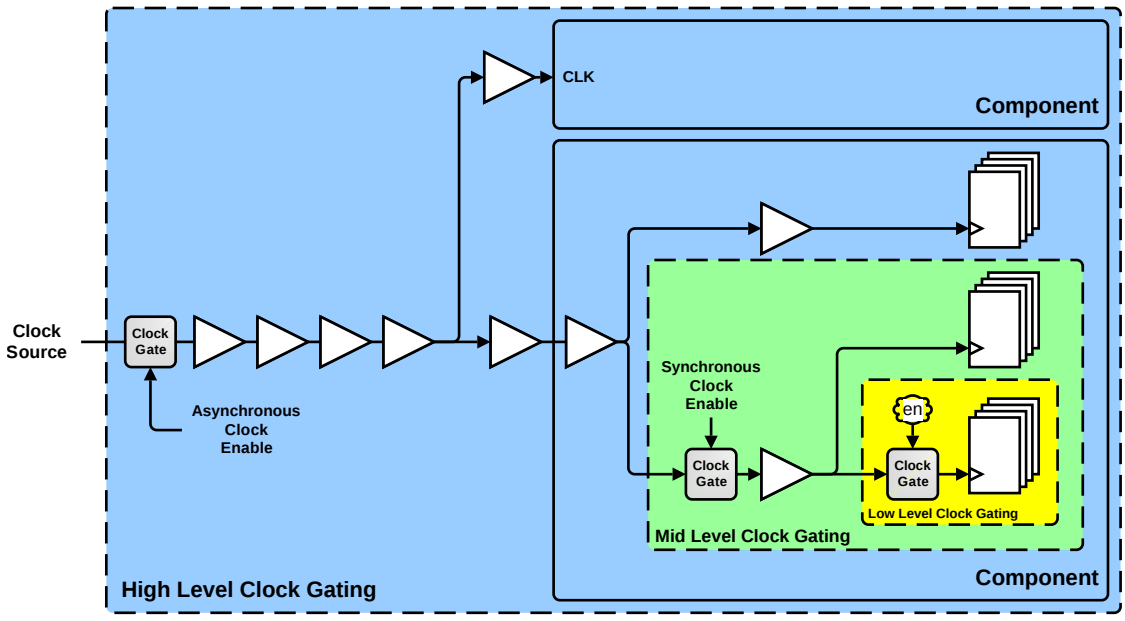


Figure 7.1: Clock gating hierarchy

Low-level Clock Gating

Low-level clock gates are inserted by synthesis tools. They are placed directly in front of a group of flip-flops and replace the enable functionality therefore saving both area and power.

Low-level gates are very granular. In the ideal case an endpoint with a low-level clock gate is gated whenever the flip-flop is not being updated. However, they gate only the flip-flops and not the clock tree and additionally cannot be placed on the entire design for reasons explained below.

The enables for these clock gates are inferred from the functional enables, expressed in the RTL, for the flip-flops. A clock gate is inserted where a functional enable is shared by a minimum number of flip-flops. This minimum number is set by the synthesis constraints and is typically based on the power-area break-even point of the re-structuring.

Figure 7.2 and Figure 7.3 illustrate a standard flip-flop enable, using multiplexor feedback, and how this is restructured by synthesis tools to add a clock gate for the flip-flops.

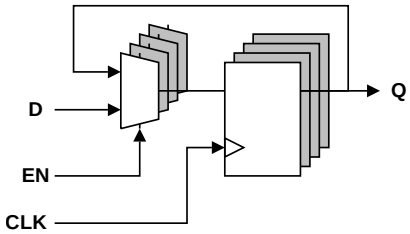


Figure 7.2: Standard flip-flop enable

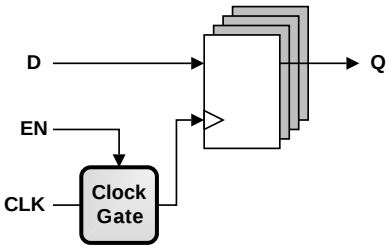


Figure 7.3: Low-level clock gated flip-flop

However, some flip-flops do not have a low-level clock gate inserted by the synthesis tool because:

- No enable is present.
- The enable is not recognized by the synthesis tool.
  - This can be due to the enable being too logically complex, or not structured in a way that the synthesis tool can easily recognize.
- Enable terms might be too logically complex to create within a required timing window.
- The number of flip-flops controlled by an enable is less than the minimum threshold set by the synthesis constraints.

Therefore, low-level clock gating, while very important, is not enough to produce a fully power efficient system due to the lack of clock-gating coverage on sub-sets of flip-flops and most of the clock tree.

Mid-level Clock Gating

These clock gates are instantiated by the designer in the RTL to gate complete blocks of logic that are idle during periods of operation.

The enables are controlled by the surrounding logic and are typically enabled and disabled synchronously in a single clock cycle to be transparent to functional operation. As these enables need to meet synchronous timing requirements the clock gates are still placed low in the clock tree. This placement avoids skew between the logic and the clock gate that shrinks the enable timing window.

While gating larger parts of the design, these mid-level clock gates do not gate large portions of the overall clock tree. The exact amount of gating depends on the timing requirements of the design and the logical complexity of the enables. Additional gating is required to make a fully power efficient system.

High-level Clock-Gating

These clock gates are inserted per-clock domain and are placed ideally at the root of the clock tree. This placement results in near zero dynamic power when a clock domain is idle.

The clock latency between the clock root and the endpoints is typically greater than the timing window allowed for a synchronous signal to propagate. Therefore, enable control signaling must be treated as asynchronous to the clock endpoint.

This creates problems for dynamic clock gating due to the delay between:

- The device being idle, and the clock being gated.
- A request for the device to be active and the clock becoming available.

Consequently, there needs to be a method to provide guarantees associated with clock supply and removal to ensure the correct operation of components. This is described in 7.1.2 *High-Level Clock Gating Methodology*.

Although this technique provides the maximum saving, the granularity with which it can be applied is much lower, so it is important that it is used in combination with the other gating levels.

7.1.2 High-Level Clock Gating Methodology

This section describes a methodology for managing high-level clock gating using the Q-Channel low-power interface to provide functional guarantees. Multiple components can be combined into a single clock domain with a common high-level clock gate.

Figure 7.4 provides an example where a Cache Coherent Interconnect (CCI) and Network Interconnect (NIC) share a common high-level clock gate:

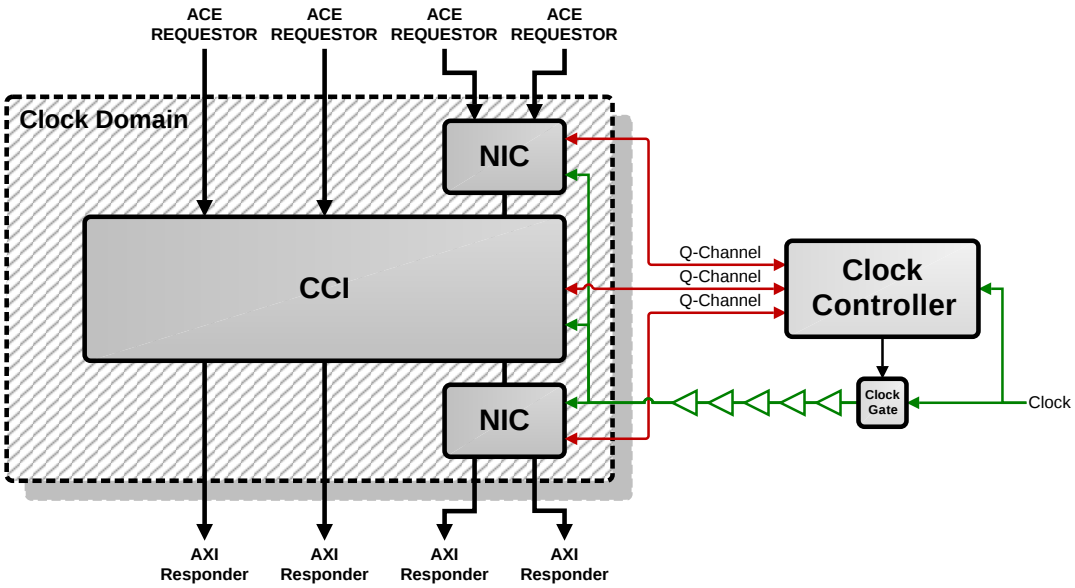


Figure 7.4: High-level clock gating for multiple interconnect components

7.1.3 High-Level Clock Domain Selection

When implementing high-level clock gating it is recommended to partition clock domains at either asynchronous boundaries or at other natural divergences in the clock tree such as a power domain boundary.

Figure 7.5 shows a simple example of components partitioned into two clock domains separated by an asynchronous boundary.

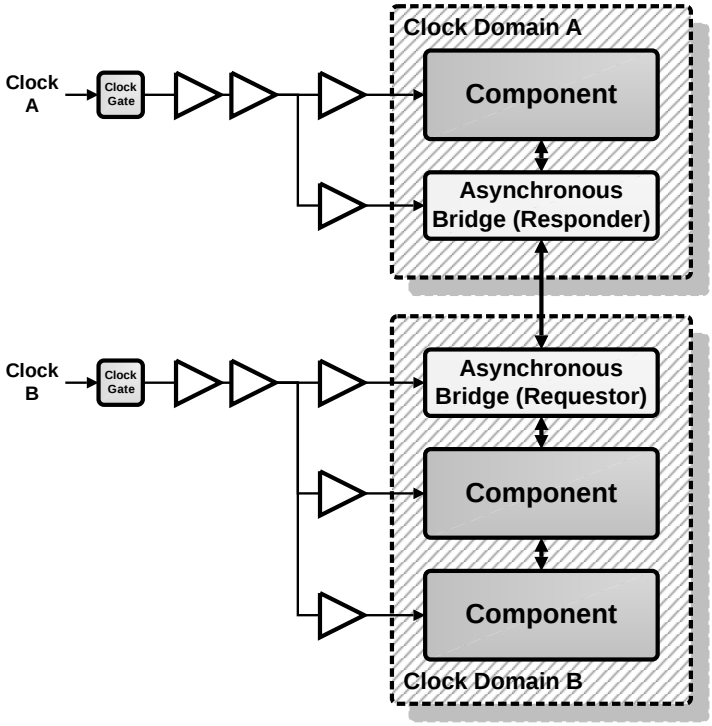


Figure 7.5: Clock domains example

The two clock sources can be gated independently at their clock root. During active periods, any lower level clock gating implemented within components provides complementary benefit.

This arrangement is most effective when all the components within a clock domain have similar high-level activity requirements, such as a flow of bus transactions through them.

Conversely, if components are combined into a clock domain with highly mismatched activity requirements then the effectiveness of the high-level clock gating is reduced. A trade-off exists between the number of clock domains implemented, any latency introduced by additional asynchronous crossings, and the benefit from high-level clock gating.

It is also possible to implement high-level clock gating for multiple synchronous clock domains with communication between them. However, this creates non-common clock insertion paths that places pressure on clock tree balancing. The consequences of are that:

- The high-level clock gate is placed lower in the tree to increase the common clock path.
  - As shown by the red arrow in Figure 7.6.
  - This reduces the effect on timing but reduces the effectiveness of the clock gating.
- The increased balancing effort leads to higher clock buffer power.
  - This increases overall power when the paths are not clock gated.

– This increases the number of buffers on each path so increases overall power.

These circumstances also occur if a free-running clock domain has synchronous interfaces to a high-level clock gated domain.

Figure 7.6 shows an example of the second case.

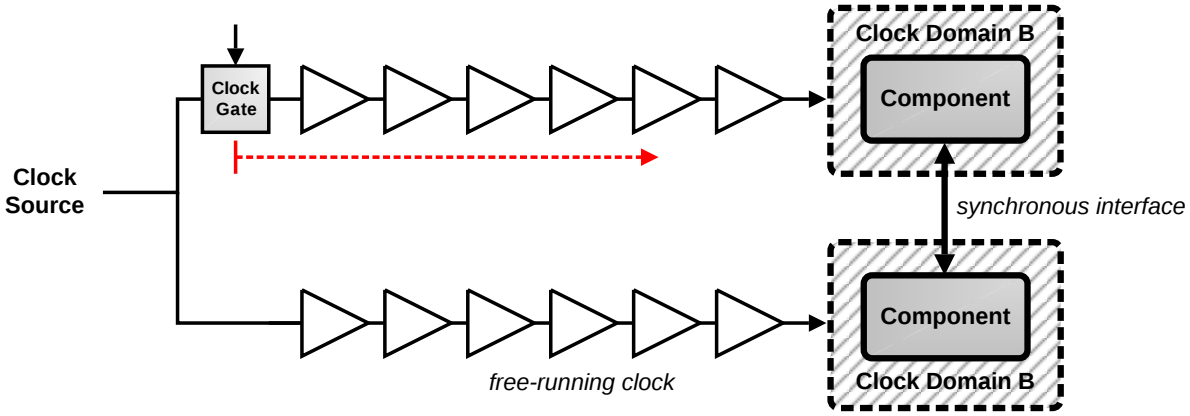


Figure 7.6: Clock insertion pressure

In Figure 7.6 the dotted arrow shows the pressure on clock gate placement caused by the difficulty of implementing balanced clock paths for both the high-level clock gated and free-running clock domains because of the synchronous interfacing between them.

7.1.4 Clock Gating Control Integration

High-level clock gating is implemented using a clock controller component for each clock domain. The clock controller supports clock gating for multiple components with one Q-Channel interface to each component.

This type of clock gating is supported by most Arm components. Most components use a Q-Channel. Some earlier components use an AXI LPI, use of this interface with a Q-Channel clock controller is restricted. The restrictions on the use of AXI LPI with a Q-Channel controller are detailed in 6.2.3 .

NOTE: The use of AXI LPI is deprecated by the PCSA and must only be used for interfacing with legacy components.

The clock controller is described in 6.5.2 *Clock Controller*.

Component design considerations for Q-Channel clock gating are detailed in 8.2 *Component High-Level Clock Gating*.

Clock Controller Connections

Figure 7.7 shows the clock gating arrangement for a single component within a clock domain.

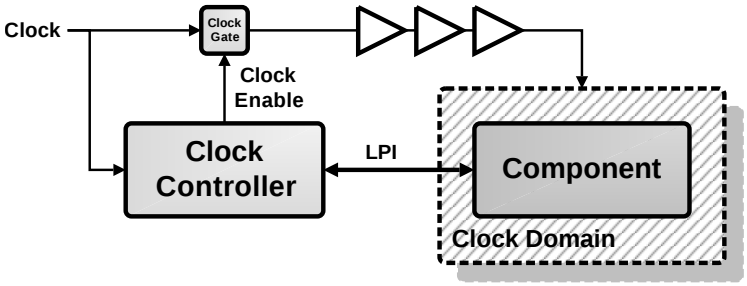


Figure 7.7: Clock gating with a single component

There can be many components within a clock domain. Each component can have a clock control Q-Channel. These are combined and managed by the clock controller.

Figure 7.8 shows this clock gating arrangement for multiple components within a clock domain.

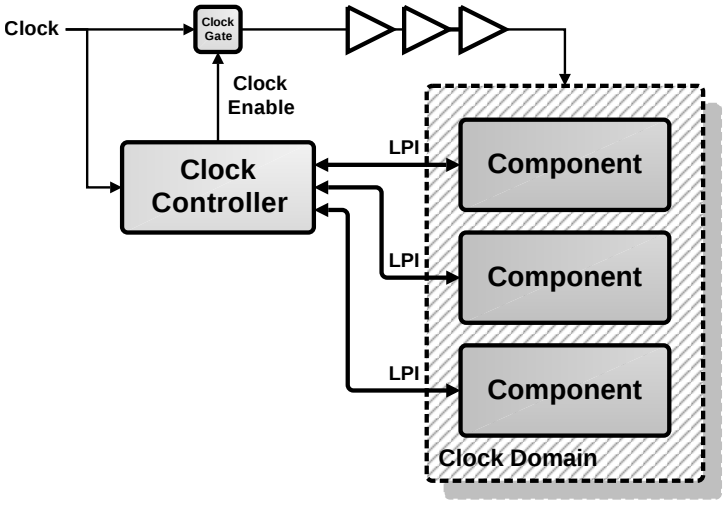


Figure 7.8: Clock gating with multiple components

If a component has multiple clock domains it requires multiple clock control Q-Channels, one for each clock domain.

Figure 7.9 shows an example of a clock gating arrangement including a component with multiple clock domains. An example of this could be a component which has separate functional logic and bus interface clock domains that can be gated independently. The high-level gated bus interface clock could be shared with many components, such as an interconnect, while the functional clock might be dedicated to that component.

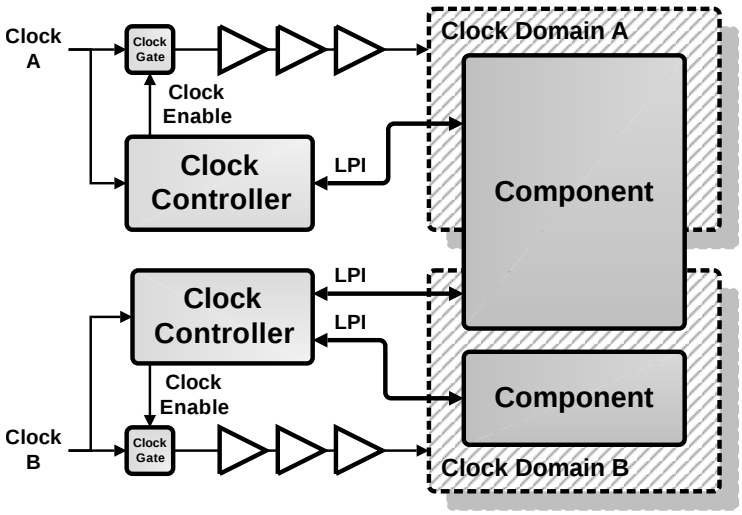


Figure 7.9: Clock gating a component with multiple clock domains

Clock Controller Placement

The aim is to place the clock controller at the root of the clock tree, using the free-running source clock as its input. The clock controller provides a synchronous clock gate enable output. This synchronous enable control is required so that the clock controller can ensure the clock availability guarantees of the Q-Channel protocol.

This placement of the clock controller means the Q-Channel interface between it and the components must be treated asynchronously to allow for the clock tree insertion delay between them. The Q-Channel handshake with the component provides a robust asynchronous interface to facilitate this.

Figure 7.10 shows an example with two components in one clock domain that can be used for discussion of physical design constraints.

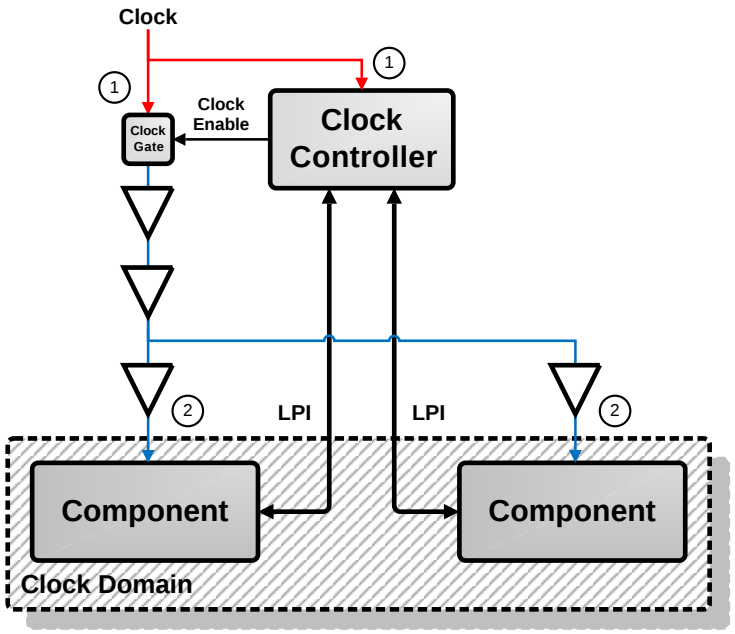


Figure 7.10: Clock gating constraints

To achieve the maximum benefit from high-level clock gating the following is recommended:

- The clock controller and the high-level clock gate clock input are placed in one clock endpoint balancing group. This is marked as group 1 in Figure 7.10.
- The components of the clock domain are placed in a second clock endpoint balancing group. This is marked as group 2 in Figure 7.10.
- The two groups are balanced separately.
- All LPI signals, between the two balancing groups, are treated as asynchronous in both directions.

Clock Domain Crossing

Clock domain crossing requires both synchronization of signals and careful management of clock speed differences. For a commonly used protocol, such as a bus interface, a reusable domain bridge component that addresses these problems, such as CoreLink ADB-400, will normally be used.

A domain bridge typically consists of:

- A responder interface that receives transactions and passes them over an asynchronous boundary interface.
- A requestor interface that receives the transactions from the asynchronous boundary interface, converts them back to the appropriate protocol and re-transmits them to downstream components.

Figure 7.11 shows the clock control arrangement between clock domains connected with an asynchronous domain bridge.

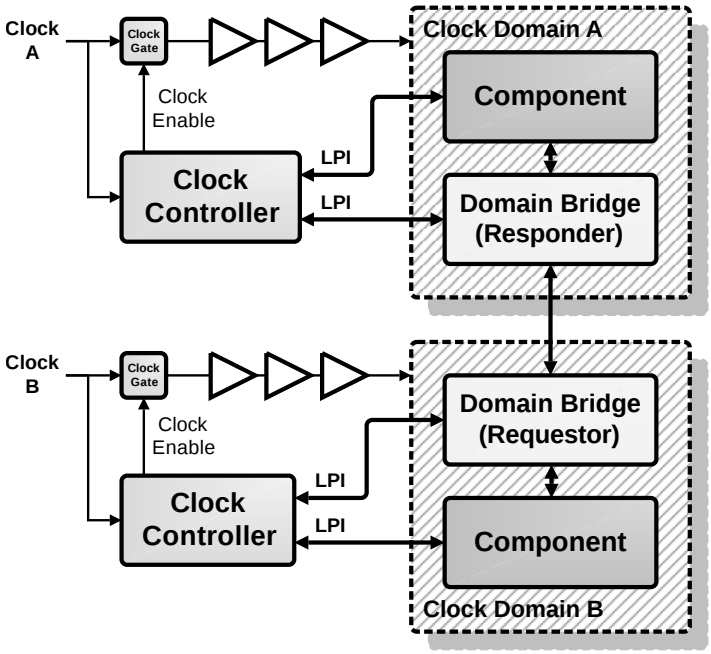


Figure 7.11: Clock domain crossing with an asynchronous domain bridge

The asynchronous domain bridge might be split into two halves. When the domain bridge is used at a voltage or power domain boundary one half can be placed in each domain. For more information see 7.2.7 *Voltage and Power Domain Boundaries*.

