**Freescale Semiconductor**
White Paper

Document Number: QORIQHSRPWP
Rev. 0, 05/2013

# Hardware Support for Robust Partitioning in Freescale QorIQ Multicore SoCs (P4080 and derivatives)

*by* *Eric Bost*
*Freescale Semiconductor, Inc.*

## About this Document

This document describes the hardware-level features that have been integrated into the P4080 to manage partitioned systems. This document is intended for embedded systems architects and designers who require robust partitioning capabilities and full control of hardware resources to achieve real-time goals through their further integration of hardware and software functionalities.

**Contents**

# Before you begin

Please see the following list of abbreviations and definitions:

| Abbreviation | Definition |
|---|---|
| AMP | Asymmetric Multi-Processing |
| ATMU | Address Translation Mapping Unit |
| BMan | Buffer Manager |
| CCSR | Configuration Control and Status Registers |
| CPC | CoreNet Platform Cache |
| DMA | Direct Memory Access |
| DPAA | Data-Path Acceleration Architecture |
| DSA | Direct Storage Access |
| EA | Effective Address |
| eLBC | Local Bus Controller |
| FMan | Frame Manager |
| GS | Guest State |
| ISBC | Internal Secure Boot Code |
| LAW | Local Access Window |
| LAWR | Local Access Window Register |
| LIOD | Logical I/O Device |
| LIODN | Logical I/O Device Number |
| LPID | Logical Partition ID |
| LSU | Load/Store Unit |
| MESI | Modified Exclusive Shared Invalid |
| MMU | Memory Management Unit |
| MPIC | Multicore Programmable Interrupt Controller |
| OCeaN or OCN | On-Chip Network |
| PAMU | Peripheral Access Management Unit |
| PAACT | Peripheral Access Authorization and Control Table |
| PAACE | Peripheral Access Authorization and Control Entry |
| PID | Process ID |
| PPAACT | Primary Peripheral Access Authorization and Control Table |
| PPAACE | Primary Peripheral Access Authorization and Control Entry |
| QMan | Queue Manager |
| RA | Real Address |

| Abbreviation | Definition |
|---|---|
| RTIC | Run Time Integrity Checker |
| SMP | Symmetric Multi-Processing |
| SoC | System-on-Chip |
| SPAACT | Secondary Peripheral Access Authorization and Control Table |
| SPAACE | Secondary Peripheral Access Authorization and Control Entry |
| sRIO | Serial RapidIO |
| TLB | Translation Lookaside Buffer |
| VA | Virtual Address |

# Introduction

Partitioning is becoming a common requirement in various application areas. In high-end, real-time embedded areas, such as Avionics and Transportation, it is often referred to as “robust partitioning” to account for the high level of isolation and protection requested between partitions in a static configuration approach with relatively low virtualization usage.

In the domains of data-centers and networking nodes, partitioning is implemented in a more dynamic way to make the best usage of the available resources for variable (in time) application needs with extended virtualization usage for sharing underlying physical resources.

In this document, “P4080 multicore architecture” refers to the architectural features that have been introduced by Freescale in the P4080. All the mechanisms described in this document also apply to the other multicore platforms (“platform” here stands for “family”) and derivatives introduced so far by Freescale as QorIQ P-series, including P4040, P5020, P5010, P5040, P3041, P2040, and P2041.

As a general stand-point, the overall model and architectural features presented in this document are present in the new generation (QorIQ T-series) of Freescale multicore System-on-Chip (SoC) architectures.

# 1 Overall partitioning model

## 1.1 P4080 architecture - memory hierarchy and interconnect

Figure 1 illustrates the P4080 SoC architecture.

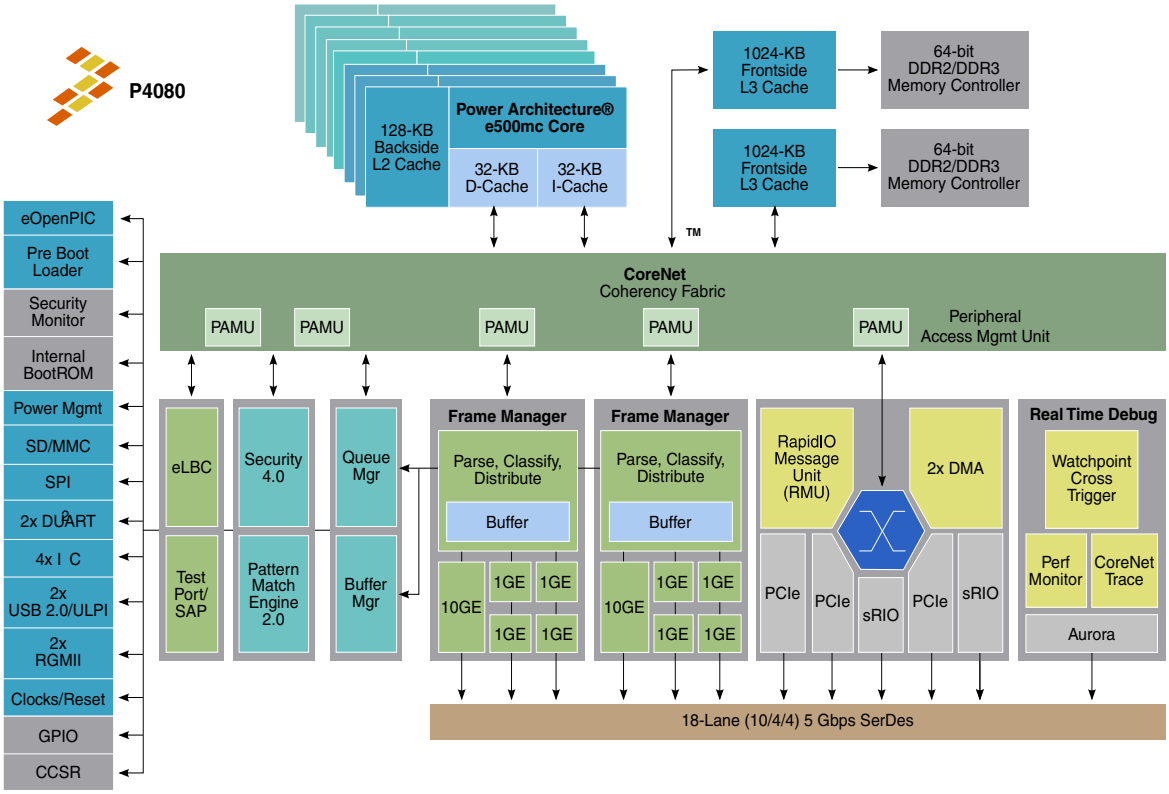


**Figure 1. P4080 Block Diagram**

The P4080 integrates eight Power Architecture® e500mc cores. Each core has its own private level-1 (L1) and level-2 (L2) caches. Being backside, L1 and L2 caches can provide deterministic latencies, which helps minimize data-to-core latency and limits the number of core/software processing interactions with other simultaneous activities in the SoC (such as peripheral DMAs).

In addition to the private L1 and L2 caches, the SoC provides a large (2MB) level-3 frontside cache, called CoreNet Platform Cache (CPC), that can act as a cache to the DDR system memory or internal lower-latency fixed-address SRAM.

The interconnection between the eight cores and the other SoC modules, including shared cache and system memory, peripheral units, and hardware accelerators, is performed at the physical level through CoreNet coherent fabric. CoreNet fabric is a high-bandwidth switch interconnect that allows several concurrent transactions to flow in parallel between the various hardware initiators and targets within the SoC.

Each core is attached to CoreNet fabric through a dedicated point-to-point connection, which allows several cores to be fed by data coming from a shared cache or external memory. Compared to previous bus-based architecture, the Power Architecture-based e500mc and e5500 core system interface logic has been completely redesigned to map to the CoreNet fabric interconnect.

## 1.2 P4080 partitioning

In the P4080, “partitioning” initially refers to the logical grouping of hardware resources, such as CPU/cores, portions of memory, I/Os, and accelerators, in support of various application requirements. A given resource can be used either privately (by a single partition) or can be shared (by multiple partitions).

The most common use for partitioning P4080 hardware resources is to consolidate several independent software environments that were previously implemented on different hardware subsystems. Each partition (that is, set of hardware resources) typically hosts a software environment made of an OS and its applications. For that reason, a partition extends to this software environment, making the partition a combination of hardware and software resources.

