ARM® Power Control System Architecture

Version 1.0

Architecture Specification

Confidential

Power Control System Architecture Specification

Copyright © 2015 ARM. All rights reserved.

Release Information

The following changes have been made to this specification.

Change History

Date	Issue	Confidentiality	Change
15 July 2015	A	Confidential	Beta
15 December 2015	B	Confidential	Release 1.0

Web Address

http://www.arm.com

Contents

Power Control System Architecture Specification

1 Preface

This specification describes the Power Control System Architecture. This section contains the following subsections:

- *About this Specification* on page 1-2.
- *Using this Specification* on page 1-3.
- *Conventions* on page 1-4.
- *Additional Reading* on page 1-6.
- *Feedback on Documentation* on page 1-7.

1.1 About this Specification

This specification describes an approach to the *Power Control System Architecture* of SoC based on ARM components. It defines version 1.0 of the Power Control System Architecture (PCSA).

Version 1.0 of PCSA primarily addresses small scale single-chip systems based on ARM components.

Future versions of PCSA are envisaged to also address larger scale and multi-chip systems in addition to keeping pace with the evolution of ARM components from a power control perspective.

1.1.1 Intended Audience

There are two intended audiences for this specification:

- SoC architects and designers designing power managed System-on-chip (SoC) 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.

1.2 Using this Specification

This specification is organized into the following chapters:

Chapter 1 Preface

Read this for an introduction to this specification.

Chapter 2 Background

Read this for background context to Power Control System Architecture (PCSA).

Chapter 3 Introduction

Read this for an introduction PCSA.

Chapter 4 Overview

Read this for an overview of concepts used in PSCA.

Chapter 5 System Partitioning

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

Chapter 6 Power States

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

Chapter 7 Power Control Framework

Read this for a description of power control framework concepts and components.

Chapter 8 System Power Control Integration

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

Chapter 9 Power Control Flows

Read this for information on power control flows for specific ARM components.

Chapter 10 Component Design Considerations

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

Chapter 11 Glossary

Read this for definitions of terms and abbreviations.

1.3 Conventions

This section describes conventions used in this specification.

Typographical conventions

The following typographical conventions are used:

italic	Introduces special terminology, denotes internal cross-references, and citations.
bold	Denotes signal names, and is used for emphasis in descriptive lists, where appropriate.
SMALL CAPITALS	Used for a few terms that have specific technical meanings.

——— **Note** ————————

All Q-Channel and P-Channel interface state references are in italics, for example *Q_STOPPED*, and refer to the Q-Channel and P-Channel state names in the *Low Power Interface Specification, ARM® Q-Channel and P-Channel Interfaces*.

————————————————

Timing diagrams

Figure 1-1 shows the conventions used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

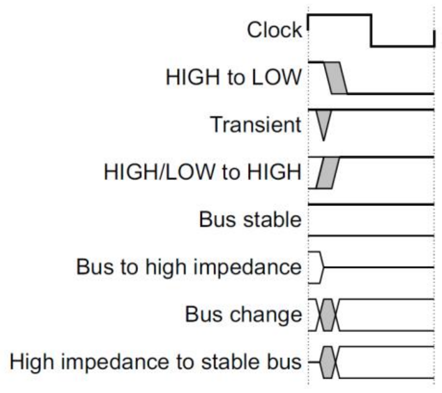


Figure 1-1 Key to timing diagram conventions

Timing diagrams sometimes show single-bit signals as HIGH and LOW at the same time and they look similar to the bus change shown in Figure 1-1. If a timing diagram shows a single-bit signal in this way, then its value does not affect the accompanying description.

Signals

In general, this specification does not define hardware signals, but it does include some signal examples and recommendations. The signal conventions are:

Signal level The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

Lower-case n At the start or end of a signal name denotes an active-LOW signal.

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by 0b, and hexadecimal numbers by 0x. 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.

1.4 Additional Reading

This section lists relevant publications from ARM and third parties.

1.4.1 ARM Publications

See the Infocenter, http://infocenter.arm.com for access to the following ARM documentation:

- *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition* (ARM DDI 0406).
- *ARM ®Architecture Reference Manual, ARMv8 for ARMv8-A architecture profile* (ARM DDI 0487).
- *ARM® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* (ARM IHI 0031).
- *ARM CoreSight™ SoC-400 Technical Reference Manual* (ARM DDI 0480).
- *AMBA® AXI™ and ACE™ Protocol Specification* (ARM IHI 0022).
- *Low Power Interface Specification, ARM Q-Channel and P-Channel Interfaces* (ARM IHI 0068).
- *ARM® Generic Interrupt Controller Architecture Version 2.0 Specification* (ARM IHI 0048).
- *ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0* (ARM IHI 0069)
- *ARM® CoreLink™ GIC-400 Generic Interrupt Controller Technical Reference Manual r0p1* (ARM DDI 0471).
- *ARM® CoreLink™ GIC-500 Generic Interrupt Controller Technical Reference Manual r0p0* (ARM DDI 0516).
- *ARM® CoreLink™ CCI-400 Cache Coherent Interconnect Technical Reference Manual r1p4* (ARM DDI 0470).
- *ARM® CoreLink™ CCN-504 Cache Coherent Network Technical Reference Manual r2p1* (ARM 100017_0201_00_en).
- *ARM® System Memory Management Unit Architecture Specification Version 2.0* (ARM IHI 0062).
- ARM® *CoreLink™ MMU-500 System Memory Management Unit Technical Reference Manual* (ARM DDI 0517).
- *ARM® Cortex-M3 Device, Generic User Guide* (ARM DUI 0552).
- *Power State Coordination Interface* (ARM DEN 0022).
- *Server Base System Architecture (*ARM DEN 0029*).*

For access to the following documentation please contact ARM:

- *ARM® Power Policy Unit, Architecture Specification Version 1.0* (ARM DEN 0051).
- *ARM® Clock Controller, Architecture Specification Version 1.0* (ARM DEN 0052).
- *Client Base System Architecture* (ARM DEN 0045).
- *Trusted Board Boot Requirements – CLIENT* (ARM DEN 0006).

1.5 Feedback on Documentation

If you have comments on this documentation, e-mail errata@arm.com. Provide:

- The title.
- The number, ARM DEN 0050B.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

2 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 a number of 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 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 an efficient system with an improved time to market. This increases the time and resource available for differentiating activities.

3 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.

3.1 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.

——————————————

4 Overview

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

4.1 Power Control Challenges

Figure 4-1 shows a simplified SoC example. The example is illustrated in terms of high level functions.

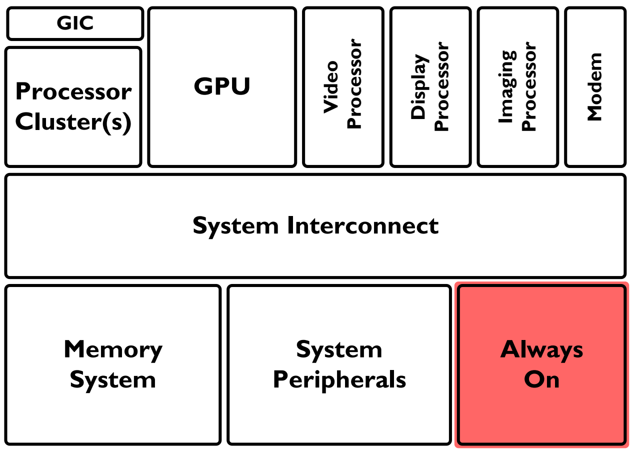


Figure 4-1 - Example SoC system

The example is mobile centric, but can be used to demonstrate power management challenges at a high level in any SoC configuration. It is notable that mobile SoC typically have very high functional diversity, compared to SoCs that are designed for other markets, which increases power management complexity.

In addition to the primary functions that the example shows, including processors, communications functionality and common system functions, an always on area is identified. 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, in practice, 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 through the use of standard infrastructure components, low power interfaces and associated methods. This approach is referred to as the *Power Control Framework*.

4.2 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 4-2 shows a concept level illustration of the SCP.

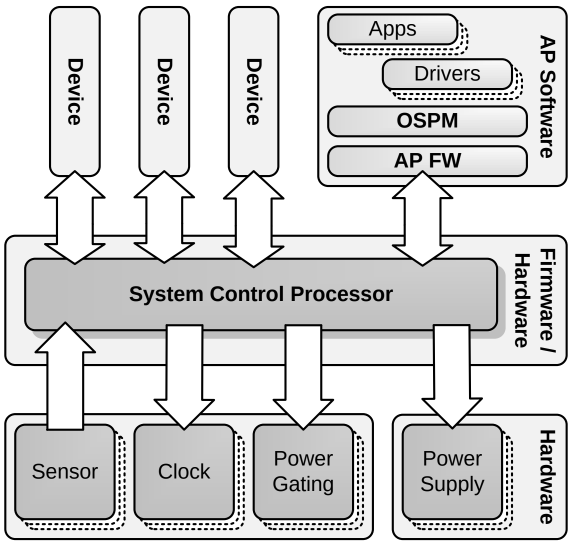


Figure 4-2 - System Control Processor concept

In the upper part of Figure 4-2, the application processor (AP) software stack is shown as a requestor of SCP services. Other devices in the system might also have the capability to directly generate requests for resources that the SCP controls. The SCP reconciles requests for shared resources from all of these agents.

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 *System Control Processor* on page 7-4.

4.2.1 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.
- **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-down 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 OSPM and device requests 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.
- **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 a number 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.

4.2.2 Trusted Operation

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

See the *Trusted Board Boot Requirements – CLIENT* specification for more information on this topic.

4.3 Power Management Software

Figure 4-3 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.

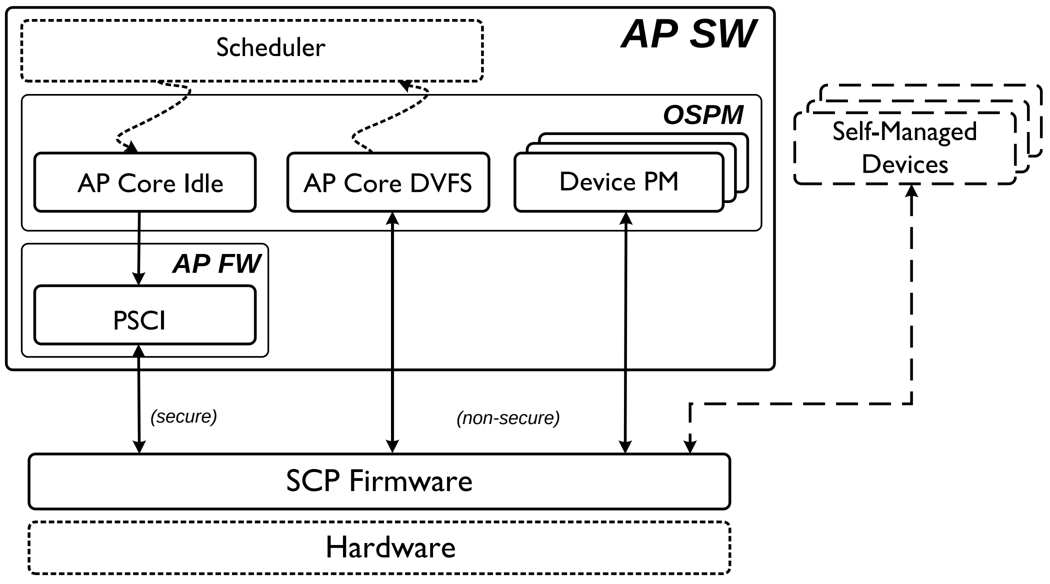


Figure 4-3 - Simplified power management software stack

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

- Core power management.
- Device power management.

4.3.1 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 4-3, 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 different 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 4-3 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 AP software stack as a whole.

————————————

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 4-3 shows that DVFS requests, resulting from the interaction between the OSPM frameworks, are sent through an SCP 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 energy and thermal optimization policy. This enables the path where OSPM only provides a high level performance request and the SCP evaluates all available techniques and considerations to provide an optimized performance point.

4.3.2 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 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 4-3 self-managed devices, depending on capability, might have an independent software interface path from their own software stacks to directly request SCP actions.

4.3.3 System Control Processor Firmware

The SCP firmware is implementation specific. While an overview of anticipated capabilities is given in *Services* on page 4-3, 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.
- **System services:** Services provided by the SCP without OSPM direction.

The SCP software interface is expected to provide, at minimum, commands for 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.

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 support response 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, such as, for example, an arriving wake request that must be delayed until a dependent power off sequence is complete.

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.

4.4 Power Control Framework

A primary aim of PCSA is to describe a standard approach for system power control integration of SoCs based on ARM components. A key component of this approach is the power control framework. Figure 4-4 shows a high level illustration of power control framework concepts.

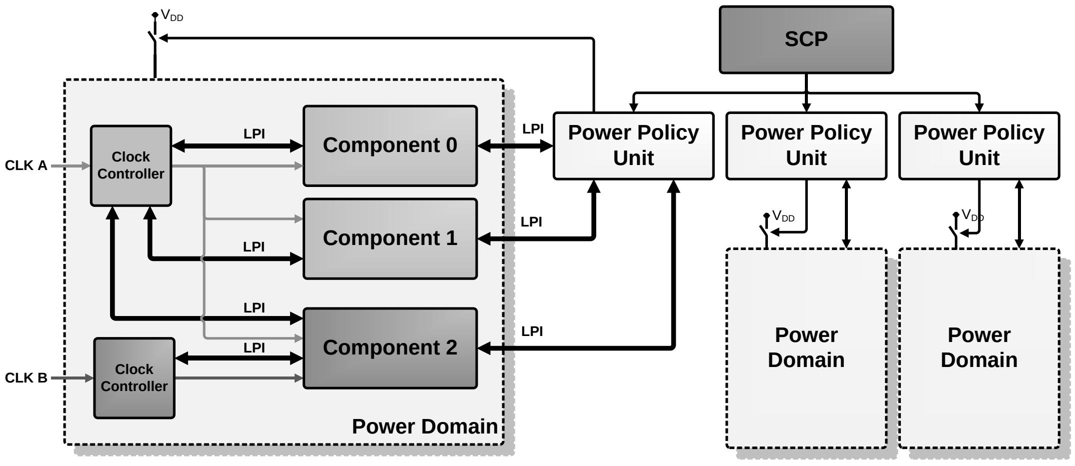


Figure 4-4 - Power control framework concept

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 component known as the *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.

Clock controller components are targeted at components supporting high level clock gating, which includes many ARM CoreLink^{TM} system components. This approach enables the clock to be gated high in the tree when components are idle.

For a detailed description of the power control framework, see *Power Control Framework* on page 7-1.

5 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:

- *Voltage domains* on page 5-2.
- *Power domains* on page 5-5.

——— **Note** ———————

In many SoC operating scenarios, dynamic power consumption is dominant and clocking strategy is therefore critical. This topic is addressed in the following sections of this specification:

- In *Clock Control Integration* on page 8-2, from a high level clock gating and implementation perspective.
- In *System Example* on page 9-47, from a clock domain partitioning perspective, showing by example, that the considerations are highly system specific.

———————————————

5.1 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 I/O, analog functions and logic domains. In this document the scope is limited to 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 of 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.

5.1.1 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.

5.1.2 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 *System Logic* on page 5-2.

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 *Always On Logic* on page 5-7.

5.1.3 Processor Clusters

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

In a big.LITTLE system, each cluster is required to have a dedicated voltage supply. This is a critical success factor when combined with big.LITTLE software.

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

In applications where DVFS is not required, or the cost is too high, then the processor cluster is considered to be a member of V_{SYS}.

5.1.4 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 considered to be a member of V_{SYS}.

5.1.5 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 also 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 in the form of function groupings, 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 considered to be a member of V_{SYS}.

5.1.6 SoC Partitioning Examples

Figure 5-1 and Figure 5-2 adapt the components of Figure 4-1 to provide simplified examples of low-cost and high-end mobile systems, respectively.

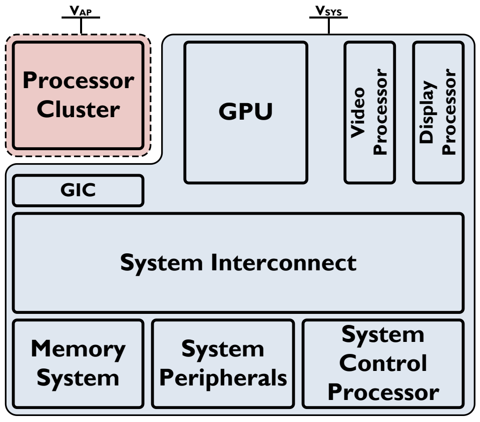


Figure 5-1 - Voltage domains: Low cost mobile SoC example

Figure 5-1 is a simplified illustration of a low cost mobile SoC. Processor DVFS is supported as a minimum requirement. All other primary logic domains share a supply with the system logic.

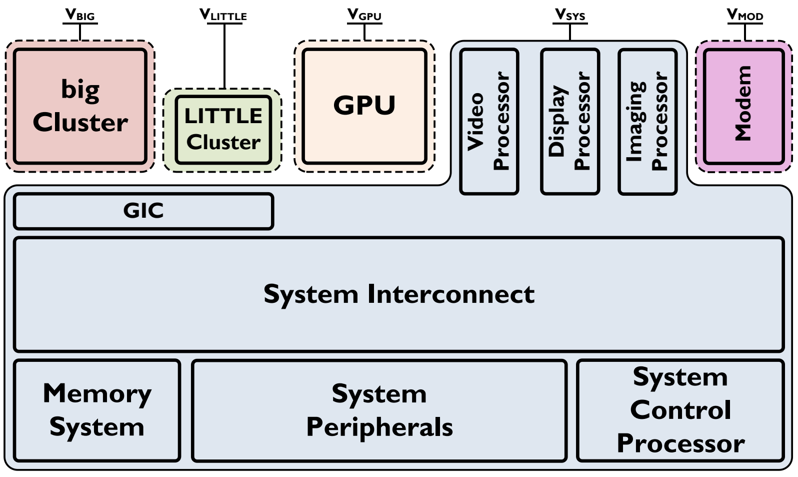


Figure 5-2 - Voltage Domains: High end mobile SoC example

Figure 5-2 is a simplified illustration of a high end mobile SoC. This is a big.LITTLE system with voltage domains for independent DVFS of each processor cluster and the GPU. An integrated modem also has an independent voltage domain.

5.2 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 segmented into a number of power gated domains as well as an always on power domain.

The motivation for partitioning a voltage domain into a number of power domains is to facilitate techniques for static leakage power mitigation. These include modes where power is removed, with loss of context, and also 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 a SoC based on ARM components.
- Power domain partitioning requirements that must be respected. These include availability and power ordering relationships between vital components in the system.

5.2.1 Power Modes

This section defines the operating 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 *Power States* on page 6-1.

ON

A power domain always supports an ON mode. This is the normal operating 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 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 a number of 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 then 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 operating 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 *Power Control Framework* on page 7-1 and integration of these mechanisms in *System Power Control Integration* on page 8-1.

5.2.2 Power Domain Choices

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

From a practical perspective, implementation and control overhead constraints mean that power gated domains are typically large with their boundaries 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 for power off of functions not required for significant periods of time can be determined.

Also, the operating 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.

5.2.3 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 might then gather these functions into a single power gated domain.

In a large scale SoC this can represent a significant area. The SoC architecture could then be organized to partition this area into further power domains. However, challenges can arise from the pervasive nature of common paths through interconnect 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.

5.2.4 Always On Logic

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 of 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 logic 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 5-3:

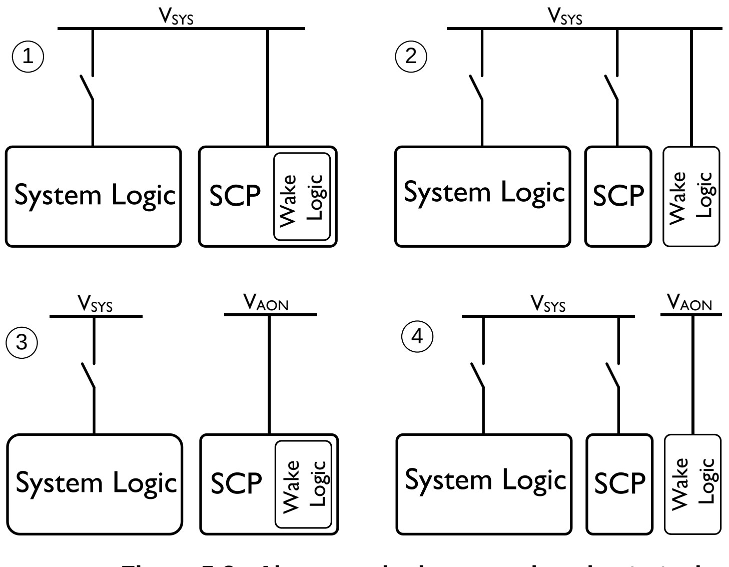


Figure 5-3 - Always on logic power domain strategies

Taking each of these options in turn:

1. In this case the system logic power domain has on-chip 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 sufficient 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 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 switches are obviated.
4. This option is, in practice, strategy 3 in combination with the power domain options of strategy 2.

5.2.5 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 5-1 shows an example using Cortex-A profile products:

Table 5-1 - Cortex-A profile power domain and modes example

Power Domain	Supported			Power Modes
	Cortex-A57	Cortex-A53	Cortex-A17	
Each core	Yes	Yes	Yes	ON, RET[a], OFF
Advanced SIMD/FP pipeline in each core	No	Yes[b]	No	ON, RET[a], OFF
L2 RAMs[d]	Yes	Yes	Yes	ON, RET[c], OFF
Cluster	Yes	Yes	Yes	ON, OFF
Debug domain	Yes	No	No	ON, OFF

a. Retention mode autonomously managed transparently to software.

b. On / Off modes follow core domain. Retention mode autonomously managed as core.

c. Autonomous retention transparent to software at run time and explicit directed retention at power off.

d. On / Off modes follow SCU-L2 cluster domain

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 temperature, 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. The cluster shared cache and SCU RAMs might also support retention modes, during cluster logic off and also autonomously while the cluster logic remains operational.

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 voltage domain portion of DVFS bridges, for functional and CoreSight^{TM} use, and also CoreSight infrastructure dedicated to the cluster.

Debug Domain

If a debug power domain is supported, 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.

Partitioning Example

Figure 5-4 shows an example logical view for a MP4 processor 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. 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 ordering requirements for the processor subsystem from the voltage supply (V_{AP}), available first, downwards. Power off ordering is the reverse sequence.

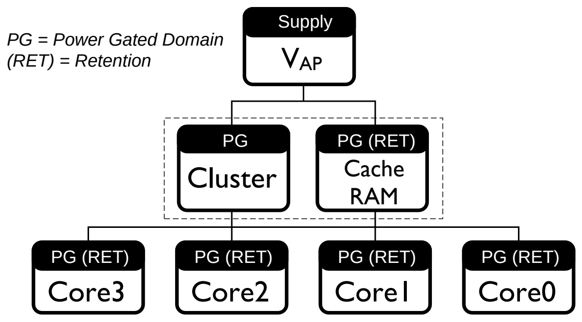


Figure 5-4 - Processor power domains example – logical view

For clarity Figure 5-5 shows that in the physical construction all on-chip switches are implemented in parallel and that the logical ordering is by control sequencing. In further examples this representation will be omitted.

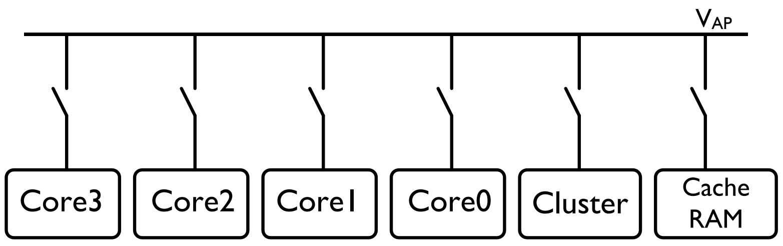


Figure 5-5 - Processor power domains example – physical view

5.2.6 CoreSight Logic

Shared CoreSight infrastructure, with the exception of Debug Access Ports which must be always on, can have a dedicated power gated domain. This would be anticipated to be within the system logic voltage domain.

If this is not implemented this logic would be incorporated into the system logic power domain.

5.2.7 Graphics Processor

The descriptions in this section are intended to be generally applicable to ARM Mali^{TM} 400-series, T600-series, T700-series and T800-series GPU products.

These 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 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 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 domain in SoC sleep states.

Core and Core Group Power Domains

The implementation of power gated domains for the 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.

Partitioning Examples

Figure 5-6 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.

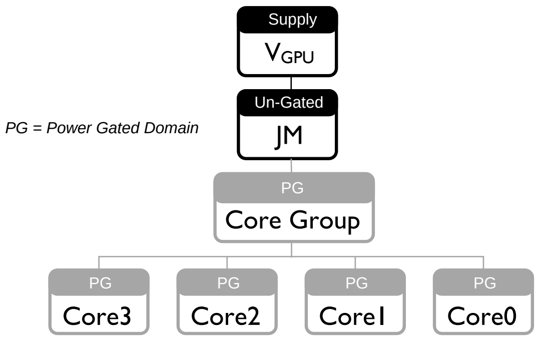


Figure 5-6 - GPU power domains example 1 – logical view

Figure 5-7 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.

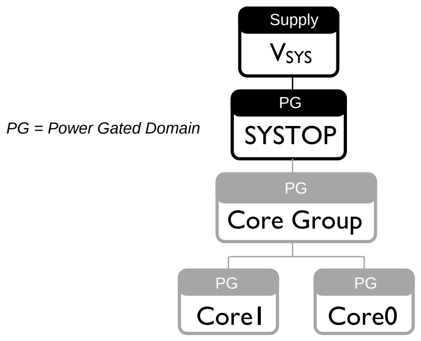


Figure 5-7 - GPU power domains example 2 – logical view

5.2.8 Video Processor

The ARM Mali^{TM} V500 and V550 video processors have an integrated power management capability supporting power gating of processing cores. In a similar arrangement to Mali GPUs a top level control module is required to be placed in a relative always on domain that must be powered before the driver software can schedule any work. This top level control module contains a function, known as the core scheduler, which controls when cores are powered on and off.

From a high level system power control and OSPM perspective only the availability of the power domain containing the core scheduler needs to be considered.

The top level control module is assumed to be integrated into an external power domain such as a system logic domain. The only constraints are that the external power domain is in the same voltage domain as the processor cores and always on whenever the video processor is required.

Core Power Domains

If core power gating is implemented it is possible to implement both a single power gated domain for all cores or a power gated domain for each core.

If core power gating is not implemented then the cores are part of the same external power domain that contains the top level control module.

Partitioning Example

Figure 5-8 shows an example logical view for an MP4 video processor.

In this example the top level control module has been merged into the system logic domain containing shared resources (SYSTOP). When SYSTOP is powered the core scheduler is available. Per-core power gated domains are supported. The lighter color shading again reflects that these domains are managed by the core scheduler and are not system visible.

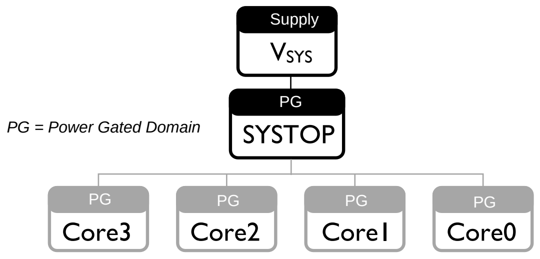


Figure 5-8 – Video processor power domains example – logical view

5.2.9 Display Processor

The ARM Mali™ DP550 display processor supports a single power gated domain in a single display configuration and one or two power gated domains in a dual display configuration.

The supported power gated domains are externally managed.

5.2.10 Other Functions

For the remaining SoC functions the choice to implement power gated domains would typically be 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 on the basis of the utility and capability for dynamic dimensioning of the system.

Typically, a fairly 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 which 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.

5.2.11 Power Domain Hierarchy Requirements

A number of fundamental relationships between critical resources must be maintained in a SoC based on ARM components 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. Detailed power flows are provided in *Power Control Flows* on page 9-1.

The *Server Base System Architecture (SBSA)* and *Client Base System Architecture* (*CBSA)* describe 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* and *ARM Architecture Reference Manual, ARMv7-A.*
- Wake-up semantics for two wake-up methods:
  - Interrupts from the Generic Interrupt Controller (GIC)
  - Always on domain wake events
- System memory availability

SBSA and *CBSA* also describe power state semantics consistent with these requirements. Power states are covered in *Power States* on page 6-1.

Figure 5-9 shows a power domain hierarchy similar to that shown in *SBSA* and *CBSA* which conforms to these requirements. Other examples that conform to the requirements, that are not simply degenerate subsets, are possible.

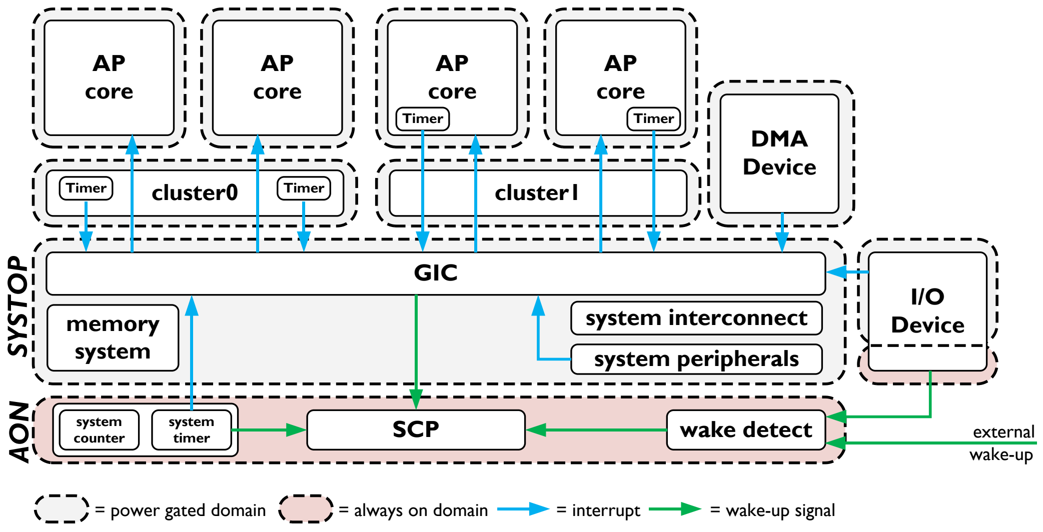


Figure 5-9 - Example power domain hierarchy

In Figure 5-9 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 entirely *OS managed* and dependent on the presence of the GIC and the availability of system memory at all times when it is powered.
- An I/O device with *self-managed* capabilities that has a power gated domain portion and always on power domain portion capable of generating wake events to express resource requests to the SCP.

The following sections describe the requirements arising from *SBSA* and *CBSA* 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, of the generic timer, is required to provide a consistent incrementing view of time for the entire time the SoC is powered on.

The system counter exports its count value and this 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 the system wake-up timer. 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 wired as GIC 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 implementations compliant with the GICv3 architecture, the GIC CPU interface must be incorporated in the AP core power domain while the associated redistributor, the distributor and any interrupt translation service (ITS) support can be in other power domains.

In this release of PCSA only monolithic implementations of GICv3 are considered. In a monolithic implementation, such as GIC-500, all logic other than the GIC CPU interfaces is implemented in a single power domain. This logic is herein referred to as the GIC.

The GIC must be powered on with a relatively always on relationship to all AP cores. However, it is not required to be always on, and can be powered down in SoC sleep states. In the example of Figure 5-9 the GIC is placed in the system logic domain (SYSTOP).

When the GIC is powered off an AP core can only be woken by signals routed as 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.

Since the 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.

While level 0 of SBSA and CBSA only require a system specific timer it is recommend that systems comply with at least level 1 of SBSA and CBSA and 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 the system wake-up timer.

Other always on domain wake events are implementation defined. Common examples as illustrated in Figure 5-9:

- **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

Clearly, 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* and *CBSA* 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 5-9 the path to system memory is in the same power domain (SYSTOP) as the GIC and therefore the system memory will be available whenever any AP core is powered on.

However, components, such as the I/O device in the example, 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 *Access Control* on page 8-18.

Power Ordering Requirements

The power ordering requirements of the example shown in Figure 5-9 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 fully OS managed devices. This class of device is represented in Figure 5-9 by the DMA device.
- The I/O device in Figure 5-9 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.

5.2.12 SoC Partitioning Example

Figure 5-10 shows a high end mobile SoC example as a summary for voltage and power domain partitioning.

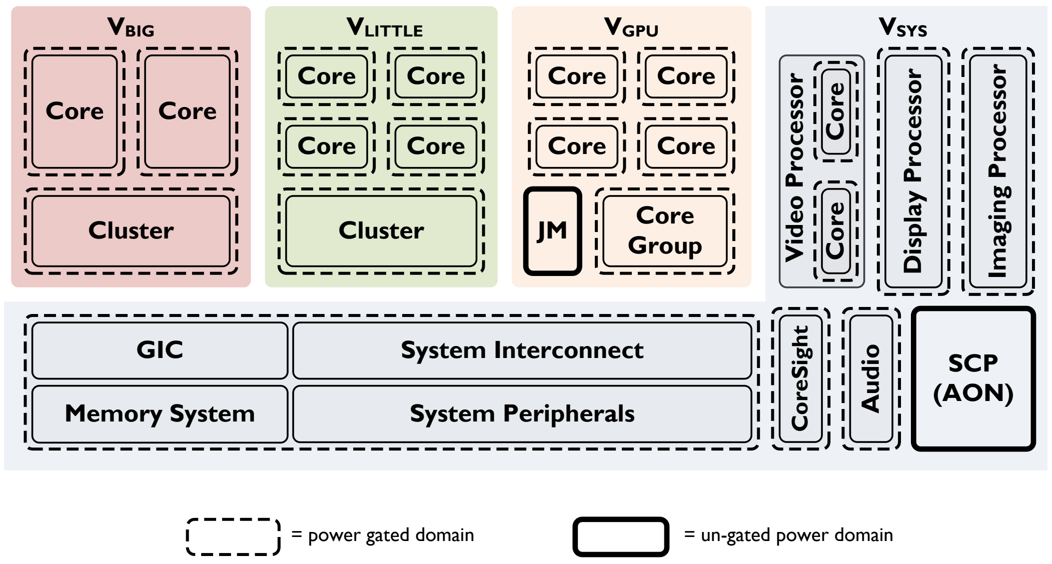


Figure 5-10 - SoC voltage and power domain partitioning example

Figure 5-10 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 only 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.

6 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.

The remainder of this chapter is divided into the following sections:

- *Power States and Power Modes* on page 6-2.
- *Power State Hierarchy* on page 6-3.
- *Coordination by System Control Processor* on page 6-9.

6.1 Power States and Power Modes

6.1.1 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.

6.1.2 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 *Power Modes* on page 5-5. A component within a power domain can also have power modes that are related to dynamic power reduction which might be visible as power states to software.

———————————

6.1.3 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 a number of 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.

6.2 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:

- *Core Power States* on page 6-4.
- *Cluster Power States* on page 6-5 and *Device Power States* on page 6-7.
- *SoC Power States on page 6-7.*

Figure 6-1 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.

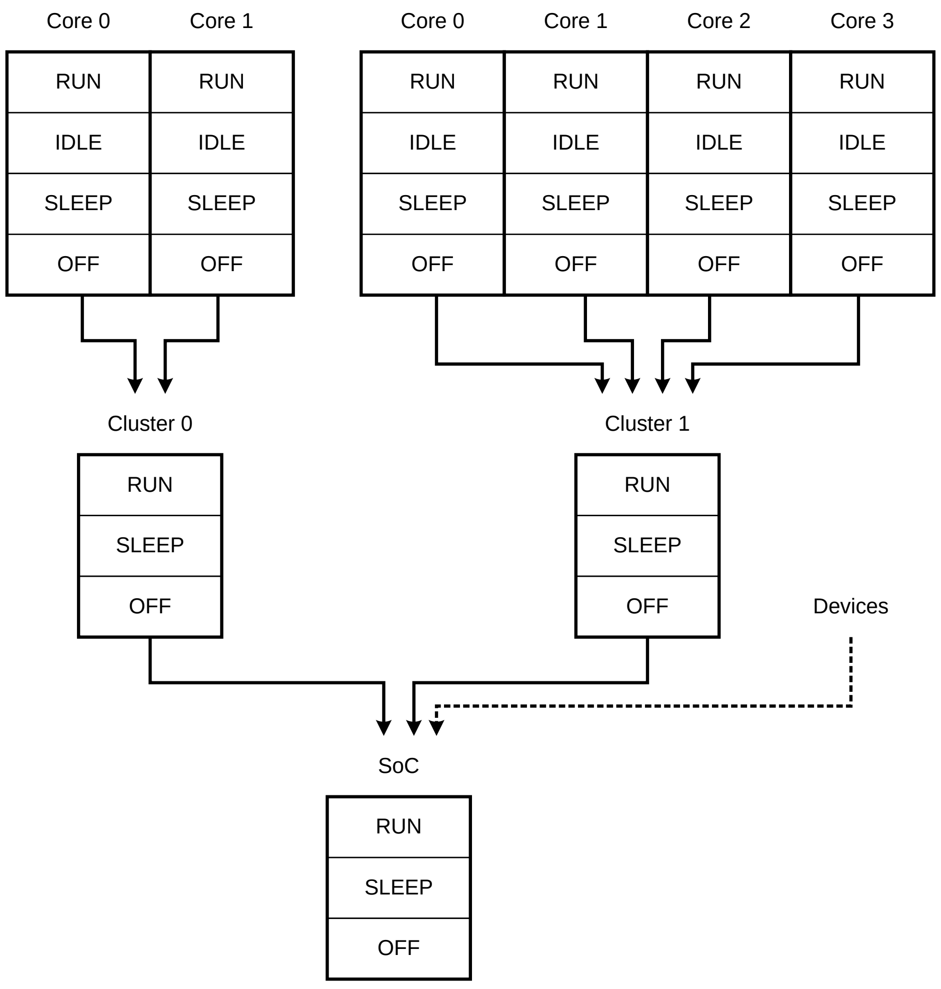


Figure 6-1 - Power state hierarchy example

6.2.1 Core Power States

The *Server Base System Architecture* (SBSA) and *Client Base System Architecture* (CBSA) define 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 (**EDBGRQ**). 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 (**EDBGRQ**). 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 of these core power states will always be available. Their availability depends on the operating modes that are supported and implemented.

Example Mapping of Core Power States to Modes

Table 6-1 uses the supported features of Cortex-A53 to provide an example mapping of AP core power states to the available core power modes:

Table 6-1 - Cortex-A53: mapping AP core power states to modes

Power State	Power Mode		Note
	Core	Advanced SIMD / FP	
RUN	ON	ON	Core running code
		RET	Core running code, 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-A53 does not implement support for the IDLE_RETENTION state sematic as the core will respond to external access of debug registers in WFI or WFE.

The grouping in the mapping 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 as a consequence 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.

6.2.2 Cluster Power States

This document defines the following AP cluster power states.

RUN

Cluster is powered on, and can support any core moving to any power state.

SLEEP_RETENTION

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

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

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 to wake when a core in the cluster receives a wake capable interrupt.

———————————

Example mapping of Cluster Power States to Modes

Table 6-2 uses the supported features of Cortex-A53 to provide an example mapping of cluster power states to available power modes:

Table 6-2 - Cortex-A53: cluster power states mapping to modes

Cluster Power State	Core Power State	Power Mode		Note
		SCU-L2	L2 Data RAMs	
RUN	Any	ON	ON	Cores able to run code
	Each Core in either: IDLE_STANDBY or SLEEP or OFF		RET	Dynamic L2 RAM retention
SLEEP_RETENTION	All Cores in SLEEP or OFF, with at least one in SLEEP	OFF	RET	Cluster off. Cores can wake from interrupts. Static L2 RAM 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.

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.

6.2.3 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 then requests the SCP to perform any required actions.

6.2.4 SoC Power States

The SoC power states used in this document are for illustration purposes only. The power states for a 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 is able to 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.

Fully 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**, **SLEEP**1 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.

Table 6-3 shows an example of how these power states might map to power domains in a system.

Table 6-3 - SoC power states

SoC Power State	Processor Cluster States	OS Managed Device States	Power domain Mode		System Memory	Note
			GIC	Wakeup timers, system counter		
RUN	Any	Any	ON	ON	Available	AP cores able to run, can wake 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

6.3 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.

6.3.1 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. At a higher wake latency constraint the SCP might be switched to a low frequency clock and a higher speed clock is turned off. At a lower, reconciled, 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/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 has to the rest of the system.

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

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

6.3.2 Cluster Power States

While the OSPM may provide mechanisms to enable cluster power state selection, it might be attractive 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 must be able to enforce a period where no new cores are allowed to become active in the cluster. This is to remove the risk of a core becoming powered on and creating a race condition in the cluster power off sequence.

To be able to manage cluster power mode transitions, the SCP must be able to control any required shared cache flushing process and also 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 and this is a consideration in terms of the access SCP has to the rest of the system.

7 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:

- *Power Control Framework Overview* on page 7-2.
- *Low Power Interfaces* on page 7-3.
- *System Control Processor* on page 7-4.
- *Power Management Infrastructure Components* on page 7-9.