To enable high-level clock gating there must be a wake-up signal, asserted when a transaction enters at one side of the domain bridge, that forms a contribution to the **QACTIVE** at the opposite side of the bridge. The **QACTIVE** signal is driven HIGH whenever one side of the bridge has a transaction pending for the other side of the bridge.

Figure 7.12 shows this arrangement for **QACTIVE** generation at one side of an asynchronous domain bridge.

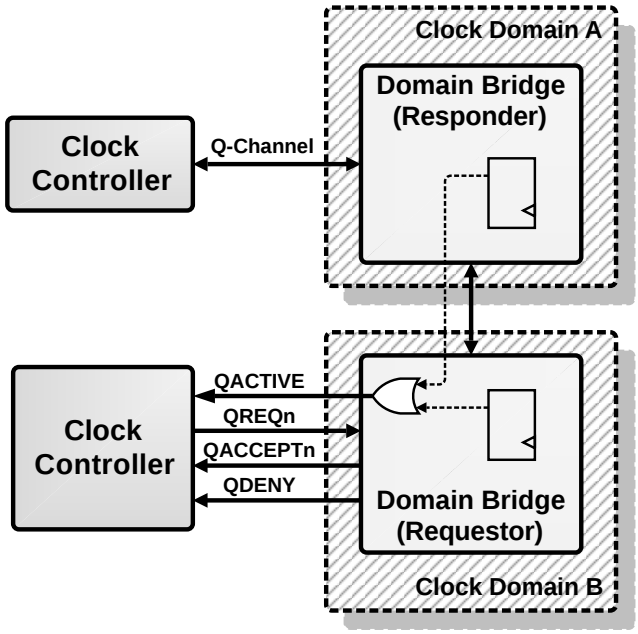


Figure 7.12: QACTIVE generation between asynchronous domain bridge slices

An asynchronous domain bridge can be implemented with:

- **QACTIVE**-only clock control support, such as CoreLink ADB-400 up to release 2.
- A complete Q-Channel for each clock domain, such as CoreLink ADB-400 release 3.

A domain bridge implementing a full Q-Channel for clock control at each side supports high-level clock gating as a standalone component without dependencies on other components.

In the case of a domain bridge with **QACTIVE**-only interfaces, the **QACTIVE** signals must not be used directly for any clock control. High-level clock gating can only be supported if the domain bridge is connected directly to a component with full Q-Channel clock gating support. The connected component is responsible for managing the transaction flow according to the clock guarantees provided by the Q-Channel handshake.

In the case of a connected component not supporting Q-Channel based high-level clock gating then the clock at that side of the bridge must be provided by the system whenever the bridge is required to be operable.

Figure 7.13 shows the detail of the connections in case of **QACTIVE**-only asynchronous domain bridge with components supporting a full clock control Q-Channel at both sides.

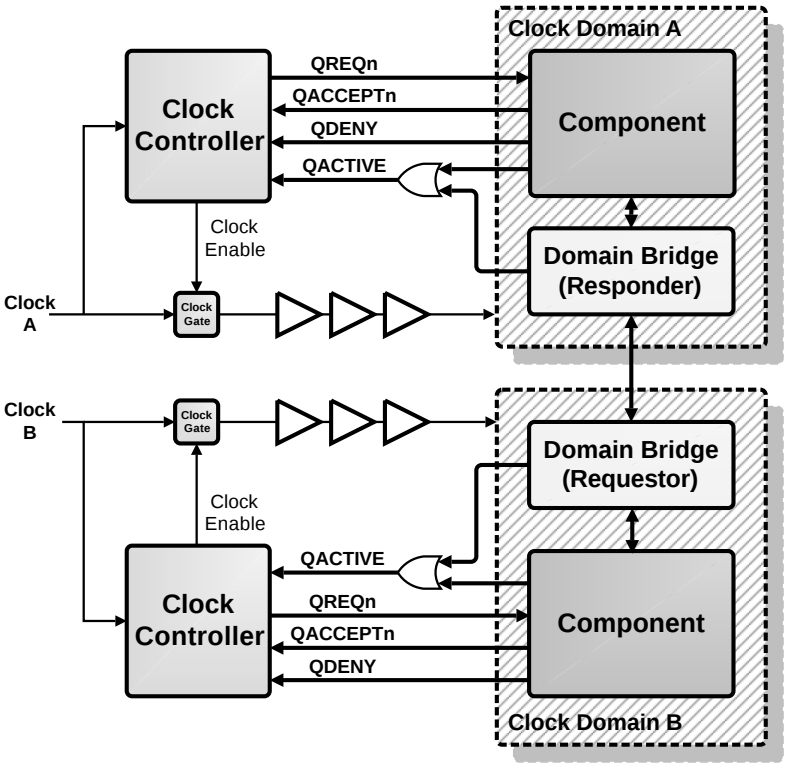


Figure 7.13: Integration of QACTIVE-only asynchronous domain bridge

Clock Domain Scope

In some cases, components without any explicit support can be incorporated into a high-level clock gated domain. This can be achieved when a component only requires clock activity during periods when another component with LPI clock gating support will guarantee clock supply is maintained.

Typically, these conditions are satisfied only when there is a dependency between the operation of the component guaranteeing clock supply and the component reliant on that guarantee.

Figure 7.14 shows an example of this concept.

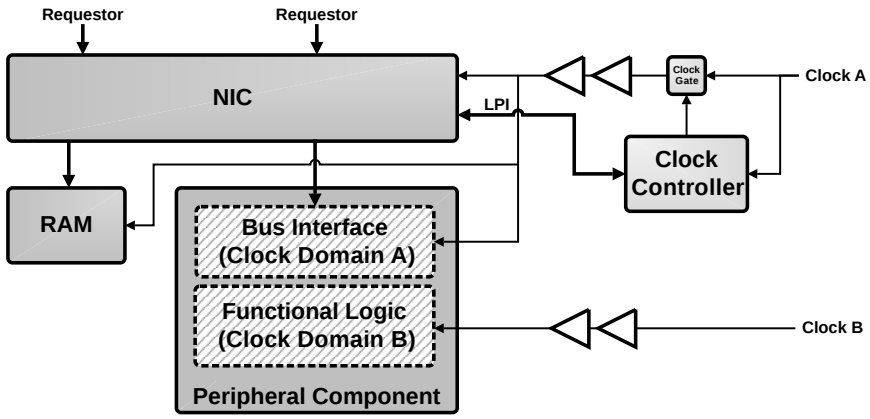


Figure 7.14: Clock domain scope example

In Figure 7.14 the NIC interconnect uses its LPI to guarantee clock supply whenever it has outstanding transactions at its interfaces.

A clock domain, using clock A, contains the NIC, the RAM, and the bus interface portion of the peripheral component. Neither the RAM nor the bus interface portion of the peripheral component provide LPI support for clock gating.

A clock domain, using clock B, contains the functional logic portion of the peripheral component. Clock B might be gated by an independent clock controller, but this is not considered further in this example.

If the RAM and the bus interface portion of the peripheral component only require a clock while bus transactions are outstanding, then the clock controller can be used to gate clock A.

This technique can be applied, with careful analysis of topology dependencies and component clock requirements, to interconnect components without LPI clock gating support downstream of components with LPI clock gating support.

NOTE: In all cases a detailed analysis of the clocking requirements for each component must be carried out. For the RAM component example this is likely to be straightforward. However, the peripheral component could, for example, rely on bus interface clock activity for capturing status changes, such as interrupts, from its functional logic and therefore would not be suitable.

7.1.4.1 Clock Controller Reset

The clock controller must be reset in one of the following conditions to ensure the Q-Channel protocol is maintained between controller and components.

- All Q-Channels are in the *Q_STOPPED* state.
- The reset is also applied to it and all connected Q-Channel components at the same time.
  - This ensures that any consequent protocol violation cannot be observed by the connected components.

7.2 Power Control Integration

The power mode of a component is controlled by the SCP using a PPU. As the PPU is in a different voltage or power domain, and typically in a different clock domain, the power control interfaces must support asynchronous operation. More information on the PPU can be found 6.5.1 *Power Policy Unit* and [6].

The Arm LPIs provide robust clock domain crossing semantics for asynchronous interfacing. The interface handshake protocols provide guarantees to ensure components are in safe state for power mode transitions.

The clock control LPI and the power control LPI must be separate interfaces on a component. They are required to be separate as they have independent control points.

Once an LPI request has been accepted by all controlled components in a power domain the PPU communicates with the PCSM. The PCSM manages the physical changes of the domain such as power switches, retention controls and voltage levels.

The following sections give an overview of how components are connected to create managed power domains.

7.2.1 Power Domain Wake Events

Signals from various parts of the system can be used as stimulus to wake a power domain. Such signals should be level sensitive and stay asserted until the request is met or while the requirement is still applicable.

Due to the nature of these requests in most cases they are required to cross an asynchronous clock domain boundary or a large physical distance. In these cases, it is impractical to balance the clock to ensure the signal is captured correctly.

Additionally, when a SoC is in a low power state it is likely that clocks will be gated, and if the wake-up has a transient nature, such as a pulse, it means that even if it acts as a stimulus to re-enable clocks it will be unlikely the pulse will still be present when the receiver is ready to capture the signal.

Sources such as pulse-based interrupts therefore cannot be used as wake-up signals or must be conditioned so they are level sensitive.

7.2.2 Hardware Abstraction with Power Policy Units

For the SCP firmware to directly manage many low-level signals would be time consuming and expensive in terms of processor runtime, interrupt inputs and system control outputs. This might lead to the requirement to run the SCP core at a higher frequency or to choose a higher performing core in a power sensitive area. It will also adversely impact the response to power mode changes due to interrupt latency and conflicting processing tasks.

To manage this scenario the PCSA makes use of hardware abstraction by including power policy units (PPU). SCP firmware makes a policy decision for the power domain based on the requirements of the system but delegates the low-level management to a PPU. Each domain has its own PPU.

The PPU also introduces a level of autonomy where the hardware can enter and exit low-power modes without needing to involve the SCP firmware.

This autonomous operation is especially important for power modes that require fast response times. An example of this is logic or RAM retention states, where no state is lost, and the operation can save power and be transparent to software. This must add minimal latency, so performance is not adversely affected. With the PPU programmed for dynamic operation it can request entry and exit from allowed power states based on the **PACTIVE** status.

Figure 7.15 shows an example power-gated domain arrangement using a PPU to interface with the power domain components.

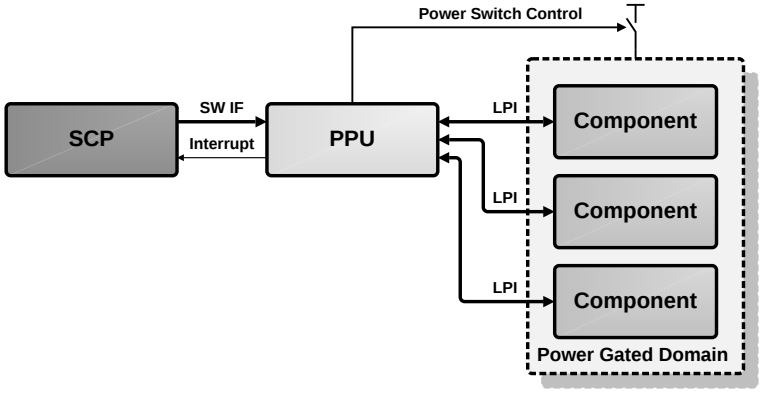


Figure 7.15: Power-gated domain control with a PPU

The PPU manages the physical changes of the domain such as power switches, retention state, and voltages levels without needing to interrupt the SCP firmware.

The PPU software registers must be accessible to SCP firmware. The PPU must be placed in a power domain that is relative always-on to the power domain it is controlling.

An overview of the PPU and its interfaces is given in 6.5.1 *Power Policy Unit*. Complete details of the PPU can be found in the *ARM Power Policy Unit Architecture Specification* [6].

PPU placement is discussed further in 7.2.3 *Distributed PPUs*.

7.2.3 Distributed PPUs

The placement of PPUs is an important consideration. The simplest approach is to place all the PPUs with the SCP in the always-on domain. With all PPUs in one hierarchy, integration concerns such as address mapping, interconnect, clocking and resetting are straightforward. However, there are several reasons why this might not be the best choice.

Firstly, in a complex system there might be many component LPIs and it might not be desirable or practical to connect large amounts of wires across the SoC to the always-on domain.

Secondly, to enable the PPU to react quickly, where fast entry to and exit from power modes is required, it is advantageous to use a clock that is close to the power domain clock frequency. It is not desirable to route high speed clocks to multiple subsystems to achieve this.

These concerns can be resolved by distributing the PPUs throughout the system placing them close to the power domains they control. From a communication perspective, the distributed PPUs can then be programmed through re-use of the existing SoC interconnect.

This placement means that the clock sources supplied to the power domain under control can be used locally to maximize PPU responsiveness. Even though the PPU might use the same clock source as the component it is controlling, as it is in a different power, or even voltage, domain the interface is still required to be treated as asynchronous to minimize timing closure costs.

The distributed approach can also assist the encapsulation of functionality into subsystem building blocks, easing the construction of large systems.

Figure 7.16 shows an example of this type of arrangement.

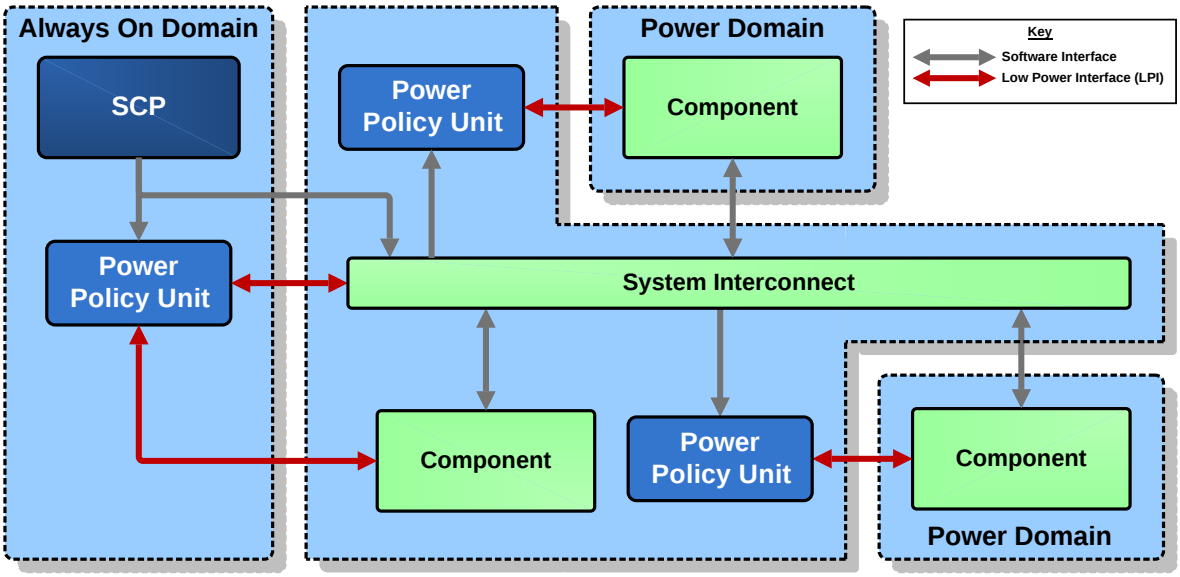


Figure 7.16: Distributed PPU overview

This approach requires a hierarchical arrangement of power domains, such that the PPU for a domain must be powered-on, and be accessible by the SCP, before further sub-domains can be powered-on. There must be a minimum of one PPU in the always-on domain to power-on the first power-gated domain. However, more than one PPU might be needed in the always-on domain depending on the power domain hierarchy of the system.

7.2.4 System of Systems

Some SoC subsystems might themselves be complex entities and there might be reasons for such a subsystem to have its own local control processor (LCP). With some minor differences, the subsystem power management structure reflects that of the SoC, therefore such an arrangement is called a *system of systems*.

Figure 7.17 shows an example of a system of systems.

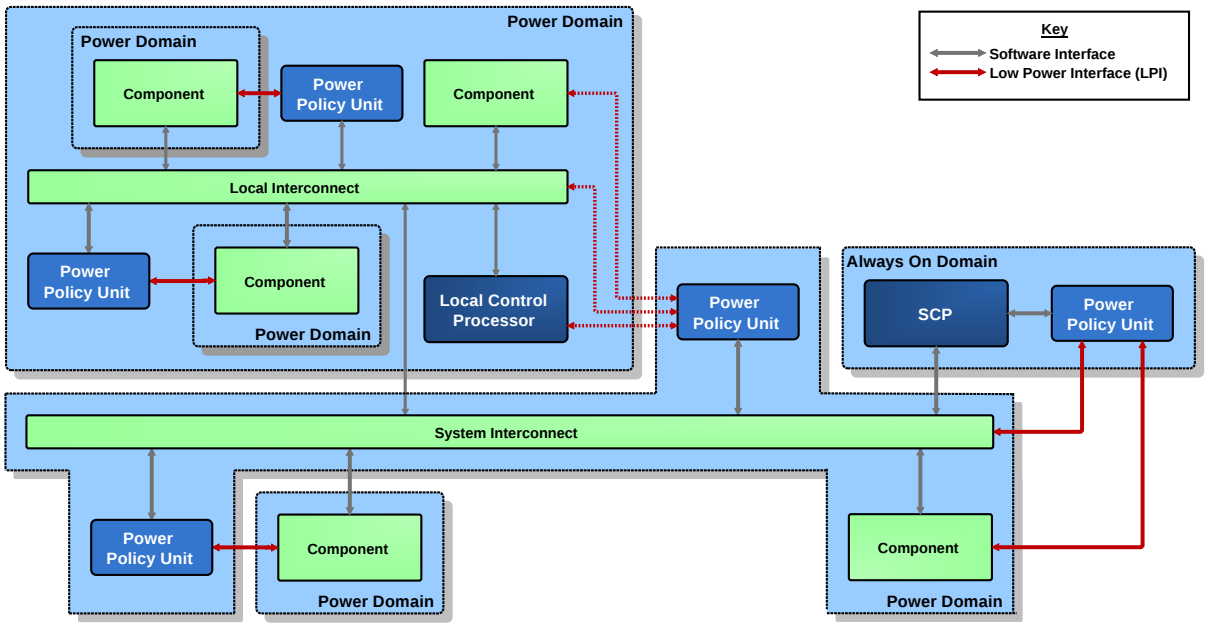


Figure 7.17: System of systems overview

The subsystem in Figure 7.17 has its own LCP. This approach can be applicable to both devices with self-managed capabilities and as a method to scale the SCP capability by offloading local tasks into subsystems.

The LCP of the subsystem in Figure 7.17 is different from the SCP. It is relatively always-on to the power domains it is controlling, therefore it can be powered-off when its sub-domains are powered off. To allow the LCP to be powered off, it requires an LPI interface to a PPU in a relatively always-on domain. This PPU is controlled by the SCP and might also control other entities within the same power domain level as the LCP, shown by the dashed connections, or the quiescence of these components might be managed by the LCP with an additional PPU.

PPUs are used within the subsystem to control sub-domains.

Subsystems of this complexity will also typically require a messaging capability to facilitate firmware to firmware communication between LCPs and the SCP. This capability is described in 6.4.1 .

This structure can reduce integration complexity if the subsystem is delivered pre-verified with existing firmware. From a power control perspective only the top level LPI and any messaging capability needs to be integrated.

7.2.5 Component Integration Layer

Some components might have interfaces that do not directly match those of the PPU. In such cases an integration layer approach can be used to adapt the interface between the PPU and the controlled components.

For example, this can be used to adapt a component without LPI support, or with additional power control signals which need to be managed through power mode transitions.

Figure 7.18 shows miscellaneous signaling combined into an LPI in a component integration layer.

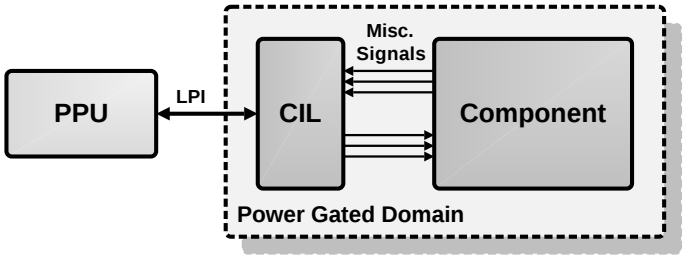


Figure 7.18: Single component integration layer example

7.2.6 Voltage and Power-Gated Domain Clock Gating

Before a domain can be powered-off or put into full retention it must be clock-gated externally. This prevents the following:

- Parasitic switching in the domain causing current draw that can slow the power on process.
- Corruption of retention registers.

This is managed by the PPU and is separate from any dynamic high-level clock gating control.

For power modes where some part of the component is operational, such as FUNC_RET and MEM_OFF, the external clock is still required so cannot be gated. Internal clock gating within the component is required for the parts that are off or in retention.

The PPU is in a different clock domain to the power domain it is controlling. Therefore, the clock enable from the PPU for the controlled power domain's clocks are required to be synchronized to the clock being gated. There might be several clock inputs to the controlled power domain, so the single clock enable from the PPU must be synchronized separately for each clock gate.

Figure 7.19 shows and example of this arrangement.