A software model with several OSes running concurrently is referred to as Asymmetric Multi-Processing (AMP), as opposed to Symmetric Multi-Processing (SMP), where a single OS instance manages all the resources.

## 1.3 What is meant by “robust” partitioning

Figures 2 and 3 illustrate the robust (secure) partitioning models that can be supported by the P4080 multicore architecture. In both cases, each logical partition has access to only a portion of the hardware resources.

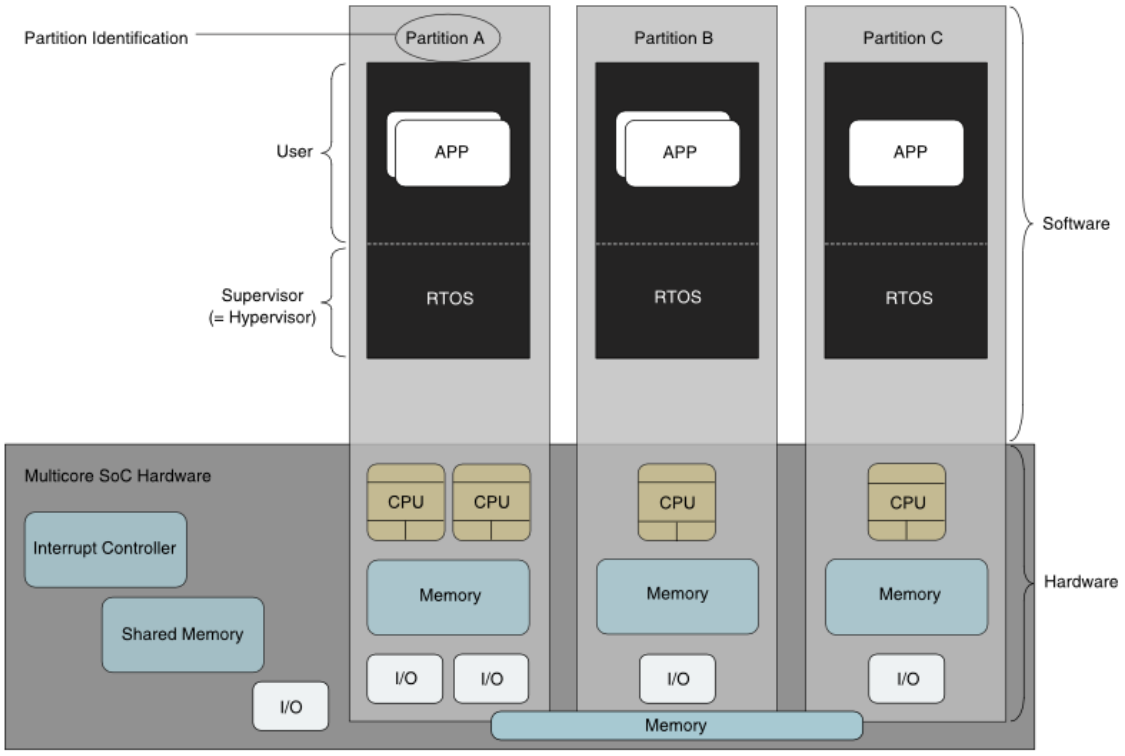


**Figure 2. Two-level Partitioning Model**

Figure 2 illustrates a cooperative model where all partitions are managed with an equal level of trustability. In this case, the system code running on each core can be seen as part of a global distributed executive. In

this model, where only two privilege levels are implemented, system functions run at the hypervisor level of the e500mc core, which is equivalent to the supervisor level in the traditional two-level user/supervisor model.

Other variants derived from Figure 2, based on two hierarchical levels, are possible. For example, a single kernel instance running at the hypervisor level and distributed among various cores.

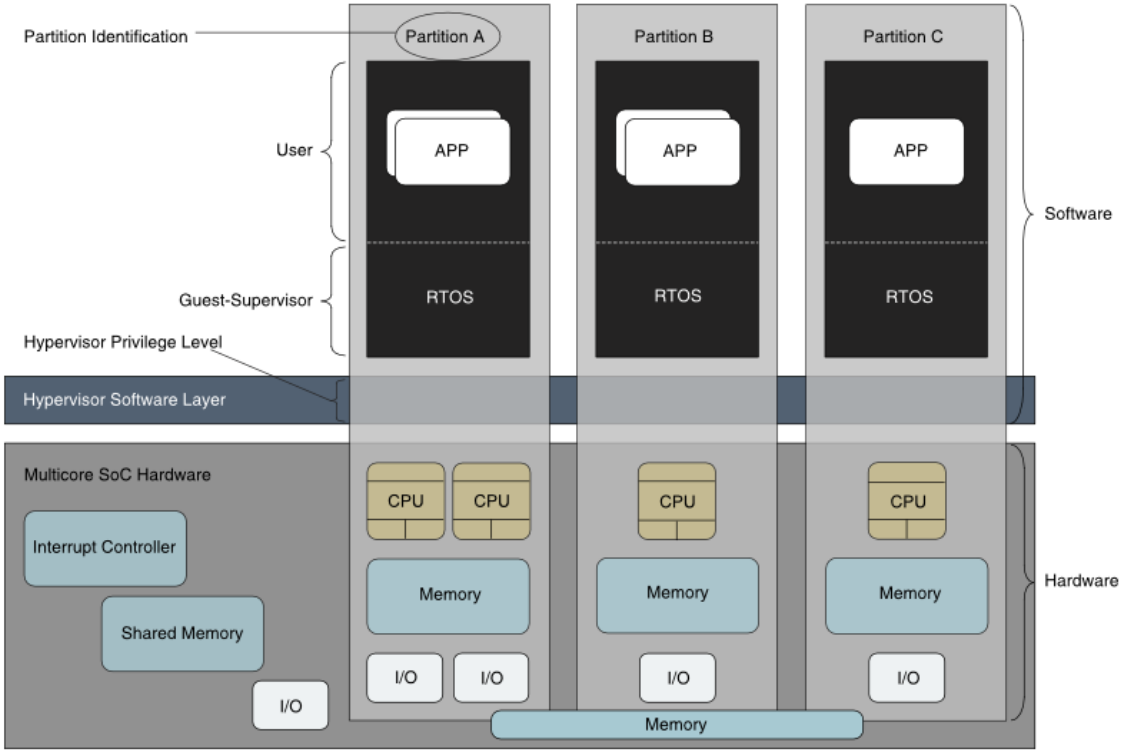


**Figure 3. Three-Level Partitioning Model**

Figure 3 shows the supervised AMP model where different OSes with non-equal levels of trustability must co-exist. In this case, as in the traditional user/supervisor model where an OS never trusts its underlying applications and restricts direct access to the system resources, the non-trusted partitions must be controlled to not go beyond the usage of resources that are strictly allocated to this partition. To account for this level of protection, the addition of a third privilege level, called “hypervisor,” is provided.

In these partitioning use cases, each core belongs to only one partition. There is no architectural restriction in P4080 that prevents several partitions from running on one core (as already implemented on single-core processors). In the context of this document, however, it is assumed that embedded critical systems will tend to allocate each core to a single partition.

For the purpose of describing the wide range of available mechanisms for partitioning a system, the three-level model in Figure 3 is referenced throughout this document.

**NOTE**

This does not imply that any partitioned system requires three privilege levels.

In addition to protecting private resources, robust partitioning also accounts for the need to share some resources, such as memory regions, high-speed I/O interfaces, or low-speed console controls and debug interfaces. To prevent uncontrolled accesses from each partition to the shared resources, it may be necessary to virtualize such resources. In the supervised AMP model, this can be achieved with the use of a hypervisor, as introduced in the next section.

## 1.4 Temporal partitioning

In critical, real-time application, robust partitioning may encompass both spatial and temporal aspects. Temporal partitioning is more challenging in the context of a multicore SoC because of the high amount of physical interaction that happens inside the SoC.

Section 4 addresses this concern and suggests guidelines for configuring robust partitioning or restricting usage domain in order to meet time determinism requirements.

## 1.5 P4080 hardware-based features for partitioning and virtualization

A complete multicore partitioning infrastructure is designed into the QorIQ P2/P3/P4/P5 multicore devices. This infrastructure is composed of many features, spread over the different blocks in the P4080 SoC.