7.1 Power Control Framework Overview

Figure 7-1 shows a high level illustration of power control framework concepts, including the primary components and how they are interfaced.

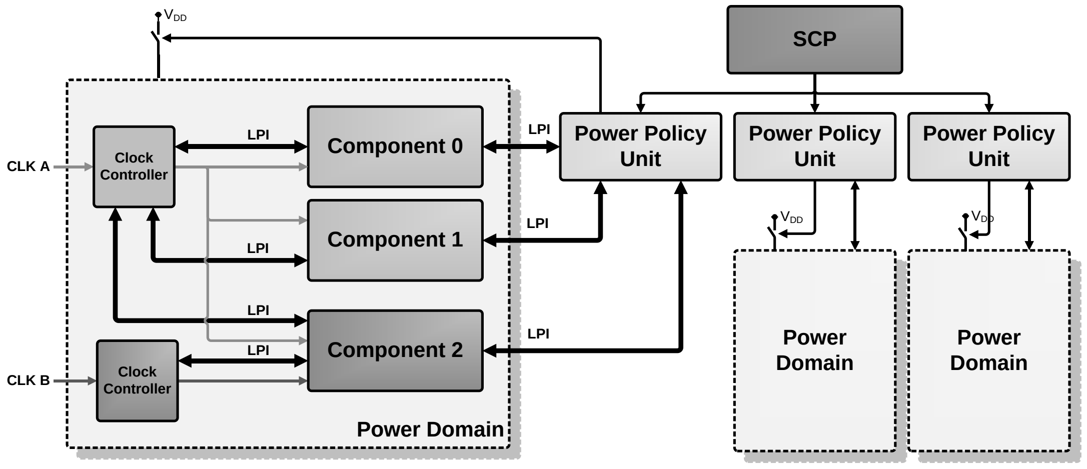


Figure 7-1 – Power control framework overview

7.1.1 Power Control Framework Low Power Interfaces

The Low Power Interfaces (LPI) described here are:

- *Q-Channel* on page 7-3.
- *P-Channel* on page 7-3.
- *AXI LPI* on page 7-3.

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.*

The protocol of the AXI LPI is described in detail in the *AMBA® AXI™ and ACE™ Protocol Specification.* From a power control framework perspective this interface is only for use with legacy components and only with the restrictions described in *AXI LPI* on page 7-3.

7.1.2 Power Control Framework Infrastructure Components

The power control framework infrastructure components described here are:

- *System Control Processor* (SCP) on page 7-4.
- *Power Policy Unit* (PPU) on page 7-9.
- *Clock Controller* (CC) on page 7-12.

More details about integrating these components and related requirements for other components can be found in *System Power Control Integration* on page 8-1.

The architecture of the *power policy unit* (PPU) is described in detail in the *ARM® Power Policy Unit Architecture Specification Version 1.0*.

The architecture of the *clock controller* (CC) is described in detail in the *ARM® Clock Controller Architecture Specification Version 1.0*.

7.2 Low Power Interfaces

There are a number of low power interfaces (LPI). These interfaces are complementary and should be selected as appropriate to the circumstances in which they are used.

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.*

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.*

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 *Restrictions on the use of AXI LPI* on page 7-3.

Full details of the AXI LPI protocol can be found in *AMBA® AXI™ and ACE™ Protocol Specification*.

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

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 **CACTIVE** is HIGH at this time 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.

———————————————

7.3 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 *System Control Processor* on page 4-2.

Figure 7-2 shows an example SCP hardware overview.

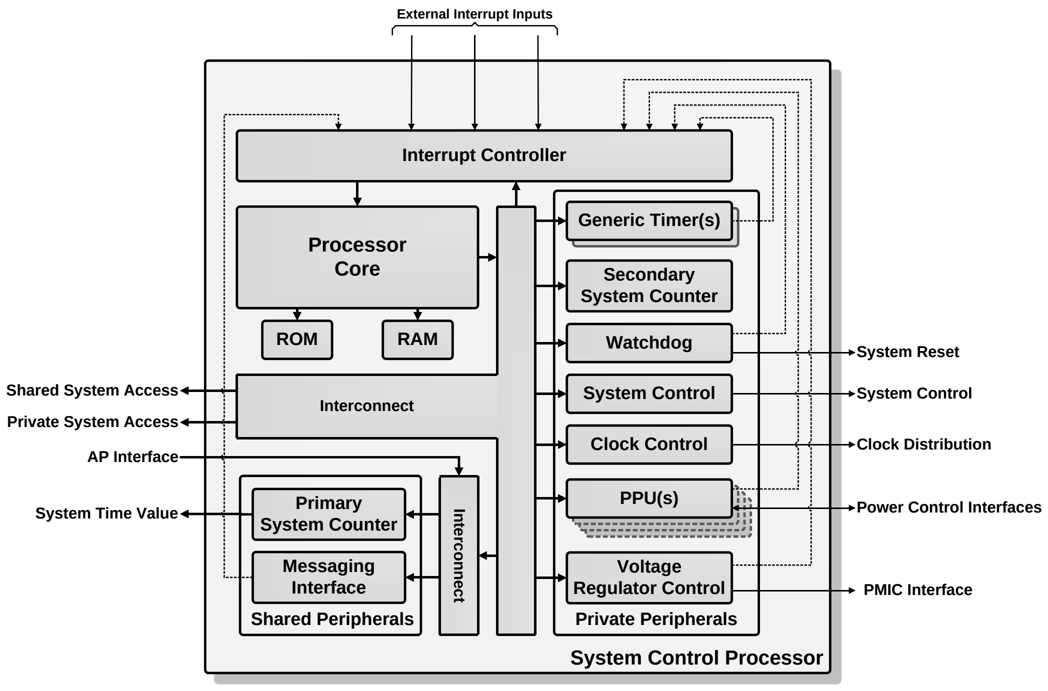


Figure 7-2 – SCP example overview

——— **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.

———————————

7.3.1 SCP Components

This section describes the components of the SCP subsystem.

Processor

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

SCP Processor 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 suite of tasks it is expected to run, especially any tasks with real-time dependencies or where tasks might need to be handled concurrently. Selection criteria include the type of processor and the frequency at which it operates.
- **Interrupt Types**: Some processors have vectored interrupts which can ease the connection, handling, and latency of interrupts, removing the need for software interaction to discover interrupt causes. The number of interrupt inputs available on the core should also be considered and if that requires an additional interrupt controller within the SCP. An additional interrupt controller adds latency to the interrupt sequence in both firmware and hardware.

- **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.
- **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 support of the SCP processor subsystem, available when the SoC is in SLEEP states, and also the integration into the broader CoreSight SoC 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 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.

Processor 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* specification. An example power control flow for boot is given in *System Initialization* on page 9-62.

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 these 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.*

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

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 *Server Base System Architecture (SBSA)* and *Client Base System Architecture* (*CBSA)* 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. 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 *Clock Controller* on page 7-12 and *Clock Gating Control* on page 8-2.

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 7-3 shows an example of this application where a SoC component acts as the requestor.

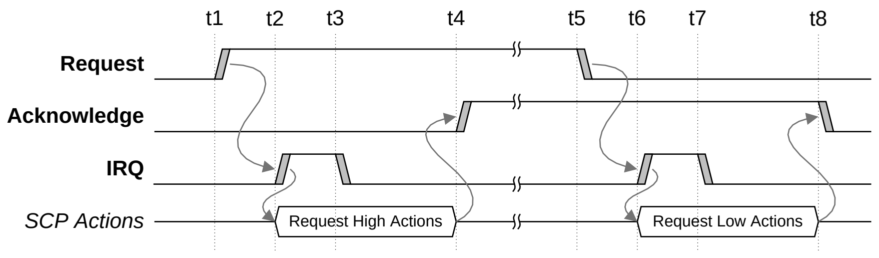


Figure 7-3 – System control handshake example

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, 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 which 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 takes action 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 facilitate communication between these agents and the SCP.

Power Policy Units

Power Policy Units (PPU) are specialized hardware components used to abstract low level control for 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 *Distributed PPUs* on page 8-11.

See *Power Policy Unit* on page 7-9 for more details on the PPU.

See *Power Control Integration* on page 8-10 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 7-2:

- **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 *Messaging Interface* on page 7-7.

For SCP peripherals 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 shared interconnect.

Access to SoC resources allows the SCP to perform actions as part of power control sequences, such as configuration and save-restore. This includes examples, such as, configuration of interconnect and memory controller components.

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

7.4 Power Management Infrastructure Components

7.4.1 Power Policy Unit

The power policy unit (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, which 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 7-4 shows the PPU interfaces.

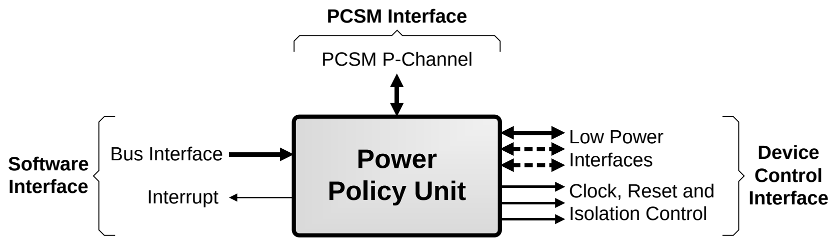


Figure 7-4 – Power Policy Unit interfaces

The PPU interfaces are:

- **Software Interface**: A bus interface for programming, for example AMBA APB, and an interrupt which 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 which 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 7-5 shows how these interfaces are connected.

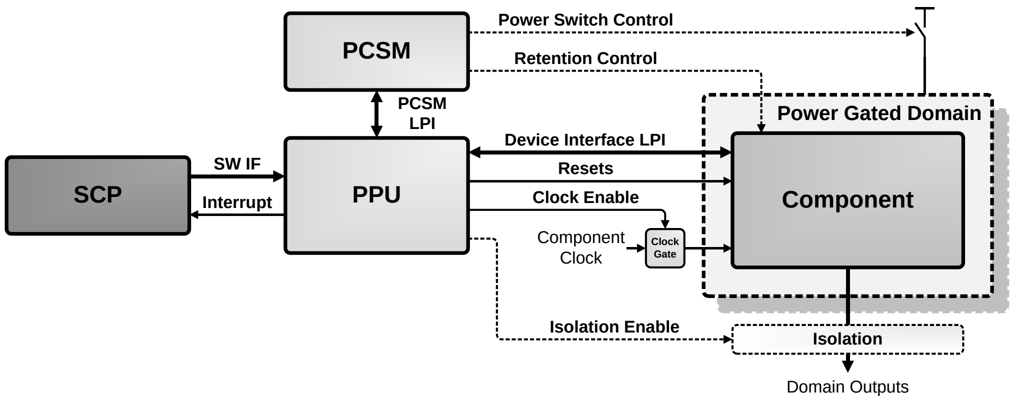


Figure 7-5 – PPU integration example

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 intent or similar means.

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.

Power Control State Machine

Low level power control details such as direct power switch control, or signal controls 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 requests to the implementation dependent controls.

Reset Control

The PPU provides reset signals 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 reset control contains multiple resets from the PPU which are used in different power modes dependent on the reset action required.

An example of this is 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 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. An example is when the power off of a domain is emulated for debug purposes. In such a case certain clocks might need to remain enabled to allow debug access to a component.

——— **Note** ———————

The PPU controls the clock to manage power domain mode changes, not activity based high-level clock gating. High-level clock gating is managed by a clock controller, for more details see *Clock Controller* on page 7-12 and *Clock Control Integration* on page 8-2.

———————————

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. An example is when the power off of a domain is emulated for debug purposes, 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 a number of policies to meet different component requirements. Not all PPUs are required to support all policies, so power mode support is design time configurable.

Table 7-1 lists the power policies supported by the PPU.

Table 7-1 – PPU power policies

Policy	Logic Mode	RAM Mode	Description
Debug Recovery Reset	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	ON	ON	Warm reset application with logic and memories on.
On	ON	ON	Logic on and any memory on. The component is functional.
Functional Retention	ON	RET	Logic on with memories retained. The component is still functional.
Memory Off	ON	OFF	Logic on with memory off. The component is still functional.
Full Retention	RET	RET	Logic and memory in retention.
Logic Retention	RET	OFF	Logic retention with memories off.
Emulated Memory Retention	ON	ON	Logic on and memories on. This mode is used to emulate a memory retention condition without removing power.
Memory Retention	OFF	RET	Logic off with memories retained.
Emulated Off	ON	ON	Logic on and memories on. This mode is used to emulate a power off condition without removing power.
Off	OFF	OFF	Logic off and memories off.

Complete details of the PPU can be found in the *ARM® Power Policy Unit Architecture Specification Version 1.0*.

For more details on PPU integration see *Power Control Integration* on page 8-10.

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 with the exception of the communication with the PCSM. This means, for example, that power switches are not turned off.

Some parts of the design can have power off emulated 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 should remain functional with resets de-asserted. The PPU supports different resets to provide this capability.

Component Interface Adaptation

Components without low power interface support, or with additional power control signals to be managed through power mode transitions, can require an integration layer to adapt the interface between the PPU and the component.

For more details see *Component Integration Layer* on page 8-13.

7.4.2 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 *Restrictions on the use of AXI LPI* on page 7-2.

Figure 7-6 shows the clock controller interfaces.

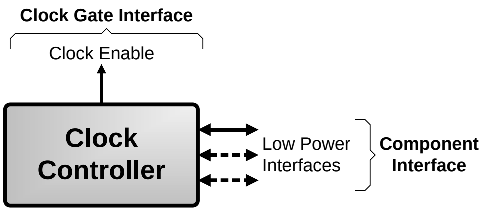


Figure 7-6 – Clock controller interfaces

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 LPI interfaces, depending on the needs of the domain.

The clock controller combines 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 applied 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.

Complete details of the clock controller can be found in the *ARM® Clock Controller Architecture Specification Version 1.0*.

For more details of clock controller integration see *Clock Gating Control Integration* on page 8-2.

8 System Power Control Integration

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

8.1 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 which 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 to balance the clock arrival time at synchronous endpoints to achieve timing closure.

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 as much of the clock tree as possible in addition to the endpoints.

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

8.1.1 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 8-1 provides an illustration of these clock gating levels.

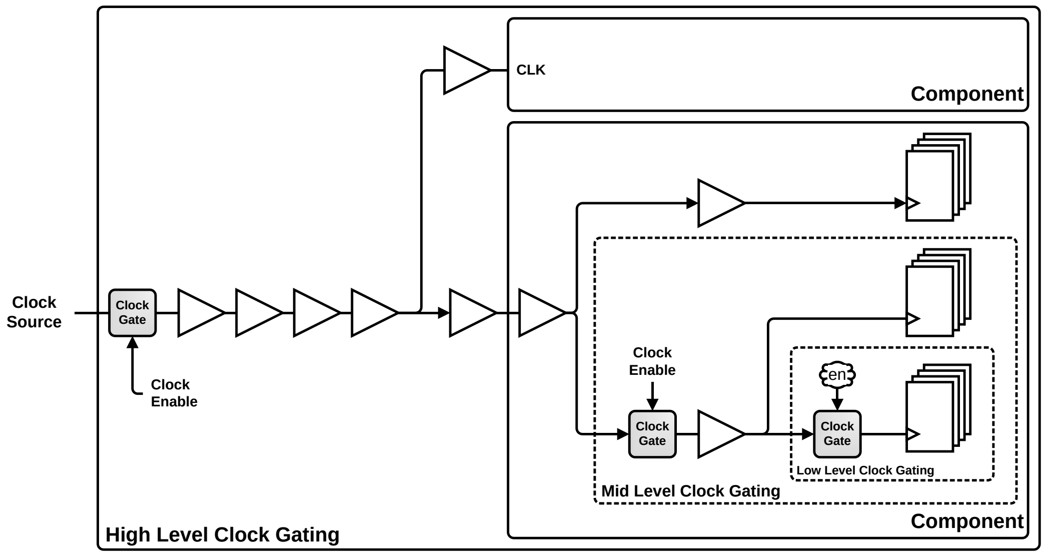


Figure 8-1 – Clock gating hierarchy

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. However, they gate only leaf parts of the clock tree.

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 breakeven point of the re-structuring.

Figure 8-2 and Figure 8-3 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.

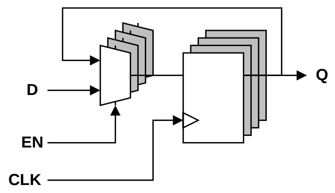


Figure 8-2 – Standard flip-flop enable

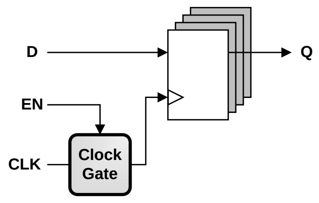


Figure 8-3 – Low level clock gated flip-flop

In the ideal case an endpoint with a local clock gate is gated whenever the flip-flop is not being updated.

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

- No enable is present, or the enable is not recognized by the synthesis tool.
- 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 the majority 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 which will be idle during periods of operation.

The enables are controlled by the surrounding logic and are typically enabled and disabled 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 relatively low in the clock tree. This placement avoids skew between the logic and the clock gate shrinking the enable timing window.

While gating large parts of the design, these mid-level clock gates do not necessarily 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, but a large portion of the clock tree can still be running. 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 *High Level Clock Gating Methodology* on page 8-4.

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.

8.1.2 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 8-4 provides an example where a Cache Coherent Interconnect (CCI) and Network Interconnect (NIC) share a common high level clock gate:

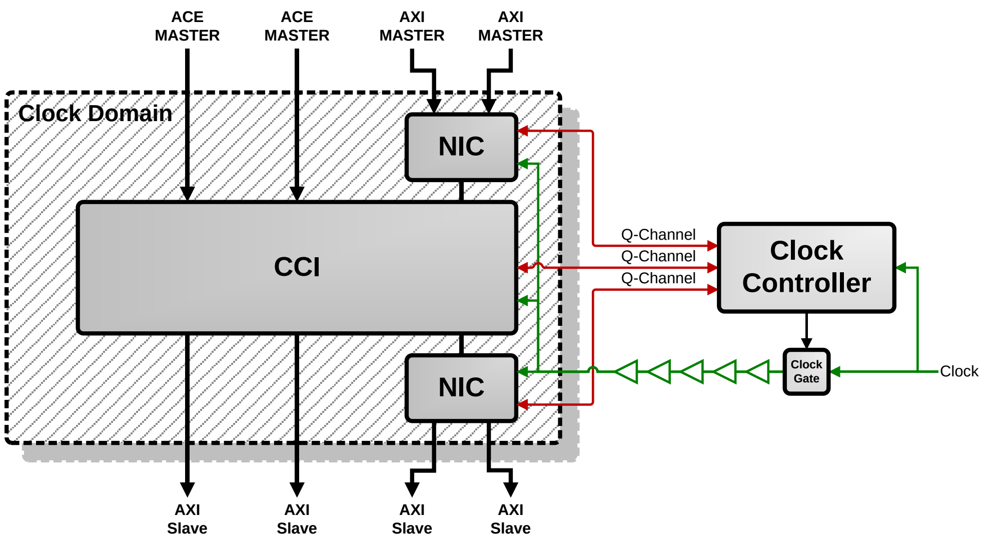


Figure 8-4 – High-level clock gating for multiple interconnect components

8.1.3 High-Level Clock Domain Selection

When implementing high-level clock gating ARM recommends partitioning clock domains at either asynchronous boundaries or at another natural divergence in a clock tree such as a power domain boundary.

Figure 8-5 shows a simple example of components partitioned into two clock domains separated by an asynchronous boundary.

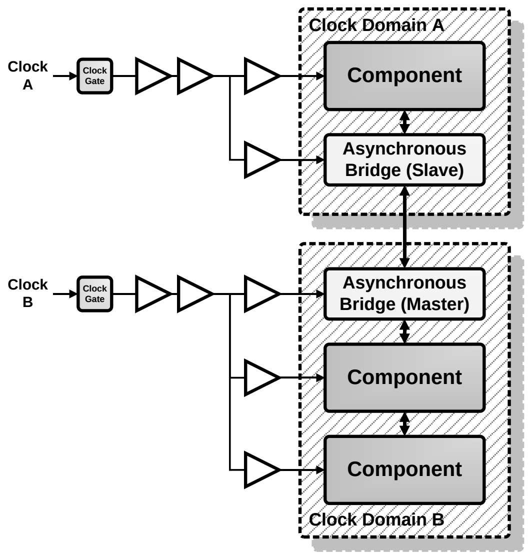


Figure 8-5 – Clock domains example

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 and related 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 in such a case between the number of clock domains implemented, and 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 multiple non-common clock insertion paths which places pressure on clock tree balancing. The consequences of this might be that:

- The high level clock gate is placed much lower in the tree.
- The increased balancing effort leads to higher clock buffer power.

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

Figure 8-6 shows an example of the second case.

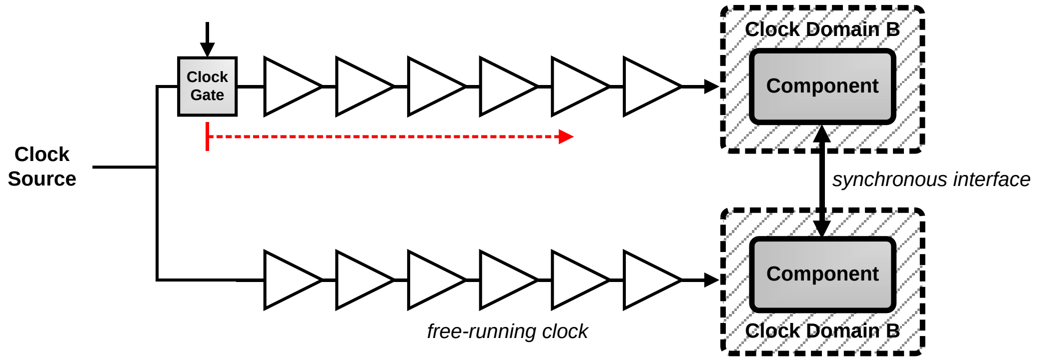


Figure 8-6 – Clock insertion pressure

In Figure 8-6 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 as a consequence of the synchronous interfacing between them.

8.1.4 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 many ARM components, particularly CoreLink components. For newer components this is with Q-Channel. For earlier components this is with AXI LPI according to the restrictions given in *Restrictions on the use of AXI LPI* on page 7-3.

The clock controller is described in *Clock Controller* on page 7-12.

Component design considerations for Q-Channel clock gating are detailed in *Component High-Level Clock Gating* on page 10-8.

Clock Controller Placement

The aim, as discussed, 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 Q-Channel.

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.

Clock Controller Connections

Figure 8-7 shows the clock gating arrangement for a single component within a clock domain.

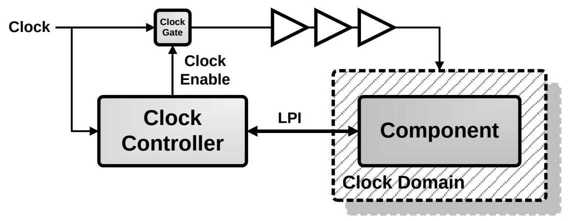


Figure 8-7 – Clock gating with a single component

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 8-8 shows this clock gating arrangement for multiple components within a clock domain.

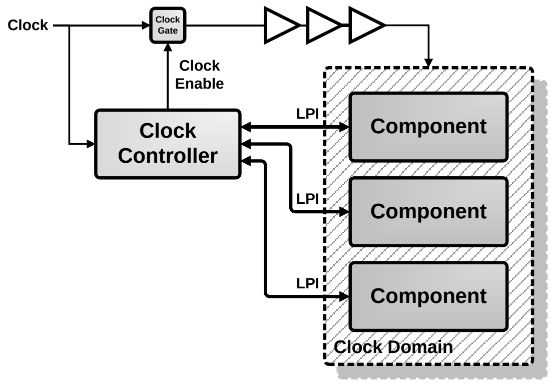


Figure 8-8 – Clock gating with multiple components

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

Figure 8-9 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 which can be gated independently. The high level gated bus interface clock could be shared with many components while the functional clock might be dedicated to that component.

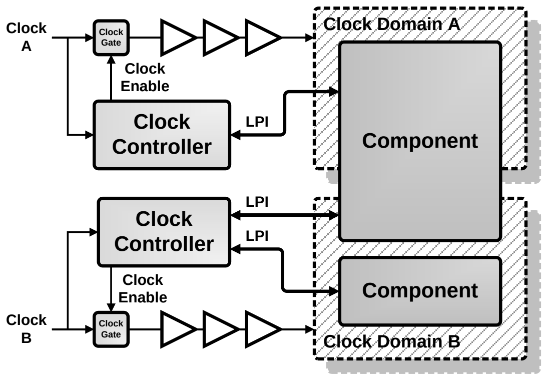


Figure 8-9 – Clock gating a component with multiple clock domains

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 slave interface that receives transactions and passes them over an asynchronous boundary interface.
- A master interface that receives the transactions from the asynchronous boundary interface and re-transmits them to downstream components.

Figure 8-10 shows the clock control arrangement between clock domains connected with an asynchronous domain bridge.

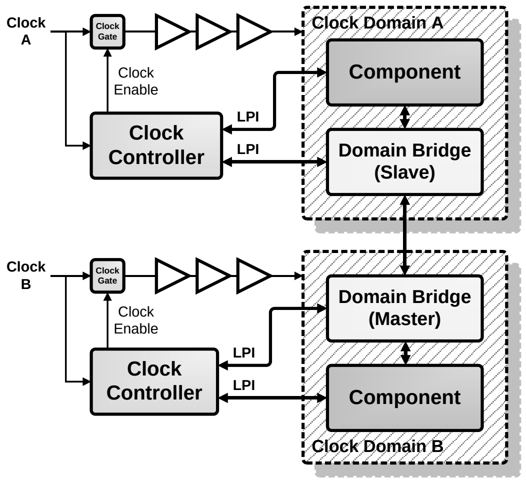


Figure 8-10 – Clock domain crossing with an asynchronous domain bridge

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 *Voltage and Power Domain Boundaries* on page 8-15.

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, which 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 8-11 shows this arrangement for **QACTIVE** generation at one side of an asynchronous domain bridge.

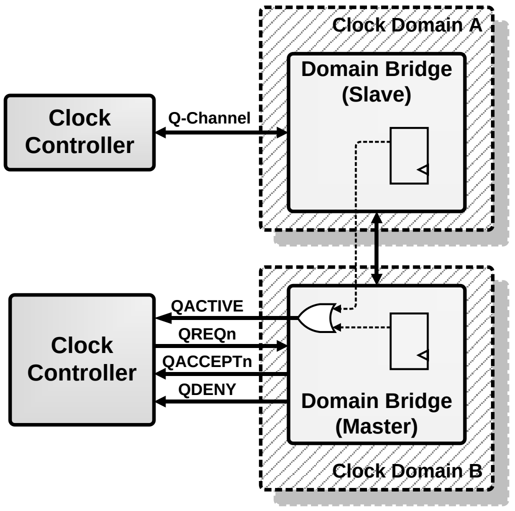


Figure 8-11 – QACTIVE generation between asynchronous domain bridge slices

An asynchronous domain bridge can be implemented with:

- **QACTIVE**-only clock control support, such as Corelink ADB-400.
- A complete Q-Channel for each clock domain.

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 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 8-12 shows the detail of the connections in case of **QACTIVE**-only asynchronous domain bridge with components supporting Q-channel clock gating at both sides.

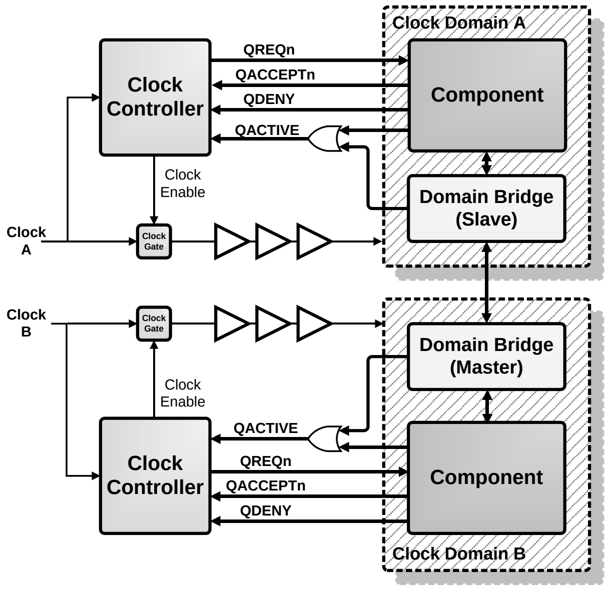


Figure 8-12 – Integration of QACTIVE-only asynchronous domain bridge

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 8-13 shows an example of this concept.

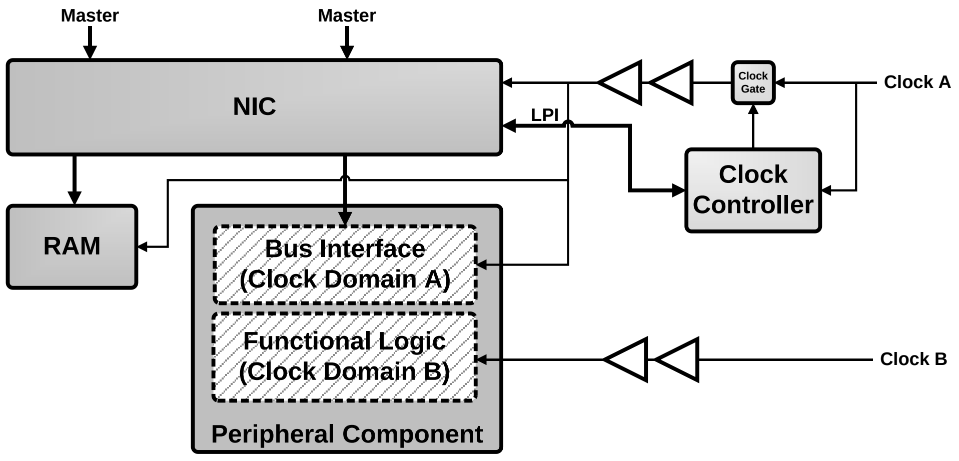


Figure 8-13 - Clock domain scope example

In Figure 8-13 the NIC interconnect uses its LPI to guarantee clock supply whenever it has outstanding transactions at its interfaces.

The RAM component and the bus interface portion of the peripheral component are connected to the NIC within the same clock domain, Clock A. Neither of these components provides LPI support for clock gating. The peripheral component also has functional logic in an independent clock domain, shown as Clock B. Clock B might also be gated by an independent clock controller, but this is not considered further in this example.

Providing that the RAM component and the bus interface portion of the peripheral component only require a clock while bus transactions are processed, they can be included safely within the gated clock domain.

This technique can also be applied, with careful analysis of topology dependencies, 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 might 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.

———————————————

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 all connected Q-Channel components.
    - This ensures that any consequent protocol violation cannot be observed by the connected components.

8.2 Power Control Integration

The power mode of a component is controlled either directly or indirectly by the SCP. As the SCP is in a different voltage or power domain, and often in a different clock domain, the power control interfaces must support asynchronous operation.

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.

——— **Note** ———————

The LPI used for clock control and the LPI used for power control are separate interfaces with independent control points.

———————————————

Once an LPI request has been accepted by all controlled components in the power domain the SCP manages either directly or indirectly 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 together to create managed power domains.

Figure 8-14 shows an arrangement for a single component in a power gated domain.

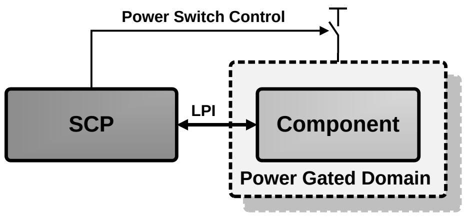


Figure 8-14 – Power gated domain control with a single component

A power gated domain can contain multiple components that are controlled together. The LPIs for the components of the power domain are combined at the SCP. All components need to accept a power mode change request before it can proceed.

Figure 8-15 shows an arrangement for multiple components in a power gated domain.

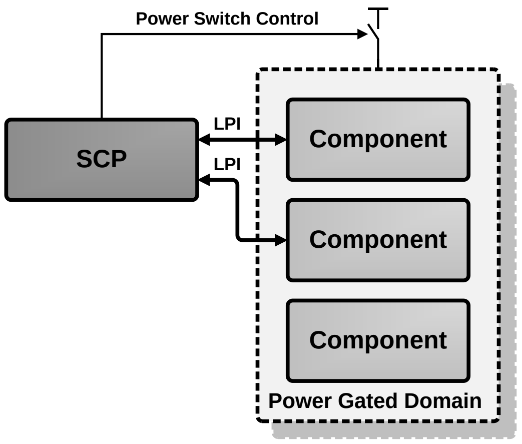


Figure 8-15 – Power gated domain control with multiple components

In Figure 8-15 one component is shown without an LPI. In some cases a component will have no specific hardware power control requirement. The component can then be simply incorporated into the control of the power gated domain according to the state of the controlled components and the system conditions.

8.2.1 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. It might also adversely impact the response to power mode changes due to interrupt latency and conflicting processing tasks.

A more scalable solution 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, or once a policy is programmed leave the timing of entering a state to the PPU when all required conditions are met.

This autonomous operation is especially important for those power modes which require fast response times. An example of this is logic or RAM retention states, where no state is lost and the operation can be transparent to software while saving power. However, this must add only minimal latency so as not to adversely affect system performance. The PPU when programmed for dynamic operation can enter and exit allowed states based on only the components LPI status.

Figure 8-16 shows an example power gated domain arrangement using a PPU to interface with the power domain components.

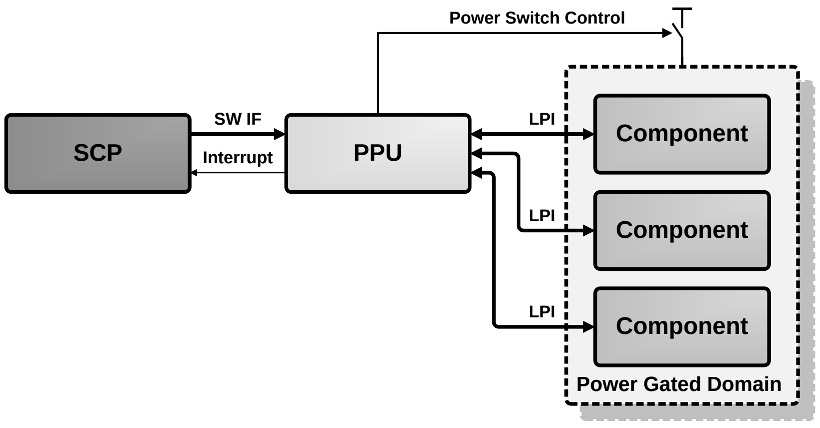


Figure 8-16 – Power gated domain control with a PPU

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 *Power Policy Unit* on page 7-9. Complete details of the PPU can be found in the *ARM® Power Policy Unit Architecture Specification Version 1.0*.

PPU placement is discussed further in *Distributed PPUs* on page 8-11.

8.2.2 Distributed PPUs

The placement of PPUs is an important consideration. From some perspectives, 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 apparently straightforward. However, there are a number of reasons that 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 all those 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 which 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 SoC interconnect.

This placement also 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 then use the same clock as the component it is controlling, it is in a different power, or even voltage, domain and treating the interface as asynchronous should be considered 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 8-17 shows an example of this type of arrangement.

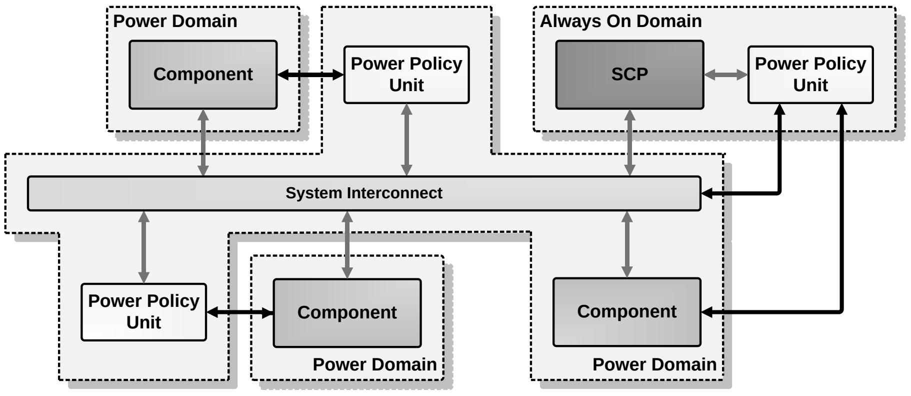


Figure 8-17 – Distributed PPU overview

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.

8.2.3 System of Systems

Some SoC subsystems might be complex entities in their own right. There might be reasons for such a subsystem to have its own local control processor. With some small differences the subsystem power management structure reflects that of the SoC, therefore such an arrangement is called a *system of systems*.

Figure 8-18 shows an example of a system of systems.

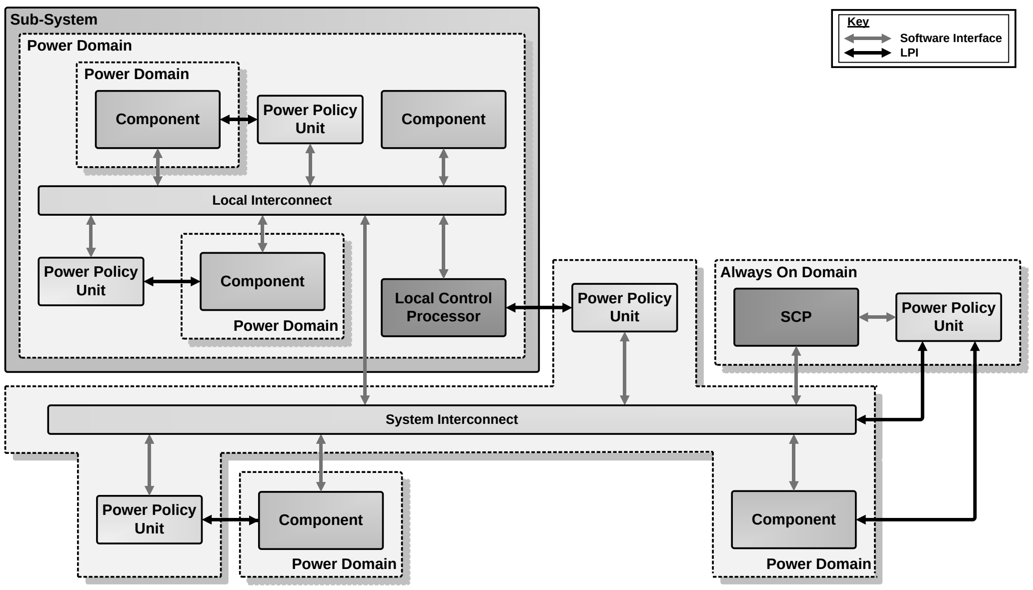


Figure 8-18 – System of systems overview

The subsystem in Figure 8-18 has its own local control processor. 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 local control processor of the subsystem in Figure 8-18 is slightly different from the SCP. It is only relatively always on and can be powered off. Therefore, it needs an LPI interface to a PPU in a relatively always on domain. This PPU is controlled by the SCP.

Standard 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 local control processors and the SCP. This capability is described in *Messaging Interface* on page 7-7.

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.

8.2.4 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.

This can be used, for example, to adapt a component without low power interface support, or with additional power control signals to be managed through power mode transitions.

Figure 8-19 shows miscellaneous signaling combined into an LPI in a component integration layer.

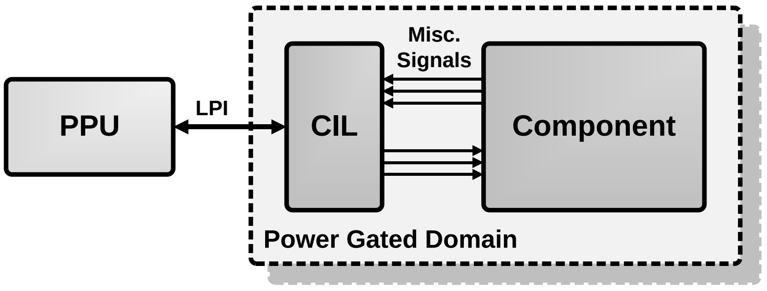


Figure 8-19 – Single component integration layer example

Another example of the integration layer approach is when multiple components in a power gated domain have a control dependency, such as a specific quiescence sequence, or when the number of channels to control exceeds the PPU capability. In this example the PPU LPI requests can be adapted to the individual component LPI requirements in the integration logic.

Figure 8-20 shows an integration layer example with LPI adaptations for multiple components.