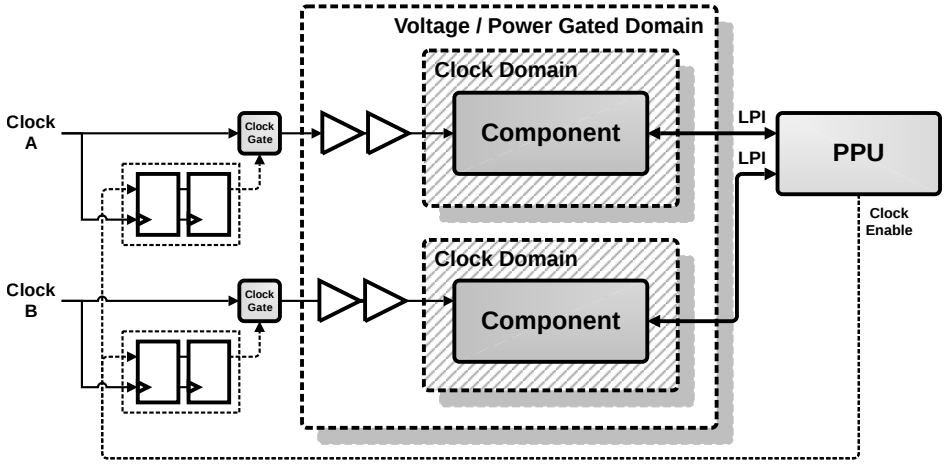


Figure 7.19: Clock gating for voltage and power domains

Depending on the topology, the PPU clock gate might be combined with the clock gate used for dynamic high-level clock gating, or it could be a separate clock gate. In either scenario the clock controller must be reset when the domain is reset to prevent violations in the clock Q-Channel, see 7.1.4.1 *Clock Controller Reset*.

Figure 7.20 shows an example of this implementation where the power control and clock control clock gates are separate. This is typically done when the clock controller is inside the power domain.

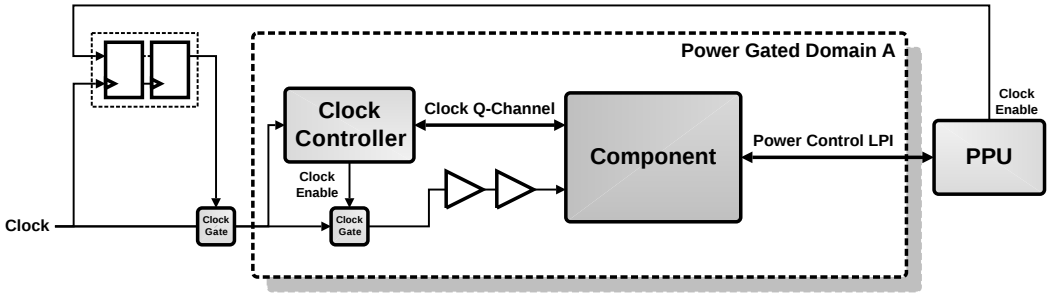


Figure 7.20: Separate clock gates for high-Level clock gating and power control example

Figure 7.21 shows an example of this implementation where the power control and clock control clock gates are shared. The clock controller must be external to the power domain in this case.

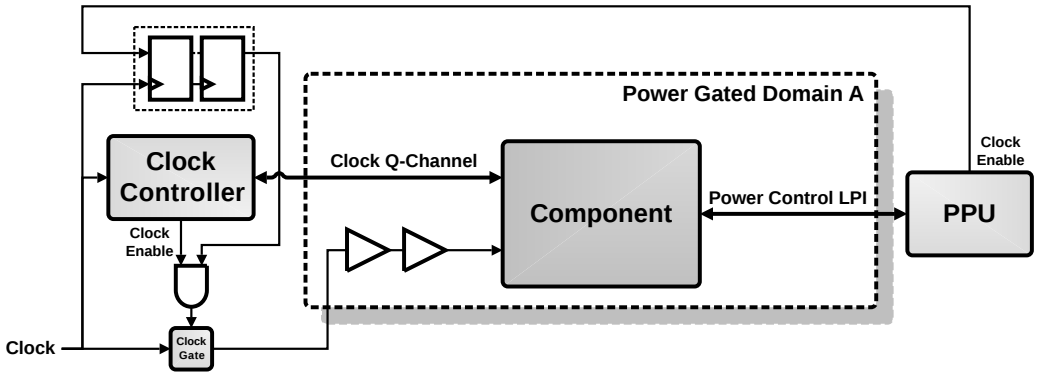


Figure 7.21: Shared clock gate for high-Level clock gating and power control example

7.2.7 Voltage and Power Domain Boundaries

The crossings between voltage domains must be asynchronous. The crossings between power domains can be synchronous or asynchronous.

The crossing of a voltage or asynchronous power domain boundary is achieved with the use of a domain bridge for the required protocol.

For the boundary between voltage domains, or asynchronous power domains, the domain bridge is split, with one half in each domain.

For the boundary between synchronous power domains, the domain bridge can be split, with one half in each power domain, or the entire bridge can be within one power domain.

The following sections describe the crossing of these domains and the effect on the power control and high-level clock control.

Voltage Domain and Asynchronous Power Domain Boundaries

When crossing between voltage domains, level shifters must be placed between the two domains to manage the difference in voltage levels between the two voltage supplies. Closing timing across such a boundary is difficult because of the considerable number of voltage-supply cross corners that need to be analyzed. Therefore, it is treated as an asynchronous interface for all signals.

When crossing between asynchronous power domains, the crossing needs to be treated as an asynchronous interface for all signals because of the asynchronous nature of the clocks in the two power domains.

A domain bridge used to cross between voltage or asynchronous power domains is split into separate requestor and responder interface components. Each of these uses a separate asynchronous clock. These two components are then instantiated either side of the domain boundary. This means the clock domains are constrained to their respective domains and isolation and, in the case of voltage domains, level shifter cells can be added at an easily identified hierarchy level.

Figure 7.22 shows an example of this structure for a voltage domain. An asynchronous power domain is the same, but without the requirement for level-shifters.

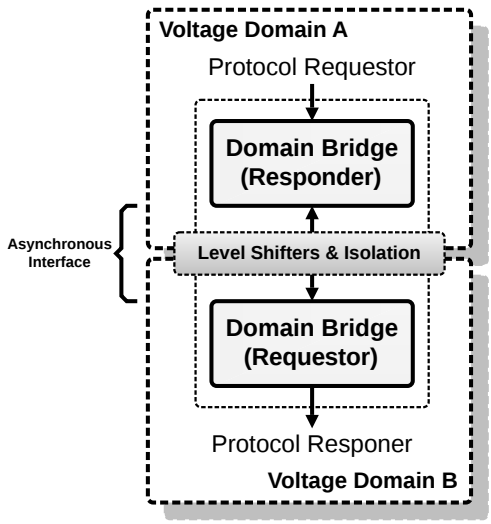


Figure 7.22: Voltage domain crossing domain bridge structure

Figure 7.23 shows an example voltage, or asynchronous power domain crossing using a domain bridge with clock and power control connections.

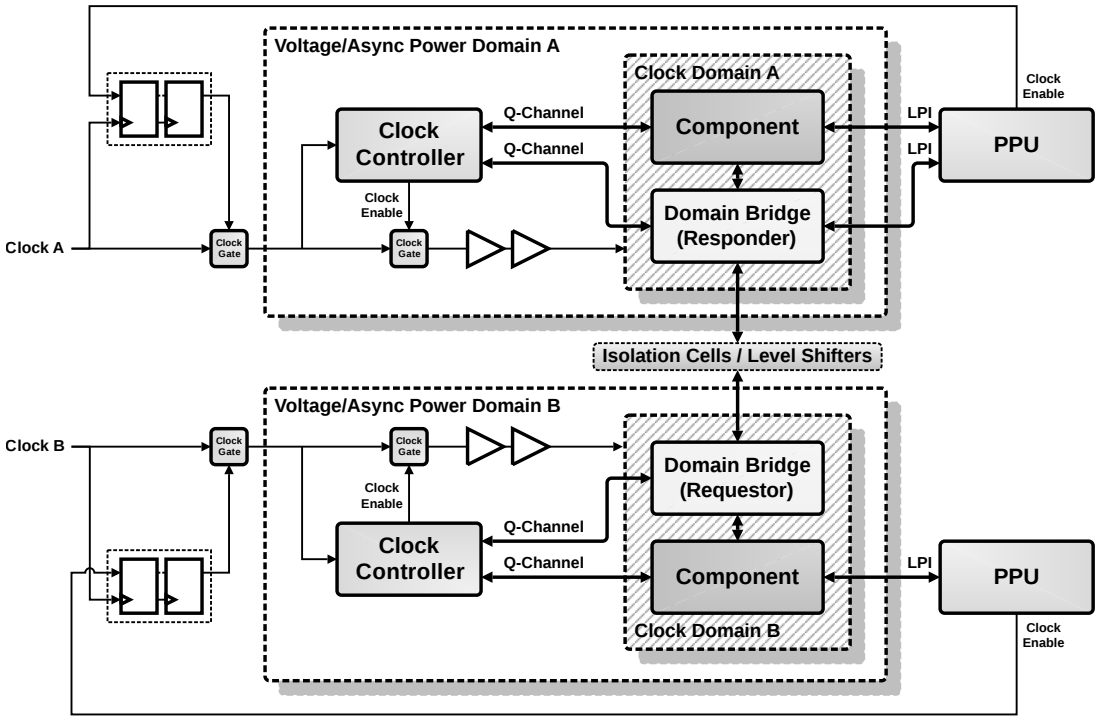


Figure 7.23: Voltage or asynchronous-power domain boundary example

The connections in Figure 7.23 for the domain bridge power control assumes that domain B is RAON, therefore domain A controls the domain bridge quiescent state as it will be the first to be powered-off and the last to be powered-on. For more information see 7.2.7 *Power Control for Domain Bridges*.

When the domain bridge power control LPI is quiescent, any domain bridge component that is powered-on must still respond to clock control Q-Channel requests. This is required because:

- The clock is required to be requested to complete subsequent power mode transition.
- The clock Q-Channel might be required to return to a quiescent state to ensure correct interfacing to system components.
- Another component in the voltage or power-gated domain might require the clock and the clock controller will handshake with all components in the domain.

In Figure 7.23 the domain bridge power control LPI is shown connected to the responder portion of the domain bridge in the upstream power domain. The location in this example reflects its location on the CoreLink ADB-400. However, the location of the LPI on the bridge is arbitrary. For more information see 7.2.7 *Power Control for Domain Bridges*.

From a high-level clock control integration perspective, this example has no dependencies. The clock control implementation is achieved with a simple combination of clock control Q-Channels from the components within each domain.

Power Control for Domain Bridges

When the power, or reset, to the two sides of the domain bridge is managed independently there must be means to ensure that the bridge is in a safe quiescent mode before either side is powered-off or reset. The domain bridge must only exit from the quiescent mode once both sides are powered-on.

The first concern is the correct operation of the bridge itself, as the internal state is required to be managed such that the bridge does not malfunction when only half of the bridge is reset or powered-off. Another concern is access control of transactions attempting to enter the bridge.

The domain bridge is entered and exited from the quiescent mode with a single power control LPI. Where this LPI is controlled from is dependent on the relationship between the two domains. This section will assume the domain bridge power control LPI is a Q-Channel.

NOTE: Multiple power control LPIs on a domain-bridge are possible, but a detailed description is beyond the scope of this document.

Domain Relationships

Where the domain bridge is controlled from depends upon the relationships between the domains it spans. The possible relationships are as follows:

- **Always-On (AON)**: One domain is always powered, whilst the other can power-off.
- **Relatively Always-On (RAON)**: Both can power-off, but one always powers-off first, and powers-on last.
  - An example of this is the system logic which is relatively always-on to a processor cluster.
- **Independent**: Both can power-off, and the order in which they power-on and off is not fixed.
  - An example of this is a debug domain, whose state might depend upon if debug is enabled independent of the state of other system components.

These domain relationships can also affect the required placement of isolations cells.

Domain Bridge Connections

For domains with an AON or RAON relationship, where one domain will always be powered-off before the other, the domain bridge power control Q-Channel is controlled from the domain which powers-on last and powers-off first.

Figure 7.24 shows the control of a domain bridge power control Q-Channel when the domains have a RAON relationship. The level shifters (LS) are only required when this domain bridge spans a voltage domain boundary.

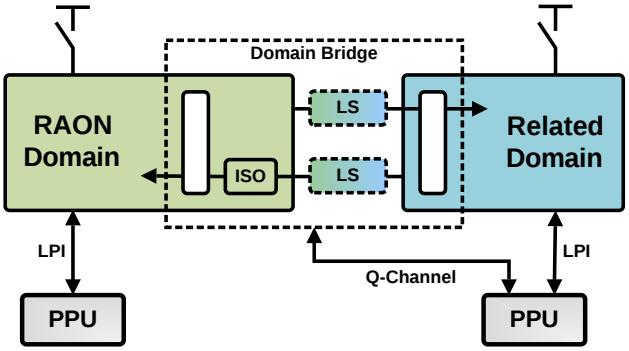


Figure 7.24: Domain bridge control for a RAON domain

For domains with an independent relationship the domain bridge power control Q-Channel must be controlled by the domain that powers-off first and powers-on last. However, as the domains are independent this could be either

domain. This means there needs to be a component outside of both the controlled domains that can manage the bridge control requirements between the two domain PPUs. This component is a Low Power Combiner (LPC). For more information see 6.5.4 *Low Power Combiner*.

Figure 7.25 shows the control of a domain bridge power control Q-Channel when the domains are independent. The level shifters (LS) are only required when the domain bridge is crossing voltage domains.

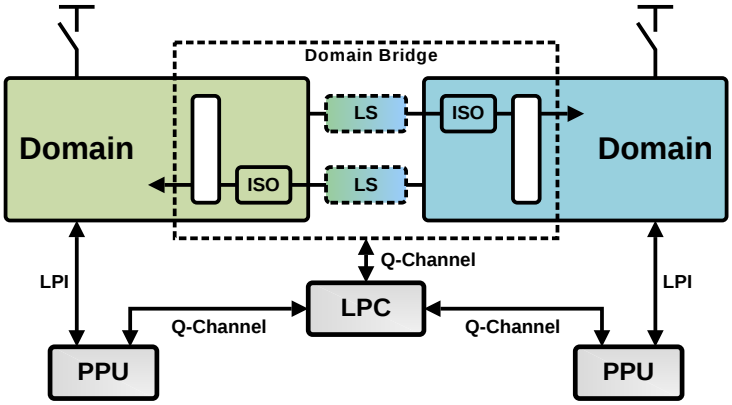


Figure 7.25: Domain bridge control for independent domains

Synchronous Power Domain Boundaries

Figure 7.26 shows an example arrangement where the clocks to two power domains are synchronous.

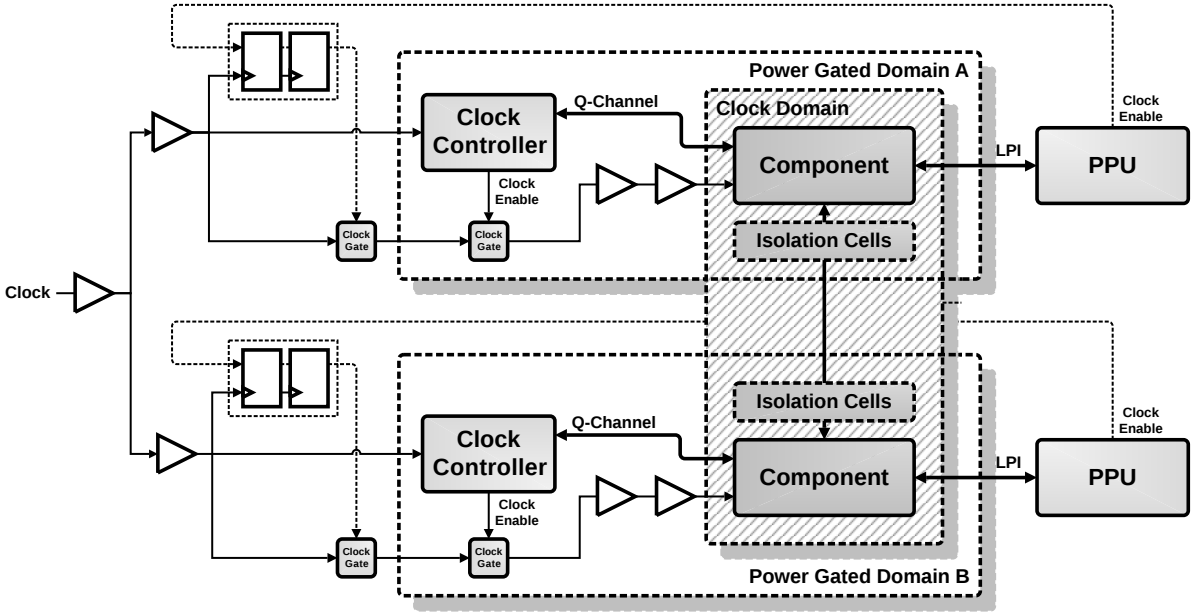


Figure 7.26: Synchronous power-gated domain boundary example

In this arrangement, the clock control for the two domains must be split. When one domain is powered-off it will not be able to respond to a clock controller Q-Channel handshake. This split in the clock tree can impact the overall effectiveness as only the non-common parts of the clock tree can be gated.

In Figure 7.26 the power domain clock gating strategy of Figure 7.20 is chosen. The clock controller is within the domain as otherwise isolation application can result in a protocol violation on the Q-Channel.

The clock controller can be placed outside of the domain provided it is reset along with the power domain. Alternatively, in this scenario the clock controller, see 6.5.2 *Clock Controller*, supports a hierarchical power control Q-Channel that can be used to ensure the downstream Q-Channel is quiescent before the domain isolation is asserted.

The clock controller also supports a hierarchical clock control Q-Channel that can be used to add an additional clock controller to gate the clock at the root of the clock tree in addition to at the power domain boundaries.

7.2.8 Access Control

It is desirable to allow part of the system to be in a non-functional powered-off or retention mode even though a bus transaction might arrive from a part of the system that is powered-on. This type of capability is referred to in this specification as *access control*.

Applications for access control include:

- To ensure a controlled power cycle for a resource, by ensuring all accesses to it are complete before power-off and providing responses during power-off.
- To preserve the availability of a resource while allowing it to opportunistically enter a low-power mode. The resource is woken automatically when an access is attempted.
- Preventing access to a resource from selected interfaces during post reset or run time configuration.

Each component with power or clock control capabilities must manage its interfaces correctly to provide access control.

For the purposes of explanation, it is convenient to consider a standalone access control gate component. Figure 7.27 shows a simple example.

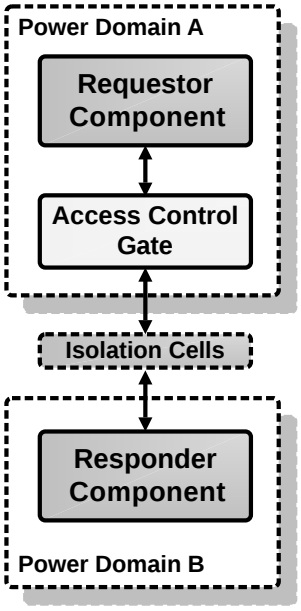


Figure 7.27: Access control example

In Figure 7.27 the access control gate component, when in a gated state, breaks the path between the requestor and responder components. This allows the responder component in power domain B to be placed into an inaccessible state in support of the access control applications previously outlined.

The power domain arrangement of Figure 7.27 shows the access control gate in the requester component power domain. This location enables the generation of wake-up requests and any required sequential responses to the requester component when the responder component power domain is off. In this case, all the access control gate functionality is implemented in one place.

In some combinations of bus protocols and response models, isolation values alone can provide a static solution for the required response. This allows the access control functionality to be implemented in the downstream power domain. However, any wake-up logic must always be implemented in the upstream power domain.

Access control requires two capabilities. The first is capability to safely enter and exit the gated state. The second capability is the response to the arrival of transactions during the gated state.

When the gate function is enabled, by either software control or a low-power interface request, the following occurs:

- Any outstanding transactions are completed before entering the gated state. The safe handling of any transactions arriving during the transition to gated state is protocol and implementation specific.
- Once in the gated state, transaction attempts are either stalled or receive a response from the access control component according to the implemented response model.
- The access control component can be requested to exit the gated state at any time that the downstream resources are available. After exiting the gated state, the access control component is transparent.

The following interface management response models are considered useful and are chosen according to the application:

- **Stall and wake:** A transaction arriving at a gated access control component is initially stalled. A wake-up signal is asserted indicating that the downstream power domain must be made available. Once available the access control is un-gated and the transaction progresses as normal. This model is used to preserve the availability of a resource while allowing it to opportunistically enter a low-power mode.
- **Error:** The access control component generates an error response to any arriving transaction. There is no request to change any power domain mode. This might be used for debug purposes in case an access is attempted in error.
- **Accept and ignore:** The transaction is accepted by the access control component, to prevent blocking the interface, but is not forwarded and has no effect on any power domain mode. This might be used with some types of broadcast, trace or monitoring traffic.

7.2.9 Isolation and Reset Control Considerations

When a component changes power mode isolation can be applied or removed, and resets can be asserted. Isolation control and resets are controlled asynchronously. Therefore, component outputs need to be conditioned so any potential asynchronous change does not affect functioning domains to which they communicate.

Isolation cells provide constant values on the outputs of powered-off or retained power domains to prevent unknown values propagating to operating domains. The isolation control is applied asynchronously to remove timing constraints on the control signal propagation.

Reset assertion is asynchronous to ensure all registers are correctly reset regardless of the availability of system clocks.

To prevent outputs changing during reset and isolation assertion and release it is strongly recommended to apply the following rule:

- Reset value = Idle value = Isolation value.

Care must be taken, especially with reset values, as an output might be driven by multiple registers through combinatorial logic. In this case it must be ensured that all the outputs of source registers can be reset without changing their individual outputs to ensure no glitches are produced in the downstream logic.

In some cases, the idle value might not be the same as the isolation and reset value due to domain retention. Care must be taken to assess if an asynchronous change on such signals will cause issues in other domains. Such effects can often be subtle, such as address signals used directly to decode to selects, or signals that might cause metastability in downstream registers.

Only those signals that can affect system behavior need the considerations described below.