Robust partitioning is primarily supported by three key hardware mechanisms:

- Memory Management Unit (MMU)
- Peripheral Access Management Unit (PAMU)
- User/Supervisor/Hypervisor (a 3-level hierarchical model)

MMU is the hardware mechanism that controls all the address-based accesses initiated by the cores. PAMU is a similar mechanism that controls all address-based accesses initiated by the DMA-capable peripherals.

MMU and PAMU are the two key security features of a partitioned system. In order to enforce robustness, only trusted software must have the permission to modify MMU and PAMU content; namely, either a hypervisor software-layer or an executive kernel running at the hypervisor-privilege level.

By reporting and enforcing the configuration of MMU and PAMU only at the hypervisor level, the hypervisor software or highly trusted kernel can fully manage and control what system resources are accessible from each partition.Therefore, each partition can be isolated from other partitions' unexpected behavior (such as software bugs and viruses). For a resource that must be shared between partitions, such as a network interface or a hardware accelerator, this platform provides several means for virtualizing access to the shared resource:

- Calls-to-hypervisor
- Violation traps
- Queue-based producer/consumer facility

In Figure 3, the three-level model, an OS running under a hypervisor is called a Guest-OS. In this model, the hypervisor itself is neither an OS nor a kernel; it is a set of functions or services running at the highest privilege level that:

1. Initializes resources
2. Starts each partition's Guest-OS
3. Is invoked either implicitly when attempting to perform an unauthorized action (such as a trap-initiated virtualization) or explicitly (such as a Guest-OS calling for an HV service for virtualization).

The following variations to this model are possible:

- Aggregating all cores to a single OS with no hypervisor (full-SMP approach).
- Running a complete OS kernel at the hypervisor level and controlling partitions with or without a Guest-OS (rather than running an embedded hypervisor).
- Fully bare-board approach with no hypervisor and cooperative partitioning.

In addition to the three key mechanisms, many extensions have been integrated into the P2/P4/P3/P5 to efficiently manage the various aspects of partitioning dealing with:

- Protection
- Authorization
- Virtualization of Guest-OS
- Interrupt management
- Inter-core and inter-partition communication and synchronization
- Coherency of memory hierarchy

## 1.6 System partitioning example

Figure 4 illustrates an example of a simplified partitioned system.

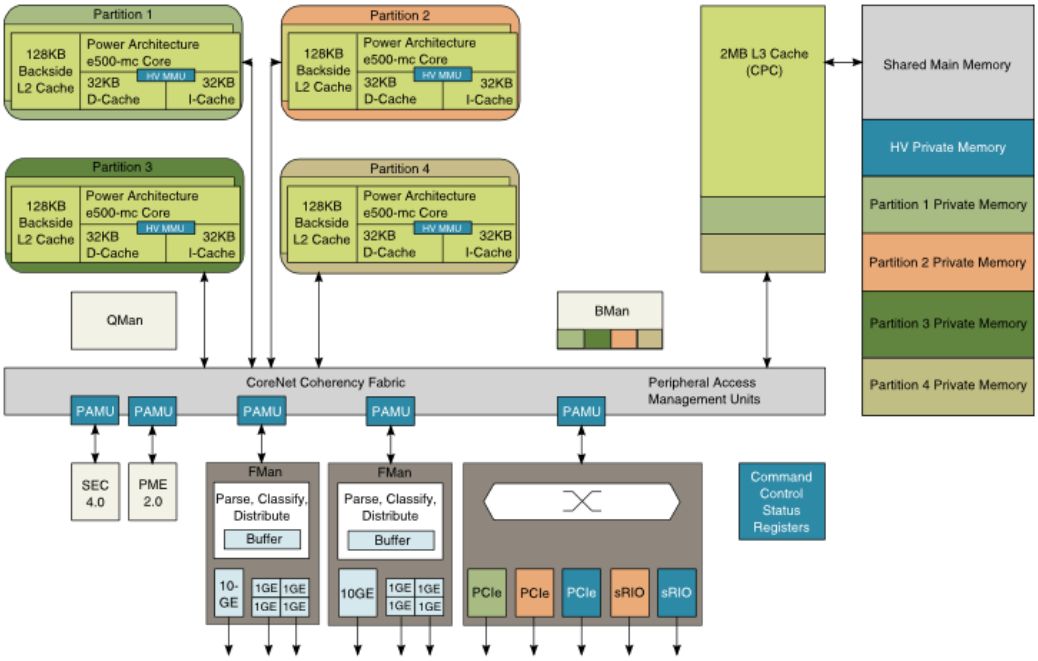


**Figure 4. Example of a Partitioned System**

In this model, there are four distinct partitions, each running on two cores. The main memory is divided into several physical regions:

- Private
- Shared between partitions; accessible at user level
- Shared among partitions; restricted to hypervisor level

This mapping is enforced by the cores' MMUs accessible only at the hypervisor level. System peripherals (PCIe and sRIO) in this example are not shared -- each is allocated to a partition usage. As such, the hypervisor is able to restrict their DMA-accessible memory range to some part of the memory region assigned to the partition through the MMU.

The shared internal memory (CPC) is partially partitioned, which provides two partition-specific sub-ranges.

**NOTE**

This CPC allocation can be done per-way. Each way is configured to work either as a cache or as a fixed-address sRAM.

## 1.7 Hypervisors

Several hypervisor technologies are proposed for the P4080 to address different purposes.

RTOS suppliers, such as GreenHills, SysGo and WindRiver, have developed their own hypervisor technology with particular focus on safety and robust partitioning.

Freescale's embedded hypervisor (so-called "Topaz") is suitable as reference or evaluation for static and robust partitioning with direct hardware access, minimal resource sharing, and limited virtualization support.

Linux's kernel-based KVM hypervisor technology provides extended virtualization and shared resource utilization with the capability to host multiple virtual machines not strictly bound to the available core(s).

Further reading on embedded hypervisor technologies:
- "Freescale's Embedded Hypervisor for QorIQ P4 Series Communications Platform" - White paper (Freescale)
- "Hardware and Software Assists in Virtualization" - White paper (Freescale)
- Ap.Notes & white papers available from RTOS suppliers

# 2 e500mc core-level mechanisms

The e500mc core integrates many enhancements to support the overall partitioning and virtualization model. These enhancements can be classified into the following categories:

- MMU
- Interrupt handling
- Inter-core/inter-partition communication (doorbells and external Ld/St)
- Miscellaneous

Detailed descriptions and considerations of these core functions can be found in the following Reference Manuals:
- e500mc and e5500 Core Reference Manuals
- EREF 2.0: A Programmer's Reference Manual for Freescale Power Architecture® Processors

## 2.1 MMU

The e500mc core MMU is implemented as a 2-level Translation Lookaside Buffer (TLB) structure. MMU is always active and, as in all common CPU architectures, its primary purpose is to map the Effective Addresses (those manipulated by application software) to the Real Addresses (physical addresses in the local SoC mapping).

In addition to being a translation stage, the MMU is a protection barrier that filters each memory-mapped access depending on the type of access and privilege level. These mapping and permission checks are performed on page granularity, each page corresponding to an entry in the TLB structure.

The e500 TLB allows definition of concurrent fixed-size 4KB pages and variable-size pages (from 4KB to 4GB). A TLB entry contains the effective-to-real address translation and all the page attributes related to access restrictions, cachability, coherency, and so on.

See Core Ref.Manual for detailed MMU.

Figure 5 illustrates the address mapping performed through the e500mc MMU.