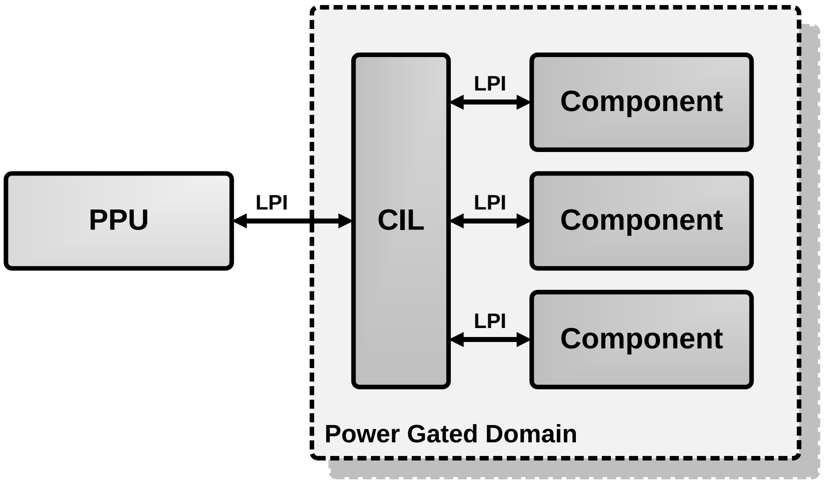


Figure 8-20 – Multiple component integration layer example

8.2.5 Voltage and Power Domain Clock Gating

Before a domain can be powered off or put into a full retention[1] state it must be clock gated externally. This is managed by the PPU and is separate from any dynamic high-level clock gating control.

The PPU will often use a different clock than the power domain, and it might be necessary to gate several clock inputs. Therefore, clock enables will have to be synchronized.

Figure 8-21 shows and example of this arrangement.

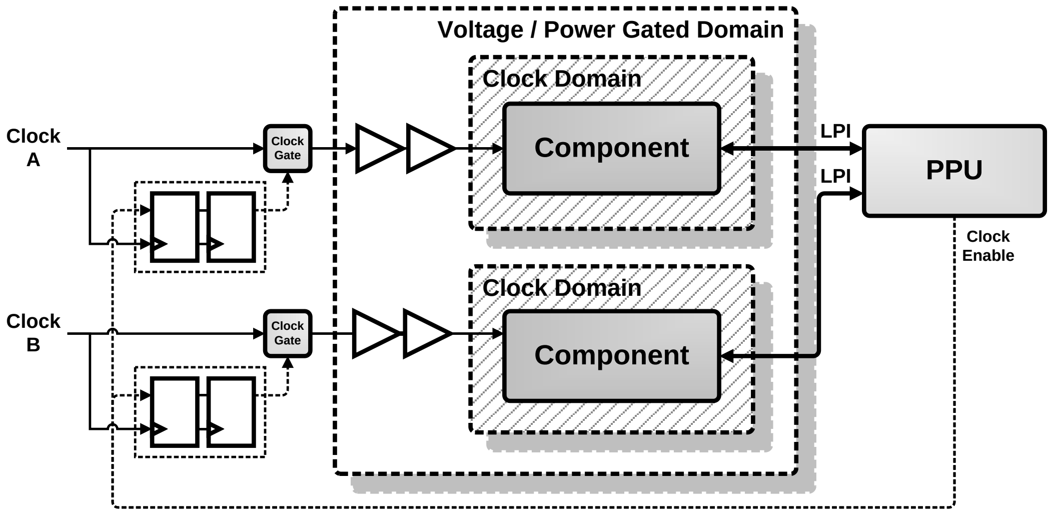


Figure 8-21 – Clock gating for voltage and power domains

Depending on the topology the power controller clock gating enable can be combined with the dynamic high-level clock gating for the domain, or it can be a separate clock gate.

[1] Components which have partial retention, such as the functional retention mode defined for the PPU in 7.4.1, where the clock is still needed to operate parts of the design while other parts are in retention, need to include internal clock gating for the retained blocks.

Figure 8-22 and Figure 8-23 show examples of these implementations.

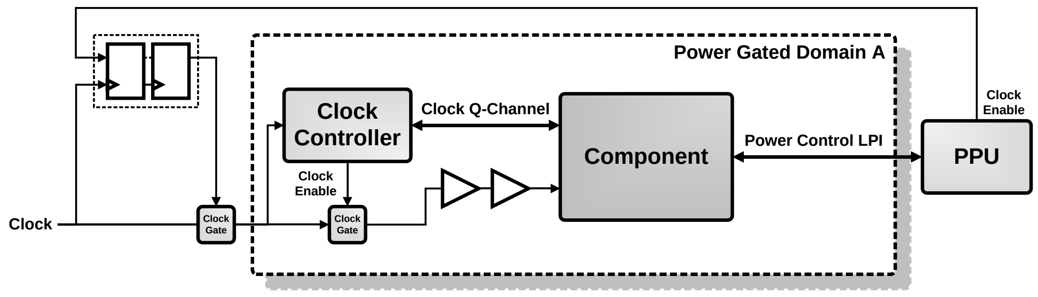


Figure 8-22 – Separate clock gates for high-Level clock gating and power control example

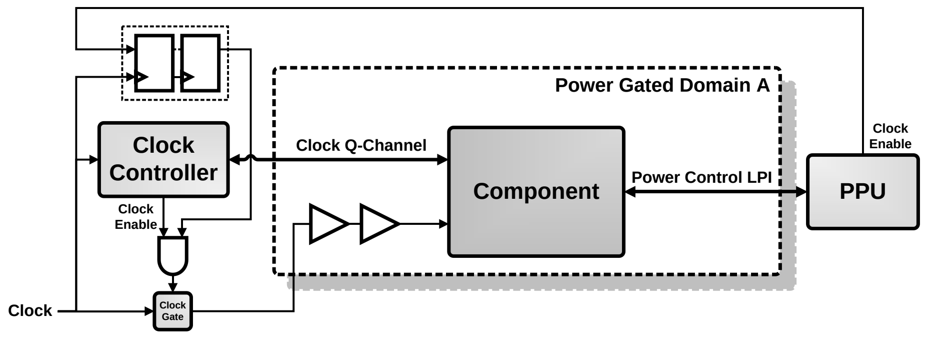


Figure 8-23 – Shared clock gate for high-Level clock gating and power control example

8.2.6 Voltage and Power Domain Boundaries

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

Voltage domains must be clocked asynchronously and the domain bridge partitioned at the asynchronous interface, one half in each voltage domain.

Power domain boundaries can be either synchronous or asynchronous. The domain bridge can be partitioned as a voltage domain bridge, with half in each domain, or the entire bridge can be within one power domain.

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

Voltage Domain Boundaries

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

A domain bridge that is used to cross between voltage domains is split into separate slave and master interface components, each using separate asynchronous clocks. These two components are then instantiated either side of the voltage domain boundary. This instantiation means the clock domains are constrained to the respective voltage domains and that isolation and level shifter cells can be added at an easily identified hierarchy level.

Figure 8-24 shows an example of this structure.

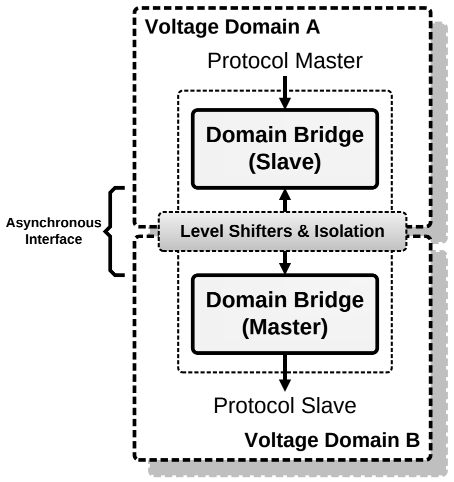


Figure 8-24 – Voltage crossing domain bridge structure

Figure 8-25 shows an example voltage domain crossing using a domain bridge with clock and power control connections.

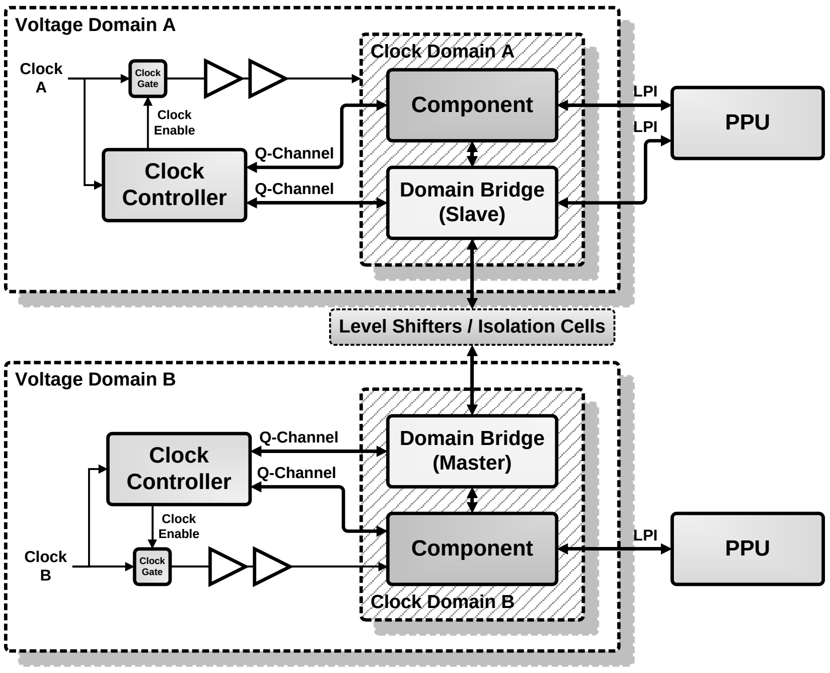


Figure 8-25 – Voltage domain crossing clock and power control example

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

The first concern is the correct operation of the bridge itself, but also transactions attempting to enter the bridge when it is not available might, depending on the capabilities of the component, also be managed.

In Figure 8-25 a power control LPI is shown connected to the slave portion of the domain bridge in the upstream power domain for this purpose. The upstream location of the example reflects that chosen for the power down interface in CoreLink ADB-400 which has semantics compatible with Q-Channel. However, since the bridge quiescence entry and exit must always be managed when both domains are powered on the location of the LPI is arbitrary.

——— **Note** ———————

Multiple power control LPIs on a bridge are not recommended. When either side of the bridge can be powered off independently the status of the each portion of the bridge is difficult to resolve without race conditions.

———————————

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

Power Gated Domain Boundaries

The boundaries between power gated domains can be synchronous or asynchronous.

Synchronous Power Gated Domain Boundaries

Figure 8-26 shows an example arrangement where the clocks to two power gated domains are synchronous.

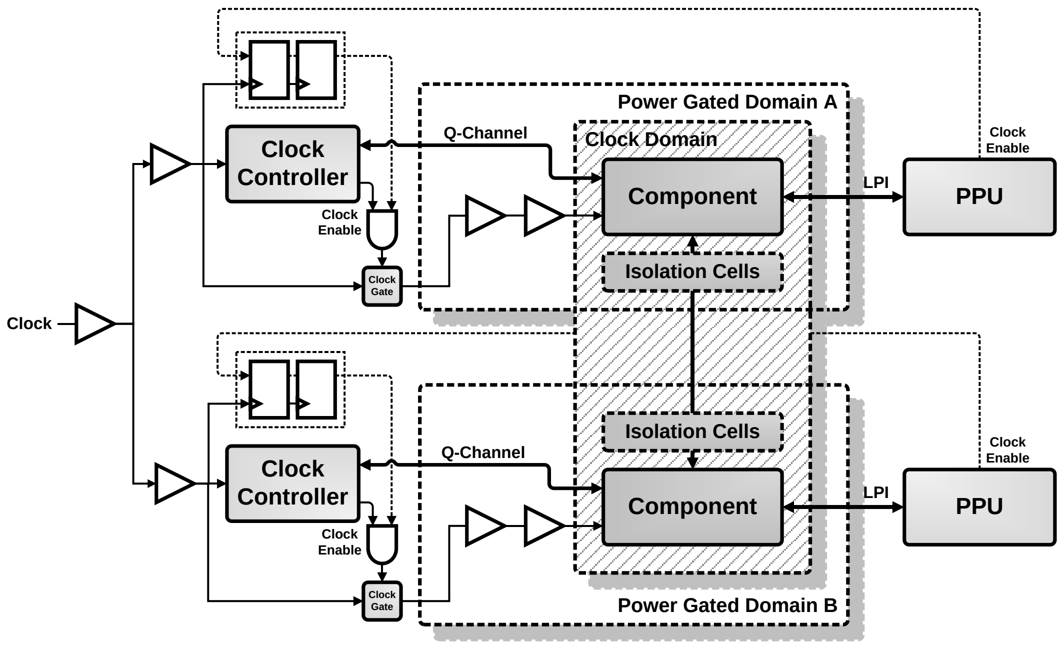


Figure 8-26 – Example of a synchronous power gated domain boundary

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 8-26 the power domain clock gating strategy of Figure 8-23 is chosen. The shared clock gate is a natural divergence point as the clock tree branches at the power domain boundaries.

A similar arrangement using the power domain clock gating structure shown in Figure 8-22 is also possible. In this case the clock controllers are placed inside the respective power gated domains. This arrangement has the same control constraints.

In both cases other system specific functionality and guarantees might be used to provide additional higher level clock gating at the common clock root for both domains. This can be achieved when both clock controllers agree the clock can be removed.

It is also possible to implement a solution with a single clock controller that is relative always on to both power gated domains. That can include placement within one of the power gated domains provided it has an always on relationship to the other power gated domain. However, this clock controller is required to be more complex than that outlined in *Clock Controller* on page 7-12. The clock controller in this case would require awareness of the power domain states for each Q-Channel to prevent handshake attempts with a powered off domain.

Asynchronous Power Gated Domain Boundaries

When the boundaries between power gated domains are asynchronous, the crossing must be managed using an asynchronous domain bridge for the required protocol.

Figure 8-27 shows an example with a domain bridge across the two power gated domains.

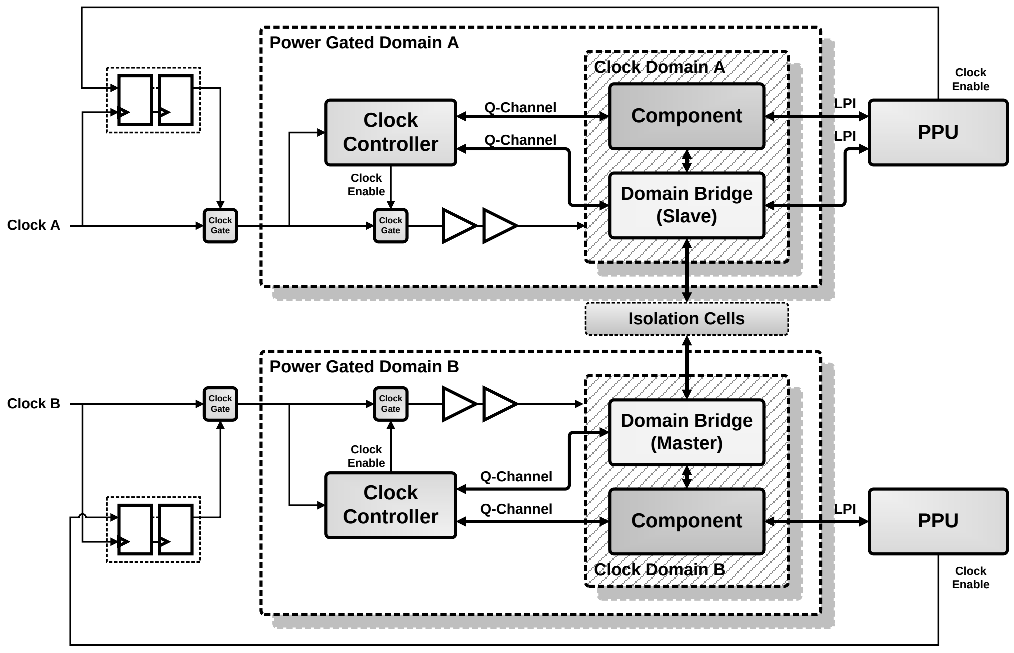


Figure 8-27 – Example of an asynchronous power gated domain boundary

The example of Figure 8-27 is similar to the voltage domain case of Figure 8-25, without level shifting, combined with power domain clock gating example of Figure 8-22.

In the case that either power gated domain is off no transactions can flow across the asynchronous domain bridge and the bridge is in its quiescent state. However, any domain bridge portion still powered on must respond to a Q-Channel handshake from its clock controller. This is because another component in the power gated domain that remains on might require the clock, and the clock controller will handshake with all components in the clock domain.

8.2.7 Access Control

In some cases it might be 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, at any time, from another part of the system which 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 entering 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 access is attempted.
- Preventing access to a resource from selected interfaces during post reset or run time configuration.

Access control support might be integrated into a master itself or into system components such as network interconnects and domain bridges. However, it is convenient to consider a standalone access control gate component. Figure 8-28 shows a simple example.

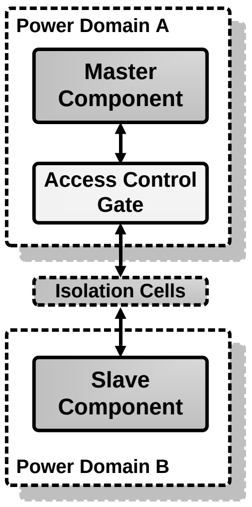


Figure 8-28 – Access control example

In Figure 8-28 the access control gate component, when in a gated state, breaks the path between the master and slave components. This allows the slave 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 8-28 shows the access control gate in the master component power domain. This location enables the generation of wake-up requests and any required sequential responses to the master component when the slave component power domain is off. In this case all of 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 transactions in-flight 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 gated state response models are considered useful and would be chosen according to the application:

- **Stall and wake:** A transaction arriving at the 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 required 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 in with some types of broadcast or trace traffic.

9 Power Control Flows

This chapter provides power control flows for critical system components and a system example.

The sequences required to manage components are given from both generic and specific ARM component perspectives as appropriate.

It contains the following sections:

These flows are representative of available ARM components at the time of the document release. Specific product documentation should always be consulted.

——— **Note** ————————

The following control flows assume the use of the components described in *Power Control Framework* on page 7-1. However, the signaling and ordering at component level is valid for any power control implementation.

————————————————

9.1 ARM Cortex A-Profile Processors

This section describes power control flows for Cortex-A profile multi-core processor products that consist of a cluster containing one or more cores. These processors support individual core power domains and a cluster power domain. The core and cluster power domains must be implemented within one voltage domain.

Some processors also support an additional debug power domain within the cluster. Since this debug logic is small it is commonly merged into the cluster power domain and this implementation is assumed in this section. This means that the cluster power domain must be on during a debug connection.

The cluster contains logic external to the cores. This includes a shared cache, snoop control logic, support for debug through core power off, and other infrastructure logic. The cluster power domain must be on before any cores are on and only powered off when all cores are off.

Although the cluster infrastructure logic can physically include core specific resources, such as timers, any state save-restore or migration of context is logically owned as part of core power control. The shared cache content is the only cluster specific state that must be managed as part of cluster power control.

——— **Note** ———————

The descriptions given in this section are intended to be generally applicable to Cortex-A profile processors. Due to micro-architectural differences not all details can be included and therefore the exact details described in the documentation provided with ARM products must always be observed.

———————————————

Not all processors support all of the features described in this section. Table 9-1 lists the features that are supported by the processor products considered within the scope of this specification.

Table 9-1 – ARM processor power features

Processor	ARM Arch. Version	Power Management Feature							
		Adv.-SIMD Dynamic Retention	Core Power off	Core Dynamic Retention	L2 RAM Dynamic Retention	Cluster Dormant Mode	Cluster Power off	L2 Cache Hardware Flush	Debug Power Domain
Cortex-A7	v7-A	NO	YES	NO	NO	YES	YES	NO	YES
Cortex-A15	v7-A	NO	YES	YES [1]	NO	YES	YES	NO	YES
Cortex-A17	v7-A	NO	YES	YES	YES	YES	YES	YES	NO
Cortex-A53	v8-A	YES	YES	YES	YES	YES	YES	YES	NO
Cortex-A57	v8-A	NO	YES	YES	YES	YES	YES	YES	YES
Cortex-A72	v8-A	NO	YES	YES	YES	YES	YES	YES	YES

[1] From version r3p0

9.1.1 Wake Scenarios

A core power on is typically performed as the result of one of the following events:

- System boot.
- Interrupts from the Generic Interrupt Controller (GIC).
    - When a core is off interrupts are routed via the power controller, using specific signalling from the GIC. For more information see *Generic Interrupt Controller* on page 9-30.
- A request from the OSPM running on a different core or cluster.
- Always on domain wake events.
- A debug power up request using the EDPRCR.COREPURQ bit.
    - The cluster is required to be powered on for this bit to be programmed

If the cluster or supporting system power domains are off when the wake event occurs then these domains must be powered on in the correct order before a core can be powered on.

The cluster is typically woken as a result of a wake event for one of its cores. However, it might be woken without a corresponding core power on to service snoops when in a low power state that retains coherent data in its cache. This is described in *Using Dormant Mode in a Coherent System* on page 9-11.

9.1.2 Cluster Power Control Sequences

Figure 9-1 shows the connections referred to in the following description of processor cluster power domain control sequences. It is not intended to show an entire system.

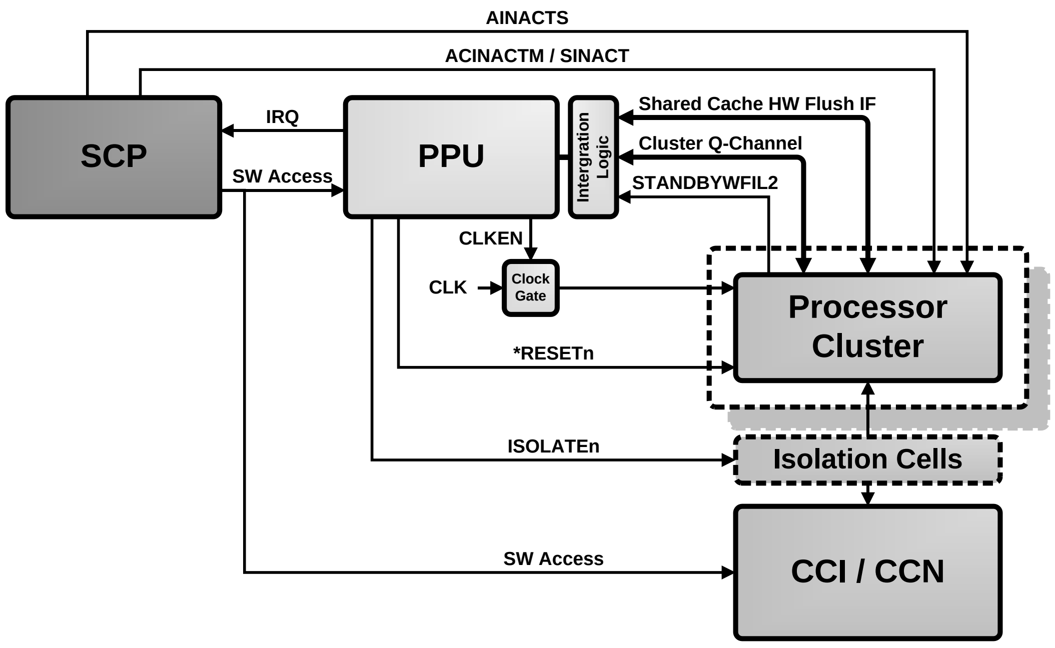


Figure 9-1 –Cluster domain power control connections

The **ACINACTM** (for processors configured with an AMBA4 ACE interface) and **SINACT** (for processors configured with an AMBA5 CHI interface) inputs on the processor, when set HIGH, provide a guarantee to the processor that it will not be sent any transactions from a connected coherent interconnect.

The **AINACTS** input on the processor, when set HIGH, provides a guarantee to the processor that it will not receive any transactions on its Accelerator Coherency Port (ACP) slave AXI interface.

The shared cache hardware flush interface allows flushing of the cluster shared cache, when powering off the cluster, to be delegated to SCP or other hardware control. This interface must only be used when all cores are powered off. This delegation also requires access to CCI/CCN registers to control connecting and disconnecting the cluster from system coherency.

If shared cache flushing is performed by AP firmware, then it is not required for the SCP to access the CCI/CCN registers.

The cluster Q-Channel is only present on processors supporting shared cache dynamic retention.

The **STANDBYWFIL2** output indicates that the processor cluster is idle and can be powered off. **STANDBYWFIL2** is asserted when all of the following conditions are met:

- All cores are in WFI.
- **ACINACTM** or **SINACT** is HIGH.
- **AINACTS** is HIGH.
- There is no internal activity in the cluster.

STANDBYWFIL2 does not guarantee that all cores are powered off. The SCP must ensure that all cores are off before powering off the processor cluster, or if the entire processor cluster is implemented as a single power domain then it must ensure that cores cannot exit WFI.

Cluster Power On

Figure 9-2 shows an example cluster off to on sequence. Lines shown in dotted grey are optional depending upon the availability and use of processor functionality.

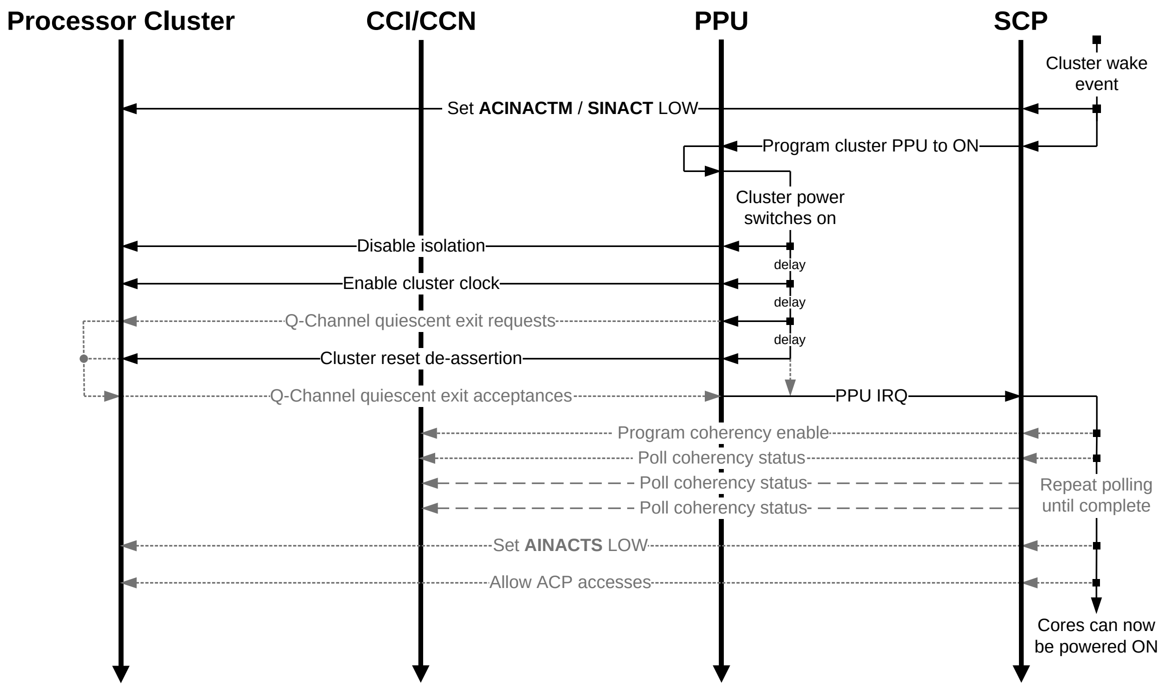


Figure 9-2 – Processor cluster power on sequence

Where 'delay' is indicated in Figure 9-2, this indicates an implementation specific delay to allow signals to propagate or meet specific requirements on signal timings.

In this example the system coherency enable sequence is performed by the SCP. However, this can also be performed by software on a processor core once it wakes.

The following details the sequence. Processor specific, but common, actions are included in brackets:

1. A wake event at the SCP indicates the cluster is required.
2. The SCP sets **ACINACTM** or **SINACT** LOW
3. The SCP programs the cluster PPU policy to ON.

The PPU then sequences the following actions:

1. The PPU turns on cluster domain power switches.
2. The PPU disables isolation cells.
3. The PPU enables cluster clocks.
4. The PPU waits for a processor specific number of clock cycles with reset asserted.
5. (The PPU makes any required Q-Channel quiescence exit requests).
    - If present, the PPU sets **L2QREQn** HIGH.
    - It also controls any domain bridges to allow access to and from the external system, for example:
        - AMBA AXI / ACE / CHI interfaces.
        - Debug APB and CoreSight trace interfaces.
        - Event interfaces.
        - GIC interfaces.
6. The PPU de-asserts cluster resets.
7. (The PPU waits for any required Q-Channel responses).
8. The PPU sends an interrupt to the SCP.
9. (The SCP connects the cluster to system coherency).
    - If not managed by the SCP this can be managed by software running on a core once it is on.
    - For this procedure see *Enabling and Disabling System Coherency* on page 9-20.
10. (The SCP sets **AINACTS** LOW).
11. (The SCP allows accesses to the ACP port).

Once this sequence is complete the SCP can power on any required cores.

Shared Cache RAM Dynamic Retention

Dynamic retention allows the cluster shared cache RAM to enter a retention state when all cores are in either WFI or WFE. This saves leakage power in addition to the dynamic power saved by clock gating. Dynamic retention entry and exit is controlled with the cluster Q-Channel.

The conditions required for dynamic retention are:

- All cores are in WFI or WFE.
- There are no outstanding snoops from the system to the shared cache.
- There are no ACP accesses.

The opportunity to enter the dynamic retention is indicated by **L2QACTIVE** going LOW.

If any of these conditions change the cluster will indicate a requirement to exit dynamic retention by setting **L2QACTIVE** HIGH.

The Q-Channel handshake is used to manage any interlocks required to ensure correct operation during entry and exit, and for the duration of the retention state.

Shared cache RAM dynamic retention is supported in selected cores. These are listed in Table 9-1 on page 9-2.

Enabling Shared Cache RAM Dynamic Retention

For the ARMv8-A cores in Table 9-1, shared cache RAM dynamic retention is enabled in the cluster by software setting the *L2 Dynamic Retention Control* field in L2ECTLR. This is a three bit field that when non-zero means dynamic retention is enabled. The value of the field indicates how many architectural timer ticks occur between the conditions for retention entry being met and **L2QACTIVE** going LOW. For full details of these values see individual product Technical Reference Manuals.

If dynamic retention is not enabled **L2QACTIVE** will stay HIGH and quiescent requests on the cluster Q-Channel will be denied.

The Cortex-A17, ARMv7-A, processor has a different use model. For this processor cluster dynamic retention is enabled by default. It can be disabled by software setting the cores L2ECTLR *L2 dynamic RAM retention mode* bit. Also in this processor there is no support for a timeout period, **L2QACTIVE** will go LOW as soon as the conditions for retention entry are met. However, the PPU supports a timeout value between sampling **L2QACTIVE** LOW and making a Q-Channel quiescence entry request.

The PPU can be programmed to manage entry and exit of dynamic retention autonomously in response to **L2QACTIVE**. For full details of programming the PPU see the *ARM® Power Policy Unit Architecture Specification Version 1.0*.

Shared Cache RAM Dynamic Retention Entry

Figure 9-3 shows the sequence for entering dynamic retention.

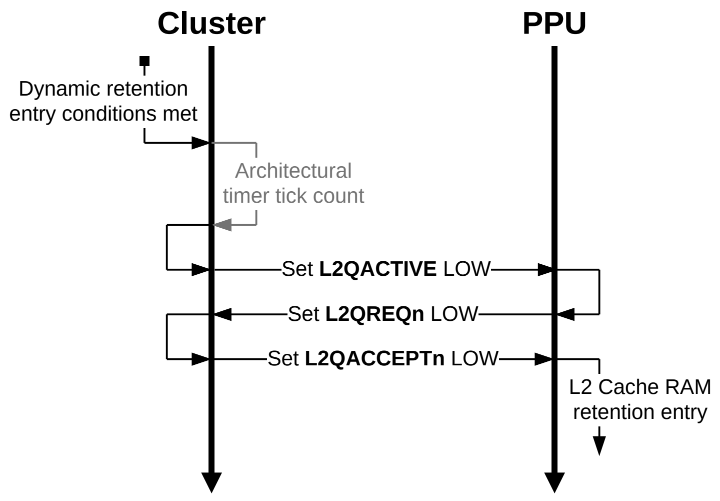

Cluster
PPU
Dynamic retention entry conditions met
Architectural timer tick count
Set L2QACTIVE LOW
Set L2QREQn LOW
Set L2QACCEPTn LOW
L2 Cache RAM retention entry

Figure 9-3 – L2 Cache RAM dynamic retention entry sequence

The following details the entry sequence, assuming that dynamic retention has been enabled in the processor and the PPU:

1. All entry conditions for dynamic retention are met.
2. The cluster waits for the required number of architectural timer ticks as programmed in the L2ECTLR *L2 Dynamic Retention Control* field.
3. The cluster sets **L2QACTIVE** LOW.
4. The PPU sets **L2QREQn** LOW.
5. The cluster responds with **L2QACCEPTn** LOW.
6. The PPU then puts the shared cache RAM into a retention state.

If there is activity in the cluster when the quiescent entry request is made, the cluster denies the request and operations will continue without interruption.

Figure 9-4 shows an example sequence when a quiescent request is denied.

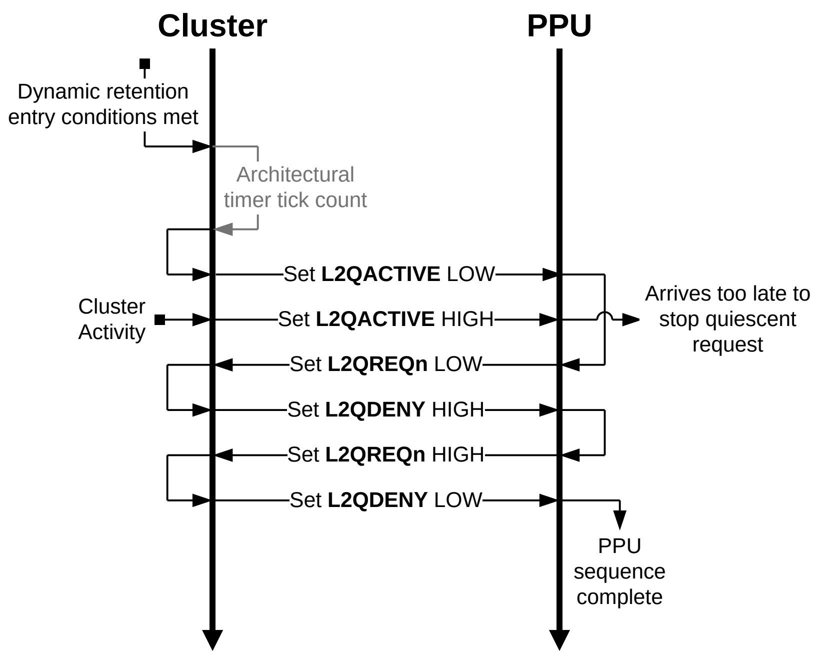


Figure 9-4 – L2 Cache RAM dynamic retention denial sequence

The following details the denial sequence:

1. All entry conditions for dynamic retention are met.
2. The cluster waits for the required number of architectural timer ticks as programmed in the L2ECTLR *L2 Dynamic Retention Control* field.
3. The cluster sets **L2QACTIVE** LOW.
4. There is cluster activity, for example a snoop arrives.
5. The cluster sets **L2QACTIVE** HIGH, but this is sampled at the PPU too late to stop the quiescent request.
6. The PPU sets **L2QREQn** LOW.
7. The cluster sets **L2QDENY** HIGH.
8. The PPU sets **L2QREQn** HIGH.
9. The cluster sets **L2QDENY** LOW.

Throughout this sequence the cluster continues operations without interruption. Once the PPU has sampled **L2QDENY** LOW it is free to make another quiescent request, but should wait until **L2QACTIVE** goes LOW again.

Shared Cache RAM Dynamic Retention Exit

Figure 9-5 shows the sequence for exiting dynamic retention.

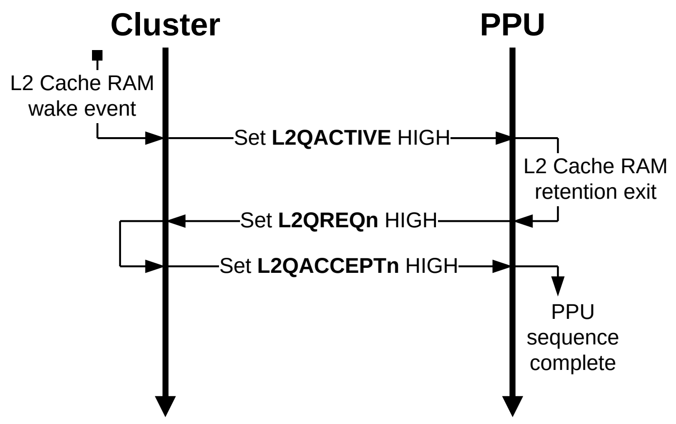


Figure 9-5 – L2 Cache RAM dynamic retention exit sequence

The following details the exit sequence:

1. A shared cache RAM wake event occurs.
    - For example, a snoop arrives.
2. The cluster sets **L2QACTIVE** HIGH.
3. The PPU brings the shared cache RAM out of retention.
4. The PPU sets **L2QREQn** HIGH.
5. The cluster responds with **L2QACCEPTn** HIGH.
6. Once the PPU has sampled **L2QACCEPTn** HIGH the sequence is complete.

The cluster can progress operations once it has sampled **L2QREQn** set HIGH, it does not have to wait until it has set the **L2QACCEPTn** HIGH or this is sampled by the PPU.

Cluster Power Off

When all cores are off the cluster can be powered off.

Where the cluster contains a shared cache this must be flushed, meaning cleaned *and* invalidated, before the cluster is powered off. This ensures data is not lost and system coherency is maintained.

In this section a lead core is defined as the last core to power off prior to powering off the cluster.

The shared cache flush can be managed in two ways:

1. Software on the lead core flushes the shared cache before it powers off.
2. The shared cache hardware flush interface flushes the cache once all cores are off.
    - This feature is only supported on selected ARM processors as shown in Table 9-1.

Figure 9-6 shows a coherent cluster power off sequence. Lines shown in dotted grey are optional and depend upon the availability and use of processor functionality.

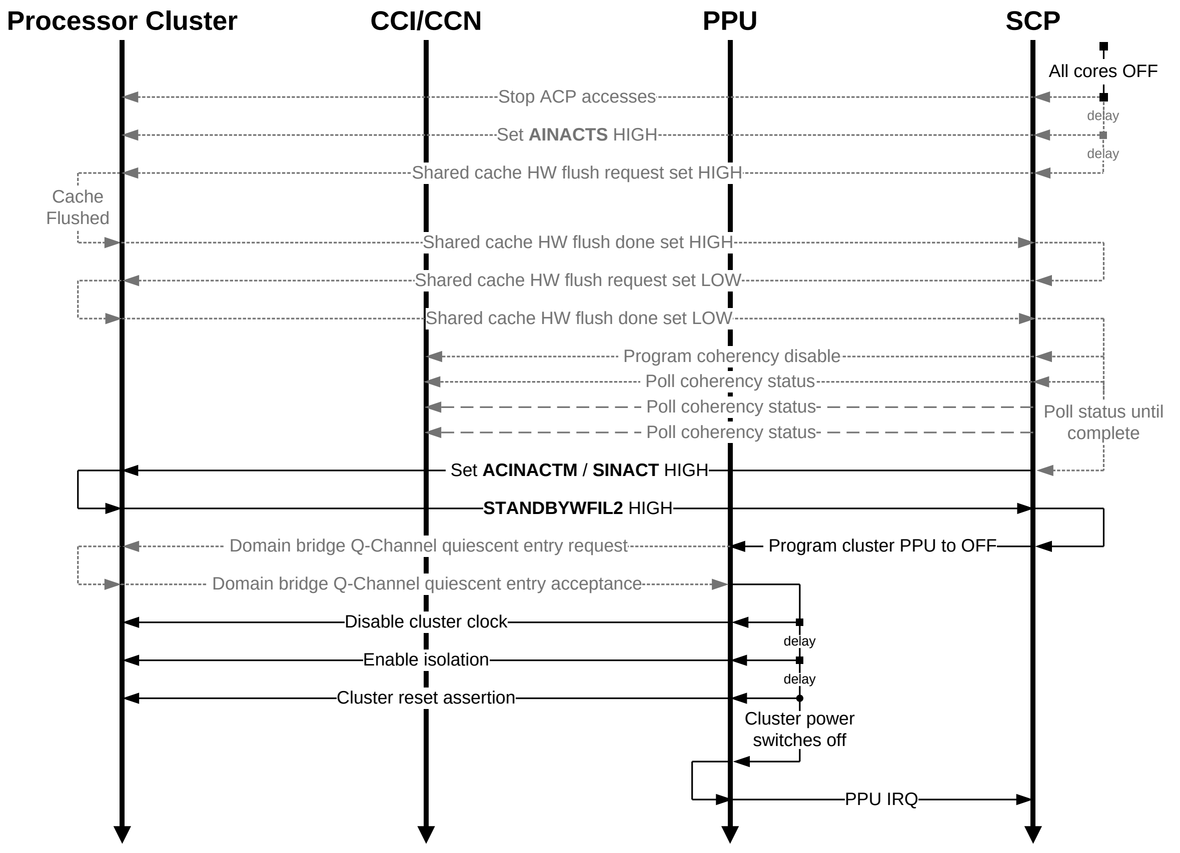

Processor Cluster
CCI/CCN
PPU
SCP
All cores OFF
Stop ACP accesses
delay
Set AINACTS HIGH
delay
Shared cache HW flush request set HIGH
Cache Flushed
Shared cache HW flush done set HIGH
Shared cache HW flush request set LOW
Shared cache HW flush done set LOW
Program coherency disable
Poll coherency status
Poll coherency status
Poll coherency status
Poll status until complete
Set ACINACTM / SINACT HIGH
STANDBYWFIL2 HIGH
Domain bridge Q-Channel quiescent entry request
Program cluster PPU to OFF
Domain bridge Q-Channel quiescent entry acceptance
Disable cluster clock
delay
Enable isolation
delay
Cluster reset assertion
Cluster power switches off
PPU IRQ

Figure 9-6 – Processor cluster power off sequence

The initial grey lines show the sequence for a hardware controlled shared cache flush. If this is not supported, then software on the lead core must perform these actions and the hardware sequence starts when **ACINACTM** or **SINACT** is set HIGH.