In all cases care should be taken as RTL simulation where isolation is not implemented can differ from real behavior where isolation is implemented. Power aware simulations which consider these aspects are recommended.

Output Isolation and Retention

For retention modes, any difference in output and isolation values can be managed in two ways:

- Use of ‘clamp last’ isolation cells.
  - This solution is only valid for isolation scenarios.
  - These are special isolation cells that latch the value of an output and apply it as the isolation value. Such cells might not be available in all technology libraries.
- Capture the output value externally to the domain before applying isolation or reset and release isolation or reset before capturing the output value again.
  - Such behavior can easily be controlled using an LPI driven by the appropriate PPU.
  - This functionality is typically considered at the system level where the scope of power domains and the effect of output signals is known.

NOTE: When a domain is in retention, isolation must be enabled. Although register values are retained, typically register outputs are not driven, and any buffers between the register output and the domain boundary are still powered-off.

Output Isolation and PACTIVE

When a component enters the FULL_RET, LOGIC_RET or MEM_RET power modes, the corresponding **PACTIVE** bit for the power mode might be HIGH from the component but will be isolated LOW, as this is the reset value.

In this case it is required for the system to maintain the required level outside of the power domain containing the component, or there might be an incorrect indication to the power controller that the component no longer requires this power mode as a minimum. This can be achieved using the techniques described above.

Similar considerations might apply for component specific operating mode **PACTIVE** outputs.

NOTE: The **PACTIVE** associated with the power mode might not be HIGH when the component enters the associated power mode. The PPU might chose to request a transition to a power mode even though it was not the component required minimum, this could be due either to the PPU programming or a contribution to the **PACTIVE** from a source external to the component.

Component **PACTIVE** behavior during these power modes is discussed in 8.3.5 *PACTIVE and Isolation*.

Output Isolation and Non-default values

In the circumstance that an output idle value is not the same as the isolation value,

- Synchronize the isolation enable.
  - This solution is only valid for isolation scenarios.
  - This allows the timing path though the isolation enable to the endpoint register to be timed so it does not cause any setup and hold issues.
  - This scenario is non-optimal and should be avoided whenever possible as it places additional requirements and constraints on aspects of the design.
- Capture the output value externally to the domain before applying isolation and release isolation before capturing the output value again.
  - Such behavior can be controlled using an LPI driven by the appropriate PPU.
  - This functionality is typically considered at the system level where the scope of power domains and the effect of output signals is known.

7.3 Reset Control Integration

7.3.1 Reset Signals

There are two basic reset types that components might support.

- **Cold reset:** resets all logic in the component. Used as the power-on reset.
- **Warm reset:** resets most of the logic in the component. Typically, the extent of the reset is all functional logic. The logic excluded from a warm reset will vary with the type of component, but typically functions such as debug and RAS syndrome registers are excluded.

These reset types, when both present, are typically implemented as shown in Figure 7.28.

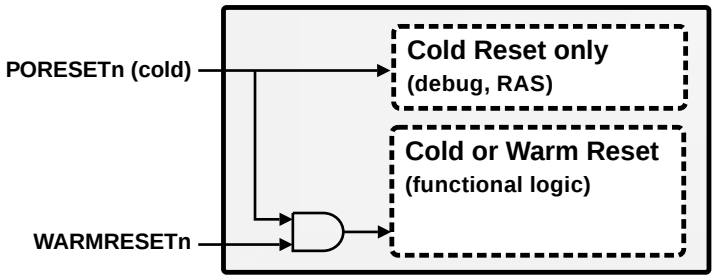


Figure 7.28: Cold and warm reset component implementation

In some cases, components will omit the internal combination of cold and warm reset signals and simply provide reset signals for each logical partition. The cold and warm reset domain signals can be easily formed outside of the component.

The reset domains of a SoC are largely aligned with the implemented power domain hierarchy. The component reset signals are aggregated into cold and warm reset domain signals at the power domain level. In practice, many components have reset signals per-clock domain, and these are aggregated into the power domain reset signals with appropriate synchronization.

Component design guidelines associated with the use of resets and power control logic are given in *Low-Power Interface Logic Reset*.

Most components have only cold reset signals, the remainder of this section outlines the reset signal requirements for components and subsystems with specific reset requirements.

Application Processors

The ARMv8-A architecture supports cold and warm resets of a PE. A warm reset excludes reset of the integrated debug logic that permits debugging across a reset of the PE logic.

The architecture provides means to request a PE warm reset using a reset management register. The required reset request sequence guarantees that the core is quiescent, and generating no bus transactions, when the reset is applied.

The external debug registers also provide a means to request a PE warm reset. It is implementation defined whether a core is reset when requested by this means. However, if the core is reset there is no guarantee that it is quiescent.

The use of warm resets without a quiescence guarantee is described in *Warm Resets*.

The ARMv8-A architecture also supports an external debug reset for debug domain logic. Reset requests for the debug domain logic are described in *CoreSight Subsystem*.

CoreSight Subsystem

Debug Reset Request describes how a debugger, using the Debug Port (DP), can request a warm reset of the debug domain logic. The debug and trace logic components typically have only one reset signal and so this is asserted for both cold and warm resets at system level.

Although the DP does not provide any control bits for requesting a system reset, the *ARM Debug Interface Architecture Specification, ADIv6.0* [9] describes that it is common for the physical interface to the debugger to include a system reset pin, **nSRST**, that is used to initiate a full system reset.

7.3.2 Reset Hierarchy

The PPU provides the following support for reset control.

- **Reset signals:** The PPU provides cold and warm reset output signals. The PPU also provides an additional warm reset output signal for the management of power domains implementing partial retention.
- **Power mode control:** The reset signals are managed appropriately by the PPU hardware for each supported power mode transition.
- **Warm reset:** The PPU supports a warm reset mode with support for both software and hardware initiated warm resets.

Each PPU controls the reset domain signals for the power domain it manages. These resets can then be used to build hierarchical reset control based on the logical power domain hierarchy.

Figure 7.29 shows a high-level reset hierarchy example.

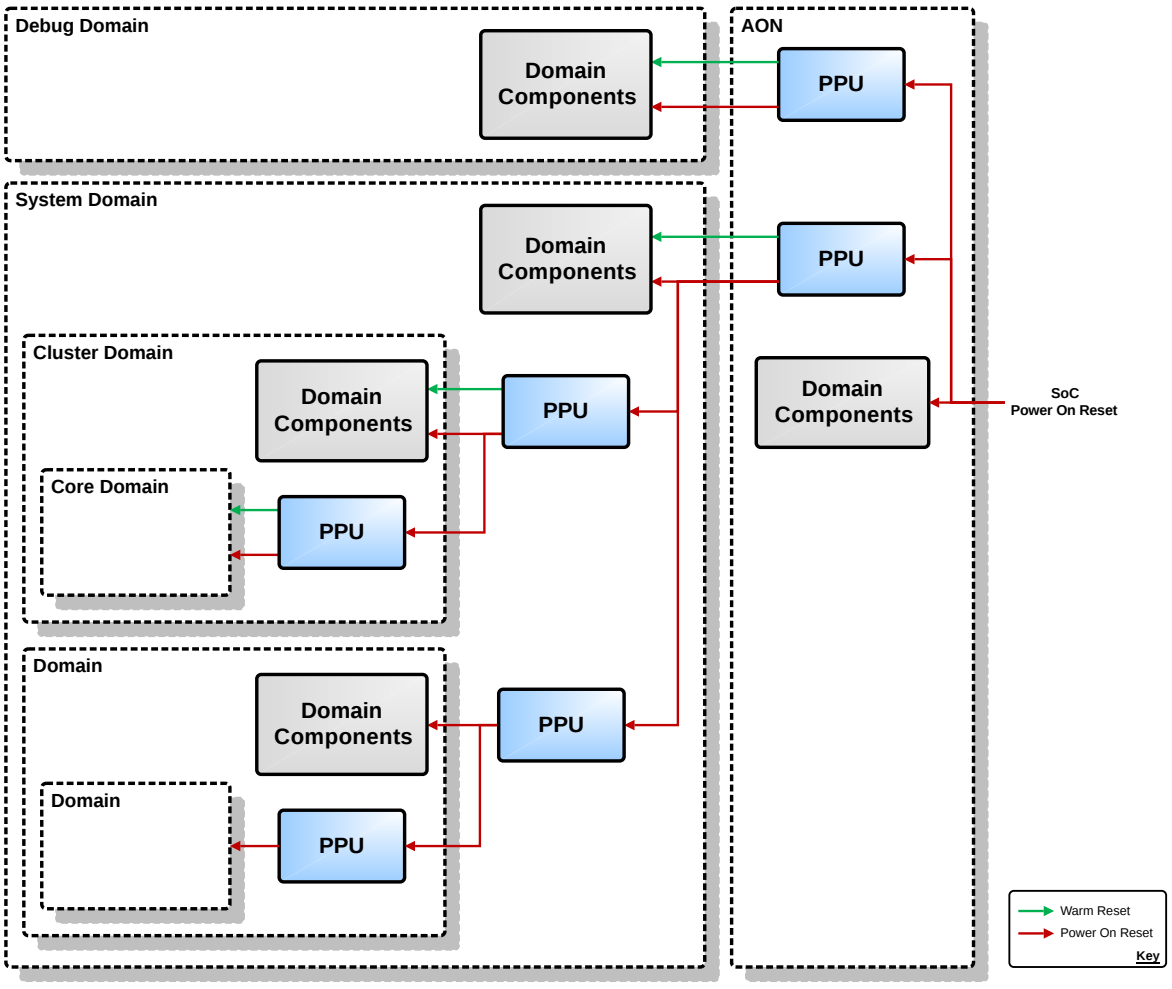


Figure 7.29: High-level reset hierarchy example

The reset hierarchy has its root in the always-on domain. At a SoC power-on, or other full system reset such as watchdog reset, the system reset is applied to the always-on domain, including any components and PPUs within the always-on domain.

The reset of the PPUs in the always-on domain will also place any powered-on domains they control into a cold reset state. This effect is hierarchical for powered-on domains through each level of PPU. On exit from system reset the PPUs in the always-on domain will place the power domains under control into their default power mode, managing resets accordingly. Typically, the default power mode for those power domains will be OFF.

As further domains in the power domain hierarchy are powered-on, any PPUs are released from reset and again the controlled power domains will be placed into their default power modes.

Each powered-on PPU then continues to manage the reset signals of the power domain it controls according to the programmed power management policy.

7.3.3 Reset Management

Power Mode Transitions

A critical requirement for entry into any power mode where the components of the domain become non-functional is to ensure quiescence. All outstanding interactions with other power domains, such as bus transactions, must have completed and the components must remain quiescent regardless of activity at their boundaries.

Where supported, the PPU device low-power interface can be used to ensure components are quiescent before entering power modes that assert resets. For components without low-power interface support then software means, or other system guarantees, are required to ensure the transition is safe.

Warm Resets

Warm resets without quiescence

As outlined in *Application Processors*, an application processor can request a safe self-warm reset with a guarantee of prior quiescence. However, warm resets are also commonly used as part of debug or system recovery operations. For example:

- A debugger can request a warm reset of the debug domain logic in attempt to re-connect with the target.
- A debugger can request a reset of a specific processor core.
- A management agent might request a warm reset of part of the system to attempt to recover RAS syndrome information.

The usage of warm resets for these purposes is generally without any guarantee of quiescence for the affected components. In the absence of specialized logic to isolate and mitigate the effect of incomplete transactions, these resets can then lead to a system lock-up requiring a full system reset. As such, these applications of warm resets have un-predictable outcomes and require careful management.

Unaligned reset domains

The required warm reset domain might not always align with power domain boundaries. This occurs when:

- The logic in the warm reset domain spans multiple power domains.
- The logic in the warm reset domain shares a power domain with other logic.

Debug infrastructure is one example where these cases can arise. Figure 7.30 shows an example arrangement where debug logic exists in both a primary debug power domain as well as within a system power domain alongside system components.

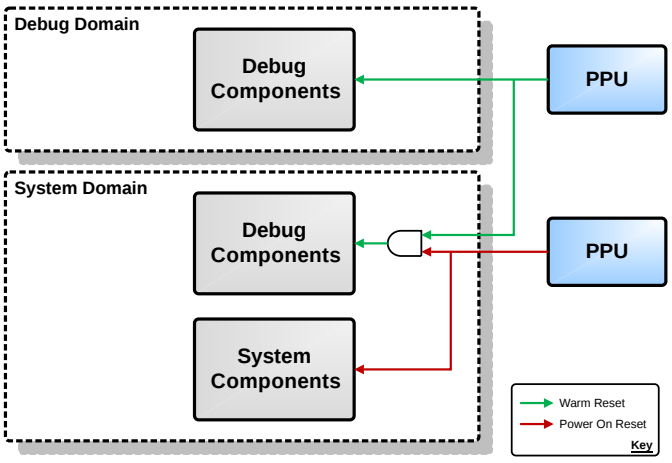


Figure 7.30: Example of warm reset domain to power domain misalignment

The arrangement in Figure 7.30 provides power-on reset for the debug logic, across the power domains, independent of their relative power sequencing. A warm reset of all available debug logic can be requested when the primary debug power domain is powered-on.

Initiating Warm Resets

The warm reset of a power domain can be initiated either by programming WARM_RST as the PPU policy or, with a P-Channel PPU, using a PACTIVE request. While many components only support cold reset signals, they can be connected at the power domain level to the PPU warm reset signal that is asserted on both warm and cold resets of the domain.

Where a specific warm reset domain is not required, the PPU WARM_RST power mode can be used to provide a power-on reset without a power gating cycle. As outlined above, in this case the warm reset signal from the PPU is used as the primary power-on reset signal.

Chapter 8
Component Design Considerations

This chapter gives an overview of component design requirements and guidelines for integration within the power control system architecture.

NOTE: Some content in this section corresponds to content within the *Low-Power Interface Specification, ARM Q-Channel and P-Channel Interfaces* [4]. In such cases the content here is intended to accord to that specification and provide complementary information.

This chapter is organized into the following sections:

- 8.1 *General Low-Power Interface Guidelines*
- 8.2 *Component High-Level Clock Gating*
- 8.3 *Component Power Control*

8.1 General Low-Power Interface Guidelines

This section provides general guidelines for components implementing Arm low-power interfaces for clock and power control.

8.1.1 Low-Power Interface Implementation

Q-Channel Implementation

A component and its clock or power controller typically use asynchronous clocks. In clock control applications, these clocks are from the same source, but due to significant phase differences the design considerations are those required for an asynchronous interface. Figure 8.1 shows the required interface signal handling for an asynchronous Q-Channel interface from a component perspective.

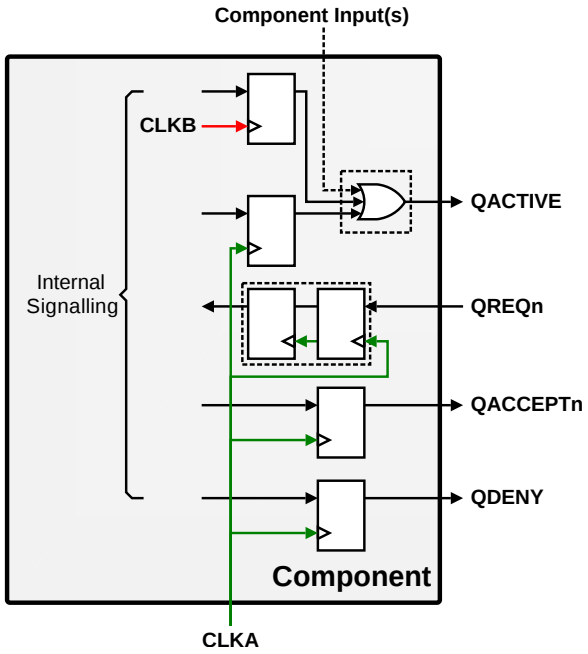


Figure 8.1: Q-Channel interface implementation

The following guidance is provided to implementers for an asynchronous Q-Channel interface:

- Synchronize all signals at their destination before use. It is strongly recommended that components include the synchronization elements for input signals. From a component perspective, this is shown for **QREQn** in Figure 8.1.

NOTE: Consequences of implementing the synchronization externally to the component are detailed in *Synchronization of Input Signals*.

- To ensure correct operation of the handshake interface **QREQn**, **QACCEPTn** and **QDENY** outputs must be registered. This allows them to be captured correctly in the destination clock domain without any glitches from combinational logic. This is shown for **QACCEPTn** and **QDENY** in Figure 8.1.
- The **QACTIVE** signal is driven either directly by a register, or by several registers whose contributions are logically combined. In Figure 8.1 **QACTIVE** sources include a register in the controlled clock domain, a register in another clock domain and component inputs which must also be registered at source. Implementation recommendations are given in 8.1.1 .

- For a clock control Q-Channel it is strongly recommended that the component Q-Channel control logic is implemented using only the controlled clock. This prevents another clock being required to complete any requests.

P-Channel Implementation

A component and its controller typically use asynchronous clocks. Figure 8.2 shows the required interface signal handling for an asynchronous P-Channel interface from a component perspective.

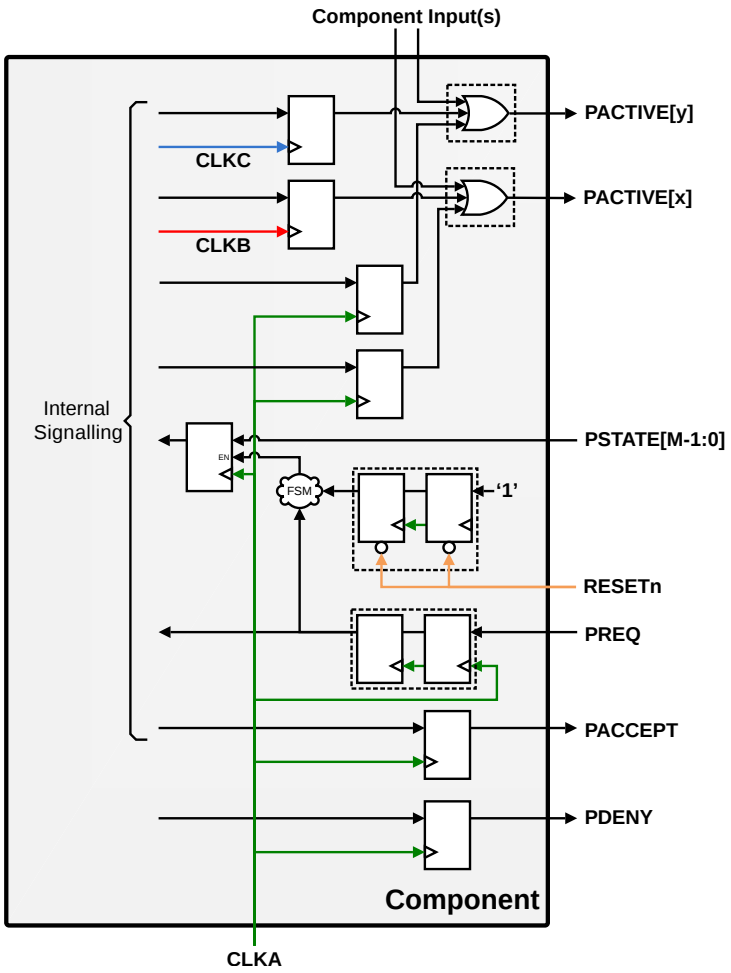


Figure 8.2: P-Channel interface implementation

The following guidance is provided to those implementing an asynchronous P-Channel interface:

- Synchronize all signals, except for **PSTATE**, at their destination before use. It is strongly recommended that components include the synchronization elements for input signals. From a component perspective, this is shown for **PREQ** in Figure 8.2. Guidance for capturing **PSTATE** is given in 8.1.1 *Capturing PSTATE*.

NOTE: Consequences of implementing the synchronization externally to the component are detailed in *Synchronization of Input Signals*.

- To ensure correct operation of the handshake interface **PREQ**, **PACCEPT** and **PDENY** outputs must be registered. This allows them to be captured correctly in the destination clock domain without any glitches from combinational logic This is shown for **PACCEPT** and **PDENY** in Figure 8.2.
- Each bit of the **PACTIVE** signal is driven either directly by a register, or by several registers whose contributions are logically combined. In Figure 8.2 **PACTIVE** sources include a component input, register in the interface clock domain, and registers in two other clock domains within the component. Implementation recommendations are given in 8.1.1 .

Capturing PSTATE

To ensure the **PSTATE** value is received correctly by the component, its capture must be enabled at reset de-assertion to ensure correct initialization or by a change on the synchronized **PREQ** signal.

It is strongly recommended that the **PSTATE** value is sampled on a change on the synchronized **PREQ**, not just when **PREQ** is HIGH, as in the event of a denial **PSTATE** can change while **PREQ** is HIGH and therefore might be captured incorrectly.

In addition, to meet the P-Channel protocol requirements for component reset and initialization specified in *Low-Power Interface Specification, ARM Q-Channel and P-Channel Interfaces* the **PSTATE** value must be sampled at reset de-assertion within a period, t_{init}, defined in component clock cycles. Capturing **PSTATE** at reset exit facilitates transition to a functional state for components where the P-Channel interface is unused.

To implement the sampling of **PSTATE** at reset de-assertion requires an enable term for the **PSTATE** capture register that is set following the exit from reset and then cleared in the following clock cycles.

Figure 8.3 shows a simplified state machine indicating when **PSTATE** is captured. This state machine is not intended to show a full description of the required P-Channel transitions but to illustrate when **PSTATE** should be captured.