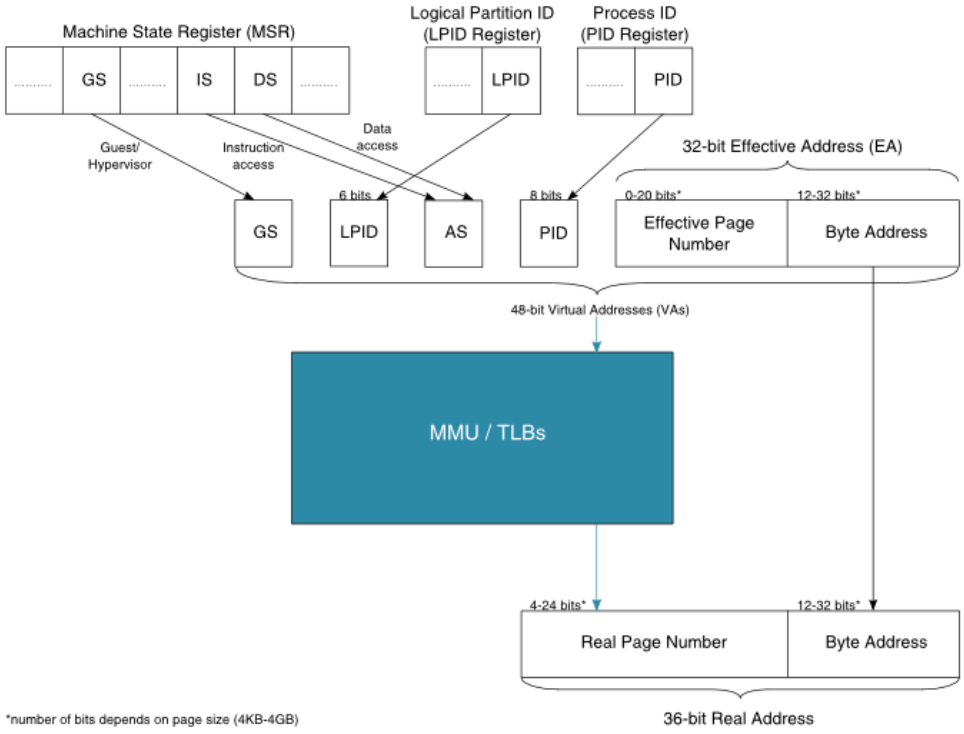


**Figure 5. e500mc MMU Address Mapping**

The 32-bit Effective Address (EA) is appended by context attributes to form a 48-bit virtual address, before searching this EA into the TLB.

Context attributes are:

- PID: Process ID (the current process in the context of which this core is currently running as managed by the OS)
- AS: Address Space (generally managed by OS to extend and differentiate EA spaces to the user and the system)
- LPID: Logical Partition ID (the partition in the context of which this core is currently running)
- GS: Guest State (in a partitioned system, indicates if the core is currently running in hypervisor or Guest-OS (including system and user) context

PID (8-bit) and LPID (6-bit) are maintained in two specific registers: PIDR (supervisor accessible) and LPIDR (hypervisor-only accessible). AS and GS bits are maintained in the core Machine State Register (MSR).

The LPIDR is new in the e500mc core. The LPID field and GS bit are new in the e500mc core and are specifically intended to support the partitioning model.

### 2.1.1 MMU and logical partitioning considerations

Each core in a multicore SoC has its own MMU and related TLB. The system software running on each core (OS and/or hypervisor) must maintain consistency in all TLBs. There is no strict enforcement for an OS running on several cores where all core MMUs share partial TLB content. To assist the system software, there are specific Power ISA TLB management instructions (such as *tlbivax, tlbsync*) that can propagate TLB updates for cores that belong to the same coherency domain. The underlying implementation of *tlbivax* partition-aware TLB invalidate instruction implies CoreNet-level broadcast and conditional execution at each core based on the virtual address context fields, including the LPID.

The LPIDR identifies the logical partition in execution. The LPIDR is used by the hypervisor to manage the address spaces between logical partitions in a manner similar to how an OS uses PIDRs to manage address spaces between processes. The LPIDR is part of the virtual address, much like the PIDR.

During the MMU validation and translation process for each new EA, the e500mc core searches for a TLB entry matching the current EA. If an entry matches, the core performs several validation checks taking into account the type of access (R/W/X), the privilege level, and the context attributes. In the e500mc core, each TLB entry includes a new Translation Logical Partition ID (TLPID) field, which is compared to the current LPID hold in LPIDR. A TLPID value of "0" identifies a global entry.

In the e500mc core, each TLB entry includes a Translation Guest State (TGS) bit. Code that runs on GS can only access pages identified as guest pages. In addition, supervisor/user filtering is performed, just as on previous cores, with consideration given to the other TLB entry attributes.

Embedded static partitioning commonly associates each core to a single partition, potentially with multiple cores per partition. In such a case, for each core, the LPIDR is initialized once and not altered. In more dynamically partitioned systems, a core state may switch from the context of a given partition to the context of another partition over time. In that case, LPIDR content may be updated. That type of system behavior and capability is application, system-dependent, and requires appropriate support from the underlying hypervisor. As defined by ISA-2.06, the e500mc core can support both static and dynamic partitioning.

With the e500mc core, only code that runs at the hypervisor state can modify the content of the TLB. That protection is key to guaranteeing the integrity of the partitioned system. If the specific instruction *tlbwe* is executed in GS, a trap interrupt allows the hypervisor software to check the validity of the attempted modification before executing the TLB write on the demand of the Guest. This allows the complete virtualization of the MMU services implemented by a Guest-OS, as illustrated in Figure 7.

The role of a hypervisor is key in initializing the system resources (like the MMUs) and allocating these resources to the various partitions. During operation, a hypervisor ensures overall integrity and provides some virtualization services.

## 2.2 Interrupt handling

In a partitioned system, it is necessary to control how interrupts are handled, based on how critical they are and their potential impact on the whole system. The P4080, like other SoCs, has two levels of interrupt control:

- Core level

- SoC level (illustrated in Figure 6)

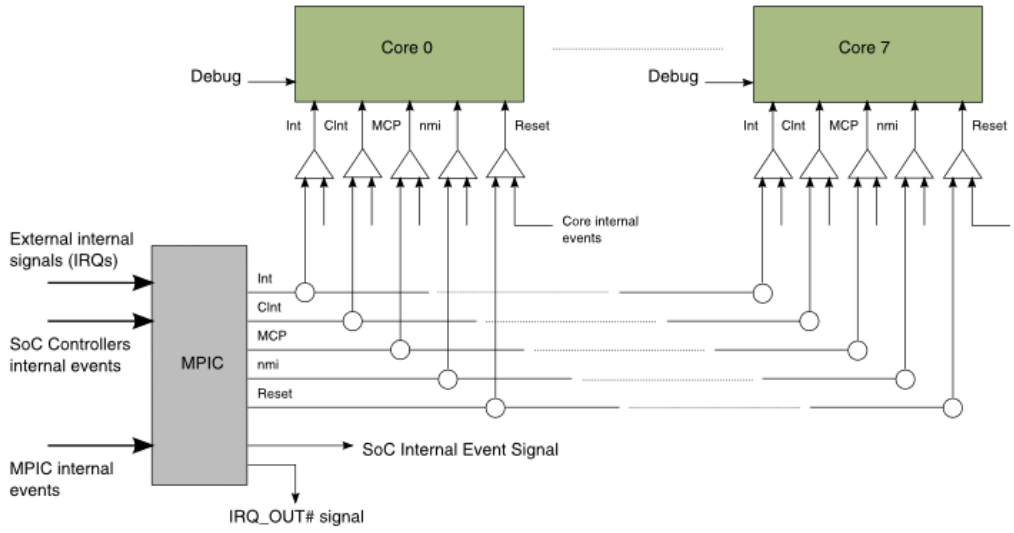


**Figure 6. Interrupt Control - Core Level and SoC Level**

At the SoC level, all exceptions go through the Multicore Programmable Interrupt Controller (MPIC), where they are prioritized, arbitrated, and redirected to the cores. The MPIC multiplexes several types of exceptions:

- All SoC block exceptions related to I/Os, accelerators, memory controllers, and so on
- Some internal signaling facilities implemented by the MPIC, including inter-core messaging, global timers, or software-controllable Core Reset and NMI exceptions
- External interrupts (those triggered from external IRQ# pins)

By configuration, each exception managed by the MPIC can be routed to any of the eight cores and can be associated to a particular core-level priority class:

- Normal - only routed to a single core
- Critical - routed to one or more cores
- Machine-check - routed to one or more cores

More internal information on the MPIC can be found in the P4080 Reference Manual.

At the core level, exceptions are either due to internal causes (those related to instruction execution flow, MMU, Timebase, or debug) or due to external causes (those routed by the MPIC). Exceptions are prioritized and classified as:

- Normal
- Critical
- Debug
- Machine-check

Each classification has its own save-/restore-state registers to ensure nesting and fast pre-emption.

More details on the interrupt management can be found in the e500mc Reference Manual.

## 2.3 Interrupt features and enhancements

This section lists the specific interrupt features and enhancements that have been incorporated into the P4080.

A key performance point between hardware and Guest-OS instances are exceptions together with interrupts from internal and external sources. The hypervisor must ensure interrupts are routed to the appropriate destination. Sometimes, interrupts are taken by the hypervisor during resource virtualization. At other times, interrupts are redirected to a Guest-OS in a logical partition.