Where ‘delay’ is indicated in Figure 9-6, this indicates an implementation specific delay to allow signals to propagate or meet specific requirements on signal timings.

The following sections detail the power off sequence. Actions which are processor specific but common are included in brackets.

Cluster Power Off with Software Cache Flush Specific Actions

The method for flushing the shared cache with software running on the lead core is dependent on the processor being used, but generally includes the following steps:

1. All cores except the lead core must be off.
2. Software disables interrupts by configuring the PSTATE.DAIF bits.
3. Software saves core context, GIC CPU interface state and core private timer state.
4. Software clears the SCTLR.C bit. This prevents further data cache allocations.
5. Software performs any other processor specific cache management.
6. Software flushes the L1 data cache.
7. Software ensures the ACP port will not be accessed, and then sets **AINACTS** HIGH.
8. Software flushes the shared cache.
9. Software disables data coherency by setting CPUECTLR.SMPEN (for ARMv8-A cores), or ACTLR.SMP (for ARMv7-A cores) LOW.
10. Software disables interrupts in the GIC.
    - For this procedure see *Generic Interrupt Controller* on page 9-30.
11. Software sets DBGOSDLR.DLK HIGH to prevent access from the external debug interface.

12. Software disconnects the cluster from system coherency.
    - For this procedure see *Enabling and Disabling System* Coherency on page 9-20.
13. Software executes an ISB and a DSB instruction.
14. Software executes a WFI instruction.
15. The core powers off with the sequences described in *Core Power Off* on page 9-18.

Once the lead core is off the cluster power off sequence can continue as specified in *Cluster Power Off Common Actions* on page 9-10.

Cluster Power Off with Hardware Flush Specific Actions

The method for flushing the shared cache by hardware is dependent on the processor being used, but generally includes the following steps:

1. All cores must be OFF.
2. The SCP ensures the ACP port will not be accessed, and then sets **AINACTS** HIGH.
3. The SCP performs a L2 Cache HW flush handshake.
    - A detailed example of this procedure is provided in *Cortex-A53 Cluster Power Off* on page 9-24.
4. The SCP disconnects the cluster from system coherency.
    - For these procedures see *Enabling and Disabling System Coherency* on page 9-20.
    - The SCP requires access to access CCI/CCN registers to perform this procedure.

The cluster power off sequence can continue as specified in *Cluster Power Off Common Actions* on page 9-10.

Cluster Power Off Common Actions

The remainder of the OFF sequence is common to both methods:

1. The SCP sets **ACINACTM** or **SINACT** HIGH.
2. The SCP waits for **STANDBYWFIL2** to be asserted, indicated as an interrupt.
3. The SCP programs the cluster PPU to OFF.

The PPU then sequences the following actions:

1. (The PPU makes any required Q-Channel quiescence requests).
    - This does not include the cluster Q-Channel used for dynamic retention.
    - This includes controlling domain bridges that provide access to and from the external system, for example:
        - AMBA AXI / ACE / CHI interfaces.
        - Debug APB and CoreSight trace interfaces.
        - Event interfaces.
        - GIC interfaces.
2. (The PPU waits for any required Q-Channel responses).
3. The PPU disables the cluster clocks.
4. The PPU enables isolation.
5. The PPU asserts resets.
6. The PPU turns off cluster domain power switches.
7. The PPU, through the integration logic, resets the cluster Q-Channel to the *Q_STOPPED* state.
8. The PPU sends an interrupt to the SCP.

Cluster Dormant Mode

Cluster dormant mode is similar to cluster power off with the exception that the shared cache RAM is retained. The use of dormant mode must be managed between the OSPM stack and the SCP by some system defined method.

In order to maintain the cache content and prevent automatic invalidation of the cache at reset de-assertion the **L2RSTDISABLE** input must be set HIGH. This must not be done when the cache RAM has been powered off.

In the power off sequence the steps of flushing the cache and removing the cluster from coherency do not need to be performed. However, the **AINACTS** and **ACINACTM** inputs must still be set HIGH. If this is not done **STANDBYWFIL2** will not go HIGH and the SCP will not start the power off sequence.

Using Dormant Mode in a Coherent System

Use of dormant mode in a coherent system is typically limited to SoC SLEEP states as the snoop or ACP access latency associated with waking the cluster is anticipated to be significant.

In a system where retained caches contain coherent data, system specific means must use one of the following methods to maintain system coherency during dormant mode.

1. Ensure that no snoops or ACP accesses will be made while the cluster is in the dormant mode.

   The system must ensure all other coherent masters are inactive when the cluster enters the dormant mode, and that the cluster exits dormant mode before any other coherent master is active.

2. Ensure incoming snoops or ACP accesses while the cluster is in dormant mode will be stalled and wake the cluster so they can be processed.

   This provides a robust mechanism for ensuring the coherent system continues to operate correctly.

   In this case the cluster must not be removed from the coherency domain. Any other master accessing the address of any data stored in the dormant cache would lead to a loss of system coherency as a result of either stale data being read from memory or the data in the dormant cache not being invalidated or updated.

9.1.3 Core Power Control Sequences

Figure 9-7 shows the connections referred to in the following description of core power domain control sequences. It is not intended to show an entire system.

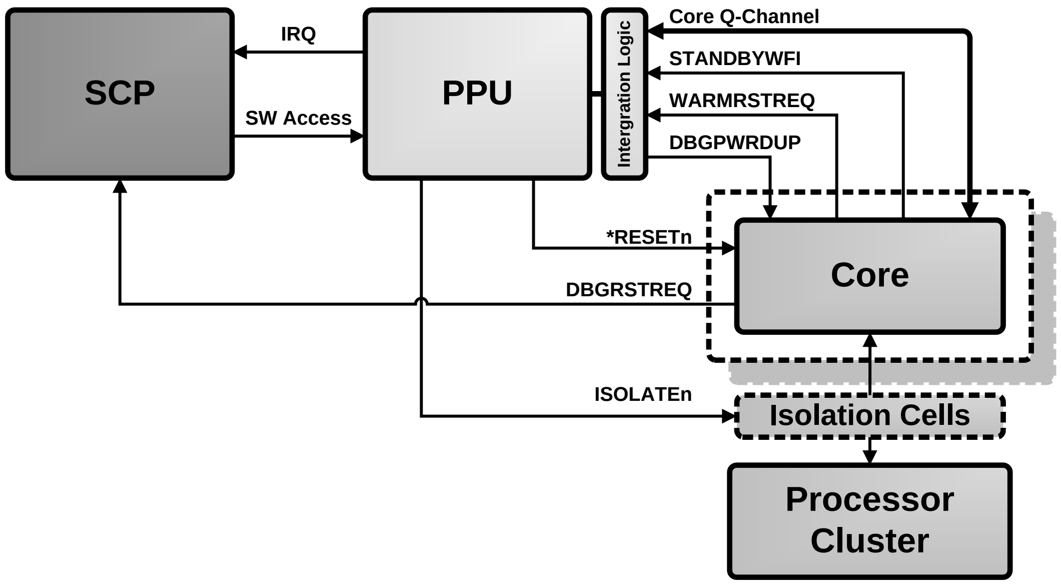


Figure 9-7 – Core domain power control connections

The core Q-Channel is present on processors supporting dynamic retention. It can also be used, optionally, in core power on and power off control sequences. This is recommended when core dynamic retention is implemented.

The **STANDBYWFI** output indicates that the core is in a WFI state.

The **WARMRSTREQ** and **DBGRSTREQ** are requests for core reset, for more information see *Core Reset* on page 9-12.

The **DBGPWRDUP** input indicates that the core is powered on. It should be set LOW before the core is powered off and set HIGH after the core is powered on.

———**Note**—————————

The **DBGPWRDUP** input is supported by all processors described within this section except Cortex-A15. For more information see the *ARM Cortex-A15 MPCore Processor Technical Reference Manual*.

————————————————

Core Reset

There are several conditions in which a core can be reset. These can be categorized as power on reset and warm reset.

A power on reset will reset all logic in the core and is used in the power on and power off sequences of the core power domain.

A warm reset is applied only to the core functional logic, but does not reset the core debug logic. Debug resources remain externally accessible during warm reset assertion.

A core warm reset is requested in the following conditions:

- For ARMv8-A cores only: a warm reset request signal, **WARMRSTREQ**, is driven by the RMR_ELx.RR bit.
    - This provides a mechanism to change the execution state the core enters after warm reset is de-asserted. This execution state is either AArch32 or AArch64 and is determined by RMR_RLx.AA64.
- A debug warm reset request signal, **DBGRSTREQ**, is driven by the EDPRCR.CWRR bit.

——— **Note** ———————

There is only one RMR_ELx.RR register at the highest exception level implemented. This is EL3 if all exception levels are implemented.

———————————————

Warm Reset Request

For a warm reset request using RMR_ELx.RR the core must be in WFI state before the reset is applied. This ensures the core is in a quiescent state and can be safely reset without any side effects on the wider system. The WFI state of the core can be determined by observing the **STANDBYWFI** output.

Therefore the recommended behavior is:

- Software sets RMR_ELx.RR HIGH to request the warm reset.
    - The setting of RMR_ELx.AA64 determines the AArch execution state entered, at the highest exception level implemented, after the warm reset.
- Software executes an ISB.
- Software executes a WFI instruction.
- The power controller samples the warm reset request, **WARMRSTREQ**, and **STANDBYWFI** HIGH and asserts the core reset.
- The power controller observes the de-assertion of **WARMRSTREQ** and de-asserts the core reset.

Debug Warm Reset Request

A debug reset request can be performed by the debugger with no synchronization to the core state.

——— **Note** ———————

Any assertion of reset when a core is not in a quiescent state, that is not in WFI or OFF, can have unpredictable effects on the system.

———————————————

Therefore, in the absence of synchronization, a debug reset request will normally require a wider system reset to ensure that the system will be able to function correctly.

This can be managed by passing the **DBGRSTREQ** signal to the SCP as an interrupt. The SCP software can then reset the required parts of the system.

Core Power On

Figure 9-8 shows a power on sequence. It assumes the cluster is already on. Lines shown in dotted grey are optional, depending upon the availability and use of the processor functions.

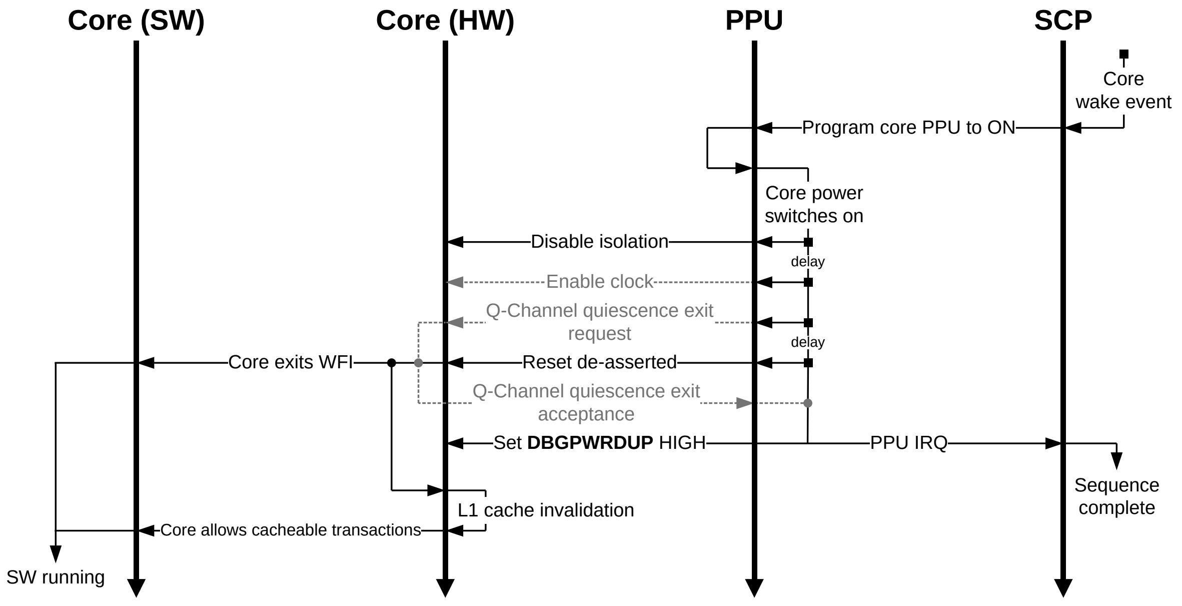


Figure 9-8 – Core power on hardware control sequence

Processors that have a single shared clock for the cores and the cluster do not need to enable the clock as it will have been enabled at cluster power on.

The following details the sequence. Actions which are processor specific but common are included in brackets:

1. A wake event at the SCP indicates the core is required.
2. The SCP programs the core PPU policy to ON.
3. The PPU turns on the core power domain switches.
4. The PPU disables the core power domain isolation cells.
5. (The PPU enables the core clock).
6. Wait for a processor specific number of clock cycles with reset asserted.
7. (Make any required Q-Channel quiescent exit request).
8. De-assert core resets.
    - The core will immediately exit the WFI state.
9. (Wait for any required Q-Channel response).
10. Set **DBGPWRDUP** HIGH.
11. The PPU sends an interrupt to the SCP to indicate the process is complete.

Software Sequence

Once the hardware sequence is complete there are several actions which need to be carried out by software. The following sequence describes these actions:

- Software enables data coherency by setting CPUECTLR.SMPEN (for ARMv8-A cores) or ACTLR.SMP (for ARMv7-A cores).
- Software enables data caching by setting SCTLR.C.
- Software restores core context, GIC CPU interface state and core private timer state.
- Software enables interrupts in the GIC,
    - o For this procedure see *Generic Interrupt Controller* on page 9-30.

Core Dynamic Retention

Dynamic retention allows a core in WFI or WFE to enter a retention state transparent to software. This saves leakage power in addition to the dynamic power saved by clock gating. Dynamic retention entry and exit is controlled with the core Q-Channel.

The required conditions, evaluated by processor logic, for dynamic retention are:

- The core is in WFI or WFE.
- There are no snoops to the core.
- There are no debug APB accesses to the core.

The opportunity to enter the dynamic retention is indicated by **CPUQACTIVE** being LOW.

If any of these conditions change, the core will indicate a requirement to exit dynamic retention by setting **CPUQACTIVE** HIGH.

The Q-Channel handshake is used by the processor to manage any interlocks required to provide safe entry and exit transitions.

Core dynamic retention is supported in selected cores; these are listed in Table 9-1 on page 9-2.

Enabling Core Dynamic Retention

For the ARMv8-A processors in Table 9-1, core dynamic retention is enabled by configuring the CPUECTLR.CPURETCTL field. This is a three bit field which when non-zero means dynamic retention is enabled for the core. The value of the field indicates how many architectural timer ticks occur between the conditions for retention entry being met and **CPUQACTIVE** going LOW. For full details of these values see individual product Technical Reference Manuals.

The Cortex-A17 and Cortex-A15, ARMv7-A, processors have different use models.

For the Cortex-A17 core dynamic retention is enabled by default. It can be disabled by software setting the cores SCUCTLR *Processor retention mode* bit for the relevant core.

For Cortex-A15 core dynamic retention is enabled for all cores by setting the *CPU WFI retention mode* bit in L2ACTLR.

In both of these processors there is no support for a timeout period. **CPUQACTIVE** will go LOW as soon as the conditions for retention entry are met. However, the PPU supports a timeout value between it sampling **CPUQACTIVE** LOW and making a Q-Channel quiescent entry request.

For all processors, if dynamic retention is not enabled **CPUQACTIVE** will stay HIGH and quiescent requests on the Q-Channel will be denied.

The PPU can be programmed to manage entry and exit of dynamic retention autonomously in response to **CPUQACTIVE** state changes. For full details of programming the PPU see the *ARM® Power Policy Unit Architecture Specification Version 1.0*.

Core Dynamic Retention Entry

Figure 9-9 shows the sequence for entering dynamic retention.

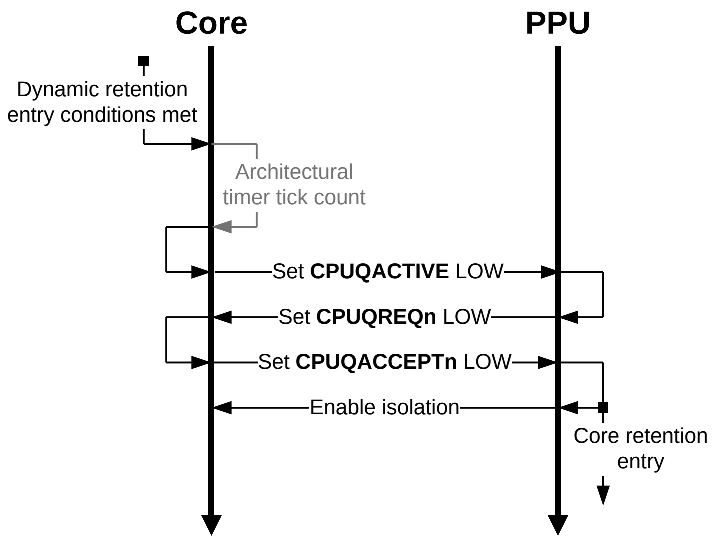


Figure 9-9 – Core dynamic retention entry sequence

The following details the entry sequence assuming that dynamic retention has been enabled in the processor and the PPU:

1. All entry conditions for dynamic retention are met.
   - The initial entry point is when software executes a WFI or WFE.
   - A core can re-enter retention after being woken, from retention, for a snoop or debug access.
2. ARMv8-A cores supporting retention wait for the required number of architectural timer ticks as programmed in the CPUECTLR.CPURETCTL field. ARMv7-A cores supporting retention do not wait for any period.
3. The core sets **CPUQACTIVE** LOW.
4. The PPU sets **CPUQREQn** LOW.
5. The core responds with **CPUQACCEPTn** LOW.
6. The PPU enables the core power domain isolation.
7. The PPU puts the logic and memories into a retention state.

If there is activity in the core when the quiescent entry request is made the core denies the request and operations will continue without interruption.

Figure 9-10 shows an example sequence when a quiescent request is denied.

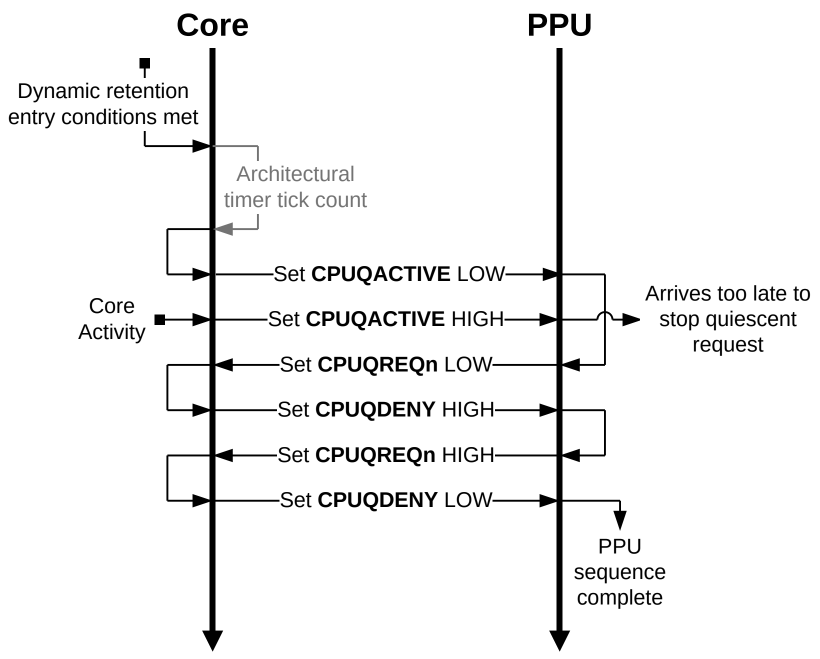


Figure 9-10 – Core dynamic retention denial sequence

The following details the denial sequence:

1. All entry conditions for dynamic retention are met.
2. The core waits for the required number of architectural timer ticks as programmed in the CPUECTLR *CPU Retention Control* field.
3. The core sets **CPUQACTIVE** LOW.
4. There is core activity, for example an interrupt arrives.
5. The core sets **CPUQACTIVE** HIGH, but this is sampled at the PPU too late to stop the quiescent request.
6. The PPU sets **CPUQREQn** LOW.
7. The core sets **CPUQDENY** HIGH.
8. The PPU sets **CPUQREQn** HIGH.
9. The core sets **CPUQDENY** LOW.

Throughout this sequence the core continues operations without interruption. Once the PPU has sampled **CPUQDENY** LOW it is free to make another quiescent request, but should wait until **CPUQACTIVE** goes LOW again.

Core Dynamic Retention Exit

Figure 9-11 shows the sequence for exiting dynamic retention.

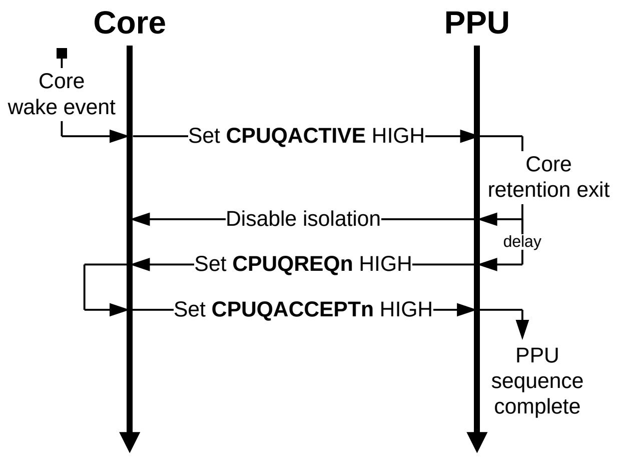


Figure 9-11 – Core dynamic retention exit sequence

The following details the exit sequence under the same assumptions as the entry sequence:

1. A core wake event happens. For example, an interrupt for the core arrives.
2. The core sets **CPUQACTIVE** HIGH.
   - The **CPUQACTIVE** generation logic is in the cluster so can operate when the core is in dynamic retention.
3. The PPU brings the core logic and memories out of the retention state.
4. The PPU disables core power domain isolation.
5. The PPU sets **CPUQREQn** HIGH.
   - Depending on the wake event the core can now exit WFI or WFE resuming execution, service a snoop or service a debug access.
6. The core responds with **CPUQACCEPTn** HIGH.
7. The PPU samples **CPUQACCEPTn** HIGH and the sequence is complete.

Core Power Off

A core is powered off in response to a specific software request. The sequence requires software to communicate to the SCP that the core should be powered off at the next WFI entry.

Software Sequence

The software actions which are required to be completed before the power off are listed below:

1. Software disables interrupts by configuring the PSTATE.DAIF bits.
2. Software communicates with the SCP to request power off at the next WFI entry.
3. Software saves core context, GIC CPU interface state, and core private timer state.
4. Software disables data caching by setting SCTLR.C LOW.
   - Software performs any other required processor specific cache management.
5. Software flushes the L1 data cache.
6. Software sets LOW:
   - CPUECTLR.SMPEN (for ARMv8-A cores).
   - ACTLR.SMP (for ARMv7-A cores).
7. Software disables the GIC CPU interface.
   - Interrupts are re-directed to the SCP as wake requests.
   - For this procedure see *Generic Interrupt Controller* on page 9-30.

8. Software sets the OS Double Lock Control bit HIGH.
    - OSDLR.DLK (for ARMv8-A cores executing in AArch64).
    - DBGOSDLR.DLK (for ARMv7-A and ARMv8-A cores executing in AArch32).
9. Software executes an ISB and a DSB instruction.
10. Software executes a WFI instruction.

Hardware Sequence

Figure 9-12 shows the hardware sequencing relative to the software sequence above. Lines shown in dotted grey are optional, depending upon the availability and use of the processor functions.

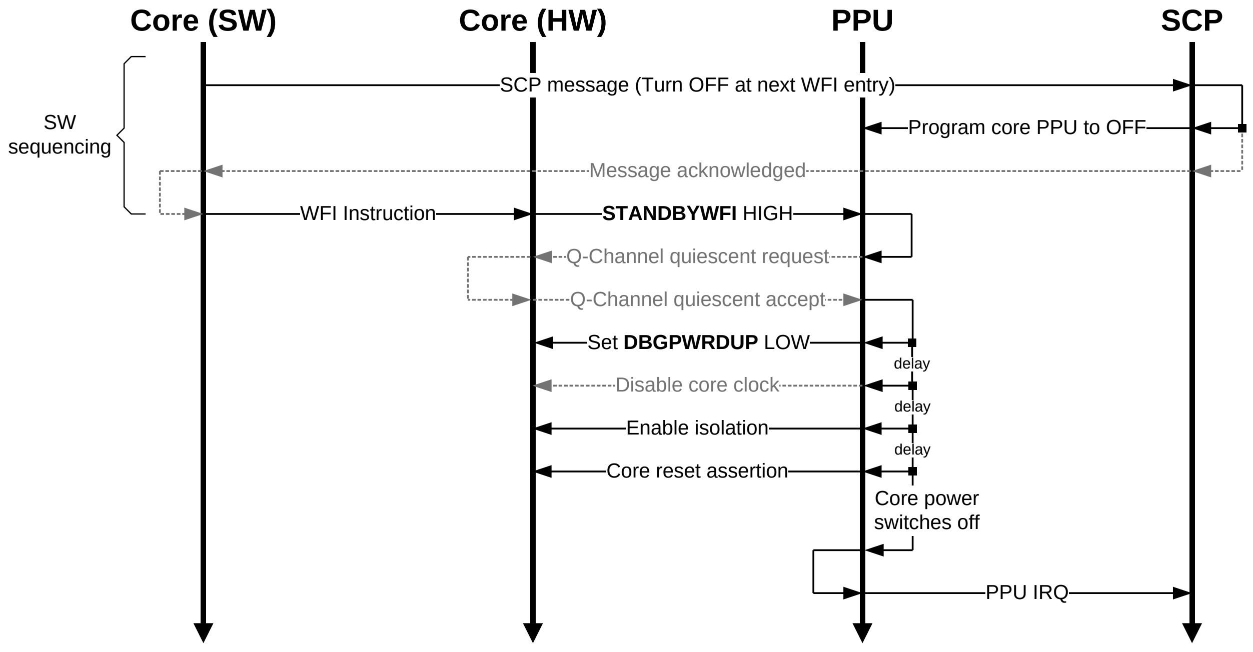


Figure 9-12 – Core power off sequence

Where 'delay' is indicated in Figure 9-12, this indicates an implementation specific delay to allow signals to propagate or meet specific requirements on signal timings.

The following details the hardware sequence. Actions which are optional dependent on features present or used, are included in brackets:

1. Software communicates with the SCP to request power off at the next WFI entry.
2. The SCP programs the core PPU policy to OFF and, optionally, acknowledges the message to software.
3. The PPU waits, through integration logic, for **STANDBYWFI** to go HIGH.
    - **CPUQACTIVE** will be set LOW if the core dynamic retention is enabled.
4. (The PPU makes any required Q-Channel quiescence requests).
5. (The PPU waits for any required Q-Channel responses).
6. The PPU, through integration logic, sets **DBGPWRDUP** LOW.
7. (The PPU disables the core clock).
    - This step is not required if the cluster and core share a single clock. In this case the processor internally gates the clock to the core.
8. The PPU enables core power domain isolation cells.
9. The PPU asserts the core power on reset.
10. The PPU turns off core power domain switches.
11. The PPU sends an interrupt to the SCP to indicate the process is complete.

9.1.4 Enabling and Disabling System Coherency

This section describes control flows for the connection and disconnection of coherent masters to and from system coherency.

Note

For processor clusters in dormant mode, where logic is powered off but data is retained in the cache RAM, there are special provisions required, these are detailed in *Cluster Power Control Sequences* on page 9-3.

CoreLink CCI

This section details the general sequence for CoreLink CCI-400 and CoreLink CCI-500.

Adding a Master to the Coherent Domain

Before a master issues any shareable transactions:

1. Set the CCI Snoop Control Register bits 0 and 1 HIGH for the CCI slave interface connected to the master.
2. Poll the CCI Status Register bit 0.
    - If this bit is HIGH then snoop enabling is ongoing.
    - If this bit is LOW then snoop enabling is complete.

When the CCI Status Register bit 0 is LOW the master is connected to the coherency domain.

Note

Once the CCI Snoop Control Register is set LOW it must not be set HIGH again until the CCI Status Register bit 0 is LOW.

Removing a Master from the Coherent Domain

Once it is ensured that the master has stopped issuing any shareable transactions and its cache is flushed:

1. Set the CCI Snoop Control Register bits 0 and 1 LOW for the CCI slave interface connected to the master.
2. Poll the CCI Status Register bit 0.
    - If this bit is HIGH then snoop disabling is ongoing.
    - If this bit is LOW then snoop disabling is complete.

When the CCI Status Register bit 0 is LOW the master is disconnected from the coherency domain.

Note

Once the CCI Snoop Control Register is set HIGH it must not be set LOW again until the CCI Status Register bit 0 is LOW.

CoreLink CCN

This section details the general sequence for CoreLink CCN-500 series products.

Adding a Master to the Coherent Domain

Before a master issues any shareable transactions:

1. Set HIGH the HNF SDCR_Set register bit which corresponds to the master node ID.

2. Poll until HIGH the HNF SDCR register bit which corresponds to the master node ID.
3. Repeat actions 1-3 for all HNFs in the CCN.

Once completed for all HNFs the master is now fully connected in the coherency domain.

Removing a Master from the Coherent Domain

Once you have ensured that the master has stopped issuing any shareable transactions and its cache is flushed:

1. Set HIGH the HNF SDCR_Clear register bit which corresponds to the master node ID.
2. Poll until LOW the HNF SDCR register bit which corresponds to the master node ID.
3. Repeat actions 1-3 for all HNFs in the CCN.

Once completed for all HNFs the master is now fully disconnected from the coherency domain.

9.1.5 Example: ARM Cortex-A53 MPCore

This section details the sequences for the ARM Cortex-A53 MPCore as an example.

This core supports the following power management features:

- Per-core power off.
- Per-core dynamic retention (with Q-Channel).
- Per-core Advanced-SIMD/FP dynamic retention (with Q-Channel).
- Shared cluster and L2 cache power off.
- L2 cache RAM dynamic retention (with Q-Channel).
- Hardware L2 cache flush support.

Cortex-A53 Cluster Domain

Figure 9-13 shows the power control connections used for the Cortex-A53 cluster domain, it is not intended to show all Cortex-A53 connections.

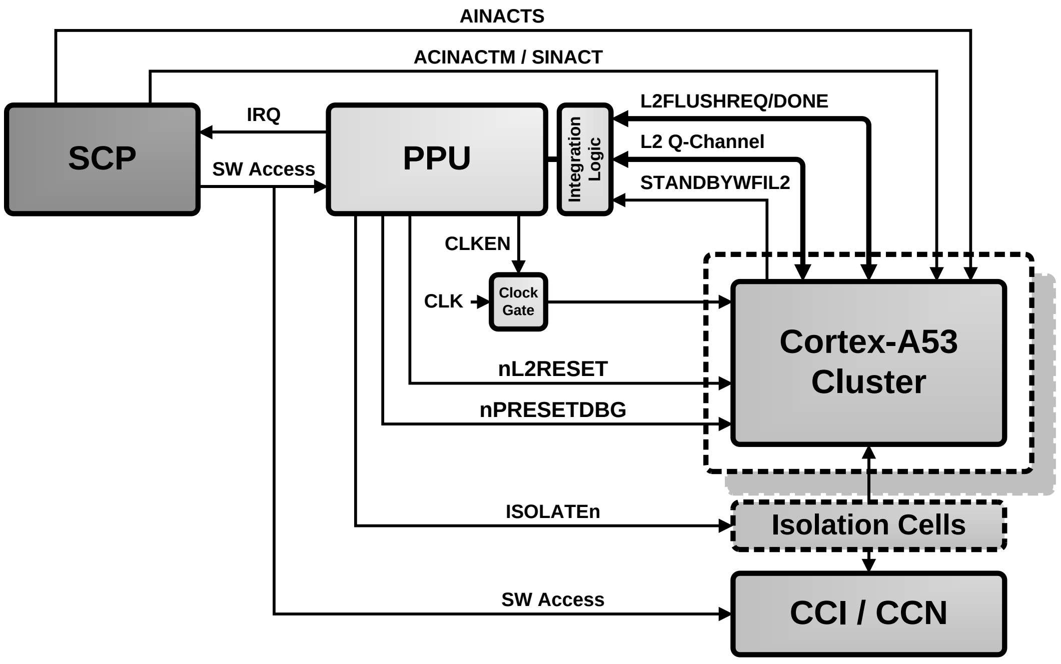


Figure 9-13 – Cortex-A53 cluster power control connections

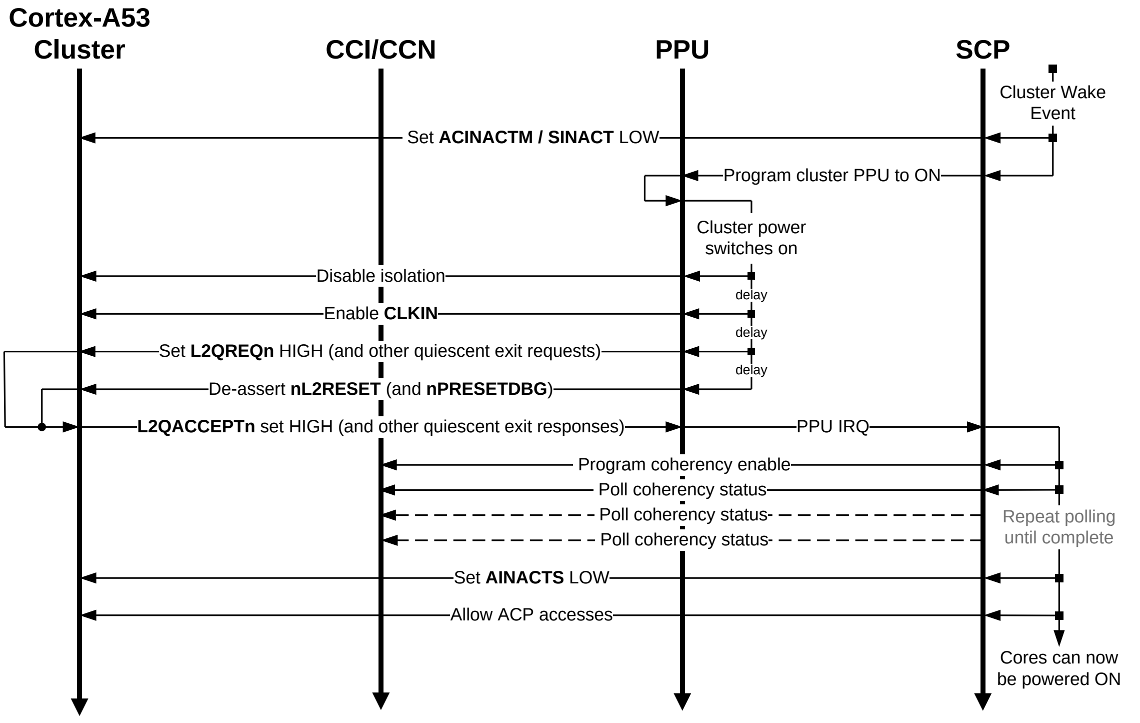
The majority of these interfaces are described *in Cluster Power Control Sequences* on page 9-3. Interfaces specific to Cortex-A53 are detailed below.

The Q-Channel is included to support L2 cache RAM dynamic retention.

If L2 cache RAM dynamic retention is not required, the Q-Channel is not required to be used. In this case the **L2QREQn** input must be tied HIGH and Q-Channel outputs left unconnected.

Cortex-A53 Cluster Power On

Figure 9-14 shows a coherent Cortex-A53 cluster power on sequence with hardware controlled shared cache flushing.



Cortex-A53 Cluster
CCI/CCN
PPU
SCP
Cluster Wake Event
Set ACINACTM / SINACT LOW
Program cluster PPU to ON
Cluster power switches on
Disable isolation
delay
Enable CLKIN
delay
Set L2QREQn HIGH (and other quiescent exit requests)
delay
De-assert nL2RESET (and nPRESETDBG)
L2QACCEPTn set HIGH (and other quiescent exit responses)
PPU IRQ
Program coherency enable
Poll coherency status
Poll coherency status
Poll coherency status
Repeat polling until complete
Set AINACTS LOW
Allow ACP accesses
Cores can now be powered ON

Figure 9-14 – Cortex-A53 cluster power on sequence

Where ‘delay’ is indicated in Figure 9-14, this indicates an implementation specific delay to allow signals to propagate or meet specific requirements on signal timings.

The following describes the sequence:

1. A wake-up event at the SCP indicates that the cluster is required.
2. The SCP sets **ACINACTM** LOW (for an ACE configuration) or **SINACT** LOW (for a CHI configuration).
3. The SCP programs the cluster PPU policy to ON.

The PPU then sequences the following actions:

4. The PPU turns on cluster domain power switches.
5. The PPU disables isolation cells.
6. The PPU enables **CLKIN**.
7. The PPU waits for 3 clock cycles with reset asserted.
8. The PPU sets **L2QREQn** HIGH and asserts any other required quiescent exit requests.
   - This includes controlling domain bridges that provide access to and from the external system, for example:
     - AMBA AXI / ACE / CHI interfaces.
     - Debug APB and CoreSight trace interfaces.
     - Event interfaces.
     - GIC interfaces.
9. The PPU de-asserts **nL2RESET**:
   - **nPRESETDBG** can also be de-asserted if debug access is required..
10. The PPU waits for **L2QACCEPTn** to go HIGH and all other required quiescent exit responses.
11. The PPU sends an interrupt to the SCP to indicate the process is finished.
12. The SCP then enables coherency for the cluster:

- This procedure is described in *Enabling and Disabling System Coherency* on page 9-20.

13. The SCP sets **AINACTS** LOW.
14. The SCP allows accesses to the ACP port.

Cortex-A53 Cluster Power Off

Figure 9-15 shows a coherent Cortex-A53 cluster power on sequence with hardware controlled shared cache flushing.

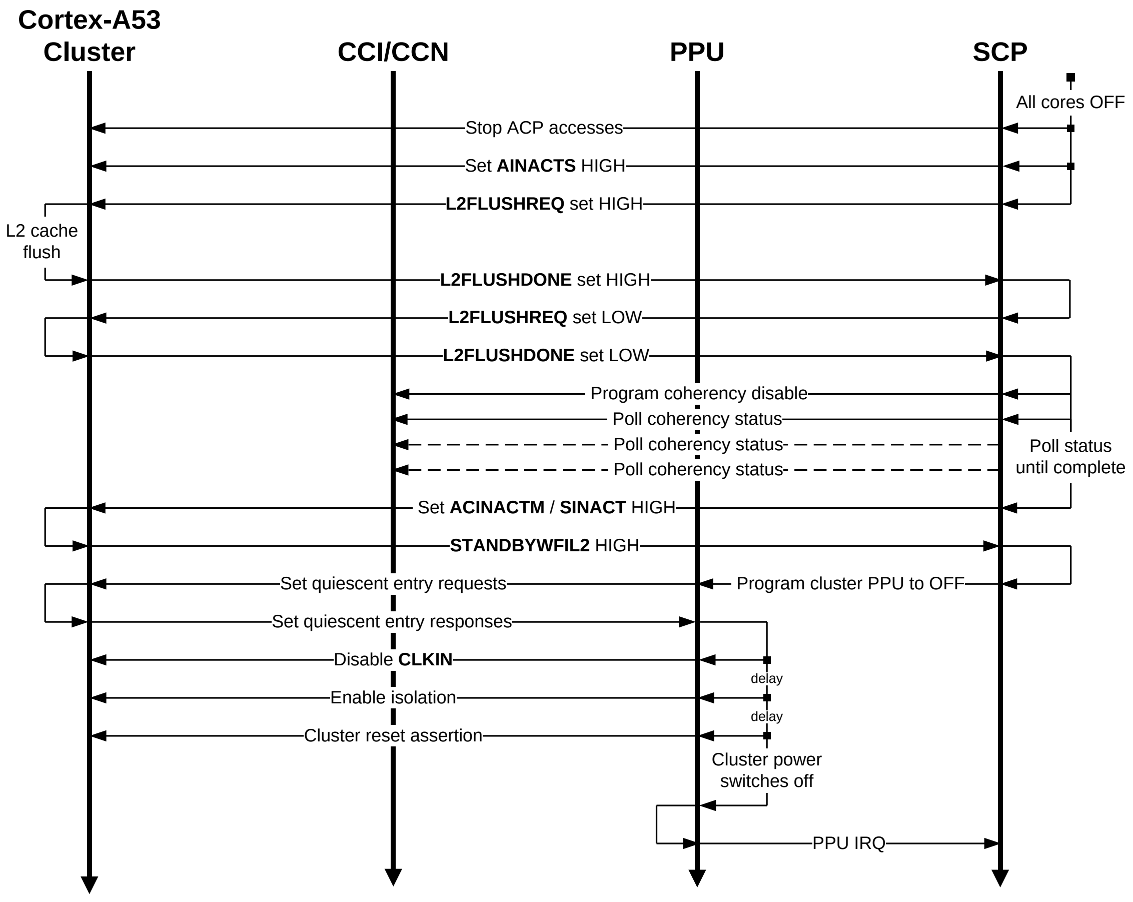


Figure 9-15 – Cortex-A53 cluster power off sequence

Where ‘delay’ is indicated in Figure 9-15, this indicates an implementation specific delay to allow signals to propagate or meet specific requirements on signal timings.