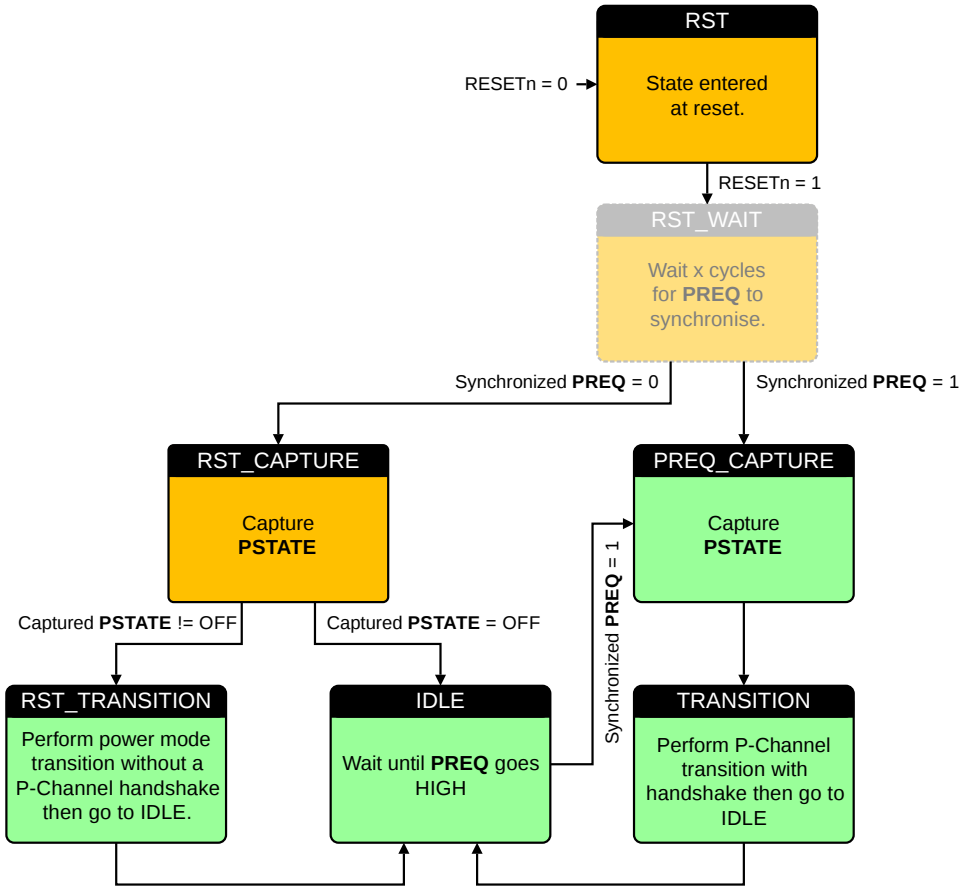


Figure 8.3: Principle of PSTATE sampling with pseudo state machine

In Figure 8.3, the orange boxes indicate that the component is still within the tinit period, the green boxes show that this period has expired. The RST_WAIT state is optional, to prevent multiple transitions within the component if this either takes a significant amount of time or effects component operation.

Synchronization of Input Signals

It is strongly recommended that components include the synchronization elements for input signals. The synchronizing elements are required to be captured in a dedicated design hierarchy. This aids identification for two purposes:

- Replacement with specific cells in physical design.
- Replacement with a bypass functionality in case of a synchronous implementation.

In a component design context, this applies to the **QREQn** and **PREQ** inputs.

The rationale for the recommendation of internal synchronization is as follows:

- There is dependency between clock and power control LPI. A clock control Q-Channel wake-up is required when the clock is not running but is required for the power control interface to transition.
  - If synchronization is not included within the component, then synchronization is required to be added externally.
  - The implementation of external synchronization causes external logic to be required to allow the unsynchronized external power control **QREQn** or **PREQ** to contribute to a clock control **QACTIVE** path.
  - Such dependencies are detailed in 8.2.3 *QACTIVE Contribution from Power Control Low-Power Interface*.
- Requiring the external implementation of synchronizers places a burden on the integrator, risks include omission of the synchronizers and inappropriate clocking and reset application of the synchronization.

QACTIVE and PACTIVE Contribution Combination

QACTIVE and **PACTIVE** signals are driven either directly by a component register, or by a component register and component inputs whose contributions are logically combined. Arm strongly recommends that this combining logic is limited, where possible, to logical OR gates.

Combining logic gates must be captured in a dedicated design hierarchy so the function can be identified and preserved through synthesis and replaced with specific technology cells if required.

The preservation of function ensures that no alternative mapping is used that might cause glitches in the path. The ability to identify the cells can also be useful for analysis purposes.

When using combining logic other than OR gates, the implications of changes on the inputs to this logic on the **QACTIVE** or **PACTIVE** output signals must be carefully considered. Although the handshake protocol guarantees functionally correct behavior regardless of changes on **QACTIVE** and **PACTIVE**, it is recommended to implement the simplest possible logic to minimize the likelihood of introducing glitches at **QACTIVE** and **PACTIVE** outputs.

One example of a circumstance requiring combining logic other than a logical OR is detailed in 8.2.3 *QACTIVE Contribution from Power Control Low-Power Interface*.

Recommended LPI Feature Support

Arm strongly recommends that components support the function to deny requests.

The denial mechanism allows a component to respond in a timely fashion and continue operations without interruption. This avoids, in a situation where the conditions at the component have changed and it is no longer idle, a potentially indefinite acceptance delay or the latency overhead of the component accepting the request and then returning to the previous state. From a controller perspective this avoids there being outstanding requests that are no longer valid that might prevent it from making a more appropriate request or performing other actions.

A denial scenario typically arises when the component becomes active between the controller sampling a **QACTIVE** or **PACTIVE** signal LOW and the arrival of a request at the component.

8.1.2 Output Management

When a component changes power mode, isolation can be applied or removed, and resets can be asserted and deasserted. Isolation and reset control are typically asynchronous. Therefore, component outputs need to be conditioned so any potential asynchronous change does not affect functioning domains to which they communicate.

To prevent outputs changing during under these conditions the component design must apply the following rule:

- Reset value = Idle value = Isolation Value.
  - Limited exceptions can be applied for components in retention modes.

Care must be taken, especially with reset values, as an output might be driven by multiple registers through combinatorial logic. If this is the case it must be ensured that all source registers can be reset without changing their outputs to ensure no glitches are produced in the following logic.

In some cases, the idle value might not be the same as the isolation and reset value due to domain retention. Care must be taken to assess if an asynchronous change on such signals will cause issues in other domains.

Any such outputs must be justified and documented so they can be managed at the system level. Strategies for managing these at the system level are discussed in 7.2.9 *Isolation and Reset Control Considerations*.

If there are power domain boundaries contained within a component any non-idle values due to retention can be managed by capturing the output value externally to the power domain but within the component before applying isolation or reset and releasing isolation or reset before capturing the output value again. Such behavior can be controlled using the component LPI interface.

In all cases care should be taken as RTL simulation can differ where isolation is not implemented. Power aware simulations that consider these aspects are recommended.

NOTE: When a domain is in retention, isolation must be enabled. Although register values are retained, typically register outputs are not driven, and any buffers between the register output and the domain boundary are powered-off.

8.1.3 Interface Management

A non-functional LPI mode is a power mode where the component cannot actively respond to signals arriving at its boundary. This is typically due to the component being in or being prepared to be put in a clock gated, full retention, or off power mode.

When a component LPI enters a non-functional low-power mode it is not able to respond to protocol transactions on its interfaces. All interfaces must be managed appropriately to ensure the system does not suffer a functional failure. While there can be many interfaces, a primary area of concern is bus interfaces and the remainder of this section is concerned with the management of these.

At entry to the low-power mode, the component must ensure that bus interfaces are in a quiescent state and then safely suspend the protocol. New transactions can arrive at the component boundary but cannot be processed by the component unless it is a state where they are guaranteed to complete correctly. This is not the case if the clock or a required power mode is not guaranteed by the associated LPI.

At exit from the low-power mode, the interface quiescence must be safely reversed before transactions are accepted.

During a low-power mode, it must be ensured that any control signals that could falsely initiate, or respond to, any protocol behavior from active parts of the system, are set to safe levels. When in the low-power mode, several possible response models to attempted bus transactions can be used. The following are the available response models:

- **Stall and wake:** A transaction arriving is stalled. A wake-up signal is asserted indicating that the downstream component must be made available (meaning the clock or power domain depending on the low-power mode). Once available the bus is un-stalled and the transaction progresses as normal.
  - This model is used to preserve the availability of a resource while allowing it to enter a low-power mode.
- **Error:** An error response is given to any arriving transaction. There is no request to change power mode as a result of the transaction.
  - This might be used for debug purposes in case an access is attempted in error.
  - Protocols that can return errors without active responses are limited.
- **Accept and ignore:** Transactions are accepted to prevent blocking the interface, but no action is taken. There is no request to change power mode as a result of the transaction
  - This might be used with some types of broadcast, trace or monitoring traffic.

NOTE: An exception to this rule is the clock control Q-Channel. This must respond to requests to enter and exit a quiescent state even when the power control LPI is in a non-functional mode. This allows a clock to be requested to complete power control transitions.

Clock Control

In clock control applications, it is strongly recommended to use the stall and wake response.

In some limited scenarios where the protocol is broadcasting information to the component and this information is not required while the component is clock gated an accept and ignore response might be appropriate.

As part of an accepted low-power interface request the component performs the entry sequence to ensure bus interfaces are quiescent. On accepting the low-power interface request the clock might be gated. A component should not rely on the clock being gated to operate correctly.

As the component is still powered-on in the clock gated state the component can use an input signal as a wake-up to combinatorially drive the clock control **QACTIVE**. Once the clock control Q-Channel has returned to *Q_RUN* the component can process transactions.

Examples of interface management for common protocols are described in the following sections.

High-level clock gating, including a specific AXI responder implementation example, is detailed in 8.2 *Component High-Level Clock Gating*.

Power Control

In power control applications, all the response models are possible.

As part of an accepted low-power interface request the component performs the entry sequence to ensure bus interfaces are quiescent. On accepting the low-power interface request power might be removed. A component should not rely on the power being removed, reset being asserted, or isolation being applied to operate correctly.

Wake-up events must come from logic in a powered-on domain.

After power is restored and the low-power interface has transitioned the component to a functional power mode it can process transactions.

Examples of interface management for common protocols are described in the following sections.

AMBA APB Interface Management

To ensure the APB interface is in a safe state, before the component enters a non-functional mode on the LPI it must manage transactions by using one of the following methods:

- **Stall and wake**: Stall transactions by setting **PREADY** LOW and **PSLVERR** LOW.
- **Error**: Provide error responses by setting **PREADY** HIGH and **PSLVERR** HIGH
- **Accept and ignore**: Provide zero data read/write ignored responses by setting **PREADY** HIGH, **PSLVERR** LOW and **PRDATA** to 0x00000000.
  - **NOTE:** In this case transactions will be lost.

For clock control a stall and wake response must be used. For power control the other cases can be considered, but it is recommended to use stall and wake, allowing the component low power modes to be managed by the LPI interface on SW access.

APB Management with Stall and Wake

When the LPI is in a non-functional state and an APB transaction is received then **PWAKEUP** going HIGH must be used to combinatorially generate a HIGH level on a **QACTIVE** or relevant **PACTIVE**. This is used as a wake request to bring the LPI to a functional state so the component can process the transaction. In circumstances where **PWAKEUP** is not present then **PSEL** must be used for this purpose.

While the LPI is in a non-functional mode the APB transaction will move from the setup phase, where **PSEL** is HIGH and **PENABLE** is LOW, to the access phase, where **PSEL** is HIGH and **PENABLE** is HIGH. Therefore, component interface design must not depend upon seeing the setup phase to complete the transaction.

Figure 8.4 shows an example of the relationship between an APB interface and a Q-Channel for the stall and wake scenario.

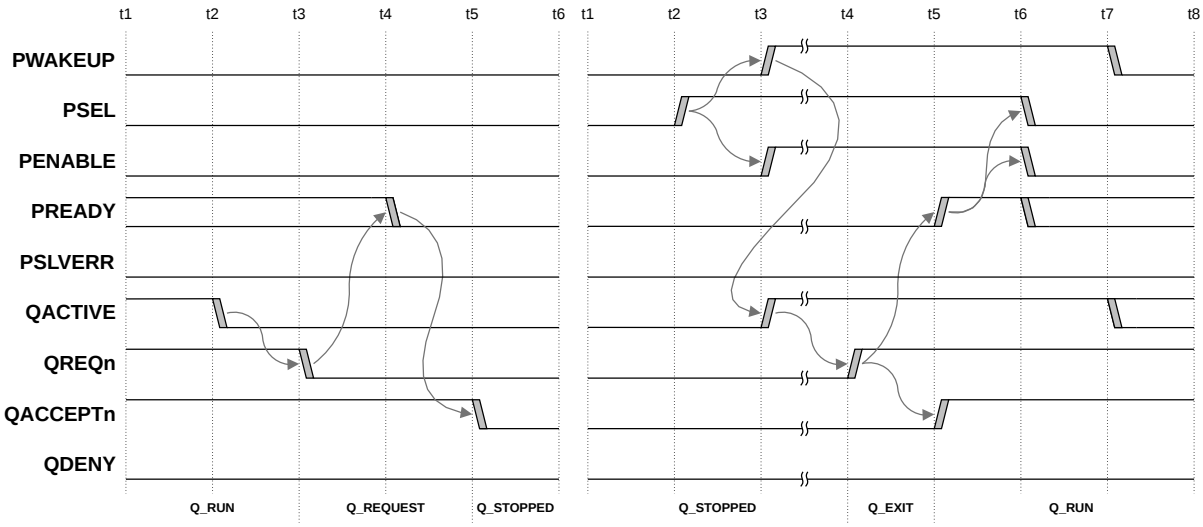


Figure 8.4: Q-Channel quiescence and stall and wake APB interface management

NOTE: The **PWAKEUP** is shown following **PSEL**, however **PWAKEUP** can occur before or coincident with the **PSEL**. The **QACTIVE** will be driven HIGH whenever the **PWAKEUP** goes HIGH.

APB Management with Error

Figure 8.5 shows an example of the relationship between the APB interface and a Q-Channel for the error response scenario.

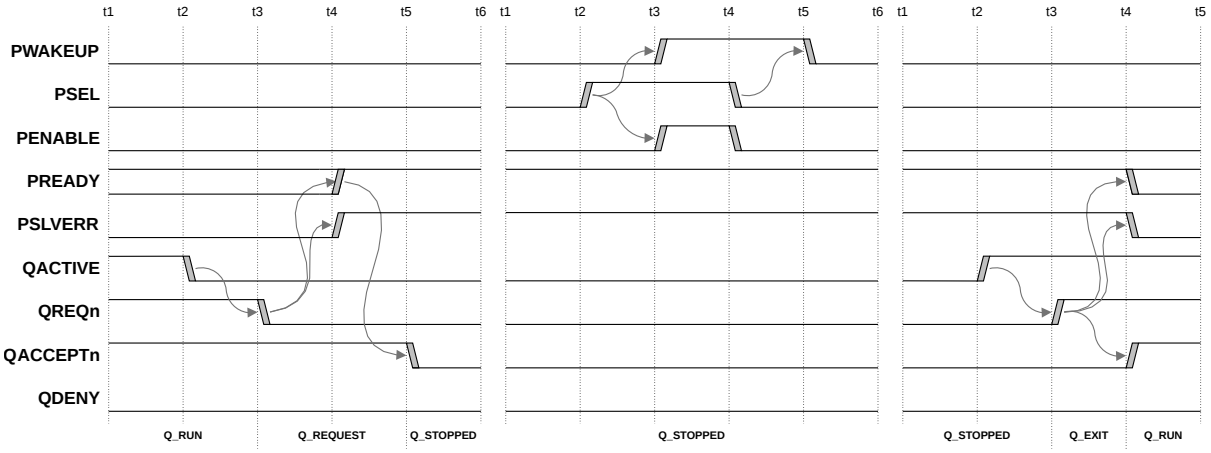


Figure 8.5: Q-Channel quiescence and error APB interface management

APB Management with Accept and Ignore

Figure 8.6 shows an example of the relationship between the APB and a Q-Channel for the accept and ignore response scenario.

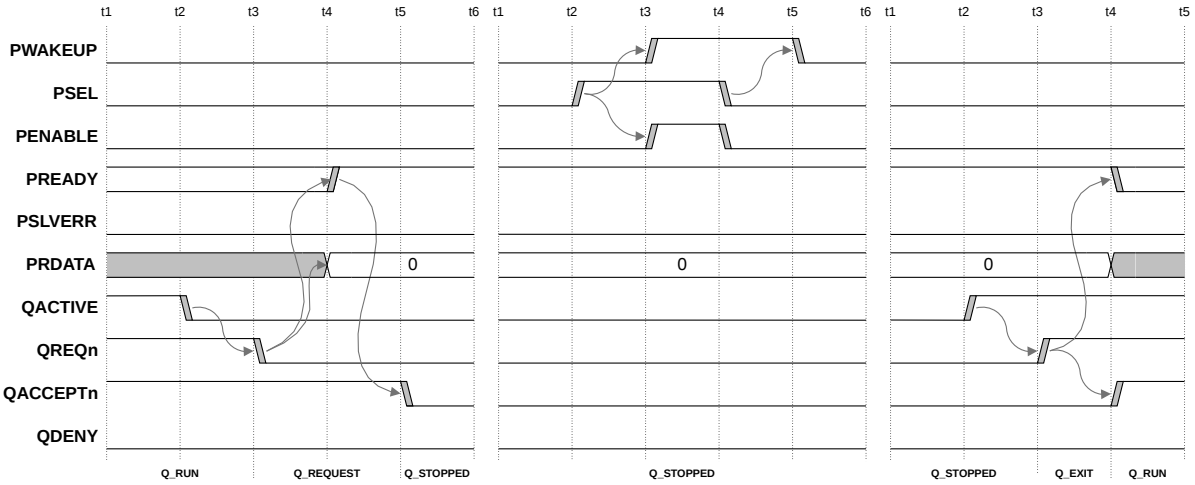


Figure 8.6: Q-Channel quiescence and accept and ignore APB interface management

AMBA AXI and ACE Interface Management

To ensure the AXI interface is in a safe state before the component enters a non-functional mode on the LPI it must manage transactions on the interface.

Where the component will be non-functional a stall and wake response must be used. This allows the component low power modes to be managed by the LPI interface on SW access. For clock control this response must be used in all cases.

The component can stall the interface by setting all ***READY** outputs LOW.

For power control if either the ‘error’ or ‘accept and ignore’ responses are required on AXI this must be provided by an upstream component that can provide an active sequential response. It is not possible to provide these responses with tied values. These responses could come from a separate access control gate within an upstream power domain, managed by an LPI from the PPU controlling the downstream domain, or the upstream part of a domain-bridge crossing the boundary of the two domains.

For ACE, it should be noted that **WACK** and **RACK** signals do not have any acceptance response. A component needs to track that these signals have propagated either beyond its boundary, or to an internal termination point before becoming quiescent.

AXI Management with Stall and Wake

When the LPI is in a non-functional state and an AXI transaction is received then **AWAKEUP** going HIGH must be used to combinatorially generate a HIGH level on a **QACTIVE** or relevant **PACTIVE**. This is used as a wake request to bring the LPI to a functional state so the component can process the transaction. In circumstances where **AWAKEUP** is not present then an OR combination, using instantiated components, of **AWVALID**, **ARVALID** and **AWVALID** inputs must be used for this purpose.

For entry to a quiescent state the component must track transactions. If a quiescent or low power request requiring the AXI interface to be stalled is received and there are outstanding AXI transitions, then the request must be denied. An AXI responder clock gating implementation example is detailed in 8.2 *Component High-Level Clock Gating*.

Figure 8.7 shows an example of the relationship between an AXI Channel and a Q-Channel for the stall and wake scenario.

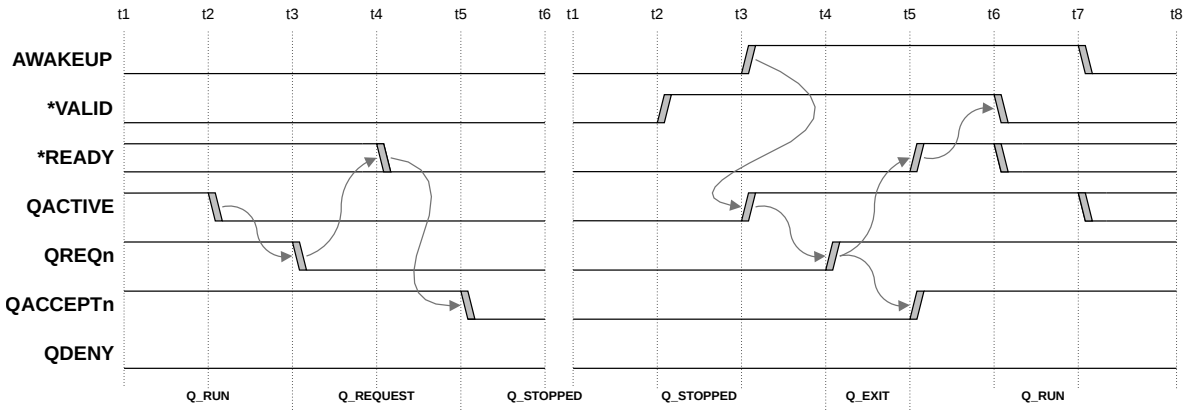


Figure 8.7: Q-Channel quiescence and stall and wake AXI interface management

NOTE: The **AWAKEUP** is shown following ***VALID**, however **AWAKEUP** can occur before or coincident with the ***VALID**. The **QACTIVE** will be driven HIGH whenever the **AWAKEUP** goes HIGH.

AMBA AXI-Stream Interface Management

To ensure the AXI-Stream interface is in a safe state, before the component enters a non-functional mode on the LPI it must manage transactions by using one of the following methods:

- **Stall and wake**: Stall transactions by setting **TREADY** LOW.
- **Accept and ignore**: Accept transaction with no effect by setting **TREADY** HIGH.
  - **NOTE:** In this case transactions will be lost.

It is not possible to generate error responses on an AXI-Stream interface.

For clock control a stall and wake response must be used.

For power control the other cases can be considered, but it is recommended to use stall and wake, allowing the component low power modes to be managed by the LPI interface on SW access.

AXI-Stream Management with Stall and Wake

When the LPI is in a non-functional state and an AXI-Stream transaction is received then **TWAKEUP** going HIGH must be used to combinatorially generate a HIGH level on a **QACTIVE** or relevant **PACTIVE**. This is used as a wake request to bring the LPI to a functional state so the component can process the transaction. In circumstances where **TWAKEUP** is not present then **TVALID** must be used for this purpose.