At the e500mc core level, if the general rule is that all sensitive exceptions (critical, debug, and machine-check) are systematically handled at hypervisor level, several non-critical exceptions can be configured to be handled at the Guest-supervisor level. This capability is configured through the EPCR hypervisor register. This applies to the following exceptions:

- Normal external exceptions - all those routed from the MPIC at normal level
- Normal core-initiated exceptions - Data Storage (DSI), Instruction Storage (ISI), Instruction TLB (ITLB), and Data TLB (DTLB)

Core-initiated exceptions are handled at either the hypervisor or Guest-supervisor level, depending on at which level the instruction flow raising the exception is running. Therefore, handling an interrupt triggered by hypervisor code through a Guest-OS handler is counter-intuitive.

To accommodate this requirement in e500mc, all core-interrupt-specific registers, which hold the save/restore context and additional state information on a normal-level interrupt, have been duplicated into a set of registers for use by the Guest.

- SRR0/SRR1/DEAR/EPR/ESR (hypervisor registers)
- GSRR0/GSRR1/GDEAR/GEPR/GESR (Guest registers)

The core selects the normal or Guest bank of registers for saving context and state information depending on at which level an exception is taken. This bank duplication is transparent to interrupt handler code as referencing one of these registers. For example, the Exception Syndrome Register (ESR) is automatically mapped by the core to either ESR or GESR, depending on the current execution level.

Guests trap when they try to write IVPR and IVOR registers. The hypervisor stores the attempted write to the GIVPR and GIVOR registers (or keeps a copy of it in memory for interrupts to which there is no GIVOR so that it can reflect it in the software, if required).

When external interrupts are directed to a Guest-OS and an interrupt occurs while executing in hypervisor mode, the interrupts remain pending until the processor transitions back to Guest mode.

At the MPIC level for critical and machine-check events, interrupts can be multi-casted to any set of cores: to all cores; to a single core; or to those cores that are part of a specific partition. In addition, the MPIC registers that are usually referenced inside normal-level interrupt handlers (including IACK, EOI and WHOAMI registers) are physically duplicated with one register bank per core. This duplication uses an address-aliasing mechanism, which allows a single, common address for each of the registers, independent

of which core is referencing these registers. Such implementation greatly improves virtualization and scalability by allowing core-independent software interrupt handlers.

Each e500mc core includes a new Interrupt Proxy mechanism for normal-level interrupts. When enabled, software can read the External Proxy Register (EPR), rather than the MPIC-level implemented register IACK, in order to identify the precise exception and read the associated vector. As soon as the interrupt transitions from “pending” to “in-service” by the MPIC and is reported to the appropriate core by activating its internal int signal, a hardware proxy mechanism automatically copies the value of the IACK register into the core-level EPR. This proxy feature can optimize interrupt latency by removing the necessity to read IACK at the cost of CoreNet latency. Another benefit is direct access to the interrupt vector from Guest-OS code without the need to call the hypervisor service to access the MPIC-level physical registers.

Interrupt latency of the e500mc and e5500 cores is 10 cycles or less, except in rare cases where the core is in the completion stage of a guarded load or a cache-inhibited *stwcx.* instruction.

There are 3 conditions that should be considered when deciding to authorize Guest-level interrupts:

- Performance
- Degree of Guest-OS virtualization
- System integrity

Figure 7 illustrates the handling of a TLB-miss exception for the case where the MMU service routine of the Guest-OS is invoked but the MMU access is done under control of the hypervisor.

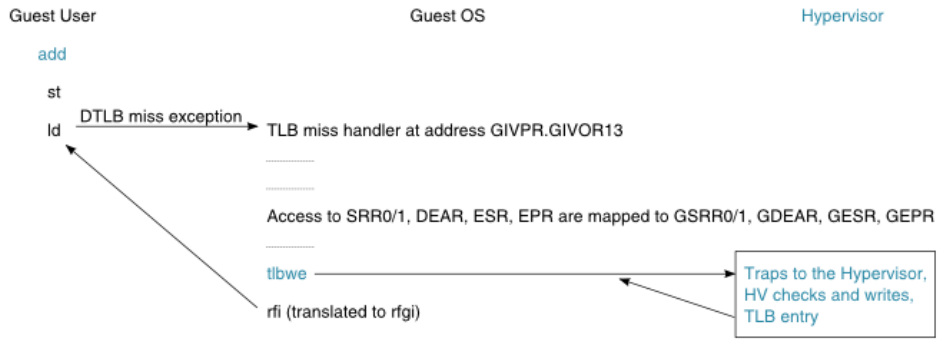


**Figure 7. TLB exception handling by Guest-OS under Hypervisor control**

## 2.4 Review of core and partition ID support

Here is a review of the various hardware-level IDs implemented in the P4080, which are available at the hypervisor or Guest-OS level:

1. Processor ID in the PIR register - a logical core ID set to an unique value at reset that can be further modified by the hypervisor.

2. Processor ID in the GPIR - a logical core ID that is set by the Guest to a unique value of cores within a partition. The same value may be used for different cores in different logical partitions. This is the value returned to a Guest-OS attempting to read the PIR register. Normally not modified after setting in the case of a statically partitioned system.
3. Per-core LPID in the LPIDR - a logical partition ID that is set by the hypervisor and taken into account in the virtual address translation performed by the MMU. The same value is shared by all cores in a given partition. Normally not modified after setting in the case of a statically partitioned system.
4. Per-core PID (Process ID) in the PIDR - managed by an OS and taken into account in the virtual address translation performed by the MMU. Each PID value represents a different address space as assigned by the Guest-OS.
5. WHOAMI read-only register - returns to the core reading this register the physical core ID value (#s 0-7 for P4080) reflecting the physical connection of this core to the MPIC. Fully static and cannot be modified; primarily intended for use by the MPIC interrupt handler and for hypervisor MPIC virtualization purposes.

## 2.5 Inter-core/inter-partition communication

In applications built on multicore architecture, inter-communication and synchronization between the cores and the logical partitions is a key area for performance, particularly when core and partition activities are closely linked.

Standard means, such as shared-memory, normal interrupts, or software-implemented messaging, are costly in terms of software overhead and latencies for achieving coherency and atomicity. The QorIQ multicore architecture integrates a set of enhanced features for efficient inter-core/inter-partition communication.

See Section 3.5 for SoC-level inter-partition communication mechanisms.

### 2.5.1 Inter-core/inter-partition doorbells

At core level, a so-called "doorbell" mechanism supports direct synchronization between cores and partitions. This mechanism provides a means for basic, low-latency core-to-core signaling. It relies on two instructions, which are defined by Power Architecture ISA 2.06 (new from e500v2 to e500mc core), *msgsnd* and *msgclr* (message send/clear), and a new set of related core-level exception vectors:

- IVOR36: Doorbell Interrupt
- IVOR37: Doorbell Critical Interrupt
- IVOR38: Guest Processor Doorbell Interrupt
- IVOR39: Guest Processor Doorbell Critical Interrupt
- IVOR39: Guest Processor Doorbell Machine-Check Interrupt

Figure 8 illustrates the basic operation of inter-core/inter-partition doorbells.