The following describes the sequence. Before the sequence begins all cores must be powered off:

1. The SCP ensures the ACP port will not be accessed then, and sets **AINACTS** HIGH.
2. The SCP sets **L2FLUSHREQ** HIGH.
3. The SCP waits for **L2FLUSHDONE** to go HIGH, indicated as an interrupt.
4. The SCP sets **L2FLUSHREQ** LOW.
5. The SCP waits for **L2FLUSHDONE** to go LOW, indicated as an interrupt.
6. The SCP disconnects the cluster from system coherency
    - This procedure is described in *Enabling and Disabling System Coherency* on page 9-20.
7. The SCP sets **ACINACTM** or **SINACT** HIGH.
8. The SCP waits for **STANDBYWFIL2** to be set HIGH, indicated as an interrupt.
9. The SCP programs the cluster PPU to OFF.

The PPU then sequences the following actions:

10. The PPU makes any required Q-Channel quiescence requests.
    - This does not include the cluster Q-Channel used for dynamic retention.
    - This includes controlling domain bridges that provide access to and from the external system, for example:
        - AMBA AXI / ACE / CHI interfaces.
        - Debug APB and CoreSight trace interfaces.
        - Event interfaces.
        - GIC interfaces.
11. The PPU waits until all quiescence request responses have been received.
12. The PPU disables **CLKIN**.
13. The PPU enables cluster power domain isolation cells.
14. The PPU asserts **nL2RESET** and **nPRESETDBG**.
15. The PPU turns off cluster domain power switches.
16. The PPU, through the integration logic, resets the cluster Q-Channel to the *Q_STOPPED* state.
17. The PPU sends an interrupt to the SCP.

Cortex-A53 Core Domain

Figure 9-16 shows the connections used in the Cortex-A53 core power control sequences.

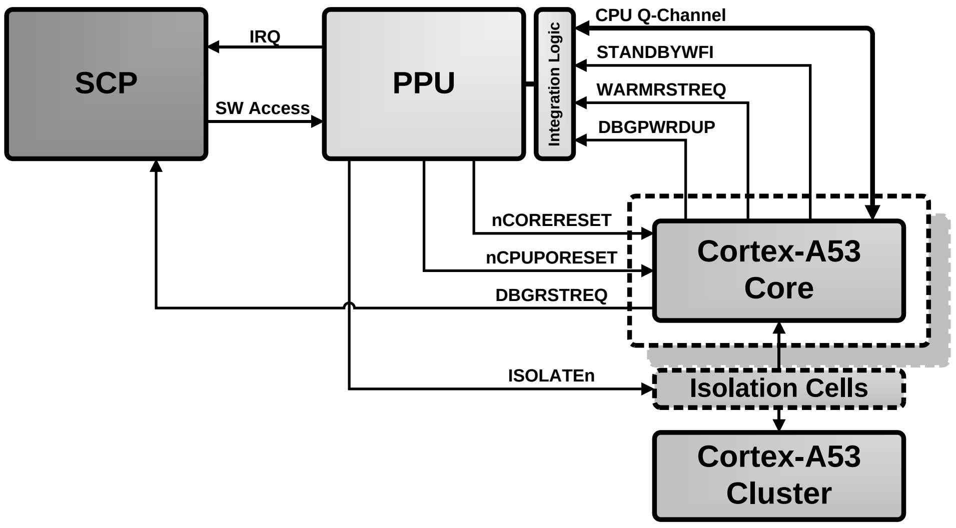


Figure 9-16 – Cortex-A53 core power control connections

The cluster and required system domains must be powered on before the core can be powered on.

The Q-Channel is included to support core dynamic retention. It can also be used, optionally, in core power on and power off control sequences. This is recommended when core dynamic retention is implemented.

If core or Advanced-SIMD/FP dynamic retention is not required, the associated Q-Channel is not required to be used. If unused the associated **QREQn** input must be tied HIGH and Q-Channel outputs left unconnected. The related steps in the sequences can be removed.

———**Note**————————

The Q-Channel for Advanced-SIMD/FP dynamic retention is omitted from Figure 9-16. However, an overview of Cortex-A53 Advanced-SIMD/FP dynamic retention is given within this section.

————————————

Cortex-A53 Core Reset

The Cortex-A53 core is warm reset with **nCORERESET**. This section describes the methods for performing a warm reset.

Cortex-A53 Warm Reset Request

The warm reset request sequence is:

1. Software sets RMR_EL3.RR HIGH to request the warm reset.
    - The setting of RMR_EL3.AA64 determines the AArch execution state entered, at the highest exception level implemented, after the warm reset.
2. Software executes an ISB instruction.
3. Software executes a WFI instruction.
4. The PPU samples **WARMRSTREQ** and **STANDBYWFI** HIGH.
5. The PPU asserts **nCORERESET** LOW**.**
6. The assertion of **nCORERESET** sets **WARMRSTREQ** LOW.
7. Wait for 3 clock cycles with reset asserted.
8. The PPU de-asserts **nCORERESET** HIGH.

Cortex-A53 Debug Warm Reset Request

The debug warm reset request is described in *Debug Warm Reset Request* on page 9-13. It is recommended to use the **DBGRSTREQ** signal as an interrupt to the SCP. The SCP can then reset the required parts of the system.

When the EDPRCR.CWRR bit is set HIGH, **DBGRSTREQ** is driven HIGH indicating to the SCP that action is required.

Cortex-A53 Core Power On

Figure 9-17 shows a Cortex-A53 core power on sequence.

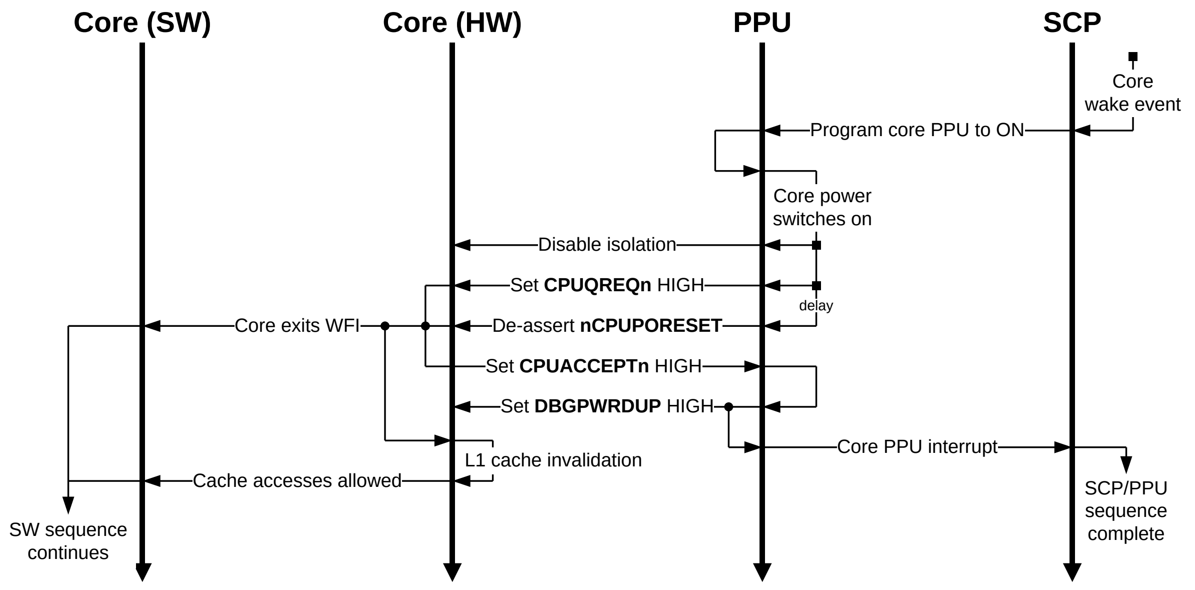


Figure 9-17 – Cortex-A53 core power on sequence

Where 'delay' is indicated in Figure 9-17, this indicates an implementation specific delay to allow signals to propagate or meet specific requirements on signal timings.

The following describes the sequence:

1. A wake event at the SCP indicates the core is required.
2. The SCP programs the core PPU policy to ON.

The PPU then sequences the following actions:

3. The PPU turns on the core domain switches.
4. The PPU disables isolation cells.
5. The PPU sets **CPUQREQn** HIGH.
6. The PPU waits for at least 3 clock cycles with reset asserted.
7. The PPU de-asserts **nCPUPORESET** HIGH.
    - It also de-asserts **nCORERESET** HIGH if this was asserted
8. The PPU waits for **CPUQACCEPTn** to go HIGH.
9. The PPU, through integration logic, sets **DBGPWRDUP** HIGH.
10. The PPU sends an interrupt to the SCP to indicate the process is complete.

Software runs once the resets have been de-asserted.

Once this hardware sequence is complete there are several functions which need to be carried out by software running on the core. The following sequence describes these actions:

1. Software enables data coherency by setting CPUECTLR.SMPEN HIGH.
2. Software enables data caching by setting SCTLR.C.
3. Software restores core context, GIC CPU interface state, and core private timer state.
4. Software enables interrupts in the GIC.
    - o For this procedure see *Generic Interrupt Controller* on page 9-30.

Cortex-A53 Core Power Off

The sequence required to be completed by software running on the core before power off are:

1. Software disables interrupts by setting the PSTATE.DAIF bits.
2. Software communicates with the SCP to request power off at the next WFI entry.
3. Software saves core context, GIC CPU interface state, and core private timer state.
4. Software disables the GIC CPU interface.
    - Interrupts are re-directed to the SCP as wake requests.
    - For this procedure see *Generic Interrupt Controller* on page 9-30.
5. Software disables data caching by setting SCTLR.C LOW.
6. Software flushes the L1 data cache.
7. Software disables data coherency by setting CPUECTLR.SMPEN LOW.
8. Software executes an ISB and a DSB SY instruction.
9. Software executes a WFI instruction.

——— **Note** ————————

Setting DBGOSDLR.DLK in AArch32 or OSDLR.DLK in AArch64, as described in *Core Power Off* on page 9-18, is not required but can still be performed.

————————————————

Figure 9-18 shows the hardware sequencing relative to the software sequence above.

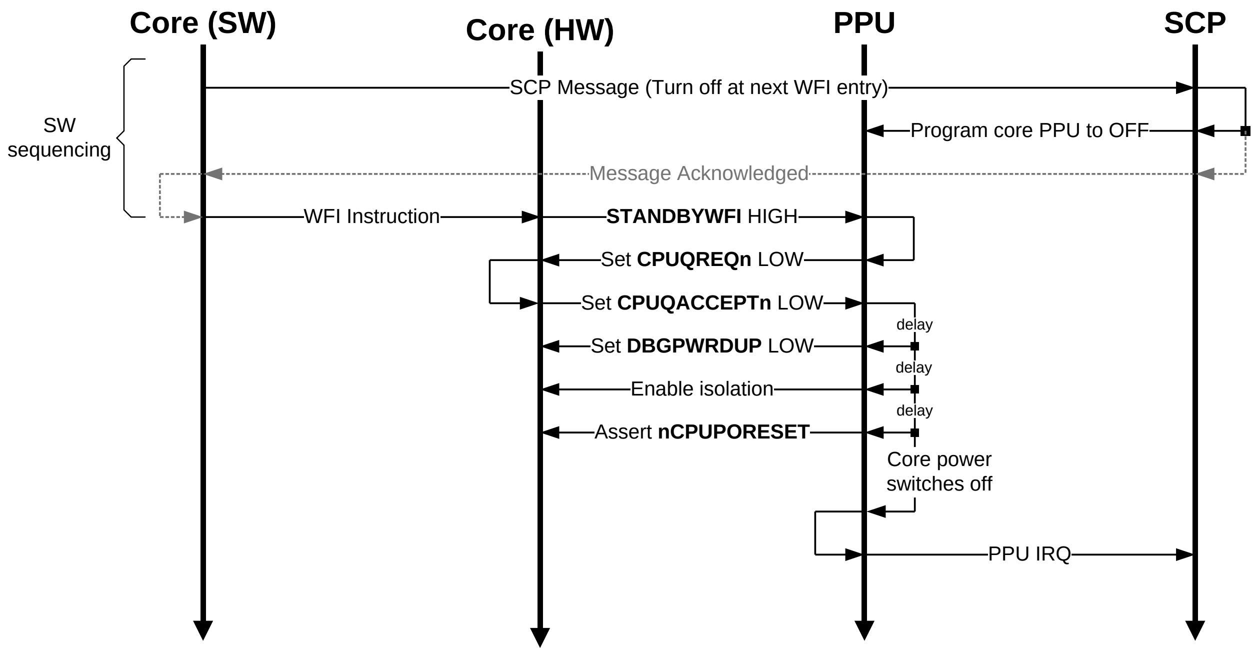


Figure 9-18 – Cortex-A53 core power off sequence

Where 'delay' is indicated in Figure 9-18, this indicates an implementation specific delay to allow signals to propagate or meet specific requirements on signal timings.

The following details the sequence:

1. Software communicates with the SCP to request power off at the next WFI entry.
2. The SCP programs the core PPU policy to OFF and, optionally, acknowledges the message to software.
3. The PPU waits, through integration logic, for **STANDBYWFI** to go HIGH.
    - **CPUQACTIVE** will be set LOW if the core dynamic retention is enabled and the retention entry timeout has expired.
4. The PPU sets **CPUQREQn** LOW.
    - Core dynamic retention must be enabled and any timeout expired for this handshake to be accepted.
5. The PPU waits for **CPUQACCEPTn** to go LOW.
6. The PPU, through integration logic, sets **DBGPWRDUP** LOW.
7. The PPU enables core power domain isolation cells.
8. The PPU asserts **nCPUPORESETn** LOW.
    - **nCORERESET** can also be asserted LOW but this is not required.
9. The PPU turns off core power domain switches.
10. The PPU sends an interrupt to the SCP to indicate the process is complete.

Core Dynamic Retention

Cortex-A53 core dynamic retention control sequences are identical to that given in

Core Dynamic Retention on page 9-16.

Advanced-SIMD / FP Dynamic Retention

Cortex-A53 Advanced-SIMD/FP dynamic retention is managed similarly to core dynamic retention with an additional dedicated Q-channel.

The Q-Channel sequences are identical to those for core dynamic retention, the main differences are in the entry and exit conditions used.

The entry condition for Advanced-SIMD/FP dynamic retention is the Advanced-SIMD/FP pipeline being idle. This is followed by timeout expiration as defined in CPUECTLR.FPRETCTL.

The opportunity to enter Advanced-SIMD/FP dynamic retention is indicated by **NEONQACTIVE** being LOW

Advanced-SIMD / FP dynamic retention is exited either when an Advanced-SIMD/FP instruction is detected in the early stages of the pipeline or if the power controller unilaterally sets **NEONQREQn** HIGH.

When enabled, Advanced-SIMD/FP dynamic retention must be entered before core dynamic retention is entered.

If not implemented **NEONQREQn** must be tied HIGH and the retention feature must be disabled by setting CPUECTLR.FPRETCTL to zero.

9.2 Generic Interrupt Controller

The generic interrupt controller (GIC) takes interrupts from many sources and distributes them to AP cores.

The GIC must be powered on with a relatively always on relationship to all AP cores. However, it is not required to be always on, and can be powered down in SoC SLEEP states. The GIC power domain hierarchy requirements are also defined in *Power Domain Hierarchy Requirements* on Page 5-13.

This section describes the following:

- Using the GIC to wake AP cores.
- Managing the GIC CPU Interface when powering a core off and on.
- Managing interrupts when the GIC is powered off.

These cases are described for the v2 and v3 versions of the GIC architecture as the behavior is different.

AP core wake-up semantics for the following methods are defined in *Power Domain Hierarchy Requirements* on Page 5-13:

- Interrupts from the GIC.
- Always on domain wake-up events.

However, from a power control integration perspective it is important to note that some signals that are interrupt inputs to the GIC might also be wired as always on domain wake-events.

The procedures given in this section related to core power control provide supporting detail for the GIC enabling and disabling requirements given in *Core Power Control Sequences* on page 9-12.

9.2.1 GIC v2

The ARM GICv2 Architecture is described in the *ARM Generic Interrupt Controller Architecture Version 2.0 Specification.*

GICv2 uses wired interrupts in the form of **nFIQ** and **nIRQ** outputs to send interrupts to AP cores. These are connected directly to inputs on the processor clusters.

The principles described here are the same for any GIC v2 compliant GIC, but product documentation should always be consulted.

Using GICv2 to Wake Cores

In addition to the normal interrupt outputs, the GIC architecture defines wake request outputs. These are connected to the SCP to indicate the associated core needs to be woken to process an interrupt. Once the core is powered the interrupt can be sent to the core.

There are two wake request outputs for each core. The **FIQOUT** and **IRQOUT** wake request signals for a specific core can be combined to produce a single wake signal to the SCP. However, the wake-up signals for multiple cores must not be combined. A separate wake requirement for each core must be separately visible to the SCP so it can determine which core to power on.

The GIC wake request outputs are never masked in the GIC. Therefore, the SCP should mask them when the related core is on to prevent it responding when a power on is not required. The SCP should un-mask the interrupt to provide a wake request only when the core is powered off.

The following procedures are required for a GICv2 implementation to manage the forwarding of interrupts when powering on and off cores.

Before powering off a core:

1. Ensure interrupt bypass is disabled in GICC_CTLR.
2. Clear the group enables in GICC_CTLR.

After powering on a core:

1. Set the required group enables in GICC_CTLR.

Waking the System when GICv2 is off

In a SoC SLEEP state the GIC can be powered off but the system can wake in response to always on domain events.

When the GIC is powered on, in response to an always on domain wake event, any signal which is also wired as GIC interrupt source input will be forwarded to the GIC.

Signals which are wired as both always on domain events and GIC interrupt source inputs might also be masked in the SCP during the system RUN state to avoid unnecessary SCP interrupts.

——— **Note** ———————

Always on domain events that are also wired as GIC interrupts must be level sensitive, or regenerated accordingly, to ensure the forwarding to the GIC after it is powered on.

———————————————

ARM CoreLink GIC-400

The CoreLink GIC-400 is an ARM implementation of the GICv2 Architecture. The details of this are described in the *ARM CoreLink GIC-400 Generic Interrupt Controller Technical Reference Manual*.

Figure 9-19 shows power management integration for GIC-400.

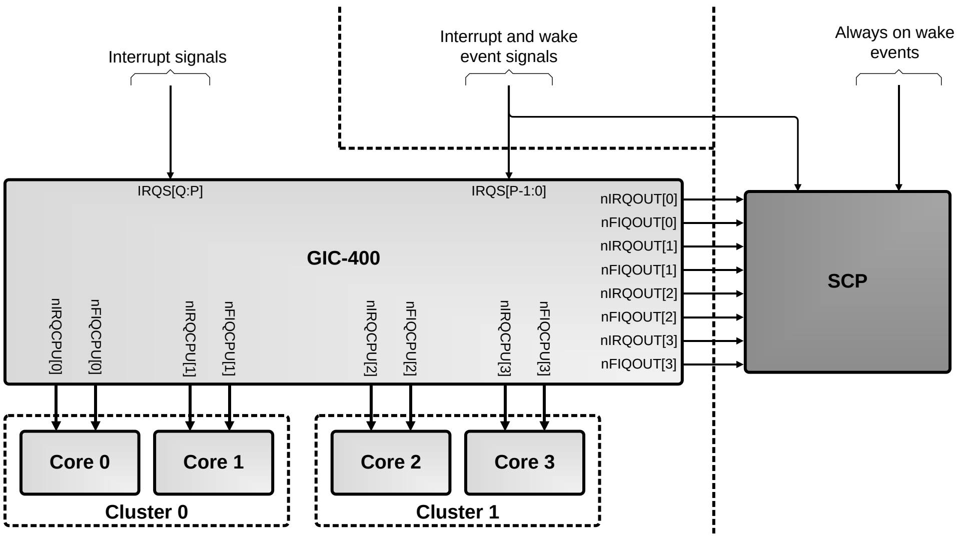


Figure 9-19 – GIC-400 power management integration

9.2.2 GIC v3

The ARM GICv3 Architecture is described in the *ARM v3 Generic Interrupt Controller Architecture Specification*.

The following flows assume the use of the GIC Stream Protocol Interface to communicate between the GIC redistributor and the GIC CPU Interface as described in the *ARM v3 Generic Interrupt Controller Architecture Specification*.

Figure 9-20 shows an example of GICv3 power management integration:

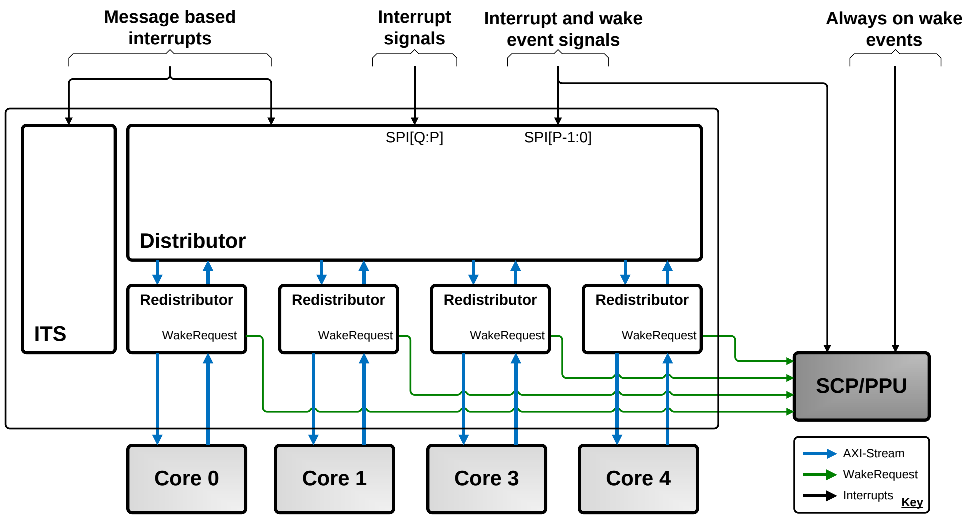


Figure 9-20 – GICv3 power management integration

When a core powers on it must connect the GIC CPU interface to the GIC redistributor, and before powering off it must disconnect this interface. This is managed with the GICR_WAKER.ProcessorSleep and GICR_WAKER.ChildrenAsleep memory mapped register bits in the connected redistributor.

When the redistributor to GIC CPU interface is disconnected and there is an interrupt pending for that core the redistributor indicates the core wake requirement using the **WakeRequest** signal.

Additionally, where present, when an Interrupt Translation Service (ITS) is powered on it must be enabled, and before being powered off it must be disabled. This is managed with the GITS_CTLR.Enable and GITS_CTLR.Quiescent bits in the ITS.

The following sections describe these flows.

Redistributor to GIC CPU Interface Connection

After powering on a core the GIC CPU interface must connect to its GIC redistributor.

Figure 9-21 shows the connection flow.

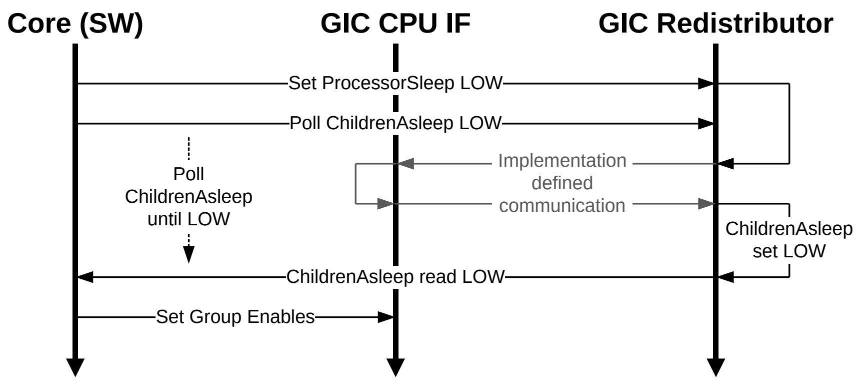


Figure 9-21 – GIC Redistributor to GIC CPU Interface connection flow

The following details the flow:

1. Software on the core sets GICR_WAKER.ProcessorSleep LOW for its associated redistributor.
2. The Redistributor completes any required communication with the GIC CPU Interface before setting GICR_WAKER.ChildrenAsleep LOW.
   - This communication is IMPLEMENTATION DEFINED but could include the GICD_CTLR.DS settings.
3. Software polls the GICR_WAKER.ChildrenAsleep for its associated redistributor until it is LOW.

4. Software sets the physical group enables in the GIC CPU Interface.

When GICR_WAKER.ProcessorSleep and GICR_WAKER.ChildrenAsleep are LOW the redistributor can forward enabled interrupts to the GIC CPU Interface.

Redistributor to GIC CPU Interface Disconnection

Before powering off a core the GIC CPU interface must disconnect from its GIC redistributor.

Figure 9-22 shows the disconnection flow.

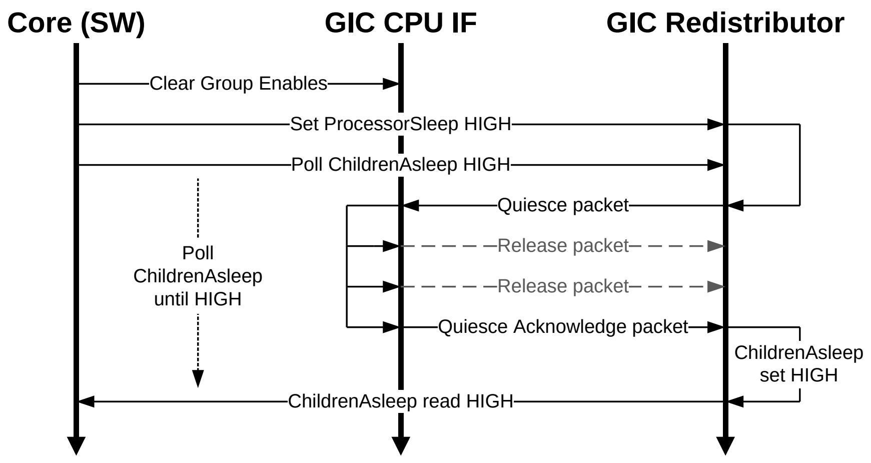


Figure 9-22 – GIC Redistributor to GIC CPU Interface disconnection flow

The following details the flow:

1. Software clears the physical group enables in the GIC CPU interface.
2. Software on the core sets GICR_WAKER.ProcessorSleep HIGH for its associated redistributor.
    - The Redistributor is no longer allowed to send interrupts to the GIC CPU Interface.
3. The Redistributor sends a Quiesce packet to the GIC CPU Interface.
4. The GIC CPU Interface then releases all pending interrupts.
5. Once all interrupts are released the GIC CPU Interface sends a Quiesce Acknowledge packet to the Redistributor.
6. The Redistributor sets GICR_WAKER.ChildrenAsleep HIGH.
7. Software polls the GICR_WAKER.ChildrenAsleep for its associated redistributor until it is HIGH.

——— **Note** ———————

To guarantee that no interrupts can be forwarded to the core the sequence above also requires that interrupt bypass is disabled in the GIC CPU interface.

————————————

Using WakeRequest to Wake Cores

To wake a core a GICv3 implementation provides a WakeRequest output signal from each Redistributor for the associated core.

When GICR_WAKER.ProcessorSleep is set HIGH the WakeRequest output will be asserted when an interrupt is received that is specifically targeted to that core. The WakeRequest signal is connected to the SCP, or PPU, as a wake event to power on the core.

When the core has powered on the redistributor to GIC CPU interface connection sequence described in *Redistributor to GIC CPU Interface Connection* on page 9-32 must be performed before interrupts are forwarded.

Waking the System when GICv3 is off

When the GIC is completely powered off the sequence for powering on is the same as for GICv2. This is described in *Waking the System when GICv2 is off* on page 9-31.

Enabling the ITS

Before the ITS can be used it must be enabled. It could have been disabled due an initial power on or reset procedure, or by being disabled by SW.

Figure 9-23 shows the enable flow. As a pre-cursor to this flow any memory structures required to support the device need to be initialized or restored.

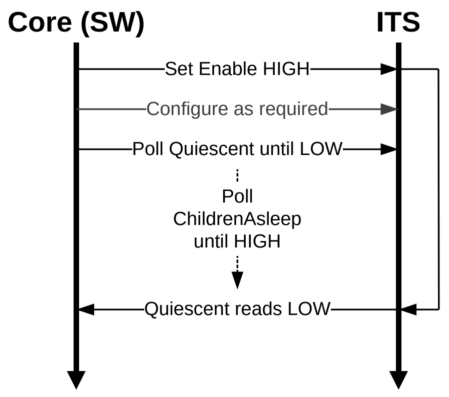


Figure 9-23 – ITS enable flow

The following details the flow:

1. Software sets GITS_CTLR.Enable HIGH.
2. Software configures the ITS as required using appropriate ITS commands.
    - For these commands see the *ARM v3 Generic Interrupt Controller Architecture Specification.*
3. Software polls until GITS_CTLR.Quiescent is LOW.

Disabling the ITS

Before the ITS can be reset or powered off it must be disabled.

Figure 9-24 shows the disable flow. As a pre-cursor to this flow any interrupts which can target the ITS must be either powered off, redirected, or disabled.

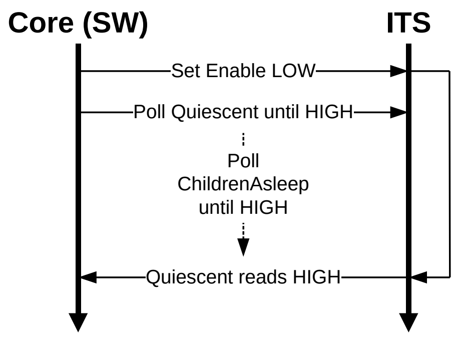


Figure 9-24 – ITS disable flow

The following details the flow:

1. Software sets GITS_CTLR.Enable LOW.
2. Software polls until GITS_CTLR.Quiescent is LOW.

ARM CoreLink GIC-500

The CoreLink GIC-500 is an ARM implementation of the GICv3 architecture. It contains the distributor and all core redistributors. It also, optionally, implements an ITS.

Full details of the GIC-500 can be found in the *ARM® CoreLink™ GIC-500 Generic Interrupt Controller Technical Reference Manual*.

Figure 9-25 shows the structure of GIC-500.

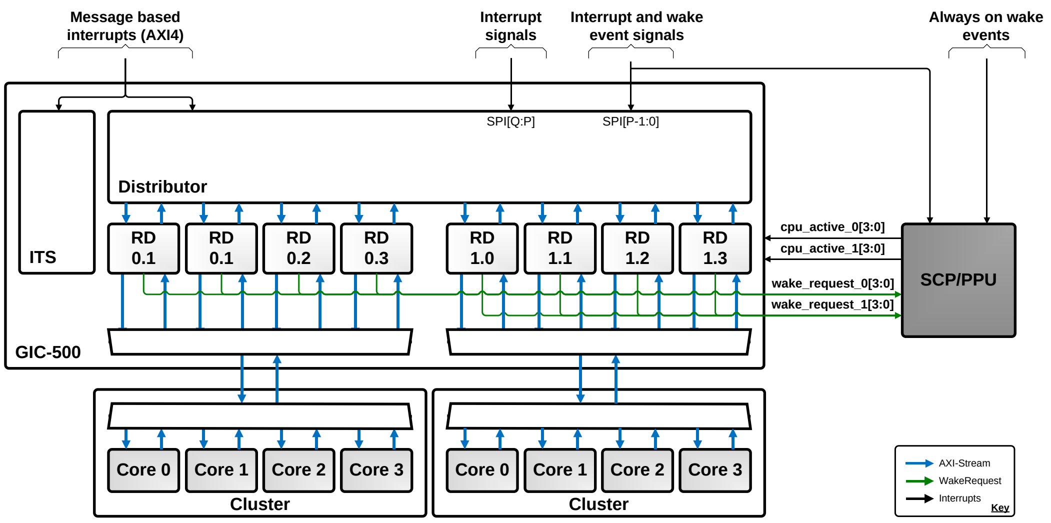


Figure 9-25 – GIC-500 power management integration

The **wake_request** outputs, which are the architected WakeRequests, are provided per core and grouped as a bus per cluster. For example, **wake_request_0[3:0]** are the WakeRequest outputs for Cluster 0. The **wake_request_0[0]** output is the WakeRequest for core 0 in cluster 0, and the **wake_request_0[1]** output is the WakeRequest for core 1 in cluster 0.

The connection and disconnection of the Redistributor interface to the cores, and the enabling and disabling of the ITS is managed as described in *GIC v3* on page 9-31.

cpu_active

The **cpu_active** inputs are a hint, when LOW, that a core is in a software transparent low power mode, such as retention. There is one **cpu_active** input to the GIC-500 per core. It makes this core less likely to be chosen for a 1 of N interrupt over other cores with **cpu_active** set HIGH. It does not guarantee that interrupts will not be sent to that core. Any interrupt specifically targeted to that core will be forwarded regardless of the status of **cpu_active**.

The **cpu_active** input for a core does not affect the behavior when that cores GICR_WAKER.ProcessorSleep bit is set HIGH. When an interrupt is targeted to a core where the GICR_WAKER.ProcessorSleep bit is set HIGH the **wake_request** output will be set HIGH regardless of the state of the **cpu_active** input.

If the **cpu_active** input for a core is set HIGH then that core will be selected as normal for any interrupts targeting multiple cores.

9.3 System Memory Management Unit

A system memory management performs virtual to physical address translation for system masters without an internal capability.

For more details on MMU functionality see the *ARM® System Memory Management Unit Architecture Specification Version 2.0.*

9.3.1 CoreLink MMU-500

The CoreLink MMU-500 consists of a Translation Control Unit (TCU), which controls and manages address translations, and one or more Translation Buffer Units (TBU). One TBU is required for each master port requiring address translation. The TBU and TCU communicate over an AXI Stream interface.

For full details of the CoreLink MMU-500 functionality see the ARM® *CoreLink™ MMU-500 System Memory Management Unit Technical Reference Manual.*

The TBU is designed to be implemented locally to the master port requiring translation. It can be configured to be in either a separate clock domain, power domain, or both. If the TBU is configured as being in a separate clock or power domain an asynchronous bridge is included between the TCU and TBU.

Figure 9-26 shows an example configuration and the main connections between the components, not including the power and clock control interfaces.

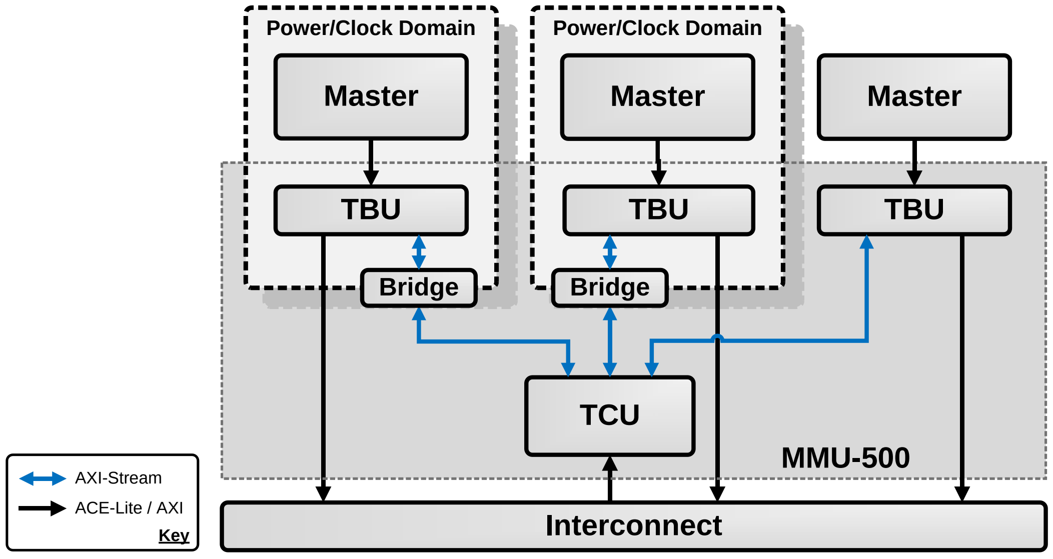


Figure 9-26 – Example MMU-500 TBU clock/power domain configuration

MMU-500 Clock Control

There is a separate clock control Q-Channel for each TBU and the TCU.

MMU-500 does not implement synchronization on its Q-Channel **QREQn** inputs, so to allow the clock to be controlled at a high level these synchronization registers need to be added. The synchronization must be added using the gated clock to ensure no duplication of the clock tree. This is allowed as the clock control **QREQn** signals will only change when the clock is guaranteed to be available.

Figure 9-27 shows the signal connections for clock control and where the synchronization registers are required. The <x> in the signal names are substituted with the TBU name given at design time.

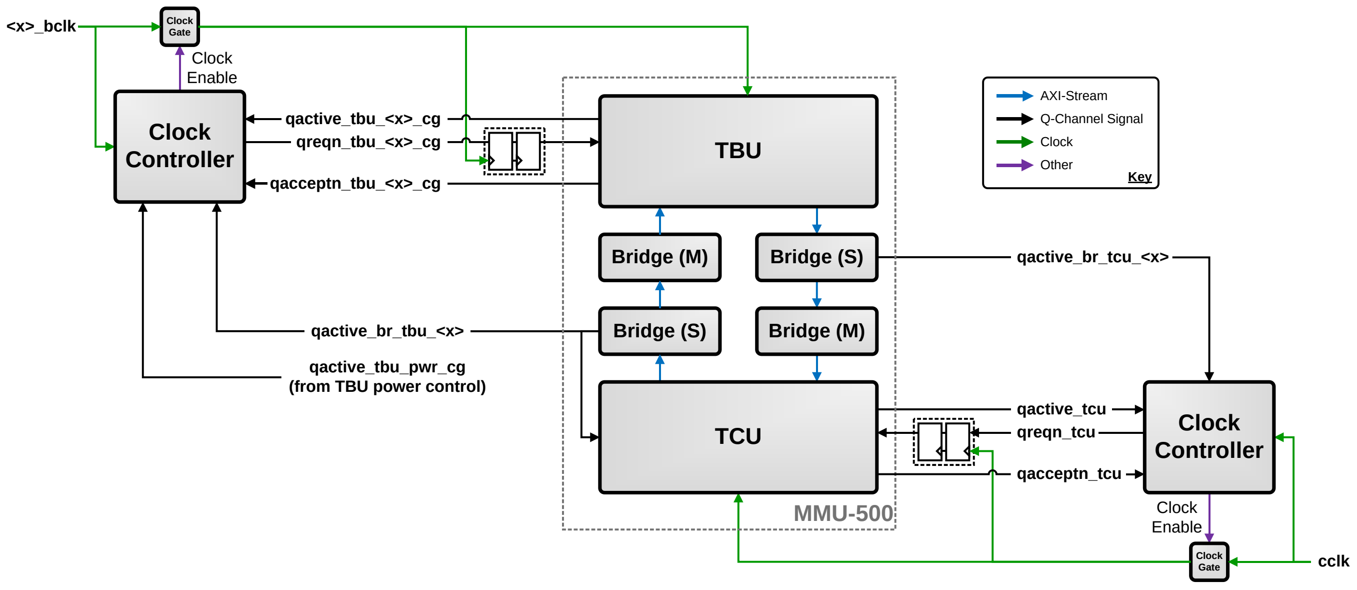


Figure 9-27 – MMU-500 Clock Control Connections

In the case where there is more than one TBU in a separate clock domain, each TBU will have its own asynchronous bridge. Therefore, all the **qactive_br_tcu_<x>** signals from all bridges need to be connected to the TCU **cclk** clock controller.

In the case when a TCU and one or more TBUs are in the same clock domain, the clock control Q-Channels from all components in the domain must be connected to the common clock controller. This is so they are used together to control the common clock. When a TBU is configured to be in the same clock and power domain there will be no asynchronous bridge component.