Figure 8.8 shows an example of the relationship between an AXI-Stream interface and a Q-Channel for the stall and wake scenario.

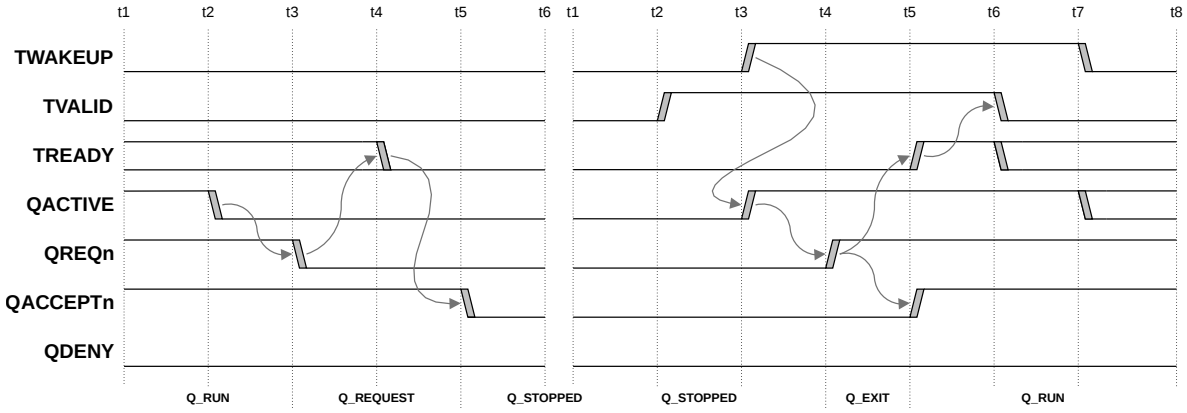


Figure 8.8: Q-Channel quiescence and stall and wake AXI-Stream interface management

NOTE: The **TWAKEUP** is shown following **TVALID**, however **TWAKEUP** can occur before or coincident with the **TVALID**. The **QACTIVE** will be driven HIGH whenever the **TWAKEUP** goes HIGH.

AXI-Stream Management with Accept and Ignore

Figure 8.9 shows an example of the relationship between the AXI-Stream interface and a Q-Channel for the accept and ignore response scenario.

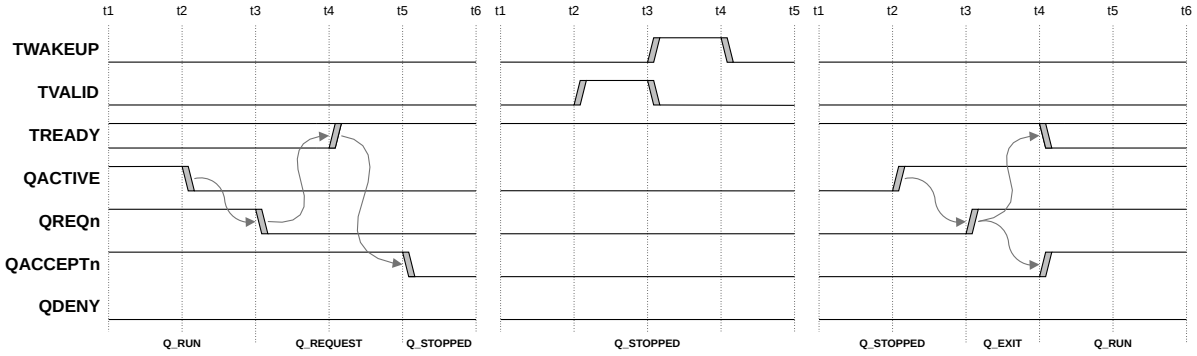


Figure 8.9: Q-Channel quiescence and accept and ignore AXI-Stream interface management

AMBA CHI Interface Management

To ensure the CHI interface is in a safe state before the component enters a non-functional mode on the LPI it must manage transactions on the interface.

Where the component will be non-functional a stall and wake response must be used by the component. This allows the component low power modes to be managed by the LPI interface on SW access. For clock control this response must be used in all cases.

A component can stall transactions on a CHI bus by deactivating links using the **LINKACTIVE** protocol. Completion of the **LINKACTIVE** deactivation also guarantees that the agents at either side of the link have safely terminated any transactions.

For power control if either the error or 'accept and ignore' responses are required on an CHI interface this must be provided by an upstream component that can provide an active sequential response. It is not possible to provide these responses with tied values. These responses could come from a separate access control gate within an upstream power domain, managed by an LPI from the PPU controlling the downstream domain, or the upstream part of a domain-bridge crossing the boundary of the two domains.

CHI Management with Stall and Wake

When the LPI is in a non-functional state then **RXSACTIVE** going HIGH must be used to combinatorially generate a HIGH level on a **QACTIVE** or relevant **PACTIVE**. This is used as a wake request to bring the LPI to a functional state so the component can process open the **LINKACTIVE'** and process transactions.

8.2 Component High-Level Clock Gating

This section describes implementation of high-level clock gating using Q-Channel low-power interfaces. Q-Channel is always used for this function.

A component should also implement low-level and mid-level clock gating internally where appropriate. The definitions used for levels of clock gating in this specification are given in 7.1.1 *Clock Gating Levels*.

8.2.1 Q-Channel Implementation

A component requires a clock control Q-Channel for each clock input. This allows the component to take part in shared system level clock gating with other components in the same clock domain to gate the shared high-level clock.

Each clock requires a separate Q-Channel interface so each clock can be gated separately, and the Q-Channels combined with other components in the same clock domain. Figure 8.10 shows an example of a component with multiple clock domains.

The clock control Q-Channels must be separate from any power control LPIs.

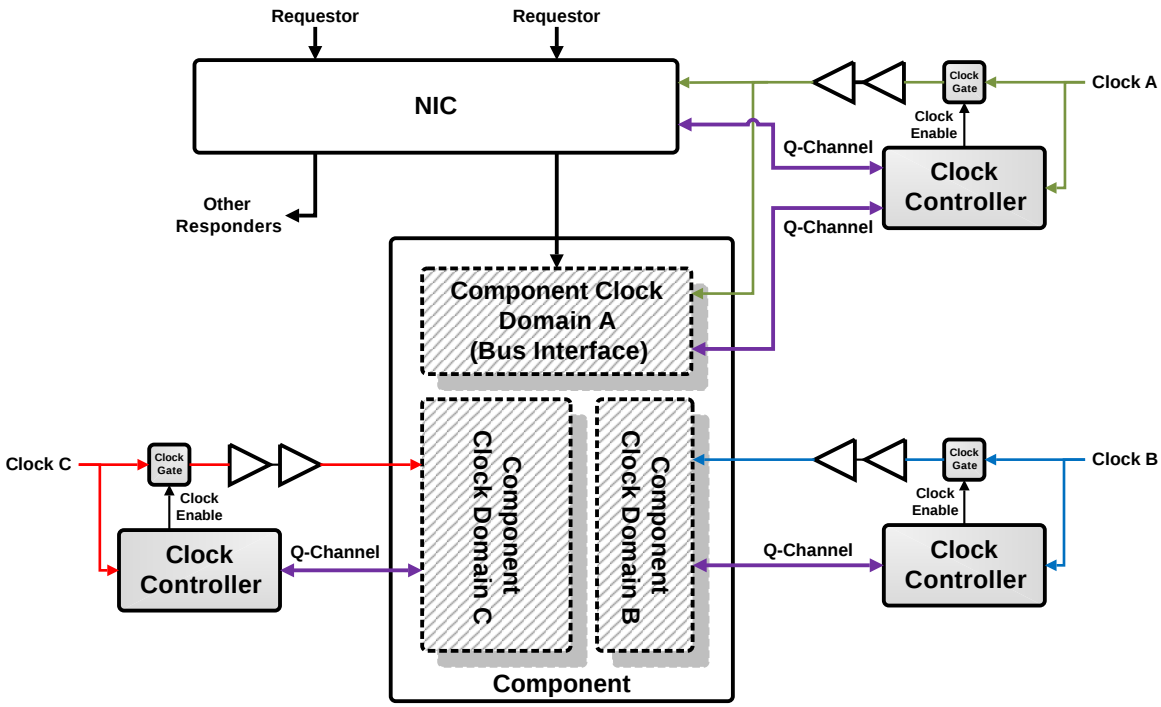


Figure 8.10: Example of a component with multiple clock domains

A common example, as shown in Figure 8.10, is a separate bus interface clock. This can be gated with the interconnect when there are no bus accesses, even when other clock domains within the component are required to be active.

8.2.2 Clock Availability during Clock Control Q-Channel Quiescence

The clock is guaranteed to be running whenever the clock control Q-Channel is not in the *Q_STOPPED* state.

However, when the clock control Q-Channel is in the *Q_STOPPED* state the clock can still be running. There can be several reasons for this, but one example is when another component managed by the same clock controller denies a quiescence request and therefore the shared clock continues to run.

Therefore, a component must not assume the clock will be gated and use this as a mechanism to ensure a component will not perform any operations. The behavior of the component in a quiescent state must be guaranteed by logical means.

In addition, a component must not use internal clock gating alone to provide this function if it is possible that the clock gates could be replaced by bypasses, as this replacement will remove the required functionality.

8.2.3 QACTIVE Behavior

QACTIVE is used for two purposes in clock control:

1. When HIGH: Indicates that the component requires a clock, regardless of the Q-Channel state. The supply of the clock is subject to the guarantees provided by the Q-Channel handshake.
2. When LOW: Hints that the component might accept a clock control Q-Channel quiescence request.

QACTIVE at Reset

The clock control **QACTIVE** contribution from internal component state must be LOW at reset. This allows the clock to be idle until component activity is required and prevents unnecessary clocks from running while a component is held in reset.

Combinatorial contributions from external sources, such as from a bus interface access or a transition on a power control LPI, can drive the **QACTIVE** HIGH while the component has its reset asserted to provide a wake function.

Contributions from a power control LPI are described in 8.2.3 *QACTIVE Contribution from Power Control Low-Power Interface*.

QACTIVE Stimuli

QACTIVE stimuli is typically derived from the following:

- Internal logic state.
- External signaling.
- Other low-power control channels.

Some specific considerations are discussed in the following sections.

A transition of the clock control Q-Channel handshake must not set its own **QACTIVE** HIGH. The clock being managed by the Q-Channel is guaranteed to be running when the Q-Channel transitions, so the **QACTIVE** is not required to be HIGH for it to be requested. Setting it HIGH could cause the interface to exit quiescence, before it becomes idle again, potentially causing the interface to oscillate between quiescent and actives states.

QACTIVE Contribution from Internal State

Internal logic state contributions to a clock control **QACTIVE** can be numerous and are not restricted.

Typical examples can include the activity of a processing element, tracking of data transfer completion, including waiting for responses from other components, and programmed settings.

QACTIVE Contribution for Clock Wake Up

To support **QACTIVE** assertion when a clock is not present there must be a **QACTIVE** contribution combinatorially from either a component input or a signal from another clock domain within the component.

QACTIVE Contribution from Power Control Low-Power Interface

When the operation of a power control LPI is dependent on the supply of a Q-Channel controlled clock a contribution to the clock control **QACTIVE** is required. This ensures the clock is supplied during power control LPI handshake sequences.

A power control LPI request can arrive when the clock control Q-Channel is in *Q_STOPPED*. Since the clock is required to progress the power mode LPI request a contribution to the clock control **QACTIVE** from the power control LPI interface logic is required.

For P-Channel power control, the clock control **QACTIVE** contribution can be provided by including the power control **PREQ** input signal, before any internal synchronization, along with the **PACCEPT** and **PDENY** as a contribution to the clock control **QACTIVE**.

Figure 8.11 shows an example of a component with power control P-Channel and a clock control Q-Channel:

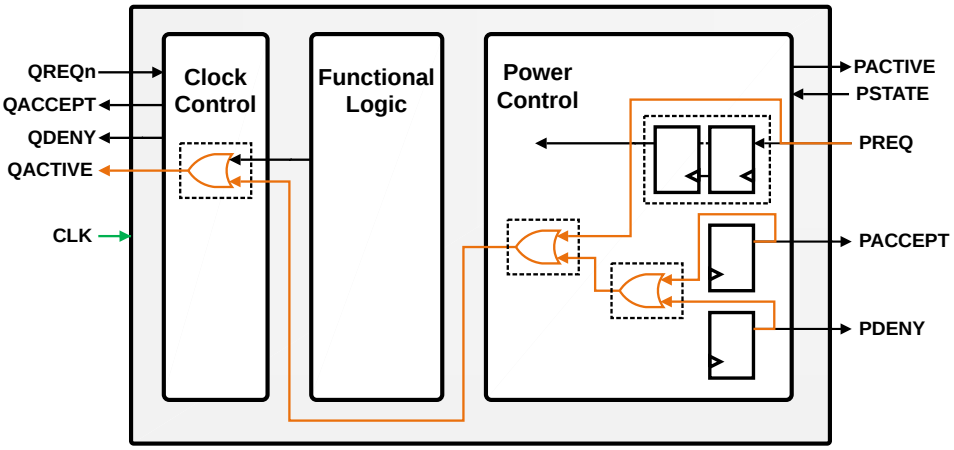


Figure 8.11: Example of P-Channel power control to clock control dependency

For a Q-Channel the wake-up condition can be provided by an XOR of the power control **QREQn** input signal, before any internal synchronization, and the **QACCEPTn** output signal, then combined with the **QDENY** output.

This is an exception to the recommendation to limit **QACTIVE** combining to a logical OR where possible. In this specific case, it is safe as the inputs to the XOR are guaranteed to change only one at a time.

Figure 8.12 shows an example of a component with power control Q-Channel and a clock control Q-Channel:

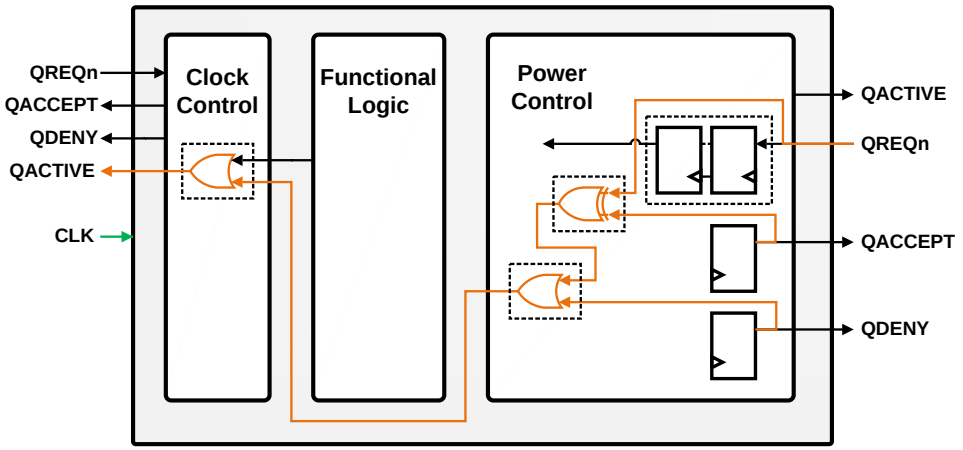


Figure 8.12: Example of Q-Channel power control to clock control dependency

QACTIVE Policy

QACTIVE should be driven LOW when the clock is not required by the component to provide the maximum opportunity for clock gating.

It is recommended that the application of any hysteresis is implemented by the clock controller at the system level as the required behavior is typically system specific and depends upon the combination of components within the clock domain.

8.2.4 Q-Channel Handshake Behavior

Q-Channel Handshake at Reset

From a component perspective, the clock control Q-Channel will always exit reset in the *Q_STOPPED* state. Component **QACCEPTn** and **QDENY** outputs must be reset LOW.

Any resynchronization elements in the component on the **QREQn** input must be reset LOW.

The component must assume the state of the **QREQn** input is LOW at reset exit until it is sampled HIGH.

A controller can set **QREQn** LOW or HIGH at reset exit. When **QREQn** is set HIGH, the interface is in *Q_EXIT* state, but this will not be sampled by a component until its reset is released.

The reset state of the component should be equivalent to the quiescent state, with all required interface management in place.

Q-Channel Quiescence Requests

A quiescence request on the clock control Q-Channel can be made to a component at any time regardless of the state of the **QACTIVE** signal.

A component must only wait to accept a Q-Channel quiescence request when a response can be given within a bounded time. The length of this bounded time is not limited and depends upon the actions required to accept the request.

If a component cannot bound the time required until it will accept it should deny the request.

A clock control Q-Channel must always respond to requests to enter and exit a quiescent state, even when the power control LPI is in a non-functional mode. This allows a clock to be requested to complete power control transitions.

Actions required to enter quiescence

A component must ensure that when it accepts a request there will be no functional failure because of the clock being disabled.

Therefore, before accepting a clock controller Q-Channel quiescence request a component must ensure:

- There is no internal activity requirement.
- All outputs and interfaces are managed as described in 8.1.2 *Output Management* and 8.1.3 *Interface Management*.
  - This includes internal interfaces where a clock domain boundary is internal to the component.

All required actions must be completed before an accept response is sent by setting **QACCEPTn** LOW. Once the accept response is sent the capabilities of the previous power mode are not guaranteed.

Q-Channel Quiescence Exit

At exit from quiescence once a component has sampled **QREQn** HIGH, in the interface *Q_EXIT* state, it can resume operations. It does not have to wait until it has accepted the request by setting **QACCEPTn** HIGH.

The controller cannot make another quiescence request until **QACCEPTn** is HIGH. The component can use this to prevent another request until it has carried out any required initialization sequence. Alternatively, it can respond immediately and deny any quiescence requests until it is idle.

8.2.5 Implementation Example: AXI Responder Interface

Figure 8.13 shows a conceptual implementation of high-level clock gating support for a component with an AXI responder interface.

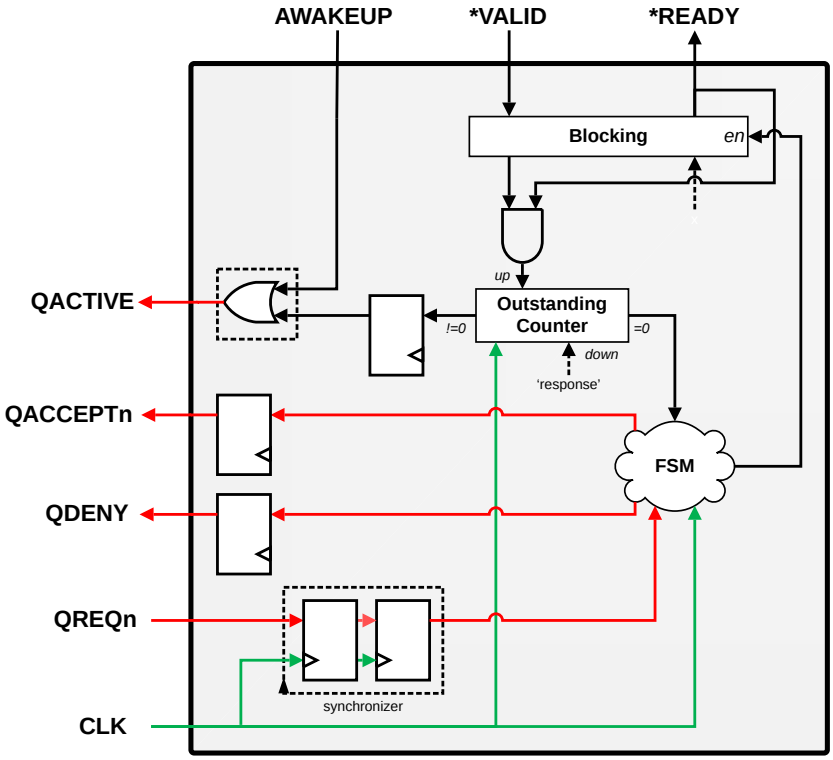


Figure 8.13: AXI responder interface high-level clock gating example

The implementation has four elements:

- A wake-up condition using external signaling as a **QACTIVE** contribution.
- Internal state tracking the clock activity requirement, once active, as a **QACTIVE** contribution.
- Interface management at the AXI responder interface to prevent transactions being accepted unless the clock is guaranteed.
- Q-Channel handshake logic.

The clock activity is controlled according to the processing of AXI transactions.

The operation of the circuit can be described as follows:

1. From a clock stopped state (*Q_STOPPED*).
   a. **ARREADY**, **AWREADY** and **WREADY** are blocked LOW. All **VALID** inputs are ignored.
   b. **QACTIVE** goes HIGH when **AWAKEUP** is asserted HIGH.
2. The Q-Channel clock controller sets **QREQn** HIGH and guarantees the clock is running (*Q_EXIT*).
   a. The **READY** outputs are un-blocked and **VALID** inputs are accepted. Transactions are now accepted.
   b. The component sets **QACCEPTn** HIGH (*Q_RUN*).

c. **QACTIVE** is maintained HIGH by an outstanding counter that remains active until all responses are completed.

3. **Accepted Quiescence**: The controller sets **QREQn** LOW (*Q_REQUEST*).
   a. This is according to controller policy specific criteria, for example when **QACTIVE** is LOW.
   b. The clock is guaranteed until the component sets **QACCEPTn** LOW. It can continue to accept transactions if any arrive. In this case, the behavior is to deny the quiescence request (see 4) rather than delay acceptance indefinitely.
   c. If all accepted outstanding transactions are completed, none are pending and **VALID** and **READY** are blocked, then **QACCEPTn** is set LOW by the component (*Q_STOPPED*). The clock is no longer guaranteed.
4. **Denied Quiescence**: The controller sets **QREQn** LOW (*Q_REQUEST*).
   a. This is according to controller policy specific criteria, for example when **QACTIVE** is LOW.
   b. If the component has returned to a busy state, for example if a transaction arrived after the controller detected **QACTIVE** LOW, it sets **QDENY** HIGH (*Q_DENY*). The clock continues to run, and the component remains functional.
   c. On sampling **QDENY** HIGH the controller must set **QREQn** HIGH (*Q_CONTINUE*).
   d. On sampling **QREQn** HIGH the component sets **QDENY** LOW (*Q_RUN*).