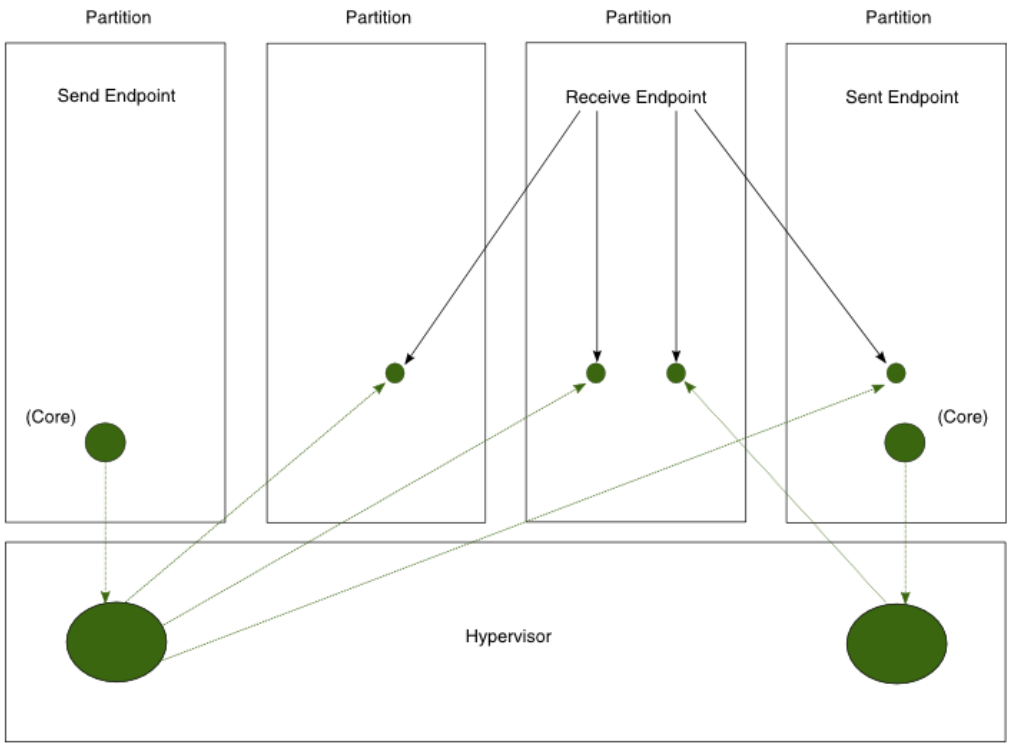


**Figure 8. Inter-core and inter-partition doorbells**

The inter-core/inter-partition doorbell operation is as follows:

1. A core issues the hypervisor-level instruction *msgsnd* with a 32-bit attribute.
2. The attribute selects the type of doorbell interrupt and the targeted core.
3. Execution signals *msgsnd* through an appropriate broad-casted message over CoreNet, which is received by all the cores.
4. Each core filters that signal according to the received attribute field values and the current state of the local PIR (or GPIR) and LPIDR.
5. A doorbell signals a core in either a specific Logical Partition or a different partition; a doorbell can also broadcast to all cores.
6. After filtering, the targeted core(s) generates an appropriate interrupt.
7. Unlike a Doorbell type, a Guest-Doorbell interrupt is kept pending if the core is running hypervisor code until the core runs in Guest state.
8. The *msgclr* instruction can be used to clear a pending doorbell; consideration must be given to the doorbell interrupt assigned level (normal, critical, machine-check).

**NOTE**

Doorbells are not cumulative.

In line with their ability to propagate direct inter-core signaling, doorbells are fully hypervisor-controlled, which means:

- The *msgsnd* and *msgclr* instructions can only be executed at the hypervisor level.
- A resulting interrupt always routes initial control to a hypervisor handler.

Besides being a general "shoulder-tap" mechanism, a core-level doorbell provides efficient services to a partitioned system, such as:

- Allowing the hypervisor to reflect asynchronous interrupts to a Guest-OS.
- Implementing hypervisor services (through hyper-calls) for fast Guest-OS-to-Guest-OS synchronization means.

The doorbell mechanism implemented by the *msgsnd* instruction does not include any user data exchange; the message carries only the signal attributes. User data exchange can be added through complementary means, such as traditional shared memory. Since the message is sent over CoreNet, it can be ordered with preceding stores using a memory barrier; thus, assuring to the receiver of the message that the stores have been performed prior to the interrupt occurring on the receiving core.

Doorbell latency from *msgsnd* executed by one core to the associated interrupt on another core can be estimated in the same range as a store operation. Given the CoreNet transaction latency and the time taken when the transaction is received to set the interrupt, typical latency is estimated in the order of 50 to 100 core cycles.

More details on msgsnd and msgclr can be found in e500mc Reference Manual and EREF.

### 2.5.2 External context load/store support

External context loads and stores are intended for use by the OS and the hypervisor to perform load, store, and cache management instructions to a separate address space while still fetching and executing in the normal supervisor or hypervisor context.

In e500mc, this is implemented through a new category of ISA v2.06 instructions called "External PID load and store instructions" and two dedicated registers: EPLC and EPSC.

- *lwepx* (load word by external PID indexed)
- *stwepx* (store word by external PID indexed)

Additional instructions exist for selecting various data types.

The OS or the hypervisor selects the targeted address space by altering the contents of the EPLC and EPSC registers for loads and stores, respectively. When the EA that is specified by the external PID load or store instruction is translated, the translation mechanism uses ELPID, EPID, EAS, EPR, and EGS values from the EPLC or EPSC registers instead of LPIDR[LPID], PID[PID], MSR[DS], MSR[PR], and MSR[GS] values.

These instructions allow an OS to manipulate a process' virtual memory using the context and credentials of the process. These instructions can also help establish hypervisor-controlled inter-Guest-OS communication means.

## 2.6 Miscellaneous (other core-level considerations)

The eight cores in the P4080 are physically independent, meaning they can be independently woken-up on reset. At Power-On or SoC-level hardware reset, Core0 always boots first; further boot code manages other core activations. Per-core selective boot and reset activation is performed through SoC registers DCFG_BRR (Boot release register) and MPIC_PIR.

Further description of SoC power-on Reset and boot sequence can be found in the P4080 Reference Manual.

As in other Power Architecture cores, each e500mc core has a timer unit, which provides several timing services:

- 64-bit Time Base (TB)
- 64-bit Alternate Time Base (ATB)
- 32-bit Decrementer (DEC)
- Fixed-Interval Timer (FIT)
- Watchdog Timer (WT)

In e500mc, TB/DEC/FIT/WT are clocked either by a SoC common clock (Plat_Clk/16) or an external common clock (RTC signal). ATB is clocked by each core-specific clock.

The common clocking of TB/DEC/FIT/WT ensures a proper time synchronization between all cores. Enabling core timebase clocking in a synchronized way can be achieved through the SoC level register RCPM_ CTBENR.

More details can be found in e500mc Reference Manual and the P4080 Reference Manual.

The e500mc System Call instruction can generate a hypercall interrupt (sc 1), in addition to the system-call interrupt (sc or sc 0)

Support for atomic update primitives for implementing memory synchronization variables (for example, semaphore and mutex) is provided through *lwarx* and *stwcx.* instructions.

The e500mc introduces ISA 2.06 so-called "Decorated Storage Instructions," which are targeted to multicore-shared and non-cacheable memory variable updates for primitives, such as clear/increment/decrement/accumulate. This is of particular interest in networking applications where many flow statistic counters must be maintained, these instructions can be used in other contexts.

More details can be found in e500mc Reference Manual, EREF, and Application NoteAN4181: "Decorated Operations for QorIQP3/P4/P5 Processors".

# 3 SoC-level partitioning mechanisms

This section introduces the architectural features integrated in the SoC (outside the e500mc cores and the MPIC that were previously discussed in this document) that can enforce or optimize system partitioning.

There are three requirements that must be considered:

1. Enforce robust partitioning for I/O-initiated transactions.

- This applies to all peripheral modules and accelerators with Direct Memory Access (DMA) capability.
- In the P4080 architecture, this is managed by the new Peripheral Access Management Unit (PAMU) structure, which works as an "IO-MMU".

2. Restrict the coherency SoC-level overhead to strict necessity.
   - This applies if hardware-level coherency is expected.
   - Restrictions depend whether the memory partitions are private or shared between the multiples cores and logical partitions.
   - In the P4080 architecture, this is managed by the coherency sub-domains.
3. Partition the usage of the shared level-3 SRAM/Cache (also called CPC).

## 3.1 Overall SoC address mapping

In order to better understand how these requirements can be fulfilled, this section gives a brief outline of how the transactions are mapped and controlled through the various structures of the SoC.

Figure 9 illustrates SoC transaction mapping and control.