There is an additional **QACTIVE** term for the TBU clock domain in the diagram called **qactive_tbu_pwr_cg**. This is required if the power control Q-Channels on the TBU and associated asynchronous bridges are used. This ensures the clock is requested when any power control Q-Channel is in transition, for more details see *MMU-500 TBU Power Control* on page 9-37.

MMU-500 TBU Power Control

When in separate power domains the TCU must be ON before any TBU is ON and the TCU can only power off when all TBUs are OFF.

Each TBU has a power control Q-Channel. Each bridge component has a power control Q-Channel, one for each direction of the AXI-Stream traffic. These Q-Channels ensure the TBU is quiescent before powering OFF and, using messages on the AXI stream interface, that the TCU will no longer forward any messages to the TBU while it is OFF.

MMU-500 does not implement synchronization on its Q-Channel **QREQn** inputs. To allow the power to be controlled from a different domain these synchronization registers need to be added external to the MMU, and must be added using the gated clock to ensure no duplication of the clock tree.

However, this means the normal mechanism whereby a power control Q-Channel transition would request the TBU clock through **qactive_tbu_<x>_cg** does not function until the **QREQn** change has propagated past the synchronization registers, which cannot happen until the clock is enabled.

Therefore a new **QACTIVE** term, **qactive_tbu_pwr_cg**, must be created. It is HIGH when any power control channel is in transition. Each power control **QREQn** input, before the synchronization registers, is XOR'd with its associated **QACCEPTn** output. These are then OR'ed together. This term is used as a **QACTIVE** input to the TBU clock controller.

Figure 9-28 shows the connections between the TBU power control Q-Channels and the PPU. A component integration layer (CIL) is used in this case to manage a specific sequence requirement.

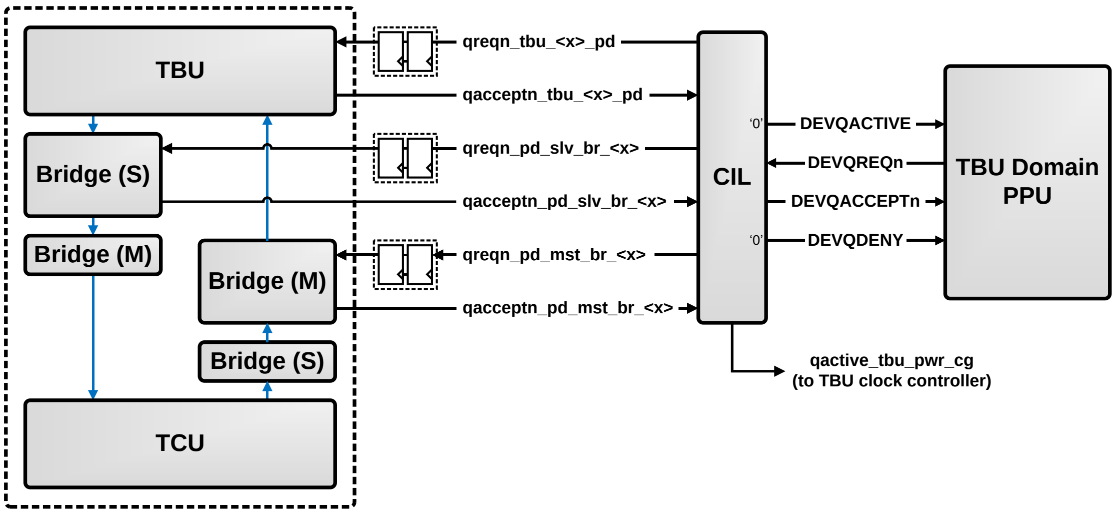


Figure 9-28 – MMU-500 TBU power control connections

The system must ensure that there will be no accesses made on the TBU ACE-Lite slave port before the power off procedure begins.

The TBU is usually combined with other components in the power domain, such as the master attached to the TBU ACE-Lite slave port interface as shown in Figure 9-26. Any power control on these components must also be managed according to their specifications before the domain isolation, power switch, or shared reset is managed.

There are no **QACTIVE** or **QDENY** signals on the TBU power control Q-Channels so these signals must be tied LOW at the PPU.

TBU Power Off Flow

Figure 9-29 shows the power off control flow for the TBU.

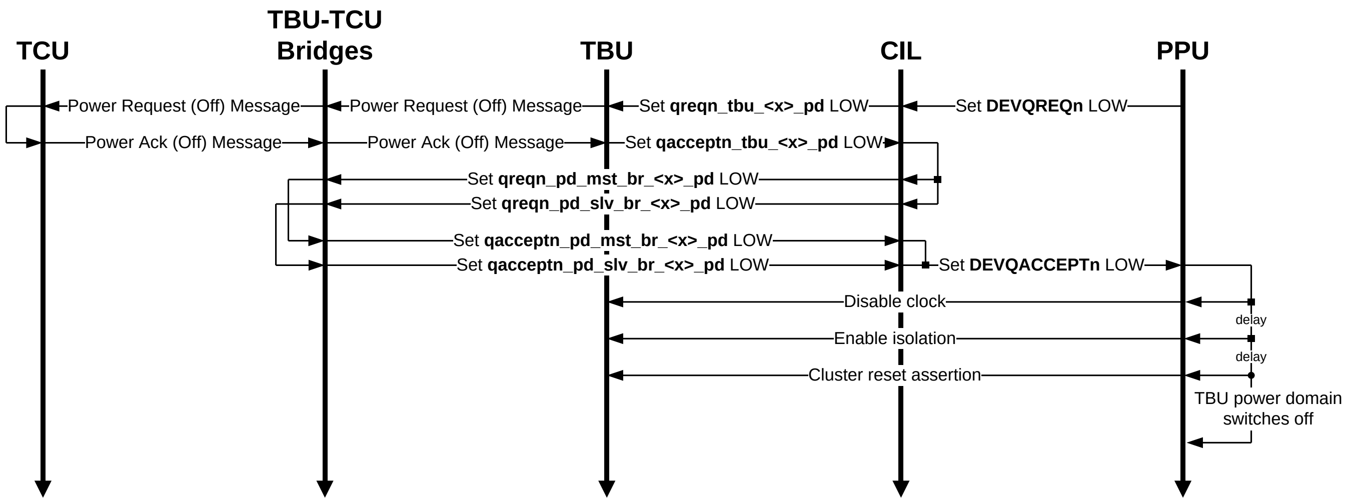

TCU
TBU-TCU Bridges
TBU
CIL
PPU
Power Request (Off) Message
Power Request (Off) Message
Set qreqn_tbu_<x>_pd LOW
Set DEVQREQn LOW
Power Ack (Off) Message
Power Ack (Off) Message
Set qacceptn_tbu_<x>_pd LOW
Set qreqn_pd_mst_br_<x>_pd LOW
Set qreqn_pd_slv_br_<x>_pd LOW
Set qacceptn_pd_mst_br_<x>_pd LOW
Set qacceptn_pd_slv_br_<x>_pd LOW
Set DEVQACCEPTn LOW
Disable clock
delay
Enable isolation
delay
Cluster reset assertion
TBU power domain switches off

Figure 9-29 – MMU-500 TBU power off flow

Where 'delay' is indicated in Figure 9-29, this indicates an implementation specific delay to allow signals to propagate or meet specific requirements on signal timings.

The following details the flow:

1. When programmed to turn off the TBU the PPU sets **DEVQREQn** LOW.
2. The CIL sets **qreqn_tbu_<x>_pd** LOW.
3. The TBU sends a power off request message to the TCU on the AXI stream interface.
    - o This will pass through the bridge between the two components.
4. The TCU sends a power off acknowledge message back to the TBU.

- This will pass through the bridge between the two components.

5. The TBU sets **qacceptn_tbu_<x>_pd** LOW.
6. The CIL sets **qreqn_pd_slv_br_<x>** and **qreqn_pd_mst_br_<x>** LOW.
    - These two channels can be controlled simultaneously.
7. The Bridge sets **qacceptn_pd_slv_br_<x>** and **qacceptn_pd_mst_br_<x>** LOW.
    - These responses can come at different times. Both must be sampled LOW before proceeding.
8. The CIL sets **DEVQACCEPTn** LOW.
9. The PPU then sequences the following with configurable delays between them:
    - Disables the domain clock.
    - Enables the domain isolation.
    - Asserts the TBU **bresetn** input.

TBU Power On Flow

Figure 9-30 shows the power on control flow for the TBU.

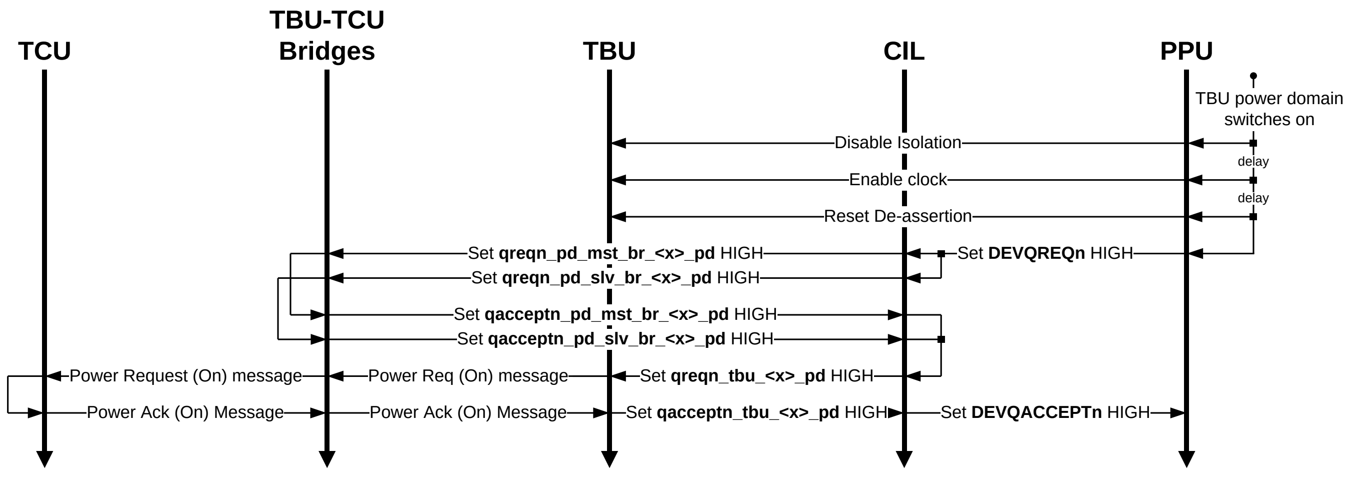

TCU
TBU-TCU Bridges
TBU
CIL
PPU
TBU power domain switches on
Disable Isolation
delay
Enable clock
delay
Reset De-assertion
Set qreqn_pd_mst_br_<x>_pd HIGH
Set DEVQREQn HIGH
Set qreqn_pd_slv_br_<x>_pd HIGH
Set qacceptn_pd_mst_br_<x>_pd HIGH
Set qacceptn_pd_slv_br_<x>_pd HIGH
Power Request (On) message
Power Req (On) message
Set qreqn_tbu_<x>_pd HIGH
Power Ack (On) Message
Power Ack (On) Message
Set qacceptn_tbu_<x>_pd HIGH
Set DEVQACCEPTn HIGH

Figure 9-30 – MMU-500 TBU power on flow

Where ‘delay’ is indicated in Figure 9-30, this indicates an implementation specific delay to allow signals to propagate or meet specific requirements on signal timings.

The following details the flow:

1. When programmed to turn on the TBU the PPU sequences the following with configurable delays between them:
    - Disables the domain isolation.
    - Enables the domain clock.
    - De-asserts the TBU **bresetn** input.
2. The PPU sets **DEVQREQn** HIGH.
3. The CIL sets **qreqn_pd_slv_br_<x>** and **qreqn_pd_mst_br_<x>** HIGH.
    - These two channels can be controlled simultaneously
4. The bridge sets **qacceptn_pd_slv_br_<x>** and **qacceptn_pd_mst_br_<x>** HIGH
    - These responses can come at different times. Both must be sampled HIGH before proceeding.
5. The CIL sets **qreqn_tbu_<x>_pd** HIGH.
6. When the TBU samples **qreqn_tbu_<x>_pd** HIGH it sends a power on request message to the TCU.

7. When the TBU receives a power on acknowledgement message from the TBU it sets **qacceptn_tbu_<x>_pd** HIGH.
8. The CIL sets **DEVQACCEPTn** HIGH.

9.4 Generic Device Power Management

The degree of device power management required varies depending on the device.

Some peripheral devices will be included in system logic power domains containing shared infrastructure logic. These are then powered on whenever the system is running and will require no more than clock enabling to be managed. More complex, typically larger size, devices can require dedicated power domains to be powered on to make them available.

When a power management action is needed the device driver software will typically express this requirement 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.

The remainder of this section describes generic power control flows for a device in a dedicated power domain which can be discretely power managed.

Figure 9-31 shows a high level representation of the entities involved in the device power management flow.

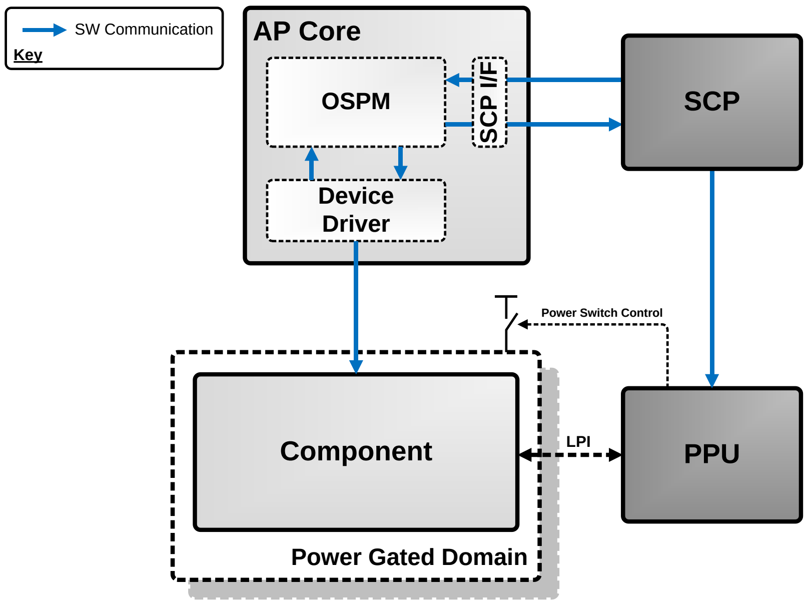


Figure 9-31 – Generic device connections

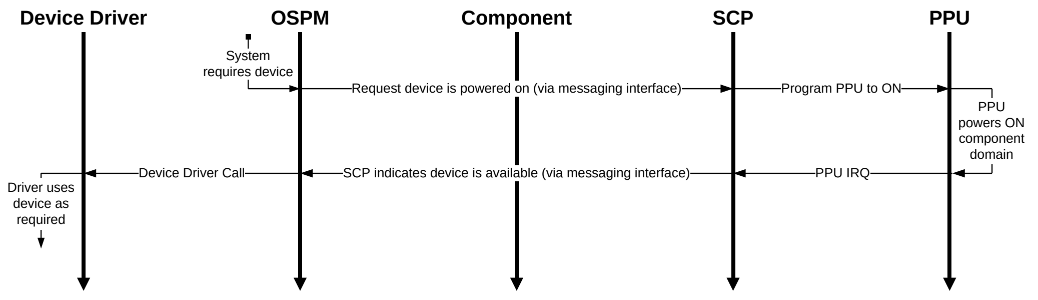
In Figure 9-31 the device hardware is represented by a component in a dedicated power gated domain. A simple example of such a component is a display processor with only external power domain management.

However, the example also extends to components with integrated power management of further power domains, such as ARM Mali GPUs and video processors. Even though there are multiple power domains only one power domain is visible at system level as a device dependency. The device driver then manages all other power domains directly through the job manager and these are invisible to the OSPM and the SCP.

The sequences described in the following sections are software initiated. This does not exclude the component also initiating autonomous power modes, invisible to software, either internally or using a low power interface to the PPU.

9.4.1 Device Power On

Figure 9-32 shows a generic flow for powering on a device.



Device Driver
OSPM
Component
SCP
PPU
System requires device
Request device is powered on (via messaging interface)
Program PPU to ON
PPU powers ON component domain
Device Driver Call
SCP indicates device is available (via messaging interface)
PPU IRQ
Driver uses device as required

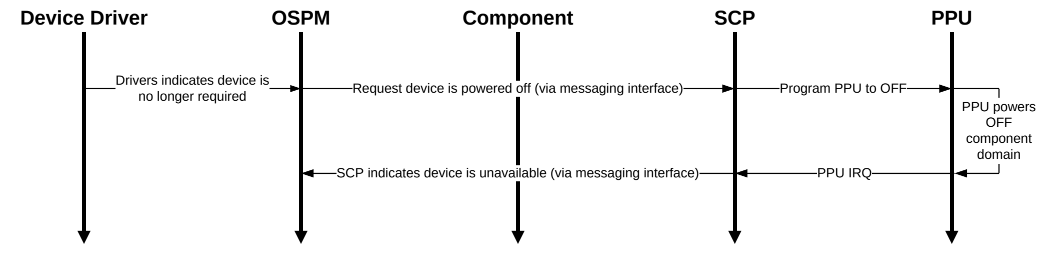
Figure 9-32 – Generic device power on flow

The following details the device power on flow:

- A device availability requirement is expressed to the OSPM by the device driver or other system software.
- The OSPM requests SCP to power on the device using the SCP messaging interface.
- The SCP programs the PPU to turn on the component.
- The PPU turns on the component domain.
    - If the component has an LPI the PPU handshakes with the device.
- The PPU interrupts the SCP.
- The SCP indicates to the OSPM that the device is available.
- The OSPM sends the device driver a callback.
- The device driver uses the component as required.

9.4.2 Device Power Off

Figure 9-33 shows a generic flow for powering off a device.



Device Driver
OSPM
Component
SCP
PPU
Drivers indicates device is no longer required
Request device is powered off (via messaging interface)
Program PPU to OFF
PPU powers OFF component domain
SCP indicates device is unavailable (via messaging interface)
PPU IRQ

Figure 9-33 – Generic device power off flow

The following details the device power off flow:

- The device driver indicates to the OSPM it has finished using the device.
- The OSPM requests SCP to power off the device using the SCP messaging interface.
- SCP programs the PPU to turn off the component.
    - If the component has an LPI the PPU handshakes with the device.
- The PPU turns off the component domain.
- The PPU interrupts the SCP.
- The SCP indicates to the OSPM that the device is unavailable.

9.5 Debug

A debugger accesses the SoC through a Debug Access Port (DAP). This resides in an always on domain so a debugger can always be connected.

Additional debug logic can be in separate power domains, either in dedicated debug power domains or distributed across other power domains. Methods of making these other domains available through the DAP DP interface are described below.

Additionally the debugger can also request for the system to reset all debug logic.

These requests are handled by the SCP so it can reconcile these requests with other system dependencies.

Debug Reset Request

The Debug Port (DP) within the DAP contains two bits that are used to request reset of the debug infrastructure. These registers drive the **CDBGRSTREQ** output to the SCP and show the status of the **CDBGRSTACK** input from the SCP.

These two signals form a four phase handshake with the SCP and can be implemented as an interrupt input and status output pair as described in *System Control* on page 7-6.

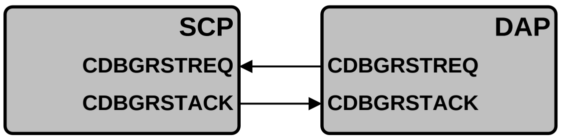


Figure 9-34 – Connections of CoreSight debug reset request

This handshake is intended to allow the debugger to attempt to reset the debug logic without affecting the functional behavior of the system. The SCP is expected to reset the DAP, the Debug APB and any other debug-only logic including debug through core power off logic in processor clusters.

The request is made in the DAP with the Control/Status Register CDBGRSTREQ bit. The acknowledgement can be observed by reading the DAP Control/Status Register CDBGRSTACK bit. For a full description of these registers see the *ARM CoreSight SoC-400 Technical Reference Manual*.

Figure 9-35 shows the timing diagram of the interface:

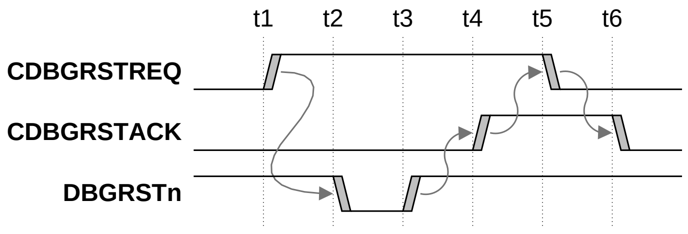


Figure 9-35 – CoreSight debug reset request timing diagram

CDBGRSTREQ is used as an interrupt to the SCP. The following details the flow:

- The DAP sets **CDBGRSTREQ** HIGH.
- The SCP receives an interrupt.
- The SCP resets the debug infrastructure.
- The SCP sets **CDBGRSTACK** HIGH.
- The DAP sets **CDBGRSTREQ** LOW.
- The SCP receives an interrupt.
- The SCP sets **CDBGRSTACK** LOW.

——— **Note** ————————

If this functionality is not required **CDBGRSTACK** must be tied LOW.

————————————————

Debug Power Requests

CoreSight Debug Power Request

The Debug Port (DP) within the DAP contains two bits that are used to request power to the debug power domain. These registers drive the **CDBGPWRUPREQ** output to the SCP and show the status of the **CDBGPWRUPACK** input from the SCP.

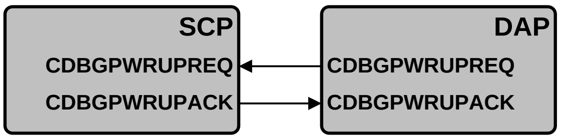


Figure 9-36 – Connections of CoreSight debug power request

These two signals form a four phase handshake with the SCP and can be implemented as an interrupt input and status output pair as described in *System Control* on page 7-6.

To request power on the debugger sets **CDBGPWRUPREQ** HIGH and waits for **CDBGPWRUPACK** to go HIGH before making any accesses to debug infrastructure.

When no longer required the debugger sets **CDBGPWRUPREQ** LOW and waits for **CDBGPWRUPACK** to go LOW.

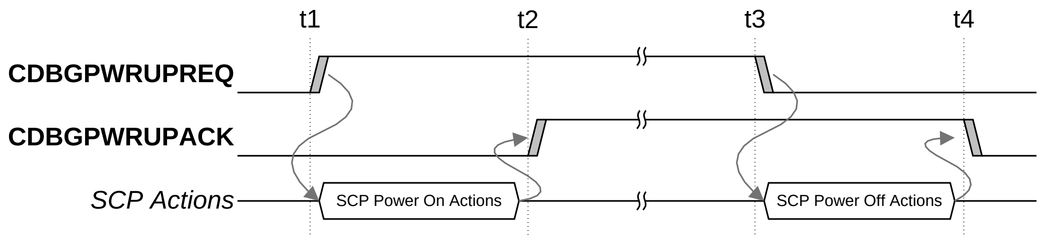


Figure 9-37 – CoreSight debug power request timing diagram

When the SCP receives **CDBGPWRUPREQ** HIGH it must power on enough debug infra-structure so that the debug agent can determine the state, but not necessarily access, debug resources.

For processor implementations without a dedicated debug power domain this will mean powering on the cluster and any interconnect needed to provide access to registers in the processor debug through core power off logic. This provides access to further bits which can be used to power on and prevent the powering off of individual cores. For more information see *Core Debug Power Requests* on page 9-45.

On a request, the SCP takes the appropriate action to power and clock the appropriate resources before acknowledging. If these resources are already available due to the system state then **CDBGPWRUPACK** can be set HIGH immediately, however these resources cannot be removed until **CDBGPWRUPREQ** is set LOW.

CoreSight System Power Request

The Debug Port (DP) within the DAP contains two bits that are used to request power to the entire system. These registers drive the **CSYSPWRUPREQ** output to the SCP and show the status of the **CSYSPWRUPACK** input from the SCP.

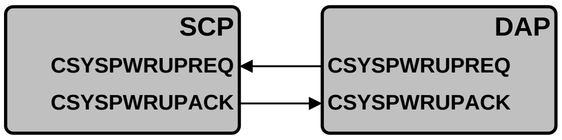


Figure 9-38 – Connections of the CoreSight system power request

These two signals form a four phase handshake with the SCP and can be implemented as an interrupt input and status output pair as described in *System Control* on page 7-6.

To request power on the debugger sets **CSYSPWRUPREQ** HIGH and waits for **CSYSPWRUPACK** to go HIGH before making any accesses to debug infrastructure.

When no longer required the debugger sets **CSYSPWRUPREQ** LOW and waits for **CSYSPWRUPACK** to go LOW.

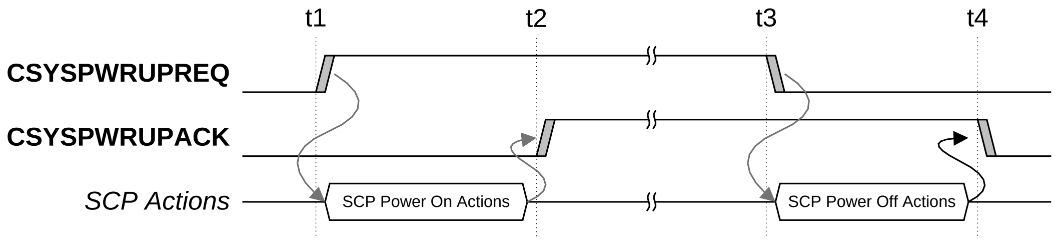


Figure 9-39 – CoreSight debug power request timing diagram

When the SCP samples **CSYSPWRUPREQ** HIGH it is being requested to power on the entire system.

On a request SCP takes the appropriate action to power and clock the appropriate resources before any acknowledgement. If these resources are already available due to the system state then **CSYSPWRUPACK** can be set HIGH immediately using a register mapped to the signal. However, these resources cannot be made unavailable until **CSYSPWRUPREQ** is set LOW.

Core Debug Power Requests

The debug agent can request individual cores to power on using the processor core External Debug Power & Reset Control Register Core Power Up Request bit (EDPRCR.COREPURQ). This sends a hardware request to the SCP to power on the processor core. This request should power on the core and make access to debug registers available.

The debug agent can also request that a powered on core remains powered by setting the Processor Core Debug Power Control Register Core No Power Down Request Bit (DBGPRCR.CORENPDRQ). This bit resides within the core so it can only be programmed when the core is on, however if the core power up request bit is set HIGH when the core exits reset then this bit will be set HIGH by default. When the core power up request bit is set LOW at reset this bit will reset LOW.

Where the functionality exists when DBGPRCR.CORENPDRQ is set HIGH, the power off mode of the processor core should be emulated. The power domain remains on but the functional reset is asserted to emulate the loss of context associated with the power off of a core. All debug state and access to debug registers must be maintained through any emulated power off.

CoreSight Granular Power Requestor

The CoreSight granular power requestor (GPR) component enables a debugger to request power on and power off of specific parts of the debug and trace infrastructure wherever the system power domain partitioning makes that possible. Without a GPR component debug power requests can only result in a power on or power off of the entire CoreSight system. The GPR then enables the implementation of finer grain power strategies for debug and test.

The GPR supports a configurable number of request acknowledge interfaces for the debugger to request the availability of the corresponding power domains. Figure 9-40 shows how these interfaces can be connected to SCP.

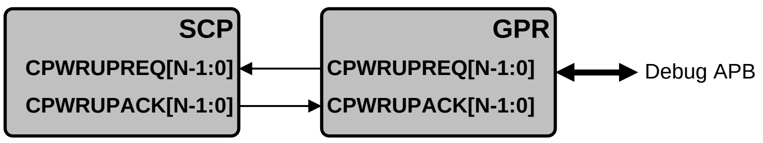


Figure 9-40 – Granular power requestor connections to SCP

The requests from the GPR can be handled by SCP using the same methods described for the overall debug power on request described in *CoreSight Debug Power Request* on page 9-44.

9.6 System Example

This section describes an example system as a means to summarize and consolidate major topics outlined in the previous chapters of this specification. In particular, emphasis is placed on the topics of partitioning, power states, power modes, clocking and power control infrastructure.

9.6.1 Block Diagram

Figure 9-41 shows the example system.

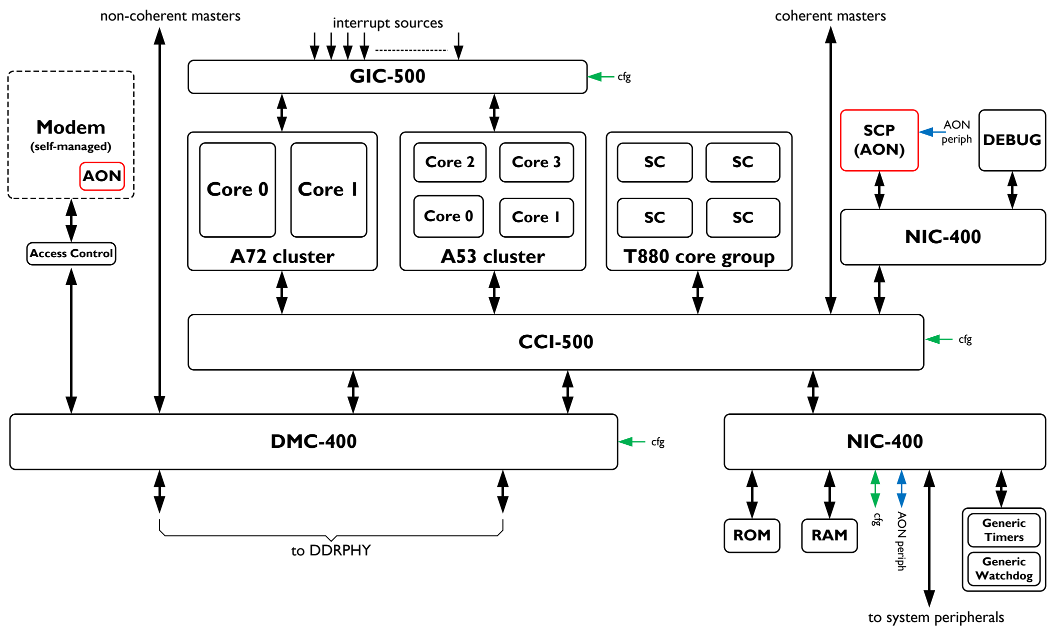


Figure 9-41 – System example: block diagram

The example system in Figure 9-41 is a partial SoC system. The example has sufficient elements for description of the topics in this section, without the complexity and scale of a complete SoC example.

The example includes a core subsystem for a mobile SoC design composed of application processor and GPU processor clusters, an interrupt controller, the memory system, some basic peripherals and a system control processor.

Additionally, an independent self-managed modem subsystem is included in the example as a shared user of the memory system. The modem subsystem, as a non-ARM component, is not considered in detail as it is included only to provide a means to describe, by example, how SCP can reconcile the power mode selection for SoC SLEEP states.

Figure 9-41 is a high level representation which excludes components relevant to power control such as domain bridges, clock generation and any distributed power control infrastructure. Also excluded is the system level debug and trace interconnect and infrastructure. These items are introduced as relevant to each topic in the following sections.

9.6.2 System Partitioning

The section describes a partitioning of the system example into voltage and power domains.

Voltage Domains

Figure 9-42 shows the voltage domain partitioning of the example system.

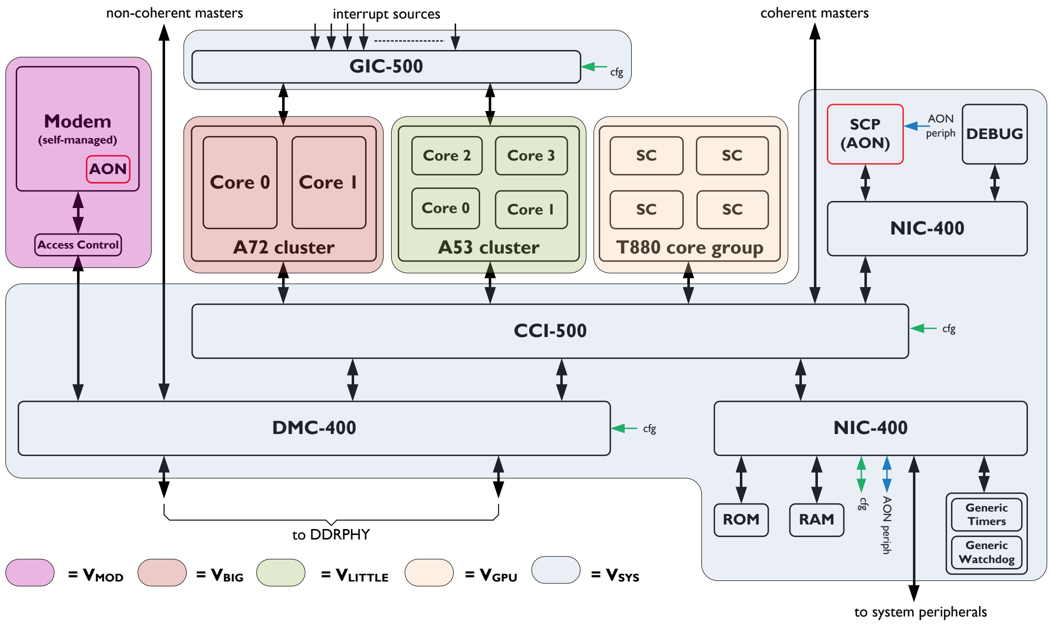


Figure 9-42 – System example: voltage domains

The following sections outline the voltage domains defined for the system example. As outlined in *Voltage Domains* on page 5-2 the scope in this specification is limited to the primary supplies for logic domains.

System Logic: V_{SYS}

The system logic voltage domain, V_{SYS}, includes the GIC, the memory system, basic peripherals, shared CoreSight debug and trace logic, and the system control processor.

In a complete system the V_{SYS} domain would be expected also to encompass parts of the system shown beyond the boundaries of the example such as the system peripherals, other masters and the DDRPHY logic.

Big Processor Cluster: V_{BIG}

The Cortex-A72 processor cluster has an independent voltage domain, V_{BIG}, to facilitate DVFS.

LITTLE Processor Cluster: V_{LITTLE}

The Cortex-A53 processor cluster has an independent voltage domain, V_{LITTLE}, to facilitate DVFS.

GPU Cluster: V_{GPU}

The Mali-T880 GPU has an independent voltage domain, V_{GPU}, to facilitate DVFS.

Modem Subsystem: V_{MOD}

The modem subsystem has an independent voltage domain, V_{MOD}.

The modem subsystem is not considered in detail. However, it is assumed to contain an always on area, with self-waking and power control capabilities, in addition to the modem functionality itself.

The access control component, between the modem and the memory controller, is also included within V_{MOD}. The integration of this is discussed in *Access Control* on page 9-59.

Power Domains

Figure 9-43 shows the power domain partitioning of the example system.

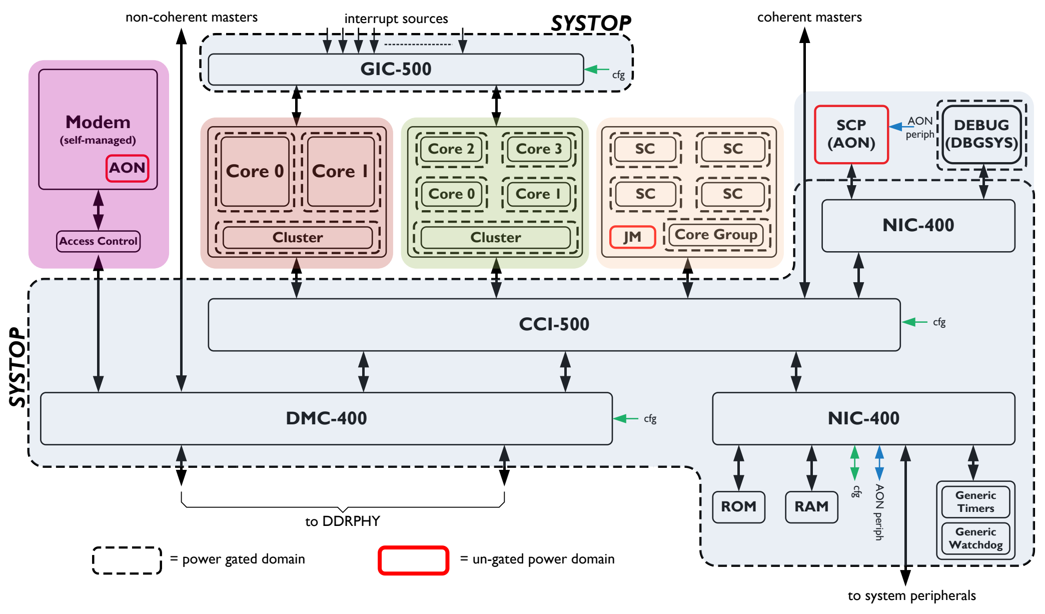


Figure 9-43 – System example: power domains

System Logic: V_{SYS}

The V_{SYS} voltage domain is partitioned into three power domains.

A primary system logic power gated domain, SYSTOP, includes the GIC, the memory system and basic peripherals.

Although not shown the SYSTOP domain is assumed to contain the clock generation subsystem including PLLs as well as selection and divide logic. The configuration of this clock subsystem is managed by the SCP.

In a complete system the SYSTOP domain would also be expected to encompass parts of the system shown beyond the boundaries of the example such as system peripherals.

The SYSTOP domain can be powered off in SoC SLEEP and OFF power states.

The following key points are noted for components within SYSTOP:

- The GIC-500 in the example system is configured without ITS and LPI support. As a result it has no memory master interface.
- The generic timers and generic watchdog instances are memory mapped and visible to AP software. Since these are not in an always on power domain the SCP timer resources are required to support wake-up of AP cores through SoC SLEEP states. The SCP timers are not visible to firmware running on the AP cores.
- The RAM component supports retention through SYSTOP power off and has a secure access portion. The retention support is required to maintain on-chip secure state through SoC SLEEP states.

An ungated power domain, SCP (AON), contains the system control processor and always on peripherals. This power domain also contains the debug access port (DAP) to allow debug access through SoC SLEEP states. The debug power request interface connections to SCP are detailed in *Debug Power Requests* on page 9-44.

Finally, a power gated domain, DBGSYS, containing all common CoreSight infrastructure apart from the DAP is implemented. This common infrastructure excludes CoreSight logic implemented in AP clusters and the SCP as well as interconnect for debug and trace between the AP clusters, the SCP and the DBGSYS domain.

In a complete system the V_{SYS} domain would be expected to include additional gated power domains for functions of significant size that do not also justify an additional voltage domain for DVFS.

Big Processor Cluster: V_{BIG}

Per-core and cluster power gated domains are supported for the Cortex-A72 cluster.

The cluster power domain also captures any integration logic such as CoreSight infrastructure and V_{BIG} portions of domain bridges.

LITTLE Processor Cluster: V_{LITTLE}

Per-core and cluster power gated domains are supported for the Cortex-A53 cluster.

The cluster power domain also captures any integration logic such as CoreSight infrastructure and V_{LITTLE} portions of domain bridges.

GPU Cluster: V_{GPU}

Per-shader core and core group power gated domains are supported for the Mali-T880 GPU.

V_{GPU} also contains an ungated power domain at top level containing the job manager and V_{GPU} portions of domain bridges. The leakage cost of this domain can be mitigated in modes where the GPU is not required by external switch off of V_{GPU}.

Modem Subsystem: V_{MOD}

Apart from the assumption that modem subsystem contains an always on area, with self-waking and power control capabilities, further partitioning is not considered.

9.6.3 Power States and Modes

Core Power States

The Cortex-A72 and Cortex-A53 AP cores in the system example are configured and implemented to support an identical set of power modes. The Advanced-SIMD / FP dynamic retention mode is not implemented in the Cortex-A53 cores.

Table 9-2 shows the core power state to power mode mapping for all AP cores in the example system.

Table 9-2 – System example: AP core power states and modes

Power State	Core Power Mode	Note
RUN	ON	Core running code
IDLE_STANDBY	ON	WFI or WFE
	RET	WFI or WFE, Dynamic Core retention (PPU Full Retention mode)
SLEEP	OFF	Off, can be woken by interrupts
OFF		Off, cannot be woken by interrupts

Cluster Power States

The Cortex-A72 and Cortex-A53 clusters in the system example are configured and implemented to support an identical set of power modes.

Support for static retention of L2 RAM, through cluster logic power off, is not supported. Correspondingly the SLEEP_RETENTION power state is not available.

Table 9-3 shows the cluster power state to power mode mapping for both AP clusters in the example system.

Table 9-3 – System example: cluster power states and modes

Cluster Power State	Core Power State	Power Mode		Note
		SCU-L2	L2 Data RAMs	
RUN	Any	ON	ON	Cores able to run code
	Each core in either: IDLE_STANDBY or SLEEP or OFF		RET	Dynamic L2 RAM retention (PPU Functional Retention mode)
SLEEP	All core in SLEEP or OFF, with at least one in SLEEP	OFF	OFF	Cluster fully off, can wake from interrupt
OFF	All cores in OFF	OFF	OFF	Off, Cores in this cluster cannot wake from interrupts.