8.2.6 Unused Clock Control Q-Channels

The component must operate correctly if the integrator chooses to not use a clock control Q-Channel. In this case, the component can assume the clock will always be available. It becomes the responsibility of the integrator to ensure the clock is managed correctly.

When the Q-Channel is unused the **QREQn** input must be tied HIGH.

For components with multiple clock control Q-Channels, if one channel is not used it must not limit the use of other clock or power control LPIs. This allows the integrator to choose which clock domains implement high-level clock gating.

8.2.7 Clock Control Q-Channel Naming Guidelines

It removes ambiguity and eases integration to give the clock control Q-Channels meaningful names according to their function. Even when there is a single interface, naming can help prevent misuse of the interface.

Clock control Q-Channels are recommended to be prefixed with the name of the controlled clock, this avoids ambiguity about the use of the Q-Channel and the clock to which it applies.

When sorted alphabetically, in simulation waves or lists, the Q-Channels will appear by function and next to the clock they control. An alternative approach is to add the clock name as a post-fix.

Table 8.1 shows an example.

Table 8.1: Clock control Q-Channel naming convention example

Naming Convention	ACLK Example
<*clk*>QACTIVE	**ACLKQACTIVE**
<*clk*>QREQn	**ACLKQREQn**
<*clk*>QACCEPTn	**ACLKQACCEPTn**
<*clk*>QDENY	**ACLKQDENY**

8.3 Component Power Control

This section describes implementation of power control using Q-Channel and P-Channel low-power interfaces.

8.3.1 Low-Power Interface Selection

Power Control can use a either Q-Channel or P-Channel low-power interface.

Q-Channel is recommended for components that have a single low-power mode as it has a simple run-stop semantic. The low-power mode entered can vary, but only according to a common configuration of both the component and the controller before quiescence is requested on the Q-Channel.

The advantage of using Q-Channel is the simpler interface that is easier to combine with other power control Q-Channels.

P-Channel is recommended for components that have multiple power or operating modes. For example, a component that can be put into retention and off. The P-Channel allows:

- The component to express varying levels of activity requirement to a power controller for entry into different modes.
  - For example, the conditions can be different for retention and off.
- The controller to express the required mode, allowing the component to take different actions, dependent on the mode requested.
  - For example, a core transition to retention manages interfaces and clocks but a transition to off adds the requirement for caches to be flushed and in-cluster coherency to be managed.
- Transitions between multiple modes.

P-Channel is only selected if Q-Channel functionality is insufficient.

Therefore, a P-Channel should be selected if a component is required to do any of the following:

- Support multiple power or operating modes.
- Accept and deny on different conditions, depending on the mode requested.
- Transition between different modes without returning to a common functional mode.

Figure 8.14 shows the differences between P-Channel and Q-Channel sequences for transitioning to and from a retention mode.

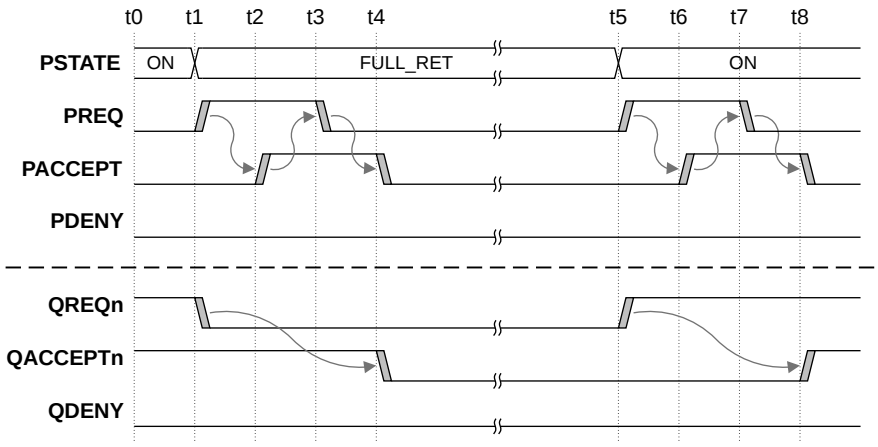


Figure 8.14: Q-Channel and P-Channel entry to and exit from retention

Figure 8.15 shows the differences between P-Channel and Q-Channel sequences for transitioning from ON to FUNC_RET then to FULL_RET.

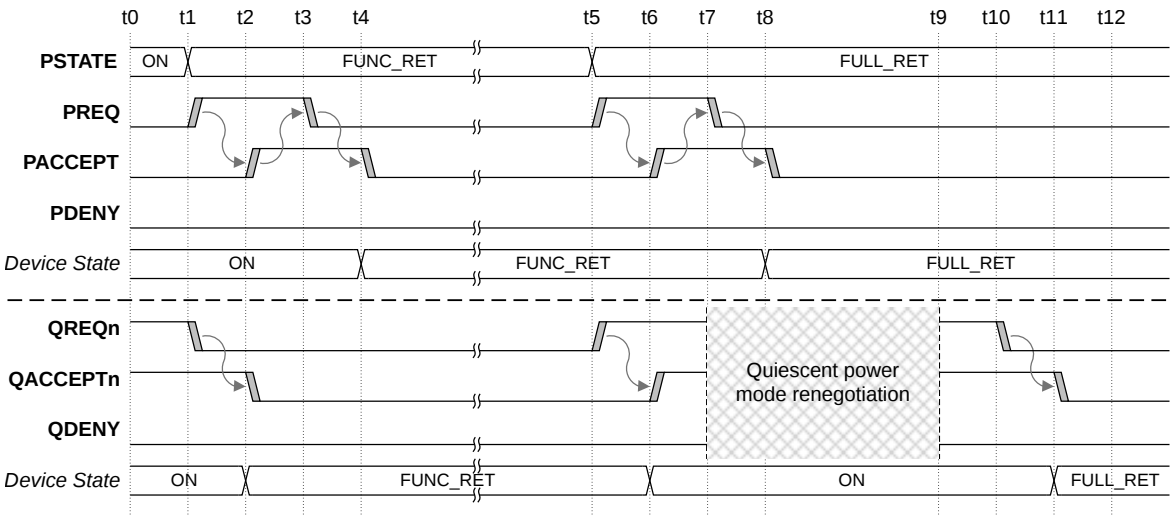


Figure 8.15: Q-Channel and P-Channel transitions between multiple modes

The P-Channel can transition directly from FUNC_RET to FULL_RET. However, the Q-Channel must transition back to ON, then renegotiate the quiescent power mode between the device and power controller, before the re-entering quiescence. Therefore, the P-Channel is a more suitable choice in this scenario.

8.3.2 General Power Control Low-Power Interface Implementation

A component must have a power control LPI for each of its power domains. This allows each component power domain to be controlled separately and for each component power domain to be combined with other components to form larger system power domains.

NOTE: With the PCSA supported power modes a single LPI can control a power domain with a single sub-domain with a known relationship and whose boundary is entirely within the controlled domain. This sub-domain might be memory or logic.

Power control LPIs must be separate from any clock control Q-Channels.

Power control LPIs are recommended to be responsive while their related clock control Q-Channel is in Q_STOPPED but the clock is running. All other interfaces must be managed as described in 8.1.3 *Interface Management*.

Low-Power Interface and Relationship to Power Domains

Depending on component structure, the low-power interface control logic can be implemented within either the managed power domain itself or within a parent power domain when a component supports nested parent-child power domain relationships.

Figure 8.16 shows a simple example with three power domains across two components:

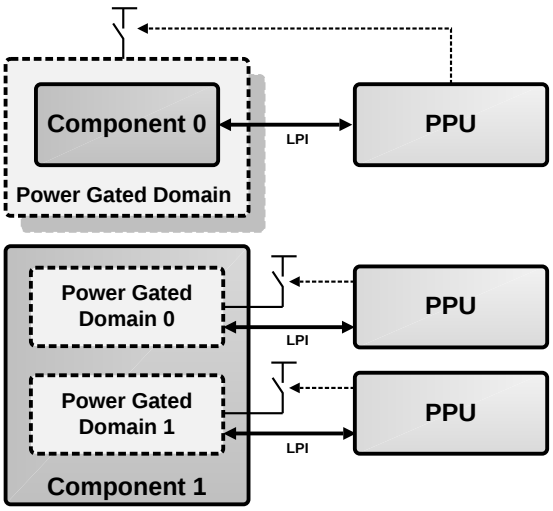


Figure 8.16: Simple power domain example

In Figure 8.16 component 0 is within one power domain and component 1 has two power domains. There is one LPI for each power domain and in all cases the LPI control logic is contained within the power domain it controls. In this arrangement each power domain can only make transitions between modes, with the corresponding LPI responses, when logic capability is available.

Also, the components cannot create wake-up requests, using either **QACTIVE** or **PACTIVE** from modes without logic in an operable state. The wake-up functionality must then be provided by the system. The wake-up request at system level can be either from hardware signaling, for example as a **QACTIVE** or **PACTIVE** contribution outside the component, or software programming of the PPU.

Transitions between different power modes without logic capability require intermediate transitions through an operable mode. Finally, since some logic capability is required at minimum to respond to LPI requests this arrangement is not suitable to support RAM only power domains.

Figure 8.17 shows an example component with a parent-child relationship between a primary power domain and two secondary sub-domains.

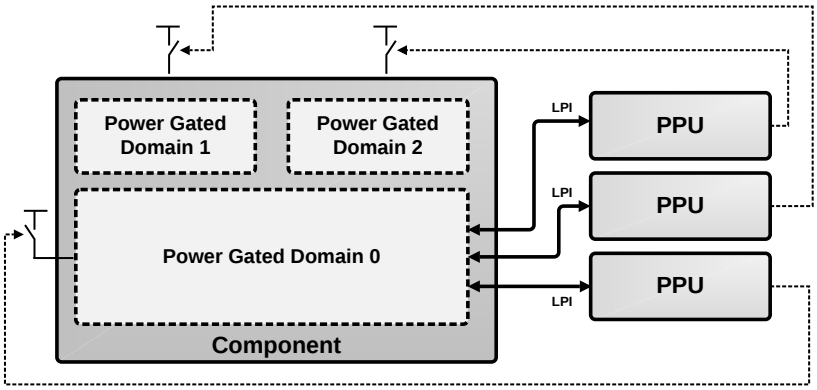


Figure 8.17: Parent-child power domain example

In Figure 8.17 power domain 0 acts as a parent domain with a first-on, last-off relationship to the child power domains, power domain 1 and power domain 2. All LPI interface and control logic is in power domain 0.

In this case, the device capabilities can include parent domain detection, and signaling, of wake conditions on behalf of secondary power domains, using either **QACTIVE** or **PACTIVE** to cause a transition to a higher power mode. This arrangement can be used in the absence of logic capability in a sub-domain, for example, a RAM only power domain.

Low-Power Interface Logic Reset

For a power domain with a single reset signal the low-power interface logic must use this reset. This applies even when the LPI logic is in another power domain, for example a parent domain in a component with primary and secondary domains as shown in Figure 8.17.

The requirement for initialization at reset in the P-Channel protocol has specific considerations, as the support for initialization is dependent on the reset used for the P-Channel logic.

Where a power domain has multiple resets the LPI logic is strongly recommended to use the power-on reset of the domain. The use of other resets, for example warm resets which affect only parts of the power domain, for the LPI logic is not recommended.

NOTE: Considerations for components with multiple resets and support for the power modes supported by the PPU architecture, outlined in 6.5.1 *Power Policy Unit*, are given in *ARM Power Policy Unit Architecture Specification* [6].

8.3.3 Power State Availability

While the capabilities of the current power mode are guaranteed, transitioning a component LPI to a lower power mode does not guarantee removal of the capabilities of a higher power mode.

There can be many reasons for this, but one example is another component within the power domain can deny a power mode transition therefore the controller will not take the actions associated with the power mode, for example applying reset.

Therefore, a component must not use the power mode conditions, for example reset being asserted, power being removed, or isolation being enabled, as a mechanism to ensure the component will operate correctly or be able to correctly transition to another power mode. The correct operation of the component must be guaranteed by logical means.

8.3.4 Power Control Q-Channel Guidelines

This section describes the implementation of power control using a Q-Channel.

A power control Q-Channel can only transition between a functional mode and a single low-power mode.

QACTIVE Behaviour

The power control **QACTIVE** signal is used for two purposes:

1. When HIGH: To indicate that the component is required to be active, regardless of the Q-Channel state. The supply of power is subject to the guarantees provided by the Q-Channel handshake.
2. When LOW: To hint that the component might accept a Q-Channel quiescence request to enter a low-power mode.
    a. The exact low power mode entered is either fixed or negotiated by some implementation defined means between the component and controller.

QACTIVE at Reset

The power control **QACTIVE** contribution from internal component state must be LOW at reset. This allows the component to remain in a low power mode until required by the system.

The rationale for this recommendation includes:

- If **QACTIVE** is set HIGH at reset assertion, taking powering off as an example, then when reset is applied, prior to power-off, **QACTIVE** would be HIGH indicating a requirement to exit the powered off state.
- If **QACTIVE** is set HIGH at reset assertion, then it is also required to be isolated HIGH for consistency of reset and isolation values. The component then indicates a requirement to exit the *Q_STOPPED* state when it is isolated throughout power-off and retention modes.

Components that are required to power on immediately after reset must have system level stimulus to provide this wake-up.

NOTE: Some components might be required to be powered-on by default at exit from a full system reset. The default mode of a power domain can be configured in the Power Policy Unit, see *ARM Power Policy Unit Architecture Specification* [6].

QACTIVE Stimuli

QACTIVE stimuli can be from several sources derived from the following:

- Internal logic state.
- External signaling.

Some specific considerations are discussed in the following sections.

QACTIVE Contribution from Internal State

Internal logic state contributions to a power control **QACTIVE** can be numerous and are not restricted. A critical consideration in power control is the loss of context that can result because of a low-power mode.

In the case of modes without loss of context, for example full retention, the conditions for setting **QACTIVE** LOW can depend only on internal hardware activity. Activity conditions can include the completion of a programmed task or lack of active transactions in an interconnect component. The setting of **QACTIVE** LOW then enables an opportunistic transition to the low-power mode.

Components that contain no software configuration or context, or that are expected to be reconfigured for each new task, can support the opportunistic approach for all modes and are recommended to drive **QACTIVE** LOW whenever possible.

The **QACTIVE** LOW assertion, using hardware activity conditions, can be according to an internal policy such as a timeout period after operations have completed, or anticipation that re-programming of context is required anyway for the next set of operations.

In the case of components that have software context or configuration the loss of context must be considered when RAM or logic will be powered-off in the low-power mode. Although the **QACTIVE** LOW condition will depend on similar activities to that described for retention modes it can consider, through register settings or machine state, that any required software actions to manage context are completed or the loss of context is acceptable.

QACTIVE Contribution for Low-Power Mode Exit

If a component power domain is in a power mode in which it cannot generate or propagate signals, for example powered-off, then wake signaling must come from outside the power domain. Although it can be possible in some low-power modes, this limits the use of component inputs for wake-up contributions as compared to clock control.

In components with multiple power domains arranged in a parent-child relationship, then a sub-domain **QACTIVE** wake-up contribution can be generated in the primary parent domain or even from another active sub-domain. While this is outside the power domain under control it is within the component and a standard LPI with all contributions included can be presented to the system.

QACTIVE contributions can be added at the system level between the component and the PPU. Examples include level sensitive interrupts, wake requests, or indications that a another component has a transaction to progress.

Figure 8.18 shows an example of an external power control **QACTIVE** contribution.

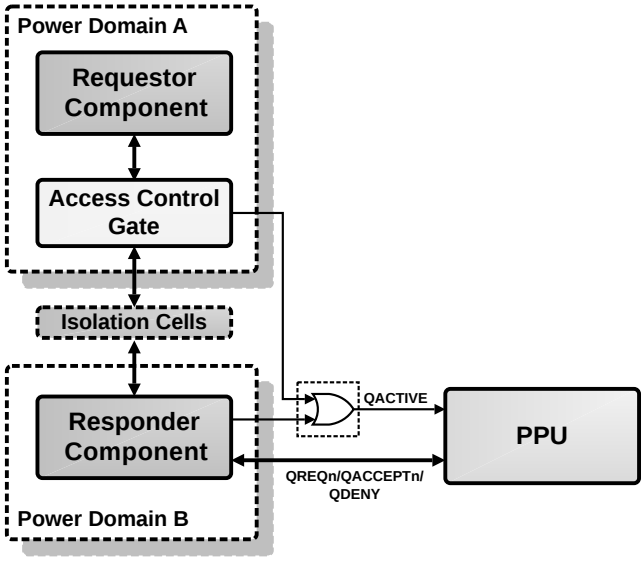


Figure 8.18: Example of external QACTIVE contribution

In Figure 8.18 a **QACTIVE** signal, indicating stalled transactions, from power domain A is used as a power control **QACTIVE** contribution between the component in power domain B and the PPU. The stalled transactions can then cause automatic power-on of power domain B when it is off, and transactions are waiting.

In the absence of suitable signaling then the low-power mode exit will require software programming of the PPU or equivalent.

Power Control Q-Channel Handshake

Q-Channel Handshake at Reset

From a component perspective, a power control Q-Channel will always exit reset in the *Q_STOPPED* state. Component **QACCEPTn** and **QDENY** outputs must be reset LOW.

Any resynchronization in the component on the **QREQn** input must be reset LOW.

The component must assume the state of the **QREQn** input is LOW at reset exit.

A controller can set **QREQn** LOW or HIGH at reset exit. When **QREQn** is set HIGH, the interface is in *Q_EXIT* state, but this will not be sampled by a component until its reset is released.

The reset state of the component should be equivalent to the quiescent state, with all required interface management in place.

Q-Channel Quiescence Requests

A quiescence request on the power control Q-Channel can be made to a component at any time regardless of the state of the **QACTIVE** signal.

A component must only wait to accept a Q-Channel quiescence request when a response can be given within a reasonably bounded time. The length of this bounded time is not limited and depends upon the actions required to accept the request, but can include transactions initiated by the component itself, such as cache flushes, required to enter quiescence even if the response time to those transactions is not guaranteed. It should not include waiting for a cessation of transactions from an external requestor but can include completing already accepted transactions even if the response time to those transactions is not guaranteed.

Some components can be required to always complete a sequence and enter the low-power mode regardless of their status. An example is a domain bridge that when requested to enter a quiescent state, prior to power-off, might first block incoming transactions to prevent any new transactions, but then complete any outstanding transactions, before accepting the quiescence request. This might take some time, depending on the traffic activity through the domain bridge, but the desired response of the system is for the request to complete rather than be denied.

Actions Required to Enter a Low-Power Mode

A component must ensure that when it accepts a request there will be no functional failure because of the low-power mode.

The requirements can change depending on the low-power mode which is requested but include ensuring:

- There is no internal activity requirement.
- All outputs and interfaces are managed as described in 8.1.2 *Output Management* and 8.1.3 *Interface Management*.
  - This includes internal interfaces where a power domain boundary is internal to the component.

All required actions must be completed before an accept response is sent by setting **QACCEPTn** LOW. Once the accept response is sent the capabilities of the previous power mode are not guaranteed.

Q-Channel Quiescence Exit

At exit from the low power mode once a component has sampled **QREQn** HIGH, in the interface *Q_EXIT* state, it can resume operations. It does not have to wait until it has accepted the request by setting **QACCEPTn** HIGH.

The controller cannot make another quiescence request until **QACCEPTn** is HIGH. The component can use this to prevent another request until it has carried out any required initialization sequence. Alternatively, it can respond immediately and deny any quiescence requests until it is idle.

Unused Power Control Q-Channel

The component must operate correctly if the integrator chooses to not use a power control Q-Channel. In this case, the component can assume that the power will always be available. It becomes the responsibility of the integrator to ensure the power domain is managed correctly.

When the Q-Channel is unused, **QREQn** must be tied HIGH.

For components with multiple power control Q-Channels, if one channel is not used it must not limit the use of other clock or power control LPIs. This allows the integrator to choose which power domains they implement.

For instance, an integrator might not implement a supported sub-domain of a component while still implementing a supported top-level power domain. In this case, the integrator can tie off the sub-domain LPI, while still using the top-level power domain LPI to provide overall component power control.

8.3.5 Power Control P-Channel Guidelines

This section describes the implementation of power control using a P-Channel.

A P-Channel can switch between multiple low-power modes. These modes and allowed transitions are specified by the component.

PSTATE and PACTIVE Usage

While the arrangement of **PSTATE** values and **PACTIVE** bits is not restricted by the *Low-Power Interface Specification, ARM Q-Channel and P-Channel Interfaces* [4] the following sections in this document specify an implementation to be followed to align to the PCSA and allow interoperability with other components and the power control framework infrastructure.

Power Modes

The PCSA defines the **PSTATE** and **PACTIVE** bit values related to the defined power modes. Components must use these values. These are listed in Table 6.1 and Table 6.2 and summarized below in Table 8.2.

Table 8.2: Recommended PSTATE and PACTIVE usage

Power Mode	PACTIVE bit	PSTATE [3:0]	Mode Priority
DBG_RECOV	10	0b1010	High
WARM_RST	9	0b1001	
ON	8	0b1000	
FUNC_RET	7	0b0111	
MEM_OFF	6	0b0110	
FULL_RET	5	0b0101	
LOGIC_RET	4	0b0100	
MEM_RET_EMU	3	0b0011	
MEM_RET	2	0b0010	
OFF_EMU	1	0b0001	
OFF	0	0b0000	Low

Typically, a component only implements a subset of the power modes. For unused power modes the component must declare the unused **PSTATE** values as RESERVED and tie the corresponding **PACTIVE** outputs LOW, rather than omit them.

PACTIVE bit 0, the indication for the OFF power-mode must be tied LOW in all cases. If all other **PACTIVE** outputs are LOW, then OFF is the default minimum required power mode regardless of the setting of this bit therefore setting it LOW simplifies component design and allows for a simpler isolation policy.

Operating Modes

Operating modes are component specific. In general, operating modes are intended to be used by more sophisticated components, where the additional complexity to both component design and integration are justified.

For further information on operating modes see the *ARM^ Power Policy Unit Architecture Specification* [6].