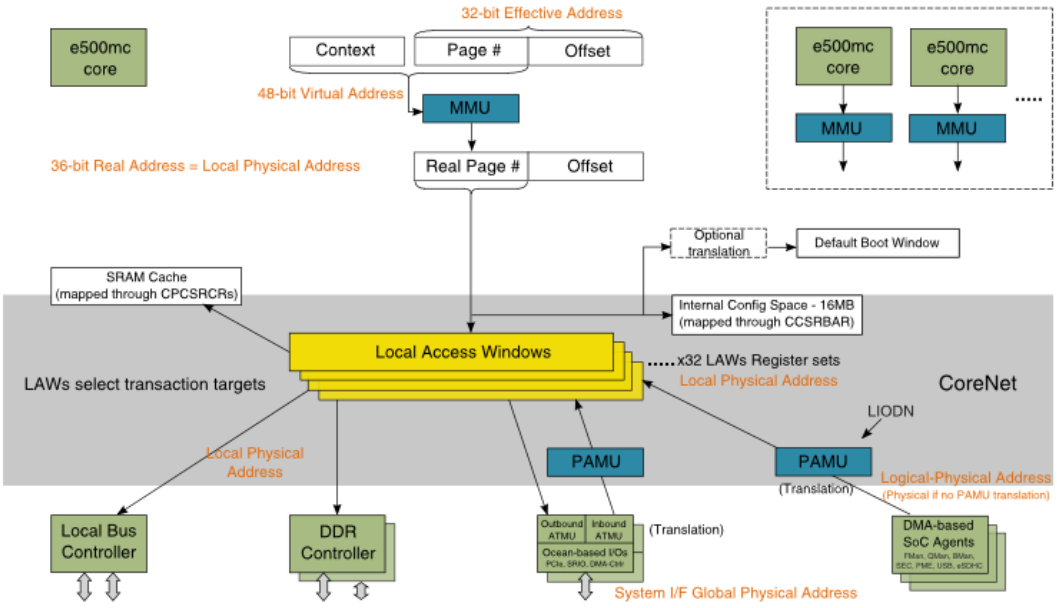


**Figure 9. SoC transaction mapping and control**

On a core-initiated transaction, the program's EA is extended with context IDs to make the virtual address. Through the MMU, the virtual address is verified and translated into the real address, which is the 36-bit local physical address. In case of a cache miss or non-cacheable access, the real address is directed to CoreNet and searched through the Local Access Window (LAW) registers.

In LAW registers, there are 32 windows, each associated to an address range (4KB to 64GB) and to a specific target, which is either a local memory interface (DDR, eLBC, internal SRAM) or a system interface (PCIe, SRIO). The LAW registers perform no address translation.

The pre-defined hardware-coded target values can be found in each processor Reference Manual (see "Global Source and Target IDs" section).

Transactions targeted to a local memory interface operate additional address mapping in the controller for chip and bank selection.

Transactions targeted to a system interface operate additional mapping and address translation (through outbound ATMU registers).

Specific local regions targeting configuration registers (CCSR) and boot windows are mapped through dedicated registers that supersede LAW registers. For the SRAM-configured portion of CPC-L3, both the LAW registers and the dedicated registers are involved.

For transactions initiated by a DMA-capable agent (I/O or accelerator), an initial mapping is performed by the inbound ATMU registers. This initial mapping applies specifically to PCIe and sRIO in order to decouple the system and local addressing.

A 36-bit address and transaction attribute is presented to the PAMU (if enabled). PAMU performs a permission check and an optional address translation. The resulting 36-bit address from PAMU is the Real Address that is directed to CoreNet and searched through the LAW registers to identify a target.

There are two important notes for DMA-initiated transactions:

1. Ocean-based controllers allow direct interconnection (for example, PCIe-to-PCIe). Therefore, transactions can be mapped directly from inbound to outbound windows and forwarded through the Ocean I/O switch without reaching the CoreNet/PAMU/LAW mechanisms.
2. The optional address translation performed by PAMU is for a hypervisor-based partitioned system where each Guest-OS and its I/O drivers manage a logical-physical address space (that is, the perceived physical address space from a guest). Each logical-physical address space is re-mapped by the hypervisor to the real physical address space of the platform. PAMU is where the hypervisor implements this logical-physical to physical-physical mapping for DMA-initiated transactions.

Description of LAW registers can be found in the P4080 Reference Manual.

## 3.2 Enforce robust partitioning for I/O-initiated transactions

DMA-initiated transactions can be controlled through PAMU. More precisely, each DMA-capable peripheral is controlled by a PAMU mechanism. As such, PAMU is implemented as a set of distributed tables called Peripheral Access Authorization and Control Table (PAACT) containing entries called Peripheral Access Authorization and Control Entry (PAACE).

In order to combine configuration flexibility and low overhead, the precise implementation of PAACTs involves:

- A two-level table structure (Primary and Secondary PAACTs) for tables residing in the external system memory with base pointers located in the CCSR.
- Two lookup-cache mechanisms (PPAACT cache and SPAACT cache) located in internal dedicated SRAM.

Figure10 illustrates the purpose of PAMU.

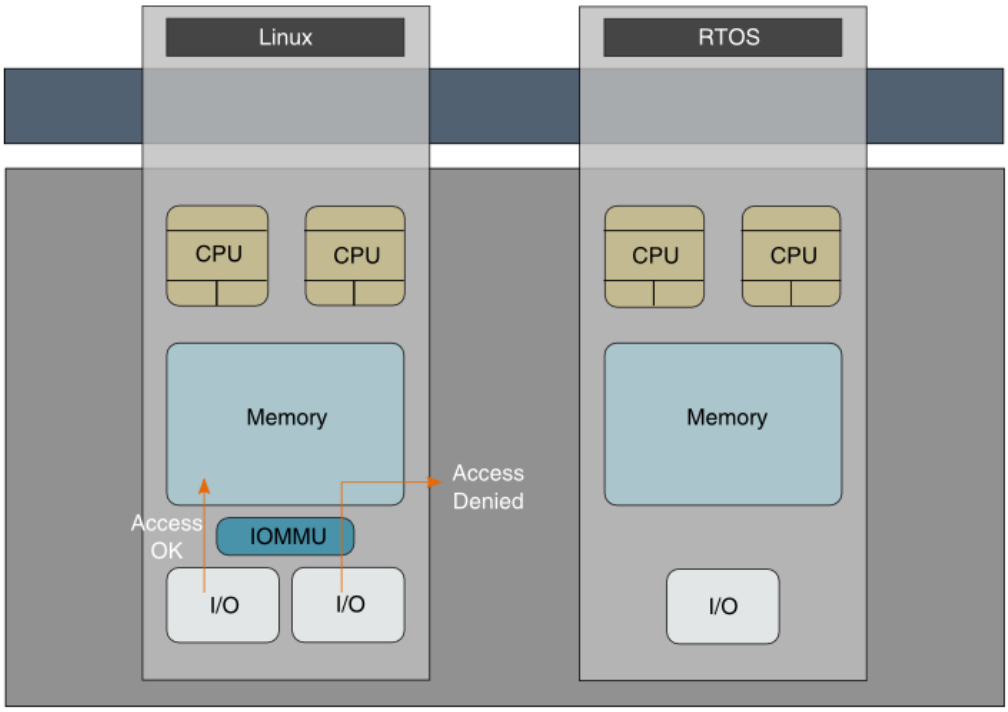


**Figure 10. The key role of PAMU**

Figure 11 outlines the overall structure of PAMU.