Device Power States

System Logic: V_{SYS}

Although the SYSTOP power domain in the system example contains a number of peripherals these are all available whenever SYSTOP is powered on. These peripherals must also be supplied with clocks without any explicit OSPM request due to their necessity to system operation.

Consequently, no device power state representation is required for these peripherals.

GPU Cluster: V_{GPU}

From an OSPM perspective a device dependency on the Mali-T880 GPU only requires the availability of the job manager. The device driver then manages all core and core-group power modes through the job manager and these are invisible to the OSPM.

The only device power state representation required, for the system example configuration, is to determine the availability of the V_{GPU} voltage to the ungated job manager power domain and the GPU clock input.

Table 9-4 shows the device power state to power mode mapping for the Mali-T880 GPU in the example system.

Table 9-4 – System example: GPU device power states and modes

Device Power State	Job Manager Domain (V_{GPU})	Power Mode		Note
		Core Group	Shader Cores	
RUN	ON	OFF	All OFF	GPU device available, inactive
		ON	All OFF	GPU device available, memory system on
		ON	At least one core ON	GPU device available, processing active
OFF	OFF	OFF	All OFF	GPU device unavailable

Modem Subsystem: V_{MOD}

The modem is an independent fully self-managed device and has no OSPM power state representation.

SoC Power States

Table 9-5 shows the SoC power state to power mode mapping in the example system. The DEEPSLEEP state is not implemented.

Table 9-5 – System example: SoC power states and modes

SoC Power State	Processor Cluster States	Device States	Power Domain Mode		System Memory	System Input Clocks		Note
			SYSTOP	SCP (AON)		REFCLK	RTCCLK	
RUN	Any	Any	ON	ON	Available	ON	ON	AP cores able to run, can wake on any enabled interrupt. OS managed devices operable.
SLEEP0	All clusters in SLEEP or OFF. At least one in SLEEP	OFF	OFF	ON	Not Available. Context is migrated or retained.	ON	ON	Wake only on always on domain wake events. Low latency wake.
SLEEP1	All clusters in SLEEP or OFF. At least one in SLEEP	OFF	OFF	ON	Not Available. Context is migrated or retained.	OFF	ON	Wake only on always on domain wake events. High latency wake.
OFF	All clusters in OFF	OFF	OFF	OFF	OFF	OFF	OFF	External wake-up only, for example, power on reset.

The power domain mode mapping is straightforward in this case since the SYSTOP power domain contains the GIC, the memory system and the base peripherals.

Table 9-5 includes two depths of SLEEP state, SLEEP0 and SLEEP1. These are differentiated according to latency constraints. In SLEEP0 the primary system reference clock, REFCLK, is kept running for a low latency wake. In SLEEP1 the reference clock is stopped and only the slow RTCCLK is kept running. These system clocks are described in *Clock Domains* on page 9-55. The implications of the SLEEP states to clock control and system time preservation in the SCP are detailed in *System Control Processor* on page 9-59.

Table 9-5 only considers the OSPM perspective on system memory availability. Table 9-6 shows the consequences on the power state to power mode mapping of the SCP reconciling a modem access request.

Table 9-6 – System example: reconciling modem access requests

SoC Power State (OSPM)	Modem Access Request	SYSTOP Mode	System Memory	Note
RUN	Any	ON	Available	No impact of modem access request until SLEEP entry attempt.
SLEEPx	NO	OFF	Not Available	Power mode selection according to OSPM power state constraint.
	YES	ON	Available	Power mode selection shallower then OSPM power state constraint to reconcile modem access request.

The implications of Table 9-6 are that when a modem access request for system memory is present then, regardless of the OSPM SoC power state selection, the SYSTOP domain will not enter OFF mode. Furthermore, if the SYSTOP power domain was in OFF mode when the modem access request was made, the domain will be powered on. Moving SYSTOP to an ON mode still satisfies the constraints of the OSPM power state selection as this is shallower than OFF mode.

9.6.4 Clocking and Clock Control

This section describes the partitioning of the core subsystem into clock domains and high level clock control support.

Clock Domains

Figure 9-44 shows the partitioning of the system example core subsystem into clock domains.

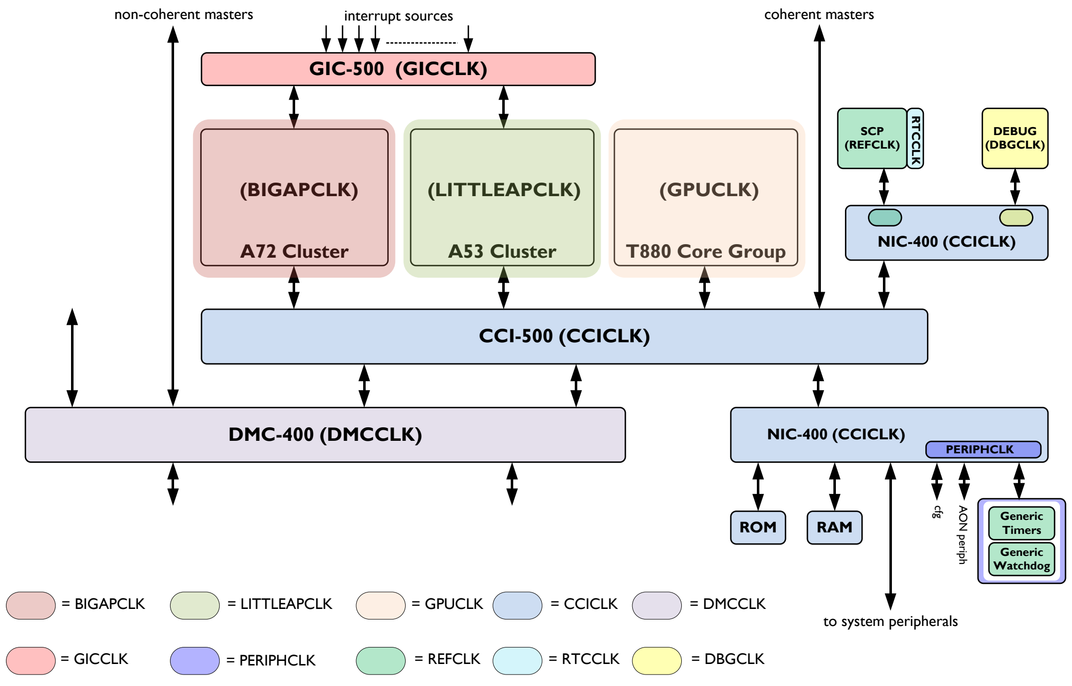


Figure 9-44 – System example: clock domains

The clock partitioning approach is according to both the requirement for independent, and diverse, control of frequency and also the globally asynchronous, locally synchronous (GALS) approach.

Where high level clock activity has divergent requirements then an asynchronous boundary is created between functional regions. In many cases this is coincident with the requirement for independent frequency control, such as between the processor clusters supporting DVFS and the coherent interconnect. In other cases, such as the boundary between the coherent interconnect and the memory controller this choice is according to a significant divergence in activity requirement.

While domain bridges are not included in Figure 9-44 they are required either as explicit bridges or as supported within the component as in the case of NIC-400. A specific example showing domain bridges, for the CCICLK domain, is shown in Figure 9-45.

REFCLK and RTCCLK are input clocks to the system. All other clocks are generated within the system. The following sections describe each clock domain.

Reference Clock: REFCLK

The reference clock, REFCLK, is the primary clock input to the system. This clock, in the range of 25 to 100MHz in this example, is the main clock of the SCP. It also provides a reference input for all clock generation in the system as the input to PLLs and as the reference for system time.

The SCP interfaces to the core subsystem using a REFCLK to CCICLK asynchronous boundary crossing within a NIC-400 component.

Real Time Clock: RTCCLK

The real time clock, RTCLK, is an input clock to the system used only in SCP. It provides a reference for wake-up generation, wake-up detection and system time preservation through deep SoC SLEEP states. A typical frequency for RTCCLK is 32,768Hz.

Coherent Interconnect Clock: CCICLK

The coherent interconnect clock, CCICLK, is used to clock the CCI-500, the primary clock domains of the connected NIC-400 components and attached RAM and ROM components.

While the CCICLK might be a similar, or identical, frequency to the memory controller clock it is separated by an asynchronous boundary for reasons of divergent activity. While the memory controller requires a clock at all times outside of memory self-refresh, the CCICLK domain only requires a clock when transactions are in-flight, and can be gated at other times.

Peripheral Interconnect Clock: PERIPHCLK

A clock is provided for the core subsystem peripheral interconnect system, PERIPHCLK. This is interfaced to through an asynchronous boundary crossing within a NIC-400 component.

GIC Clock: GICCLK

The GIC-500 clock is required to be active at all times when powered on and interrupts can be generated. An independent asynchronous clock, GICCLK, is provided. This obviates issues with activity requirement conflicts and also allows the lowest possible frequency, within latency requirements, to be used.

DMC Clock: DMCCLK

The memory controller clock, DMCCLK, is independent from the remainder of the core subsystem as outlined in the CCICLK section above.

The DMCCLK is expected to also clock parts of the DDR PHY in the complete system.

Big AP Clock: BIGAPCLK

The Cortex-A72 cluster has an independent main clock, BIGAPCLK, as required for support of DVFS.

LITTLE AP Clock: LITTLEAPCLK

The Cortex-A53 cluster has an independent main clock, LITTLEAPCLK, as required for support of DVFS.

GPU Clock: GPUCLK

The Mali-T880 GPU has an independent main clock, GPUCLK, as required for support of DVFS.

Debug Clock: DBGCLK

The primary clock for the CoreSight debug infrastructure, DBGCLK, is only required to be active when debug is connected and not during mission mode. Any dedicated debug APB interconnect also uses DBGCLK.

Although not included in Figure 9-44, for reasons of simplification, in a system with trace support then a further higher frequency trace clock, or clocks, would typically be provided for the associated infrastructure.

Domain bridges are then required for debug APB and trace interfaces at the boundary of the AP clusters.

The CoreSight subsystem interfaces to the core subsystem interconnect using a REFCLK to CCICLK asynchronous boundary crossing within a NIC-400 component.

Clock Control

SCP Control

SCP controls clock source activity and frequency selection. This control can be both implicit and also explicitly under OSPM direction.

Many clock source activity requirements are implicit according to the current system state. For example the interconnect system clock sources are required to be active whenever SYSTOP is powered and the processor cluster clocks are similarly required whenever a power domain within the cluster is active.

Other clocks would have explicit activity requirements under OSPM direction. A common example of an explicit requirement would be a system peripheral, under control of a device driver, located in a multi-function power domain.

Clock frequency selection can be implicit for common resources such as the memory system, peripheral interconnect and simple peripherals. However, this might be overridden under OSPM direction. Explicit OSPM direction is anticipated for all processor clusters at all times except boot when the SCP must select an initial start-up frequency for the boot AP core.

Interconnect Clock Gating

As described in *Clock Gating Control Integration* on page 8-6 many ARM CoreLink components support the implementation of high level clock gating and components can be combined to use a single gated clock under control of a clock controller component described in *Clock Controller* on page 7-12.

Figure 9-45 shows an example implementation of high level clock gating in the system interconnect. The example is for the CCICLK domain.

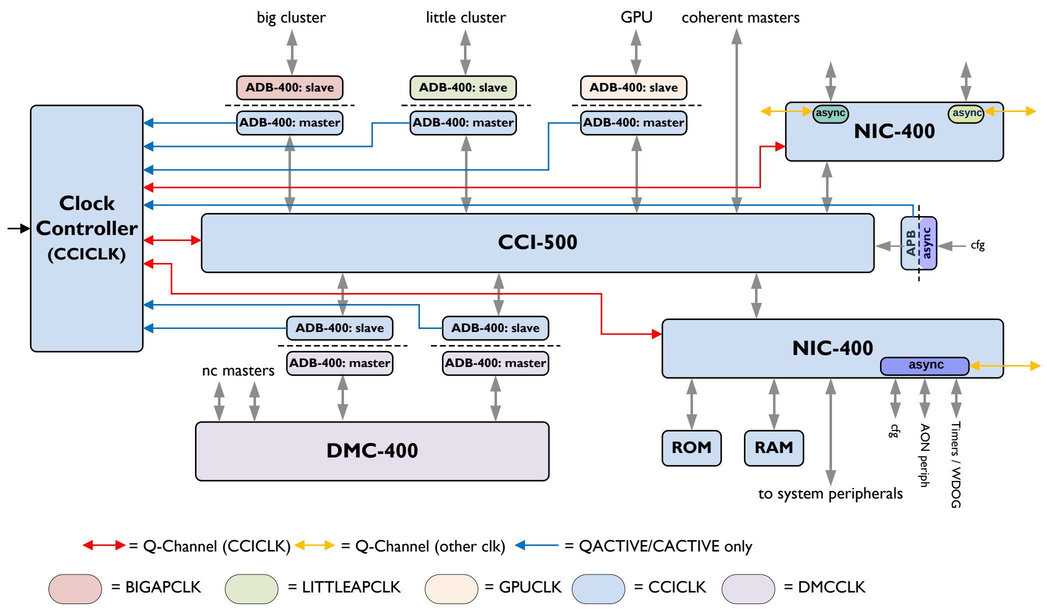


Figure 9-45 - System example: CCICLK high level clock gating

In Figure 9-45 a single clock controller component with both **QACTIVE**-only and full Q-Channel interfaces is used. All components in the CCICLK domain share the same high level clock gated source managed by this clock controller. The clock activity is according to the presence of transactions in-flight in any component within the clock domain.

Figure 9-45 also introduces the detail of the domain bridges at the clock domain boundary, including ADB-400 for all AXI and ACE interfaces in addition to an asynchronous APB domain bridge. All bridges in this case are implemented with **QACTIVE**-only, or **CACTIVE**-only in the case of ADB-400, clock control interfaces reliant on the guarantees provided by the full Q-Channel of a directly connected component. This approach is detailed in *Clock Domain Crossing* on page 8-7.

The RAM and ROM components within the CCICLK domain are assumed to be only reliant on a clock when transactions are outstanding in components upstream, in particular the lower NIC-400. This type of arrangement, and requirements for functional correctness, is described *in Clock Domain Scope* on page 8-9.

9.6.5 Power Control Integration

This section provides high level details for power control integration in the system example. The system control processor is covered in *System Control Processor* on page 9-59.

Power Policy Unit Integration

The power policy unit (PPU) for power domain control is introduced in *Power Policy Unit* on page 7-9 and integration strategies are described in *Power Control Integration* on page 8-10.

V_{BIG} and V_{LITTLE} Power Domains

The system example adopts the distributed PPU approach for the AP cluster power domains. Figure 9-46 shows a high level overview of the approach.

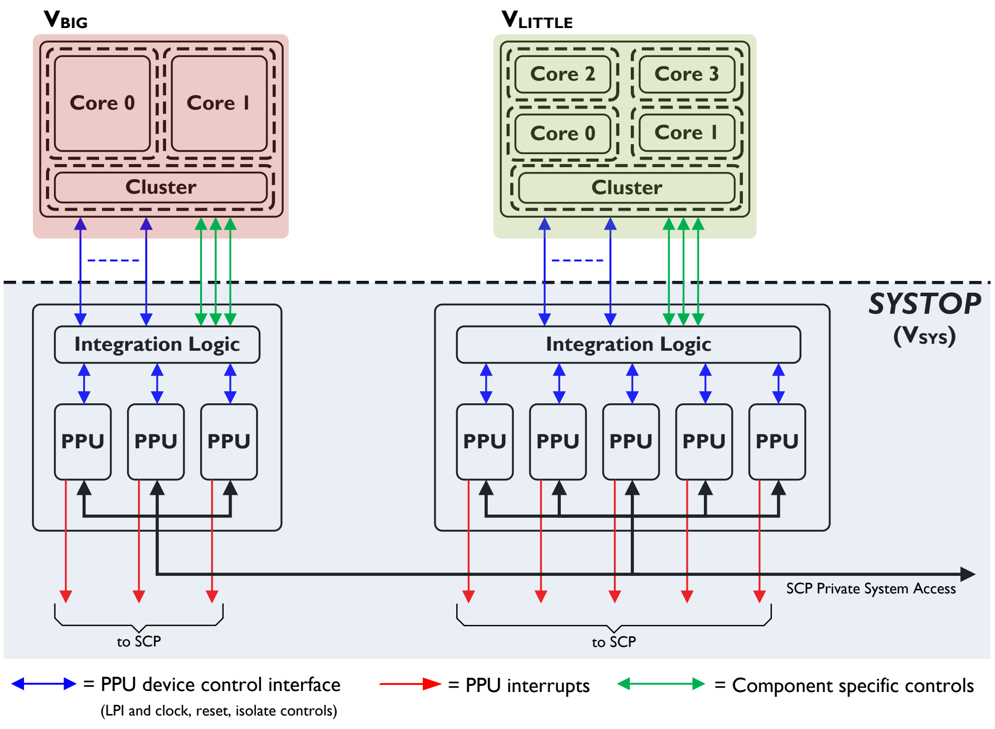


Figure 9-46 – System example: Application processor cluster PPU integration example

The PPUs and required integration logic, to manage component specific controls, are instantiated in the SYSTOP power domain which is relatively always on with respect to the AP cluster power domains.

This arrangement, disaggregated from the SCP always on domain, allows closer physical placement and use of local high speed clocks. The use of high speed clocks is critical to hardware autonomous power modes such as core and L2 dynamic retention to ensure low latency entry and exit transparent to software.

The lower level power control state machine (PCSM) detail, outlined in *Power Policy Unit* on page 7-9, is excluded from Figure 9-46 but these are expected to be integrated together with the PPUs.

Further detail for power control integration of A-profile processors is given in *ARM Cortex A-Profile Processors* on page 9-2.

V_{GPU} Power Domains

From a system power control perspective the Mali-T880 GPU only requires the availability of the job manager, an input clock, and the considerations for supply and interfacing to the VGPU voltage domain.

The core and core-group power domains are managed by the device driver which programs internal power domain controllers, with similar function to the PPU. The power domain controllers are dependent, again as the PPU, on an implementation specific power switch management function. For this purpose a PCSM created for use with the PPU is anticipated to be reusable with some small adaptations.

Since the system example selects the strategy of implementing the job manager and any required domain bridges as an ungated power domain within V_{GPU} there is no PPU integration requirement for that domain.

VSYS Power Domains

The SYSTOP power domain PPU is included in the always on SCP power domain. This is necessary since SYSTOP is relatively always on to all other power gated domains.

The DBGSYS power domain PPU is also included in the always on SCP power domain. This PPU could also be implemented in SYSTOP as the shared CoreSight infrastructure is anticipated to be dependent on resources implemented in SYSTOP such as clock sources and interconnect.

In a full system other V_{SYS} power gated domains might have PPUs instantiated in the always on SCP power domain, in SYSTOP or in any other power domain with a relatively always on relationship to the controlled domain.

Access Control

The system example includes an access control gate between the self-managed modem subsystem and the core subsystem memory controller. This is included so that:

- Power off of the SYSTOP domain can be managed safely even if a race condition means a modem memory access starts during the power off sequence.
- The modem can independently attempt memory access when the SYSTOP domain is off. A wake-up condition, as an always on domain event, is created and the SCP can make the memory system available.

The access control gate is implemented within the V_{MOD} voltage domain outside of the modem subsystem. This assumes that the modem does not support such a capability itself and that this is a system integration issue.

The choice of integrating within V_{MOD}, instead of V_{SYS}, is made so that the access control function is before, or integrated into, the domain bridge required at the voltage boundary and transactions can only enter the domain bridge when in the correct state.

An overview of access control is provided in *Access Control* on page 8-18.

9.6.6 System Control Processor

The system control processor of the system example conforms to the description in *System Control Processor* on page 7-4. For the example system the SCP is based on a Cortex-M3 core which is suitable for this class of system.

This section details the specific implications of the system example SoC SLEEP states for SCP clocking and system time preservation.

The system example SoC SLEEP states are detailed in *SoC Power States* on page 9-53.

Clocking

The primary SCP clock source for all functions is REFCLK and this clock is active in the RUN and SLEEP0 SoC power states. However, REFCLK can be stopped when the system enters the SLEEP1 SoC power state to save the power associated with its generation at system level.

The SCP also has a real time clock source, RTCCLK, active in all SoC power states except OFF. The RTCCLK domain has a dedicated timer subsystem and supports detection of external always on domain wake events.

The SCP switches to the RTCCLK at entry to the SLEEP1 SoC power state and back to REFCLK at exit from SLEEP1 to the RUN state. The SCP must ensure REFCLK clock is active before the SoC power state can return to RUN.

The SCP clocking modes are managed as follows:

- A request-acknowledge handshake interface is used to allow the SCP to request REFCLK activity and for the system to guarantee it is available and stable.
- The Cortex-M3 SLEEPDEEP mode is used as a firmware means to indicate a period where REFCLK can be stopped. Entry and exit from the SLEEPDEEP mode is used to control the REFCLK request status to the system. When REFCLK is required the Cortex-M3 is configured to enter the SLEEPING mode when idle.

For more information see the *Cortex-M3 Devices, Generic User Guide* specification.

System Time

Time Domains

The system example has the following time domains:

- **REFCLK:** This is the view of time observed by the system. This time domain can be impacted by SoC SLEEP states but this is not visible to the system.
- **RTCCLK:** This is a secondary view of time private to the SCP. It is always available when the SoC is powered on.
- **CoreSight:** This is used exclusively for the generation of CoreSight timestamps. This domain is differentiated from the REFCLK domain as it requires a generic counter that is not halted during debug. This time domain is not detailed further.

Counter and Timer Resources

The SCP always on power domain contains the following counter and timer resources:

- **REFCLK generic counter:** The primary system count value used by all REFCLK timers. The counter is programmable by both SCP and AP cores.
- **REFCLK timer:** Private to SCP.
- **RTCCLK generic counter:** Secondary counter, private to SCP.
- **RTCCLK timer:** Private to SCP. Uses the RTCCLK generic counter as its time source.

In addition to AP core private timers the SYSTOP domain implements memory mapped timers. All timers outside SCP are in the REFCLK time domain.

System Time Preservation

In the SLEEP1 SoC power state the REFCLK time domain is made unavailable. This is possible since no components that can observe system time, outside of SCP, are powered on.

However, before these components are powered on again the REFCLK time domain must first be restored to a consistent view. The SCP firmware must then program this value into the REFCLK generic counter, once REFCLK is guaranteed to be available and before entry to the SoC RUN power state.

When restoring the REFCLK counter the SCP firmware uses the RTCCLK time domain as a reference.

Time Domain Relationship

This section describes an example method for saving and restoring the REFCLK domain through a period when REFCLK is stopped. It allows for REFCLK and RTCCLK to be in a non-integer frequency relationship.

——— **Note** ———————

The method here should be considered only an example. The specific requirements of a target system must always be considered in detail.

—————————————

The following equation describes the relationship between the RTCCLK time domain and the REFCLK time domains. The current REFCLK time can be calculated from the combination of:

- The current RTCCLK time.
- The ratio between the frequency of RTCCLK and REFCLK.
- Known time values of the RTCCLK counter and the REFCLK counters, at a single point.

$$t_{REF} = \left(\left(\frac{f_{REF}}{f_{RTC}}\right)(t_{RTC} - T_{RTC})\right) + T_{REF}$$

Where:

- t_{REF} is the current time value in the REFCLK time domain.
- t_{RTC} is the current time value in the RTCCLK time domain.
- T_{REF} is a time value in the REFCLK time domain at a synchronization point in the past.
- T_{RTC} is a time value in the RTCCLK time domain at a synchronization point in the past.
- f_{REF} is the frequency of REFCLK, in Hertz.
- f_{RTC} is the frequency of RTCCLK, in Hertz.

This relationship enables REFCLK time to be linearly scaled and offset from RTCCLK time. The scaling factor is constant. The offset (T_{REF} and T_{RTC}) must be recalculated at each SLEEP1 entry so that any AP software updates are preserved. This is shown in Figure 9-47.

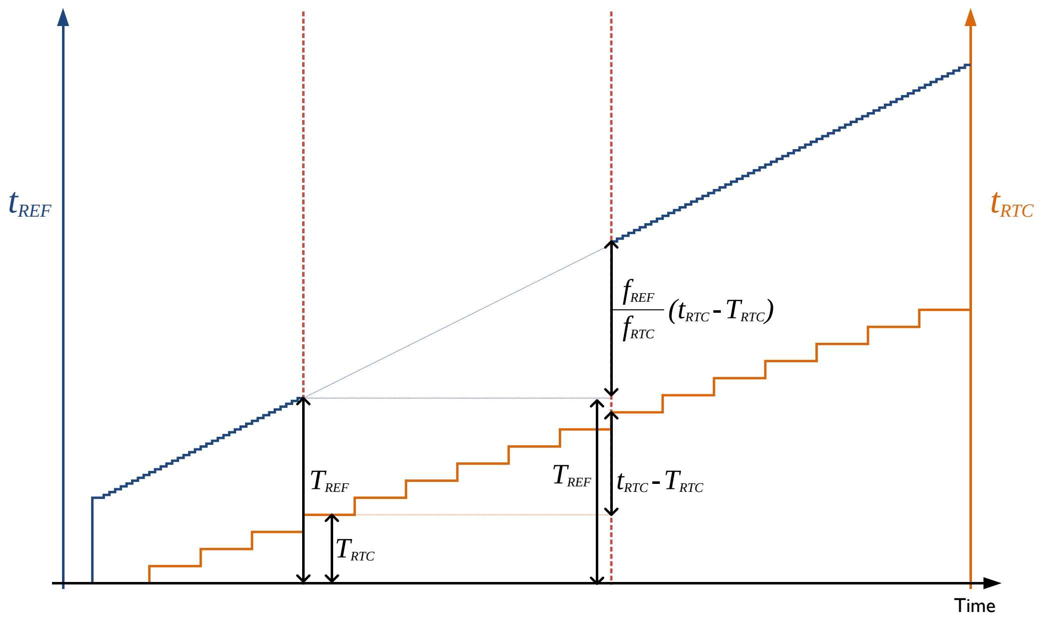


Figure 9-47 – System example: time domain relationship

The accuracy of this method is maximized by ensuring, in hardware, that when REFCLK time values are saved and restored that the sampling, using REFCLK in both cases, is synchronized to an RTCCLK rising edge. Although the synchronization delay can vary, within a few REFCLK cycles, this does not accumulate.

9.6.7 System Initialization

This section describes an example initialization sequence from power on reset through to system boot. This illustrates the functions of the SCP ROM code and how the final SCP RAM code is established in the SCP private RAM.

Figure 9-48 shows a high level representation of the system components involved in the initialization process.

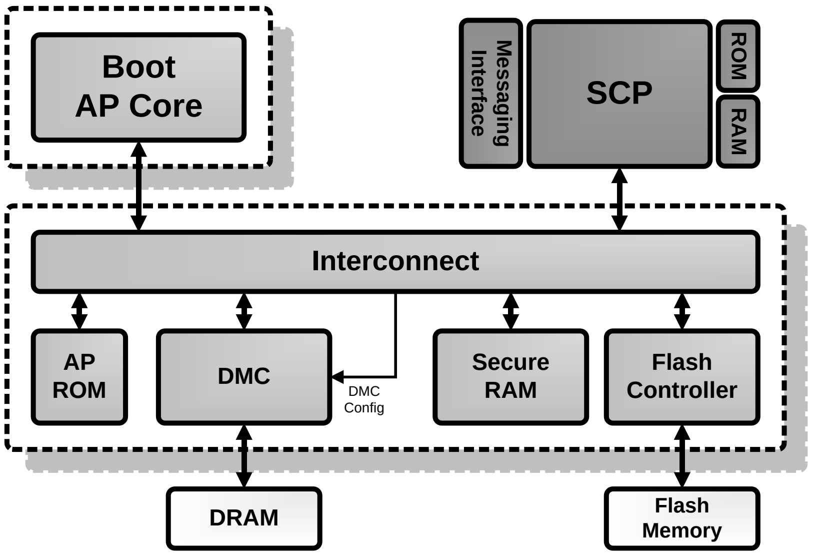


Figure 9-48 – System example: system components for boot

Figure 9-49 shows the main steps of the sequence.

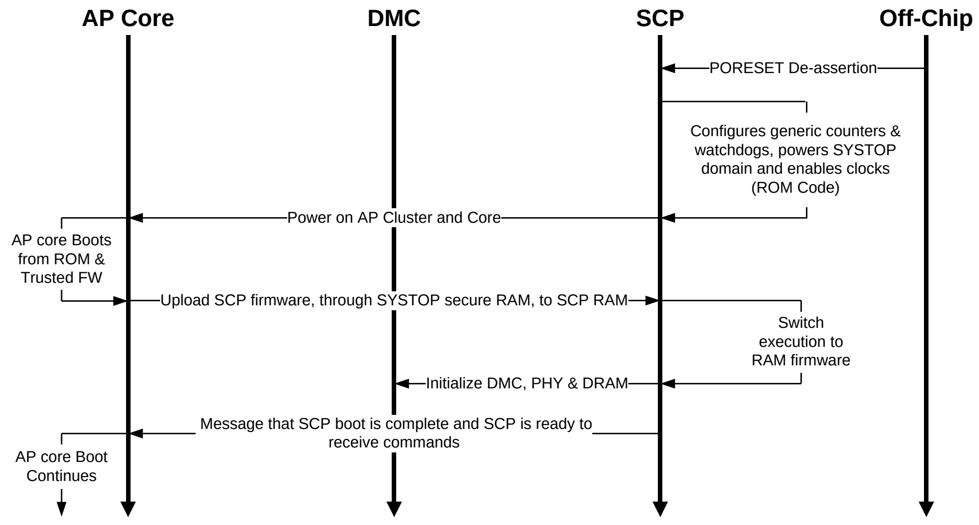

AP Core
DMC
SCP
Off-Chip
PORESET De-assertion
Configures generic counters & watchdogs, powers SYSTOP domain and enables clocks (ROM Code)
Power on AP Cluster and Core
AP core Boots from ROM & Trusted FW
Upload SCP firmware, through SYSTOP secure RAM, to SCP RAM
Switch execution to RAM firmware
Initialize DMC, PHY & DRAM
Message that SCP boot is complete and SCP is ready to receive commands
AP core Boot Continues

Figure 9-49 – System example: initialization sequence

The following describes the example sequence in more detail:

1. Power On Reset released
2. SCP requests external clocks (automatic at reset)
    - Waits until REFCLK is available.
3. SCP boots from its ROM:
    - Enables RTCCLK and REFCLK generic counters.
    - Enables SCP watchdog.
    - Powers on the SYSTOP power domain. It is assumed that the full system extends the core subsystem of the system example to include a flash controller in the SYSTOP power domain.
    - Configures and enables the clocks required to boot an AP core.
    - Powers on the boot AP cluster.
    - Powers on the boot AP core.
    - Waits for a message from the Boot AP core.
4. AP core boots from its ROM:
    - Loads AP boot firmware from flash to the secure RAM in SYSTOP.
    - Executes AP boot firmware.
    - Loads SCP firmware to the secure RAM in SYSTOP.
    - The SCP firmware is transferred to the SCP RAM by the following process:
        - AP firmware uses the messaging interface to indicate to SCP that a block of firmware is available in the secure RAM.
        - SCP copies the firmware from secure RAM to its private RAM.
        - SCP messages the AP that the block copy is complete.
        - This process is repeated as required to complete the transfer.
    - AP firmware waits for a message from the SCP to indicate its boot sequence is complete.
5. SCP switches execution from ROM code to the RAM firmware:
    - Initializes the memory controller and DRAM.
    - Sends message to AP firmware to indicate that its boot sequence is complete.
    - SCP is ready to receive commands from software running on AP cores.
6. AP core continues with its boot sequence.

10 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.* In such cases the content here is intended to accord to that specification and also provide complementary information.

————————————

This chapter is organized into the following sections:

10.1 General Low Power Interface Guidelines

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

10.1.1 Low Power Interface Implementation

Q-Channel Implementation

From a design perspective a component and its controller typically use asynchronous clocks. In clock control applications these clocks are normally from the same source, but due to significant phase differences the design considerations are those required for an asynchronous interface. Figure 10-1 shows the required interface signal handling for an asynchronous Q-Channel interface from a component perspective.

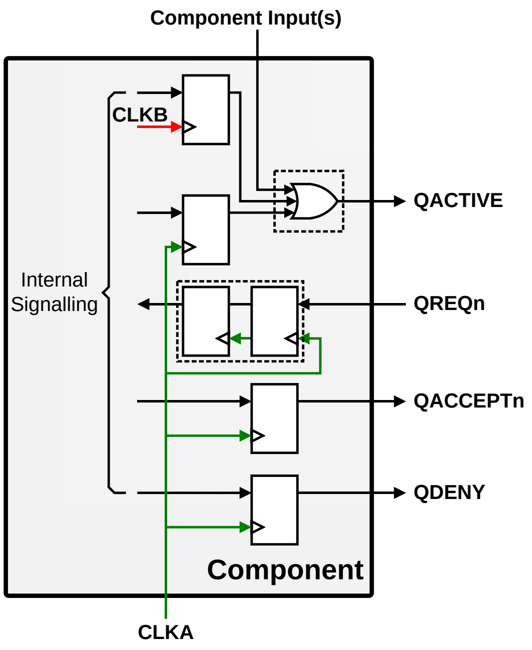


Figure 10-1 – Q-Channel interface implementation

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

- Synchronize all signals at destination before use. Specifically, it is strongly recommended that components include the synchronization elements for input signals. From a component perspective this is shown for **QREQn** in Figure 10-1.

  ——— **Note** ———————

  Potential consequences of implementing the synchronization externally to the component are detailed in *Synchronization of Input Signals* on page 10-4.

  ————————————

- To ensure correct operation of the handshake interface **QREQn**, **QACCEPTn** and **QDENY** outputs must be registered. From a component perspective this is shown for **QACCEPTn** and **QDENY** in Figure 10-1.

- The **QACTIVE** signal must be driven either directly by a register, or by a number of registers whose contributions are combined using a logical OR. In Figure 10-1 **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. Further implementation recommendations are given in *QACTIVE and PACTIVE Contribution Combination* on Page 10-4.

- 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

From a design perspective a component and its controller typically use asynchronous clocks.

Figure 10-2 shows the required interface signal handling for an asynchronous P-Channel interface from a component perspective.

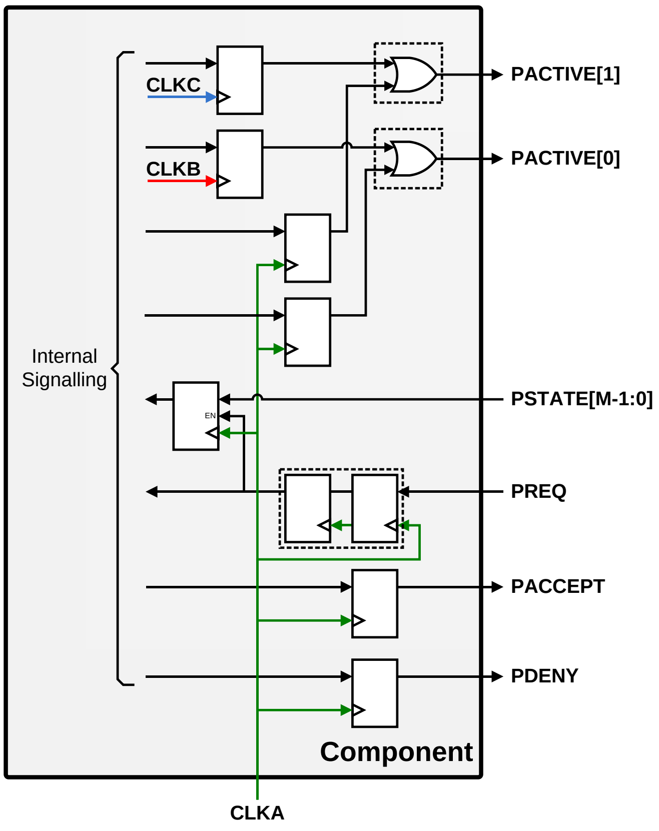


Figure 10-2 – P-Channel interface implementation

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

- Synchronize all signals, with the exception of **PSTATE**, at destination before use. Specifically, it is strongly recommended that components include the synchronization elements for input signals. From a component perspective this is shown for **PREQ** in Figure 10-2.

  ——— **Note** ————————

  Potential consequences of implementing the synchronization externally to the component are detailed in *Synchronization of Input Signals* on page 10-4.

  ————————————

- To ensure correct operation of the handshake interface **PREQ**, **PACCEPT** and **PDENY** outputs must be registered. From a component perspective this is shown for **PACCEPT** and **PDENY** in Figure 10-2.
- Each bit of the **PACTIVE** signal must be driven either directly by a register, or by a number of registers whose contributions are combined using a logical OR. In Figure 10-2 **PACTIVE** sources include a register in the interface clock domain and registers in two other clock domains within the component. Further implementation recommendations are given in *QACTIVE and PACTIVE Contribution Combination* on Page 10-4.

Capturing PSTATE

To ensure the **PSTATE** value is received correctly by the component, its capture must be enabled by a HIGH value on the **PREQ** signal synchronized in the destination clock domain as shown in Figure 10-2 on page 10-3.

In addition, to meet the P-Channel protocol requirements for component reset and initialization the **PSTATE** value must also be sampled at reset de-assertion within a period, t_{init}, defined in component clock cycles. This requirement to support **PSTATE** sampling without a handshake as one of the device reset and initialization methods requires additional circuitry which is not shown in Figure 10-2 on page 10-3.

To implement the sampling of **PSTATE** at reset de-assertion requires, in principle, an enable term for the **PSTATE** register which is set at reset and then cleared by following clock cycles.

Figure 10-3 shows the principle of **PSTATE** sampling at reset de-assertion.

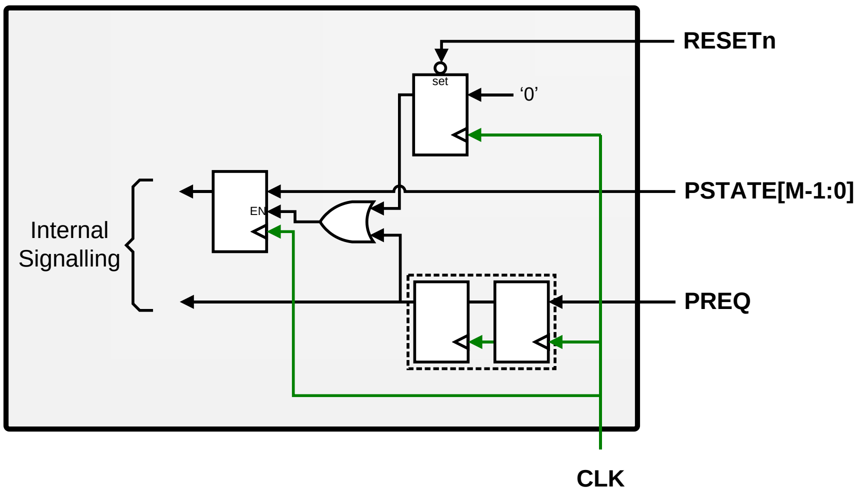


Figure 10-3 – Principle of PSTATE sampling at reset de-assertion

Dependencies between power control and clock low power interfaces are detailed further in *QACTIVE Contribution from Power Control Low Power Interface* on page 10-9.

Synchronization of Input Signals

It is strongly recommended that components internally include the synchronization elements for input signals. In a component design context this recommendation applies to **QREQn** and **PREQ** inputs.

The rationale for this recommendation is as follows:

- There can be a dependency between clock and power control interfaces requiring a clock control **QACTIVE** wake-up contribution when no clock is running. In such a case the implementation of external synchronization blocks visibility of request signals preventing the activation of the required **QACTIVE** path. A component with such dependencies must then implement the related synchronization internally. Such dependencies are detailed in *QACTIVE Contribution from Power Control Low Power Interface* on page 10-9
- Requiring the external implementation of synchronizers also places a burden on the integrator. Risks included omission of the synchronizers and inappropriate clocking.

While omission would most likely be resolved in clock domain crossing checks, the more significant risk is placing the synchronizers into the wrong clock domain or clock tree branch. To avoid the clock tree insertion pressure described in *High-Level Clock Domain Selection* on page 8-4, synchronizers must use the clock tree branch supplied to the component.

Figure 10-4 shows the correct arrangement for clocking of an external synchronizer.

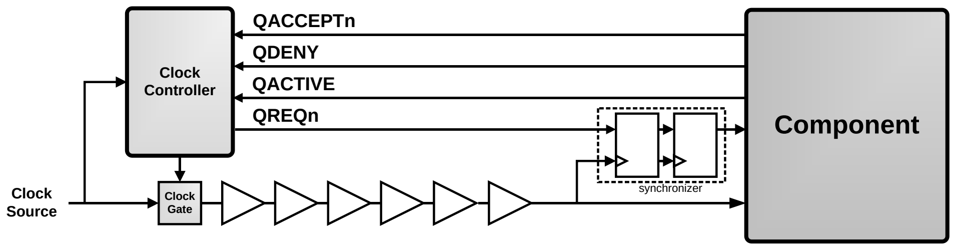


Figure 10-4 – Correct clocking of external synchronizer

Figure 10-5 shows the clock tree insertion pressure caused if an external synchronizer is clocked from the free-running clock source used as the input to the clock gate at a clock controller.