Any implementation of operating modes must maintain interoperability with the PPU specified operation.

Operating mode **PACTIVE** outputs are considered separate from power mode **PACTIVE** outputs, therefore there is no requirement to set operating mode **PACTIVE** outputs LOW to facilitate the requirement for a power mode.

If operating modes are included there must be an operating mode for all power modes that have operating mode context. This increases the compatibility between system components when placed together in a domain.

As an example, consider the power domain shown in Figure 8.19.

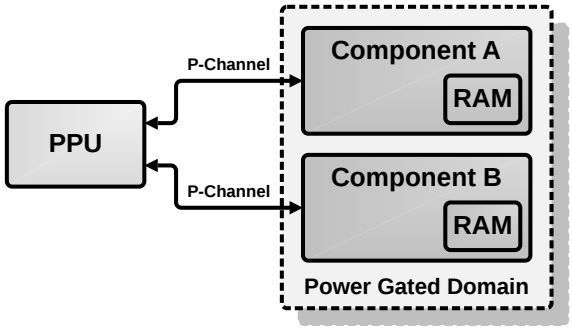


Figure 8.19: Example power gated domain

Component A supports the ON, MEM_RET and OFF power modes and has three operating modes controlling the portions of RAM that are powered on. The operating modes are No RAM, Half RAM and Full RAM.

Component B supports the ON, OFF and MEM_RET power modes without any operating modes.

Component A might not implement a MEM_RET power mode associated with the No RAM operating mode, MEM_RET(No RAM), as there is no RAM powered on to be retained making this mode essentially the same as OFF.

In this case, as shown in Figure 8.20, if component A is in ON(No RAM) and the PPU requests MEM_RET, component A will deny the transition and therefore the domain cannot enter MEM_RET. In the case, the power mode might still be useful for component B.

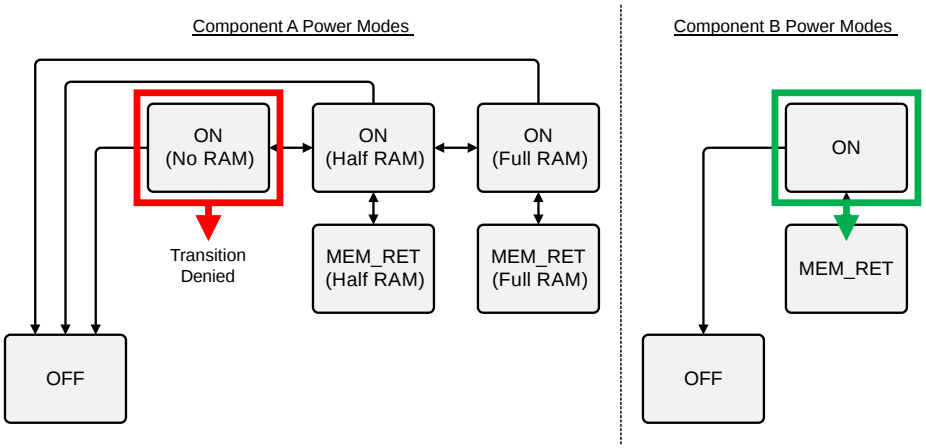


Figure 8.20: Example power transitions without a MEM_RET (No RAM) mode

If component A does implement MEM_RET(No RAM) then, as shown in Figure 8.21, if component A is in ON(No RAM) and the PPU requests MEM_RET, component A will accept the transition and the domain can enter MEM_RET.

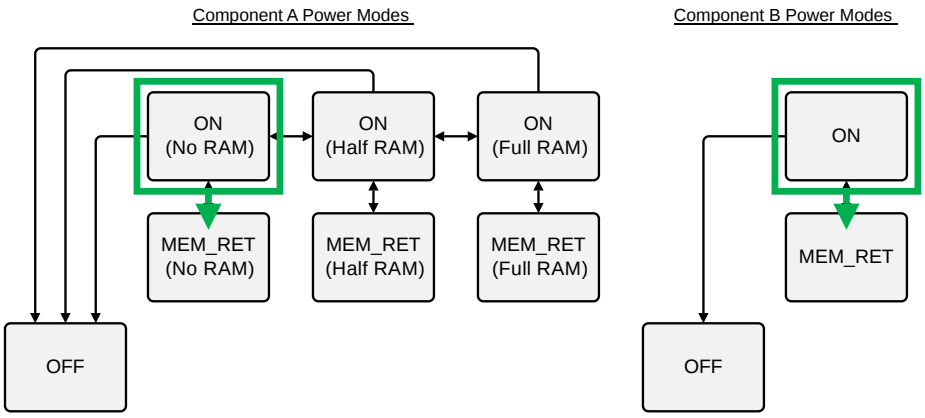


Figure 8.21: Example power transitions with a MEM_RET (No RAM) mode

Mode Transitions

A component must support the transitions as described in the *ARM Power Policy Unit Architecture Specification* [6] for the power modes it supports. It is not recommended to support additional power mode transitions not specified in this document.

A component must accept a legal transition request to a power mode at, or higher than, the minimum required as indicated on the **PACTIVE** bits.

Components must support transitions to the same mode they are already in. Accepting the transition with no change in component function is typically all that is required.

Components with the MEM_RET power mode must support the transition from OFF to MEM_RET. This allows for power control infrastructure to be powered-off while the component is in MEM_RET. The component can then be returned to ON through an OFF to MEM_RET to ON sequence while maintaining RAM contents.

PACTIVE Policy

The power mode **PACTIVE** output bits are used to indicate component power mode requirements to the power controller.

Each **PACTIVE** bit indicates the requirement for a power mode supported by the component. When HIGH it indicates a requirement for the specified mode. When LOW it is a hint that the power mode is not required. Multiple **PACTIVE** bits can be HIGH simultaneously and each bit can change independently.

The most significant **PACTIVE** bit that is set HIGH indicates the minimum power mode required by the component.

The relationship of **PACTIVE** bits to power modes is described in *PSTATE and PACTIVE Usage*.

The indication of the minimum required power mode on the **PACTIVE** bits is not required to be a power mode that is a direct transition from the current power mode.

As the **PACTIVE** bits specify a minimum required power mode then:

- if the **PACTIVE** bits indicate a minimum requirement higher than the current component power mode, then this is a requirement for the controller to move to the minimum requested power mode or a higher to progress operations. Failure to respond to such a requirement could result in a deadlock.
- if the **PACTIVE** bits indicate a minimum requirement lower than the current component power mode, then this is a hint for the controller that the component will accept a transition to this minimum requested power mode or higher.

PACTIVE at Reset

PACTIVE outputs must be LOW at reset. The rationale for this includes:

- If a **PACTIVE** bit is set HIGH at reset assertion, then when reset is applied prior to power-off that **PACTIVE** bit would then indicate a requirement to transition to that mode as a minimum power mode when it might not be required.
- If a **PACTIVE** bit is set HIGH at reset assertion, then it would also be required to be isolated HIGH for consistency of reset and isolation values. The component then indicates a requirement to transition to that mode as a minimum power mode when the output is isolated in power-off and retention modes.

Components that are required to power on immediately after reset must have system level stimulus to provide this wake-up.

NOTE: Some components might be required to be powered-on by default at exit from a full system reset. The default mode of a power domain can be configured in the Power Policy Unit, see the *ARM Power Policy Unit Architecture Specification* [6].

PACTIVE Stimulus

The stimulus for the **PACTIVE** bits typically comes from sources derived from the following:

- Internal logic.
- External signaling.

Some specifics are discussed in the following sections.

PACTIVE Contribution from Internal State

The considerations for **PACTIVE** bit contributions from internal component state are like those detailed for a power control **QACTIVE** as detailed in 8.3.4 *QACTIVE Contribution from Internal State*.

Since multiple power modes are supported, and the requirements for each power mode can be different, the contributions for each **PACTIVE** must be considered individually.

It is important to consider the priority of each mode at a given time to allow indication of the correct minimum power mode. For example, when the conditions for entering a FULL_RET power mode are met, but the additional conditions for entering an OFF power mode are also met. In this case the retention mode **PACTIVE** must be set LOW to indicate that the minimum required power mode is OFF.

PACTIVE Contribution for Low-Power Mode Exit

The considerations for **PACTIVE** bit contributions for low-power mode exit from external signals are like those detailed for a power control **QACTIVE** as detailed in 8.3.4 *QACTIVE Contribution for Low-Power Mode Exit*.

However, it is possible to use component internal or input signals as low power exit contributions to **PACTIVE** outputs in modes where some component logic is still operational such as FUNC_RET and MEM_OFF.

PACTIVE and Isolation

PACTIVE outputs are generally exempt from component output management, even if they are to be isolated to a different level the output level should be maintained when entering a power mode. This allows the external system to capture the value and maintain it externally to capture the requirement for the power or operating mode before isolation is enabled. Methods for capturing these values in the system is discussed in 7.2.9 *Output Isolation and PACTIVE*.

P-Channel Handshake

P-Channel Handshake at Reset

From a component perspective, a P-Channel will always exit reset in the *P_STABLE* state, **PACCEPT** and **PDENY** outputs must be reset LOW.

The component must assume the state of the **PREQ** input is LOW at reset exit, resynchronization of the **PREQ** in the component must be reset LOW.

At reset a component must assume an implicit power mode of OFF, with the associated component functional state, such as interface management. Any **PSTATE** initialization or subsequent P-Channel transition must be considered a transition from OFF to the requested state with the same actions normally associated with this transition.

P-Channel Requests

A request on a P-Channel can be made to a component at any time regardless of the state of the **PACTIVE** signals. When a request is made, the component should respond in a timely manner.

A component must only wait to accept a P-Channel quiescence request when a response can be given within a reasonably bounded time. The length of this bounded time is not limited and depends upon the actions required to accept the request, but can include transactions initiated by the component itself, such as cache flushes, required to enter quiescence even if the response time to those transactions is not guaranteed. It should not include waiting for a cessation of transactions from an external requestor but can include completing already accepted transactions even if the response time to those transactions is not guaranteed.

Some components can be required to always complete a sequence and enter the low-power mode regardless of their status. An example is a domain bridge that when requested to enter a quiescent state, prior to power-off, might first block incoming transactions to prevent any new transactions, but then complete any outstanding transactions, before accepting the quiescence request. This might take some time, depending on the traffic activity through the domain bridge, but the desired response of the system is for the request to complete rather than be denied.

When moving to a higher power mode the capabilities of that mode are available as soon as the component samples the **PREQ** input HIGH with the corresponding **PSTATE** value.

Therefore, the capabilities of this higher mode can be used before the P-Channel transition is completed. However, the controller cannot make another request until the accept response handshake is completed and the interface returns to the *P_STABLE* state. The component can use this to prevent another request until it has carried out any required actions by delaying the handshake completion, from *P_COMPLETE* to *P_STABLE*, by keeping **PACCEPT** HIGH until ready. This can prevent the verification overhead of having to support P-Channel denial responses during this period.

Figure 8.22 shows an example of this usage.

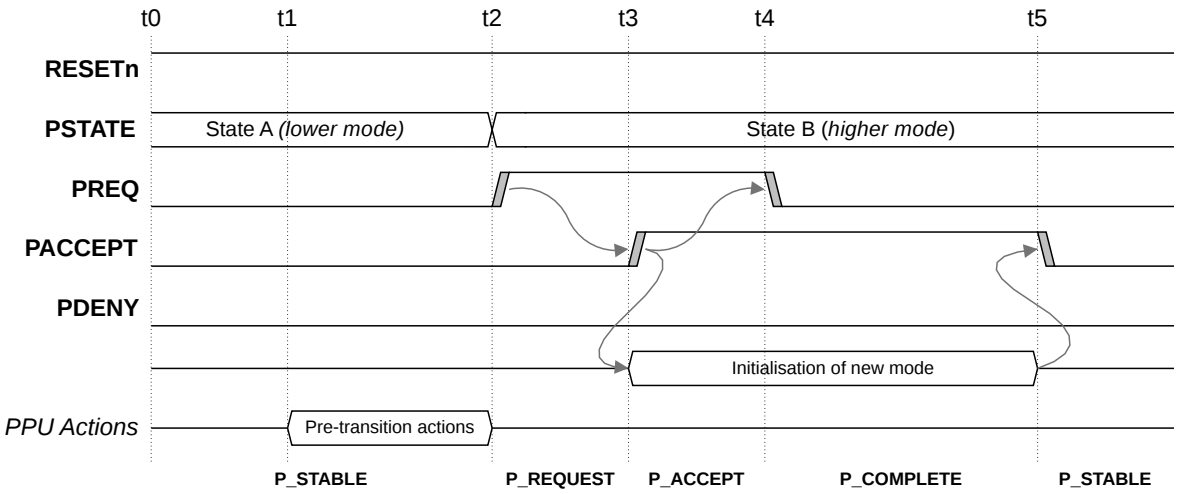


Figure 8.22: Component delayed completion on transition to higher mode

When moving to a lower power mode the reduced capabilities of that mode do not take effect until the P-Channel transition is complete. From a component perspective, this is when it sets **PACCEPT** LOW to bring the P-Channel back to the *P_STABLE* state.

Therefore, to ensure correct operation the component must complete any actions required to change mode before accepting the request. It is recommended to do this before moving to the *P_ACCEPT* state, with **PACCEPT** HIGH, rather than prior to setting **PACCEPT** LOW at the *P_COMPLETE* to *P_STABLE* transition. This allows the component to attempt actions and if not successful, or other activity is required before these actions are complete, then provide a denial response.

Figure 8.23 shows an example of this usage.

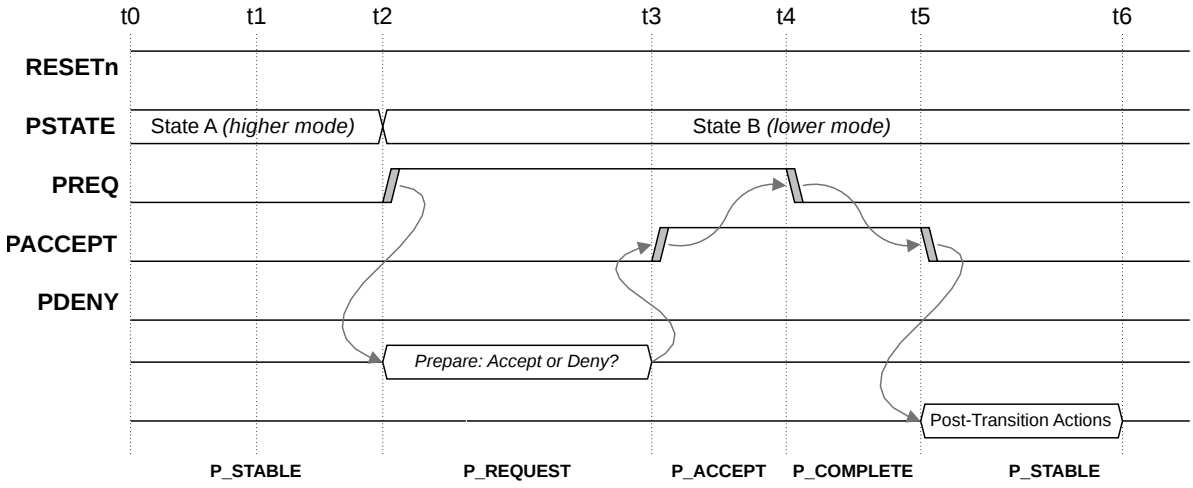


Figure 8.23: Component delayed acceptance on transition to lower mode

Denied Requests

The *Low-Power Interface Specification, ARM Q-Channel and P-Channel Interface* [4] requires a component to specify which supported transitions might be conditionally denied.

In addition to denying requests when the internal conditions are inappropriate, components are strongly recommended to deny requests when:

- The mode requested is not a legal mode of the component.
  - **PSTATE** is an invalid value.
- A transition to the requested mode, from the current mode, is not supported.

While this is not required by the P-Channel protocol it prevents deadlock in the case of incorrect controller actions and aids debug of these scenarios.

Actions Required to Enter a Low-Power Mode

A component must ensure that when it accepts a request there will be no functional failure because of the transition to the target power mode.

The requirements can change depending on the power mode which is requested but include ensuring:

- There is no internal activity requirement.
- All outputs and interfaces are managed as described in 8.1.2 *Output Management* and 8.1.3 *Interface Management*.
  - This includes internal interfaces where a power domain boundary is internal to the component.

All required actions must be completed before the P-Channel transition is complete. It is recommended that all actions are completed before **PACCEPT** is asserted HIGH to accept the transition. However, it must be done before **PACCEPT** is set LOW to complete the P-Channel transition as from this point the capabilities of the previous power mode are not guaranteed.

Additionally, for power modes where a part of the component is operational, but another part is off or in retention, the clock is still required to operate parts of the component so cannot be gated externally. Therefore, internal clock gating within the component is required for the parts that are off or in retention for the duration of the low power mode. Examples of these power modes include FUNC_RET and MEM_OFF.

Unused P-Channel

To allow a P-Channel to be unused a component must support initialization into at least one functional mode directly from reset. This is typically a functional mode where all features of the component are available.

An unused P-Channel has **PREQ** tied LOW and **PSTATE** tied to this initialization mode value, that must be sampled by the component at reset exit as described in 8.1.1 *Capturing PSTATE*.

The modes that the component can enter at reset exit must be specified by the component.

For components with multiple P-Channels if one channel is tied off it must not limit the use of other power control channels. This allows the integrator to choose which domains they will implement.

For instance, an integrator might not implement supported sub-domains of a component while still implementing a supported top-level power domain. In this case, the integrator can tie off the sub-domain LPI while still using the top-level power domain LPI to provide overall component power control.

8.3.6 Power Control Low-Power Interface Naming Guidelines

Consistent and meaningful naming according to function eases integration and prevents misuse by identifying the use, and associated domain, of the low-power interface. Additionally, by using prefixes, when sorted alphabetically, in simulation waves or lists, the low-power interfaces will appear by domain. An alternative approach is to add the clock name as a post-fix.

For a power control Q-Channel an initial PWR prefix on signals is recommended to ensure clear distinction from clock control Q-Channels. Where there are multiple Q-Channels the PWR prefix is recommended to be appended with the power domain name.

Table 8.3 shows an example for a component with a single power control channel.

Table 8.3: Power Control Q-Channel Naming Convention Example

Naming Convention	Example
(PWR/<*domain*>)QACTIVE	**PWRQACTIVE**
(PWR/<*domain*>)QREQn	**PWRQREQn**
(PWR/<*domain*>)QACCEPTn	**PWRQACCEPTn**
(PWR/<*domain*>)QDENY	**PWRQDENY**

NOTE: When a Q-Channel is used for power control the use of a P signal prefix alone is strongly recommended to be avoided. This is to prevent potential confusion with P-Channel naming.

For P-Channel where there is a single power control channel no prefix is required as the use for power control is unambiguous. Where there are multiple P-Channel interfaces, the signal names are recommended to be prefixed with the name of the domain they are controlling. An alternative approach is to add the clock name as a post-fix.

Table 8.4 shows an example for a component with multiple power control channels.

Table 8.4: Power Control P-Channel Naming Convention Example

Naming Convention	Example
<*domain*>PACTIVE	**CPUPACTIVE**
<*domain*>PREQ	**CPUPREQ**
<*domain*>PACCEPT	**CPUPACCEPT**
<*domain*>PDENY	**CPUPDENY**

Part A
Glossary

ACP

Accelerator Coherency Port. This is responder port on a processor which allows peripherals to access the system though a processor's shared cache, allowing them to be coherent with that processor. Accesses to this port can cause allocations to the cache.

ACPI

Advanced Configuration and Power Interface. http://www.uefi.org/acpi/specs

Always-On Domain

A voltage or power domain which is always-on compared to all other power domains on the SoC.

AON

Always-On (Domain).

AP

Application processor.

AP core

A processor core in the system running within the application processor software stack.

big.LITTLE

big.LITTLE is an Arm heterogeneous multiprocessing technology. High-performance Arm cores are combined with the most efficient Arm cores to deliver peak-performance, higher sustained throughput, and increased parallel processing performance, at significantly lower average power.

BSP

Board Support Package. The board support package is platform specific support software which conforms to a given operating system.

CCI

Cache Coherent Interconnect.

CCN

Cache Coherent Network.

Clock Domain

A collection of design elements supplied with a common clock source. Other clock domains which interact with the domain might be synchronous, but with independent source activity control, or asynchronous.

DAP

Debug Access Port

DDR

Double Data Rate.

Domain Bridge

A component which passes a protocol transaction from one domain to another, this includes voltage, power, and clock domains or a combination.

DRAM

Dynamic Random-Access Memory.

DVFS

Dynamic Voltage and Frequency Scaling

GIC

Generic Interrupt Controller.

GPU

Graphics Processing Unit.

HN

Home Node.

LPI

Low-Power Interface.

MMU

Memory Management Unit. A component which performs virtual to physical address translations.

MP

Multiprocessor *or* multiprocessing.

OSPM

Operating System Power Management software.

PHY

Physical Interface. A specialized input-output interfacing component, typically with timing or data recovery capabilities, such as for interfacing a SoC to DRAM or high-speed peripherals.

Power Domain

A collection of design elements within a voltage domain that share common power control. A voltage domain can have one or more power domains.

Power-Gated Domain

A power domain whose power can be removed by on-chip power switches.

PPI

Private Peripheral Interrupt.

PPU

Power Policy Unit.

RAON

Relatively Always-On (Domain)

RAS

Reliability and Serviceability.

Relatively Always-On Domain

A voltage or power domain which is always-on relative to another domain. It might be powered-off but only once the specified domain is off and must be on before the specified domain is powered-on.

RN

Request Node.

SAM

System Address Map.

SCMI

System Control and Management Interface.

SCP

System Control Processor. A processor-based capability that provides a flexible and extensible platform for provision of power management functions and services.

SCU

Snoop Control Unit.

SoC

System on Chip.

SPI

Shared Peripheral Interrupt.

UPF

Unified Power Format. The IEEE 1801 standard for expressing power intent to verification and implementation flows.

Voltage Domain

A collection of design elements supplied by a single voltage source. The voltage supply to the domain might be scaled or removed for power or performance reasons.