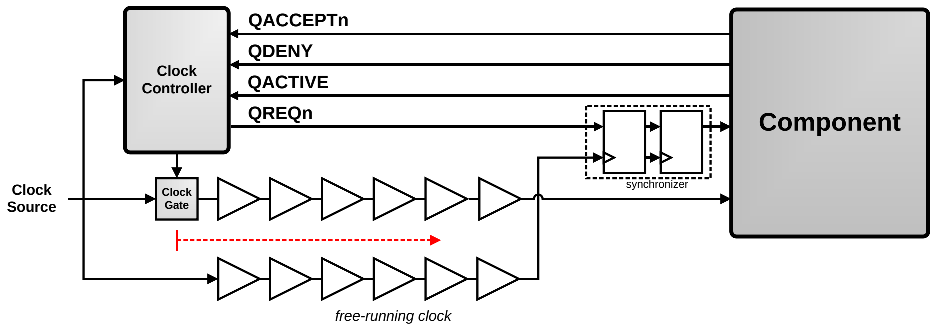


Figure 10-5 – Incorrect clocking of external synchronizer

The synchronizing elements are recommended to be captured in a dedicated design hierarchy. This aids identification for two purposes:

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

QACTIVE and PACTIVE Contribution Combination

As already described **QACTIVE** and **PACTIVE** signals must be driven either directly by a register, or by a number of registers whose contributions are combined using a logical OR

While the requirement to use registered source signals guarantees that there can be no systematic false wake-ups, from glitches at source, it is recommended that these OR gates are captured in a dedicated design hierarchy so the function can be identified and preserved through synthesis.

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

In specific circumstances practical considerations can require the use of an XOR gate in clock control **QACTIVE** generation. These circumstances are detailed in *QACTIVE Contribution from Power Control Low Power Interface* on page 10-9.

Recommended LPI Feature Support

It is recommended that components support the full set of LPI features, including support to deny requests.

The denial mechanism allows a component to respond in a timely fashion and continue operations without interruption. This enables the avoidance of potentially indefinite acceptance delay, for example in clock gating, or the latency overhead of transitions to and from modes when circumstances have changed. From a controller perspective the timely response avoids long term outstanding requests to components.

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

10.1.2 Interface Management

When a component enters a low power mode it might 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 by completing any outstanding transactions, or at safely suspending them within protocol. No new transactions can be started unless guaranteed to be processed correctly. At exit from the low power mode, the interface quiescence must be safely reversed before transactions are accepted.

This entry and exit is managed under control of requests from the low power interface for clock or power control accordingly.

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 at safe levels.

When in the low power mode, a number of possible response models to attempted bus transactions can occur and a set of models is defined in *Access Control* on page 8-18.

Clock Control

In a clock control application, the stall and wake response model is expected to be used exclusively.

As part of an accepted low power interface request the component performs the entry sequence to ensure bus interfaces are quiescent. As the component is still powered on in the clock gated state the system can provide an input wake-up signal to the component which can be used as a stimulus to exit the quiescent state using the low power interface. Once the exit sequence is completed, and the clock is supplied, the component can process transactions.

High level clock gating, including a specific AXI slave implementation example, is detailed in *Component High-Level Clock Gating* on page 10-8.

Power Control

In a power control application, all of 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.

The component will not then be in an operating state during most power control modes and cannot provide any response. As described in *Access Control* on page 8-18 any response during the low power mode is then provided either by a component in another powered on domain, which is upstream in the bus transaction flow, or using static isolation cell values.

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

After power is restored and the low power interface is used to transition the component to an operating mode it can start to process transactions.

The following sections provide a high level overview of how these different response types can be managed on AMBA interfaces.

AMBA4 AXI and ACE

A component slave interface can stall transactions on an AXI or ACE bus by holding the *__READY__ outputs LOW. *__VALID__ inputs must be ignored when the clock is not guaranteed.

A component master interface must hold ***VALID** outputs LOW in a quiescent state.

For entry to a quiescent state tracking of requests accepted, to responses received by a component is relatively straightforward. An AXI slave clock gating implementation example is detailed in *Component High-Level Clock Gating* on page 10-8.

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

A HIGH level on the ***VALID** or **AWAKEUP** inputs, at a slave interface, can be used to generate a wake request by setting **QACTIVE** or a **PACTIVE** HIGH.

Generating an error response requires sequential behavior and can only be performed outside of a component that is in a non-functional state.

AMBA5 CHI

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.

An **RXSACTIVE** input can be used to generate a wake request by setting a **QACTIVE** or a **PACTIVE** signal HIGH.

Generating an error response requires sequential behavior and can only be performed outside of a component that is in a non-functional state.

AMBA APB

A component slave interface can stall transactions on an APB bus by setting the **PREADY** output LOW.

A HIGH level on the **PSEL** or **PWAKEUP** inputs can be used to generate a wake request by setting **QACTIVE** or a **PACTIVE** HIGH.

A component can generate an error response on an APB bus by setting **PSLVERR** HIGH and **PREADY** HIGH. For APB this can be implemented with simple tie-offs or isolation values.

10.2 Component High-Level Clock Gating

This section describes implementation of high level clock gating using Q-Channel low power interfaces.

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 *Clock Gating Levels* on page 8-2.

10.2.1 Q-Channel Implementation

To provide external high level clock gating support a component requires a Q-Channel interface for each controlled clock input.

Each clock requires a separate interface so it can be gated separately, and combined with other components in the same clock domain. Figure 10-6 shows an example of a component with multiple clock domains.

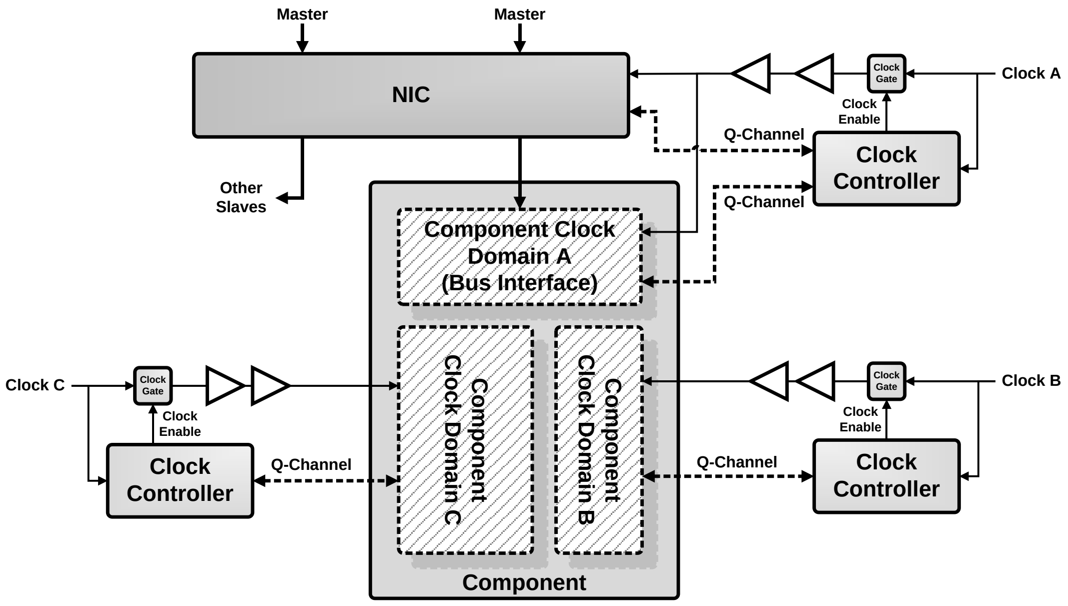


Figure 10-6 – Example of a component with multiple clock domains

A common example, as shown in Figure 10-6, 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.

The clock control Q-Channels must be separate from any power control LPIs.

10.2.2 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 a number of reasons for this, but one example is when another component managed by the same clock controller denies a quiescent 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.

10.2.3 QACTIVE Behavior

QACTIVE is used for two purposes in clock control:

1. When HIGH: To indicate 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: To hint that the component might accept a clock control Q-Channel quiescence request.

QACTIVE at Reset

It is recommended that a clock control **QACTIVE** is LOW at reset. This allows the clock to be idle until required by the component.

If the component clock is required to run at reset de-assertion for some initialization purpose then **QACTIVE** can be reset HIGH. Once any initialization is complete, and assuming the component is then idle, the clock control **QACTIVE** should go LOW.

However, if the clock controller for the component is implemented outside the component power domain, internal state contributions to **QACTIVE** that are reset HIGH have the following consequences.

- If **QACTIVE** is set HIGH at reset assertion then when reset is applied, prior to power off, **QACTIVE** would then indicate a requirement to exit the *Q_STOPPED* state.
- If **QACTIVE** 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 exit the *Q_STOPPED* state when it is isolated throughout power off and retention modes.

For the reasons resetting internal **QACTIVE** state contributions HIGH is not recommended.

It is important in this case to distinguish between internal state **QACTIVE** contributions and external signaling **QACTIVE** contributions, such as from a power control LPI as described in *QACTIVE Contribution from Power Control Low Power Interface* on page 10-9.

QACTIVE Stimulus

QACTIVE stimulus can be from a number of sources derived from the following:

- Internal logic state.
- External signaling.
- Other low power control channels.

Some specific considerations are discussed in the following sections.

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 directly 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 an LPI controlled clock a contribution to the clock control **QACTIVE** is required. This is to ensure the clock is supplied during power control handshake sequences.

Figure 10-7 shows an example of component with single power and clock control interfaces:

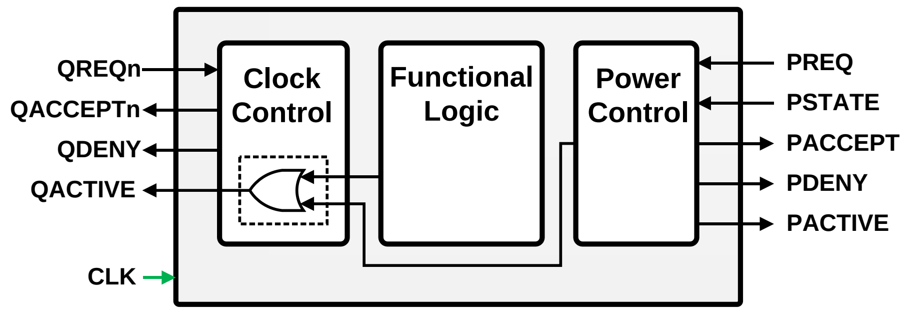


Figure 10-7 - Example of power control to clock control dependency

The example in Figure 10-7 has a clock control Q-Channel and a power control P-Channel. A power control request can, and would typically, arrive when the component function is inactive. Therefore, a power control request, in this case with **PREQ** HIGH, can arrive when the clock is stopped. Since the clock is needed to progress the power mode request this dependency must be resolved by a contribution to the clock control **QACTIVE** from the power control interface logic.

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, as a contribution to the clock control **QACTIVE** as a wake-up condition. Once the clock is running, and the power control **PREQ** is sampled HIGH, internal logic can keep the clock control **QACTIVE** HIGH until the power control sequence is completed.

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. This is an exception to the OR combination **QACTIVE** rule. In this specific case it is safe as the inputs to the XOR are guaranteed to change only one at a time. As with the OR combination this XOR should be placed in a separate level of hierarchy so the function can be correctly preserved through synthesis.

Once the clock is running and the power control **QREQn** state change is sampled internal logic must keep **QACTIVE** HIGH until the power control sequence is completed. This is important if the request is denied and the Q-Channel enters the *Q_CONTINUE* state. In *Q_CONTINUE* **QREQn** and **QACCEPTn** will both be HIGH and output of the XOR will be LOW but the clock is still required to set **QDENY** LOW to return to *Q_RUN*.

QACTIVE Policy

QACTIVE should be driven LOW whenever possible to provide the maximum opportunity for clock gating. Hysteresis can be applied internally between a component becoming idle and the **QACTIVE** output going LOW.

This might be useful if there is component specific timing which is not available to the system. However, it is recommended that the application of hysteresis is implemented by the clock controller as the required behavior is typically system specific and depends on the combination of all components using the clock.

10.2.4 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 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 clock control Q-Channel can be made to a component at any time regardless of the state of the **QACTIVE** signal. When a quiescence request is made the component should respond in a timely manner.

A component should only wait to accept a clock control Q-Channel quiescence request when a response can be given in a timely fashion. If a component cannot bound the time required until it will accept it should deny the request.

Actions required to enter quiescence

A component must ensure that when it accepts a request there will be no functional failure as a result 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 external interfaces are managed as described in *Interface Management* on page 10-6.

All required actions need to be completed before an accept response is sent. Once the accept response is sent the clock is not guaranteed to be available.

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.

However, the controller cannot make another quiescence request until the response is received and the interface is in *Q_RUN* state. 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.

10.2.5 Implementation Example: AXI Slave Interface

Figure 10-8 shows a conceptual implementation of high level clock gating support for a component with an AXI slave interface.

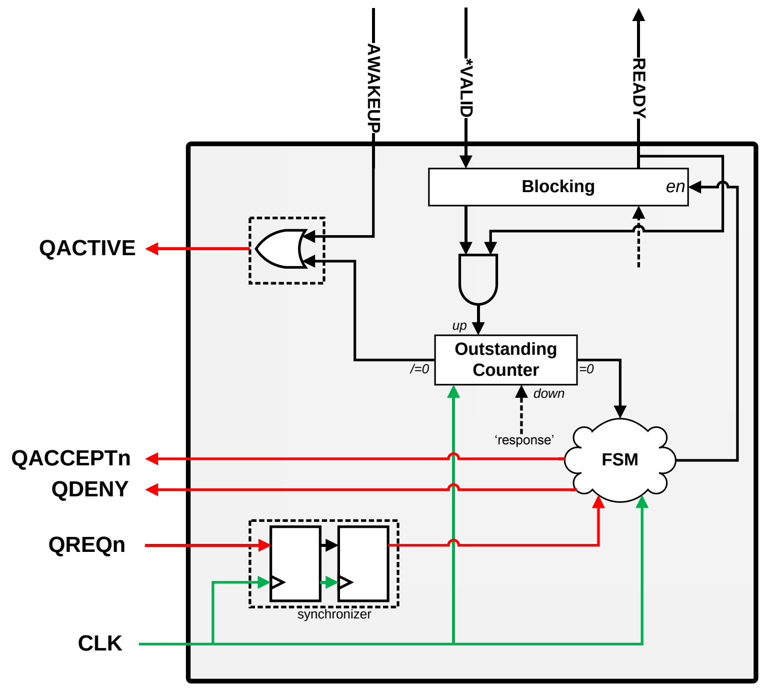


Figure 10-8 – AXI slave interface high level clock gating example

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 slave 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. Component asserts **QACTIVE** as a result of **AWAKEUP** asserted HIGH.
        i. **AWAKEUP** is logically an OR of the **VALID** inputs, but is guaranteed to be driven only from either a register or an OR combination of register sources.
2. Controller de-asserts **QREQn** HIGH and guarantees clock (*Q_EXIT*).
    a. The **READY** outputs are un-blocked and **VALID** inputs are accepted. Transactions are now accepted.
    b. Component de-asserts **QACCEPTn** HIGH (*Q_RUN*).
    c. **QACTIVE** is maintained HIGH by an outstanding counter which remains active until all responses are completed.
3. **Accepted Quiescence**: Controller asserts **QREQn** LOW (*Q_REQUEST*).
    a. This is according to controller policy specific criteria, for example at **QACTIVE** LOW.
    b. The clock remains guaranteed until the component asserts **QACCEPTn** LOW. It can continue to accept transactions if any arrive. In this case the recommend behavior is to deny the quiescence request (see 4) rather than delay acceptance indefinitely.

c. **QACCEPTn** is asserted LOW by the component, with **VALID** and **READY** blocked, once all accepted outstanding transactions are completed and none are pending (*Q_STOPPED*). The clock is no longer guaranteed.

4. **Denied Quiescence:** Controller asserts **QREQn** LOW (*Q_REQUEST*).

   a. This is according to controller policy specific criteria, for example at **QACTIVE** LOW.

   b. If the component has returned to a busy state, for example if a transaction arrived after the controller detected **QACTIVE** LOW, it can assert **QDENY** HIGH (*Q_DENY*). The clock must continue to run and the component remains functional.

   c. On sampling **QDENY** HIGH the controller must de-assert **QREQn** HIGH (*Q_CONTINUE*).

   d. On sampling **QREQn** HIGH the component de-asserts **QDENY** LOW (*Q_RUN*).

10.2.6 Unused Clock Control Q-Channels

The component must operate correctly if the integrator chooses to tie off a clock gating Q-Channel. In this case the component can assume the clock will always be available. It becomes the integrators responsibility to ensure the clock is managed correctly.

In the case where the Q-Channel is unused the **QREQn** input must be tied HIGH.

For components with multiple clock control Q-Channels, if one channel is tied off 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.

10.2.7 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.

When sorted alphabetically, in simulation waves or lists, the Q-Channels will appear by function and next to the clock they control.

Table **10-1** shows the recommend conventions along with an example.

Table 10-1 – Clock control Q-Channel naming convention

Naming Convention	ACLK Example
<*clk*>QACTIVE	ACLKQACTIVE
<*clk*>QREQn	ACLKQREQn
<*clk*>QACCEPTn	ACLKQACCEPTn
<*clk*>QDENY	ACLKQDENY

10.3 Component Power Control

This section describes implementation of power control using Q-Channel and P-Channel low power interfaces.

10.3.1 Low Power Interface Selection

Power Control can use either Q-Channel or P-Channel low power interfaces.

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 the mode is requested.

The advantage of using Q-Channel is the simpler interface which is easier to combine with other power control Q-Channels.

P-Channel is recommended for components that have multiple power modes, for example, a component which can be put into retention and off. P-Channel allows the component to indicate multiple mode requirements to the controller. It also allows the controller to specify the requested modes to the component. It supports transitions between multiple modes.

P-Channel is selected only if Q-Channel functionality is insufficient.

10.3.2 General Power Control Low Power Interface Implementation

To provide power control support a component should have a power control interface for each of its power domains. A power domain can include sub-domains, which might be memory or logic, controlled by the same LPI.

This allows each domain to be controlled separately. LPI can also be combined with power control channels from other components where a power domain is formed from multiple components.

The power control LPI must be separate from any clock control Q-Channels.

Low Power Interface and Relationship to Power Domains

Depending on component structure, the low power interface control logic can be implemented within both the managed power domain itself and also within a parent power domain when a component supports nested parent-child power domain relationships.

Figure 10-9 shows a simple example with three power domains across two components:

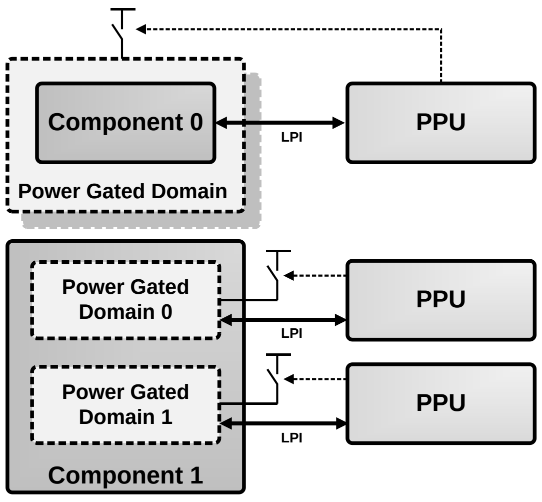


Figure 10-9 – Simple power domain example

In Figure 10-9 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 type of 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 **QACTIVE** or **PACTIVE** according to LPI selection, 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 or equivalent.

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 10-10 shows an example component with a parent-child relationship between a primary power domain and two secondary sub-domains.

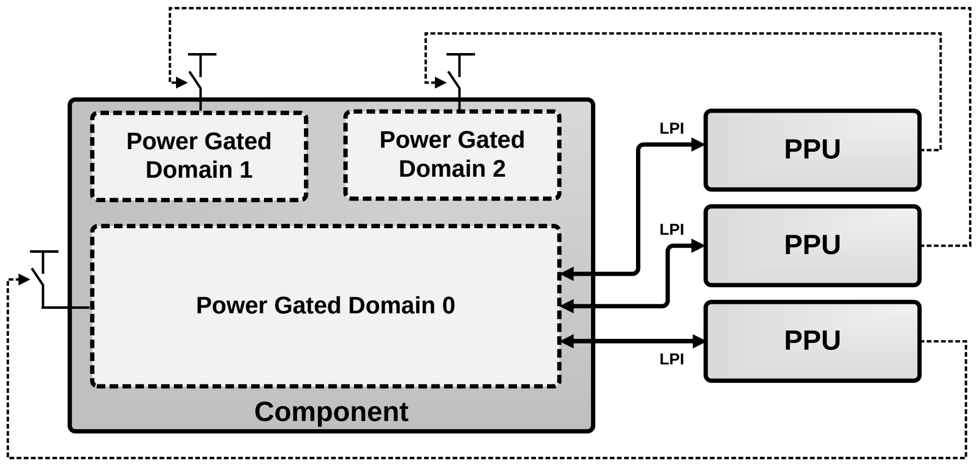


Figure 10-10 – Parent-child power domain example

In Figure 10-10 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 located 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 **QACTIVE** or **PACTIVE** according to LPI selection, to cause a transition to a higher mode. This can, for example, enable implementation of opportunistic power mode control for secondary power domains that might otherwise be complex, or impossible, to implement at system level.

Moreover, the secondary power domains can be transitioned between modes without any logic capability requirement to respond on the interface in either state. This can correspond to both increased management flexibility, between various non-functional power modes, and also the absence of logic capability in that domain in, 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 located in another power domain, for example a parent domain in a component with primary and secondary domains as shown in Figure 10-10.

In particular, the sampling of **PSTATE** at reset exit in P-Channel protocol has specific considerations, as the support for reset exit states 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 *Power Policy Unit* on page 7-9, are given in *ARM® Power Policy Unit Architecture Specification Version 1.0*.

———————————————

10.3.3 Power Mode Availability

The low power interface handshake guarantees a minimum power mode for the device. However, the actual mode might be higher. For example, if the component LPI is in off mode the power switches might still be on.

There can be many reasons for this, but one example is another component within the power domain can deny the mode entry request and therefore the controller will not take the actions to, for example, turn off power switches.

Therefore, a component must not use the power mode conditions, for example power being removed or isolation being enabled, as a mechanism to ensure the component will operate correctly. The correct operation of the component must be guaranteed by logical means.

10.3.4 Power Control Q-Channel Guidelines

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 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.

QACTIVE at Reset

For power control it is strongly recommend that component **QACTIVE** outputs are LOW at reset.

The rationale for this recommendation includes:

- If **QACTIVE** is set HIGH at reset assertion then when reset is applied, prior to power off, **QACTIVE** would then indicate a requirement to exit the *Q_STOPPED* state.
- If **QACTIVE** 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 exit the *Q_STOPPED* state when it is isolated throughout power off and retention modes.

The transition from *Q_EXIT* to *Q_RUN* can be delayed by the component, allowing **QACTIVE** to be set HIGH before **QACCEPTn** is set HIGH. By delaying the transition to *Q_RUN*, the component ensures that the controller cannot make a quiescence request in response to the initial **QACTIVE** LOW condition.

Figure 10-11 shows a reset sequence where **QACTIVE** is set HIGH by the component after reset release.

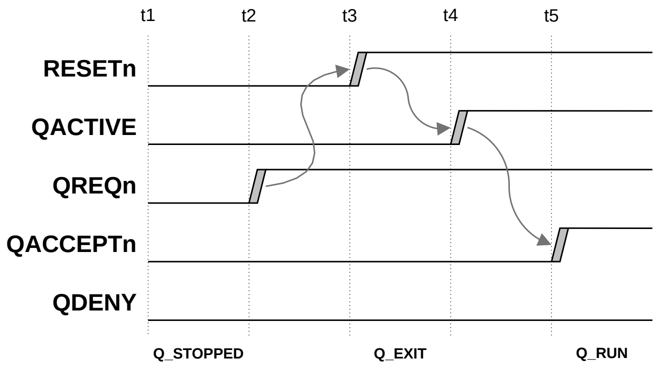


Figure 10-11 – Example of QACTIVE set HIGH after reset release

In Figure 10-11 **QACTIVE** is reset LOW at t1. At t3, the reset exit sequence is into the *Q_EXIT* state. After some clock cycles, at t4, the component asserts **QACTIVE** HIGH sequentially indicating a requirement to progress operations. The component then sets **QACCEPTn** HIGH at t5 and the interface moves to *Q_RUN* state. Only now can the controller make a quiescence request by setting **QREQn** LOW.

——— **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 Version 1.0*.

————————————

QACTIVE Stimulus

QACTIVE stimulus can be from a number of 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 as a consequence 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 as a result of an accepted quiescence request.

Components that contain no software configuration or context 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.

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 can depend on similar activities to that described for retention modes it can take into account, through corresponding register settings or machine state, that any required software actions to manage context are completed. This approach, while not restrictive, can avoid quiescence requests which will be denied.

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 unified LPI can be presented to the system.

QACTIVE contributions for low power mode exit can be formed, according to system level hardware signals, between the component and the PPU. Hardware signaling examples include a level sensitive interrupt, wake request assertion and an indication that a component has a transaction to progress.

Figure 10-12 shows an example of an external power control **QACTIVE** contribution.

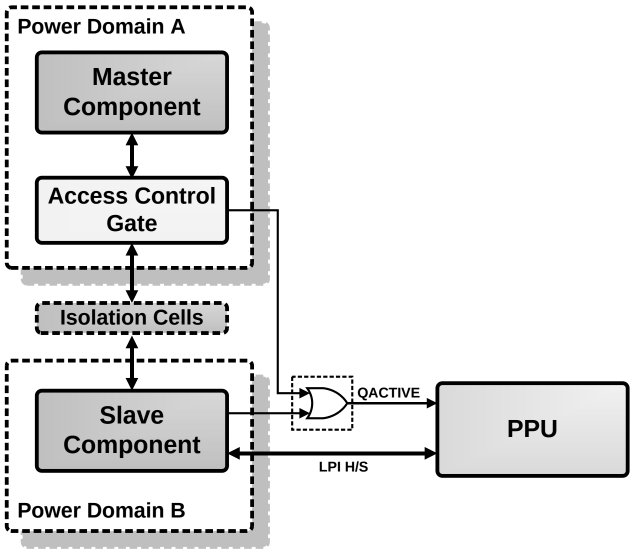


Figure 10-12 – Example of external QACTIVE contribution

In Figure 10-12 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. When a quiescence request is made the component should respond in a timely manner.

In general a component should only wait to accept a power control Q-Channel quiescence request if it knows that a response can be given in a timely fashion. If a component cannot bound the time required until it will accept it should deny the request.

However, 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 which when requested to enter a quiescent state, prior to power off, might first block incoming transactions, and 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 transition 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 as a result 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 external interfaces are managed as described in *Interface Management* on page 10-6.

All required actions must be completed before an accept response is sent. Once the accept response is sent the capabilities of the current 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.

However, the controller cannot make another quiescence request until the response is received and the interface is in *Q_RUN* state. 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 should operate correctly if the integrator chooses to tie off a power control Q-Channel. In this case the component can assume the highest power mode will always be available. It becomes the integrators responsibility to ensure any power control 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 tied off it should not limit the use of other power control channels. This allows the integrator to choose which domains in the design 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 should be able to tie off the sub-domain LPI, while still using the top level power domain LPI to provide overall component power control.

10.3.5 P-Channel Guidelines

A P-Channel can switch between multiple functional and low power modes. These modes and allowed transitions are specified by the component.

PACTIVE Behaviour

PACTIVE at Reset

For power control it is strongly recommended that component **PACTIVE** outputs are LOW at reset.

The rationale for this recommendation 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.
- 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.

If the component is required to run after reset de-assertion for initialization or other purposes then a **PACTIVE** bit can be set HIGH after reset de-assertion by the component logic in following clock cycles.

Depending on the initialization sequence used by the controller the component can either simply delay completion of the P-Channel handshake or deny any request to a lower power mode until the **PACTIVE** bit required is set HIGH.

Figure 10-13 shows an example of a **PACTIVE** bit set HIGH after reset release.

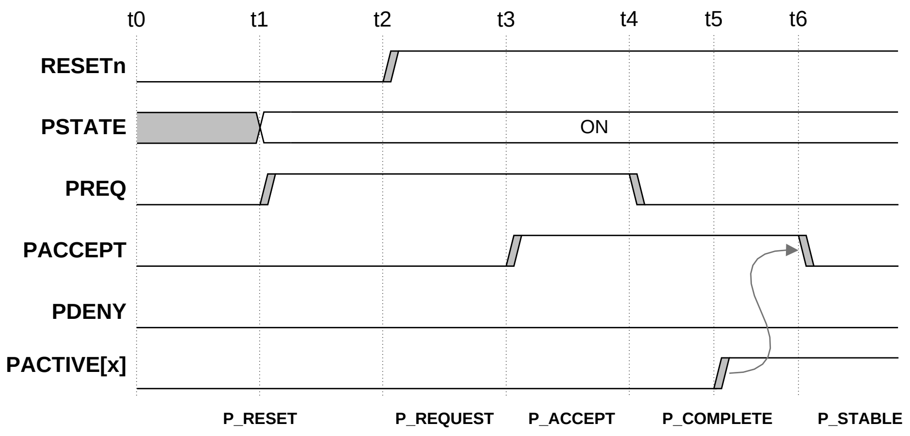


Figure 10-13 – Example of PACTIVE set HIGH after reset release

In Figure 10-13 all **PACTIVE** bits are reset LOW at t0 and **PREQ** is HIGH at exit from reset release at t2. The component drives **PACCEPT** HIGH at t3 and the interface is in *P_ACCEPT* state. The controller responds with **PREQ** LOW at t4 and the interface is in *P_COMPLETE* state. The component now sets at least one **PACTIVE** bit HIGH, at t5, before driving **PACCEPT** LOW at t6 to place the interface into *P_STABLE* state. Only now can the controller make a further request by setting **PREQ** HIGH.

——— **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 Version 1.0*.

———————————————

PACTIVE Stimulus

The stimulus for **PACTIVE** bits can come from a number of 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 similar to those detailed for a power control **QACTIVE** as detailed in *QACTIVE Contribution from Internal State* on page 10-9.

Since multiple power modes are supported, as well transitions between them without transitions to a common operating mode, the contributions for each **PACTIVE** must be considered individually.

In particular, it is important to consider the priority of each mode at a given time to enable the minimum power mode requirement to be indicated. For example, if the conditions for entering both a retention mode and an off mode are met the retention mode **PACTIVE** must be LOW to allow a request for the component to enter the off mode.

PACTIVE Contribution for Low Power Mode Exit

The considerations for **PACTIVE** bit contributions for low power mode exit from external signals are similar to those detailed for a power control **QACTIVE** as detailed in *QACTIVE Contribution for Low Power Mode Exit* on page 10-17.

However, as for internal state contributions to **PACTIVE** bits, the multiple power modes that can be supported, using different **PACTIVE** bits as activity requirements for each, means each **PACTIVE** can have differing considerations. In particular, it might be possible to use component input signals as contributions to **PACTIVE** outputs in more cases than in Q-Channel power control applications since a number of modes offer an operating capability.

PACTIVE Policy

PACTIVE output bits are used to indicate component requirements to the power controller, with each bit representing a different requirement. A **PACTIVE** bit set HIGH indicates that a transition to this mode is required to allow operations to progress. A **PACTIVE** bit being LOW is a hint that the capabilities of the mode are no longer required.

While the arrangement of **PACTIVE** bits is not restricted by the *Low Power Interface Specification, ARM Q-Channel and P-Channel Interfaces* a typical arrangement is described. In this typical arrangement the most significant **PACTIVE** bit that is set HIGH indicates the minimum power mode required by the component.

This indicates:

- A requirement that this mode is required as a minimum.
- A hint that the component will accept a request to enter this or any higher mode.

While that arrangement is sufficient in many circumstances, an exact ordering of power mode capability, low to high, is not always possible since the capabilities of modes might not be simply incremental.

For example one mode might support logic operation, but with memories off, while another might be a retention mode which retains logic and memory context. While the first mode clearly offers more functionality the second mode retains context lost in the first mode.

Therefore, if the logic on, memory off mode is treated simply as a higher mode than the full retention mode, then in a transition, according to **PACTIVE** bit states only, from the full retention mode to the logic on, memory off mode some context would be lost. Conversely, a transition according to a reversed arrangement of **PACTIVE** priority renders the logic inoperable.

Consequently, the supported policies and the legal transitions between them can require a higher level of coordination between the component and the power controller than a simple high to low

PACTIVE representation can provide. Correspondingly, the *Low Power Interface Specification, ARM Q-Channel and P-Channel Interface* requires a component to specify its supported transitions.

Recommended PSTATE and PACTIVE Usage

The mapping of **PSTATE** and **PACTIVE** to power modes and to each other is not restricted by the *Low Power Interface Specification, ARM Q-Channel and P-Channel Interfaces.* However, the *ARM® Power Policy Unit Architecture Specification Version 1.0* defines a usage of these signals for the modes it supports as outlined in *PPU Policy Support* on page 7-11.

Use of these mappings by components eases the adoption of the PPU architecture and minimizes any integration logic required to provide a suitable mapping. Table 10-2 summarizes this recommended usage. See *ARM® Power Policy Unit Architecture Specification Version 1.0* for further information.

Table 10-2 – 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 would only implement some of the modes and it is recommended to declare the unused **PSTATE** values as RESERVED and to tie LOW the corresponding **PACTIVE** outputs, rather than omit them.

P-Channel Handshake

P-Channel Handshake at Reset

From a component perspective a P-Channel will always exit reset in the *P_STABLE* state. Component **PACCEPT** and **PDENY** outputs must be reset LOW.

Any resynchronization in the component on the **PREQ** input must be reset LOW.

The component must assume the state of the **PREQ** input is LOW at reset exit.

A controller can set **PREQ** LOW or HIGH at reset exit. When **PREQ** is set HIGH this will not be sampled by a component, as a transition to *P_REQUEST*, until its reset is released.

The *Low Power Interface Specification, ARM Q-Channel and P-Channel Interfaces* requires a component to support three initialization behaviors at exit from reset. In addition to two handshake sequence behaviors there is a requirement to sample **PSTATE** within a specified period in the absence of any handshake. This sequence is also applicable to unused P-Channels. Implementation requirements for this support are described in *Capturing PSTATE* on page 10-4.

P-Channel Requests

A request on a P-Channel can be made to a component at any time regardless of the state of the components **PACTIVE** signals. When a request is made the component should respond in a timely manner.

In general a component should only wait to accept a P-Channel request if it knows that a response can be given within a known time period. If a component cannot bound the time required until it will accept then it should deny the request.

However, a component is allowed to attempt any actions required to accept the request and if unsuccessful then send a denial response. While in some case this can take significant time it can be a requirement of the component to only complete some transitions with an accepted response regardless of the delay.

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.

Figure 10-14 shows an example of this usage.

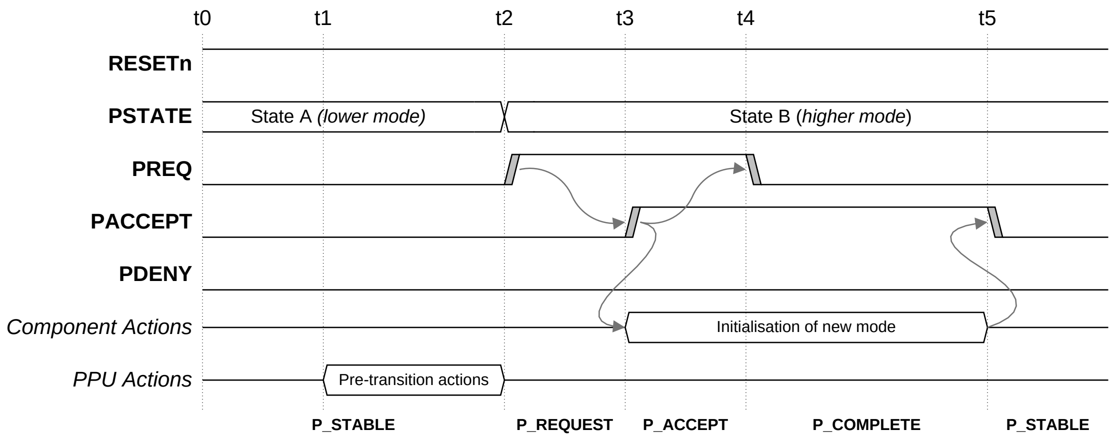


Figure 10-14 – Component delayed completion on transition to higher mode

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 provide a denial response.

Figure 10-15 shows an example of this usage.

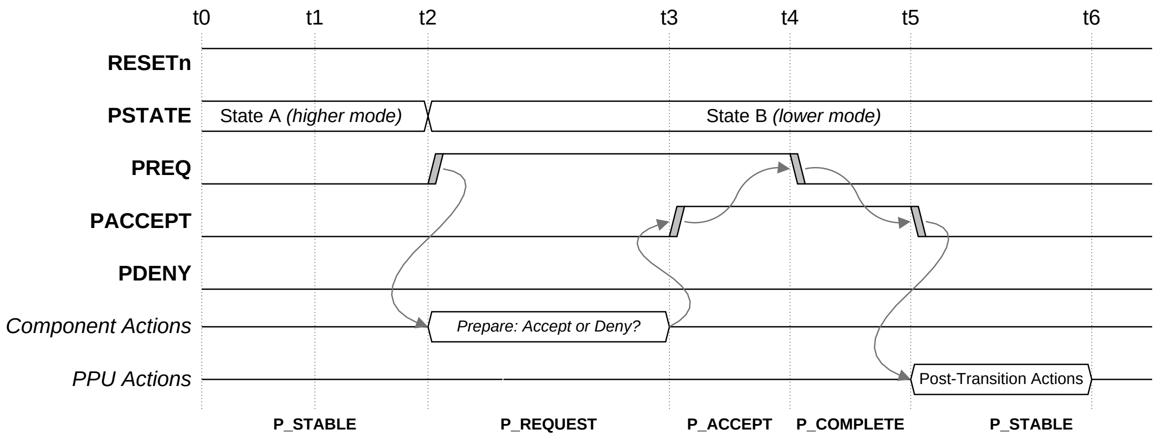


Figure 10-15 – Component delayed acceptance on transition to lower mode

Denied Requests

The *Low Power Interface Specification, ARM Q-Channel and P-Channel Interface* requires a component to specify which supported transitions might be conditionally denied and requires that a controller must not present unsupported **PSTATE** values to a component.

In addition to denying requests when the internal conditions are inappropriate, components are specifically 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.

Actions required to enter a low power mode

A component must ensure that when it accepts a request there will be no functional failure as a result 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 external interfaces are managed as described in *Interface Management* on page 10-6.

All required actions must be completed before the accept response is sent. Once this response is given the capabilities of the current mode are not guaranteed.

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 **PSTATE** tied to this initialization mode value, which will be sampled by the component at reset exit. **PREQ** must be tied LOW.

The modes which 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 should not limit the use of other power control channels. This allows the integrator to choose which domains in the design 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 should be able to tie off the sub-domain LPI while still using the top level power domain LPI to provide overall component power control.

10.3.6 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.

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 10-3 shows the recommend conventions for Q-Channel power control interfaces along with an example for a component with a single power control channel.

Table 10-3 – Q-Channel Naming Convention

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.

Table 10-4 shows the recommend conventions for P-Channel power control interfaces along with an example for a component with multiple power control channels.

Table 10-4 – P-Channel Naming Convention

Naming Convention	Example
<*domain*>PACTIVE	**CPUPACTIVE**
<*domain*>PREQ	**CPUPREQ**
<*domain*>PACCEPT	**CPUPACCEPT**
<*domain*>PDENY	**CPUPDENY**

11 Glossary

ACP	Accelerator Coherency Port. This is slave 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.
Always On Domain	A voltage or power domain which is always on compared to all other power domains on the SoC.
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	Dual 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.
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.
PPU	Power Policy Unit.
Relative Always On Domain	A voltage or power domain which is always on relative to another domain. It may be powered off but only once the specified domain is off and must be on before the specified domain is powered on..
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.
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.

12 Revisions

This appendix describes the technical changes between released issues of this specification.

Table 12-1 Issue A

Change	Location
First Release	-

Table 12-2 Differences between Issue A and Issue B

Change	Location
Correct **ACINACTS** to **AINACTS.**	*Cluster Dormant Mode* on page *9-11.*
Remove link activation from wake-up request recommendation. Only RXSACTIVE is required.	*AMBA5 CHI* on page *10-7.*
Clarify clock controller reset requirements.	*Clock Controller Reset* on page *8-9.*
Clarify which Mali products are described in this section. Also clarify that a merged power domain implementation of cores and core-groups is possible.	*Graphics Processor* on page *5-10.*
Clarify SCP reconciliation of state constraints as a coordination activity, not a delegation activity.	*Coordination by System Control Processor* on page *6-9.*