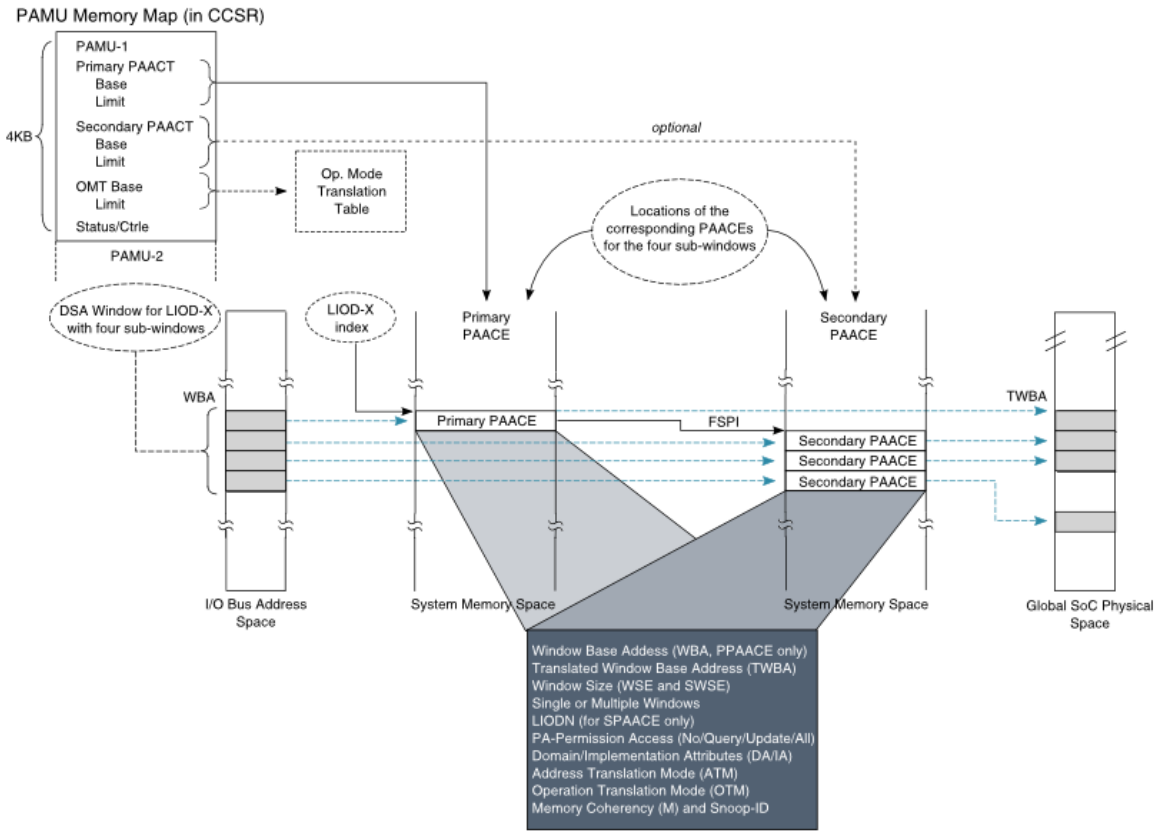


**Figure 11. Overall PAMU structures**

On an I/O transaction, a Logical IO Device Number (LIODN) 12-bit value is issued, in addition to the targeted address and transaction type. This LIODN is used in conjunction with the targeted address to perform an entry search into one of the PAMU distributed tables.

A physical I/O device may be capable of acting as one or as multiple independent LIOD(s). If the physical I/O device acts as multiple independent LIODs, it can issue multiple LIODN values. LIODN values are configurable through registers specific to each physical I/O device.

By configuring one or more PAMU entries for a given LIODN, each LIOD is allocated through PAMU to a contiguous or multiple-sub-windows portion of I/O address space called the Direct Storage Access (DSA) window. The DSA window is 2n-bytes in size (minimum 4KB).

PAMU should only be managed by an OS or a hypervisor. In a partitioned system controlled by a hypervisor, only the hypervisor can initialize and update the PAMU-entries content. Software drivers that run either at the Guest-user or Guest-supervisor level can still directly manage a DMA-based I/O or accelerator; PAMU ensures that each DMA accesses its authorized address partition.

PAMU entries include the following control capabilities:

- Authorizes - ensures operation originates from a valid I/O interface and is valid for this interface.
- Access control - restricts the type of transaction (read/write).

- Address translation - allows a hypervisor to virtualize each partition address map (through logical-physical mapping to physical-physical).

**NOTE**

This optional translation is required when I/O drivers are ported in a partitioned system and are unaware of the other partitions.

- Operation-type translation - allows PAMU to convert the Ingress I/O operation to another type of operation on the system interconnect (CoreNet).

**NOTE**

This capability relates to some specific requirement in the area of atomicity, cache optimization, and coherency purposes.

PAMU allows a hypervisor to define the memory partitions that are accessed by I/O devices to be either coherent or non-coherent. In addition, it allows a hypervisor to control the stashing of I/O data into the SoC caches.

The PAMU tables reside in main memory, as configured by the OS or hypervisor. For performance purposes, some internal SRAM regions reside in the P4080 and act as cache for the PAMU table entries. For that reason, each PAMU is considered a potential coherency agent (like the e500mc cores), which means it has limited built-in snooping capability and can be part of the coherency sub-domains.

The P4080 physically implements several different PAMU agents, each responsible for a subset of the DMA-capable blocks in the SoC. System architects may choose to merge the PAMU tables into global tables by configuring a unique, common table base-address for all the PAMUs. The allocation of the table entries in this common table to the DMA-capable sub-block depends on the LIODN values allocation.

More details on PAMU structures and implementation can be found in the P4080 Reference Manual.

## 3.3 Partitioning and coherency (coherency sub-domains)

The P4080 provides full hardware support for memory coherency. The e500mc caches support write-back; therefore, to operate in a coherent manner with other masters, snoops must be performed to keep the per-core caches coherent with the rest of the system.

In systems that must achieve strict determinism, hardware coherency can be totally or temporarily disabled in order to prevent snoop overhead. In that case, software is responsible for ensuring system coherency if it is required by performing appropriate cache-management operations (such as “flush” or “invalidate”).

In systems with hardware-coherency enabled, it may be desirable to limit the amount of snoop traffic to only those core and cache nodes that require coherency for each memory region. This optimization may be done in accordance with the logical partitioning of the system. P4080 architecture allows this optimization through the coherency sub-domains.

For core transactions, coherency is controlled through MMU. These five bits in each TLB entry define the attributes of the given memory page:

- W - Write-through caching
- I - Cache inhibited

- M - Memory coherency
- G - Guarded accesses
- E - Endianness

Access to a page whose M-bit is set initiates a coherent transaction (that is, a transaction that will guarantee system coherency for the referenced data).

The P4080 implements the Modified Exclusive Shared Invalid (MESI) coherency model. Depending on the following 3 factors, the access may either result in a fast transaction completed at the local core/cache nodes or result in a transaction forwarded to CoreNet with snoop requested to other nodes:

1. Type of access (read/write)
2. How many caches in which the data is present
3. In which state the data is present

CoreNet implementation of the MESI model supports an optimization called "Intervention." If a SoC agent were to read or write data and one cache has the data in either an Exclusive or Modified state, it produces an Intervention and provides the cache line back to the requester. Depending on whether the initial request was a read or a write, the cache line is either shared between two caches or it is set as modified in the requesting core/cache after being invalidated in the snooping core/cache.

As we introduced in section 3.1, at the CoreNet level, each new transaction address is first decoded through the set of 32 LAWs. Figure 12 shows the content of the three registers that define a LAW.

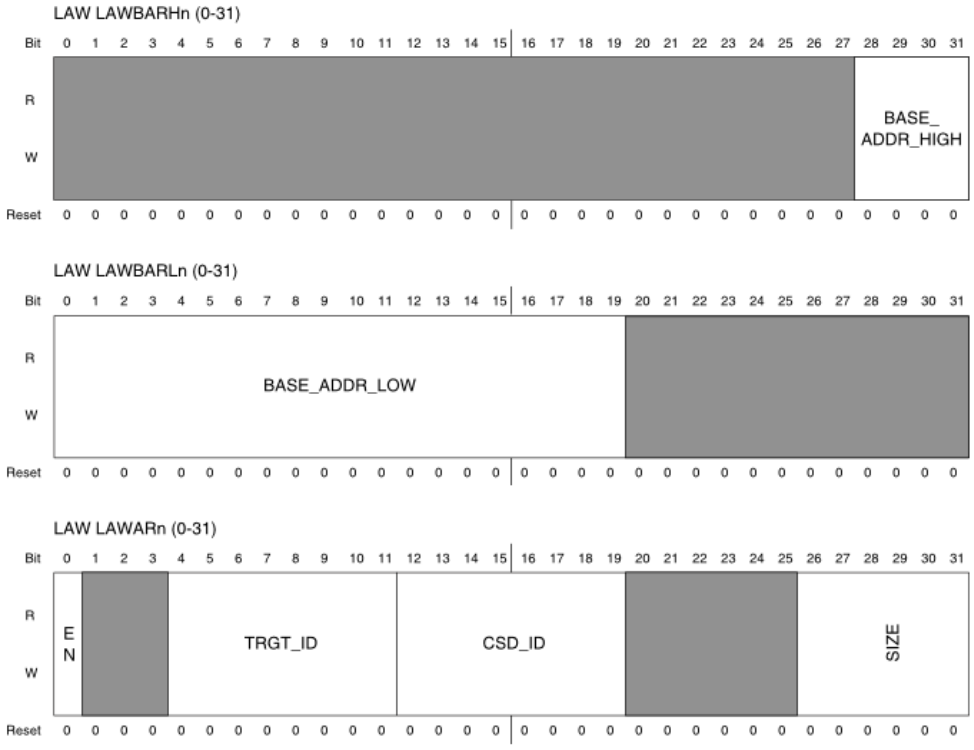


**Figure 12. Local Access Windows (LAWs) registers**

In addition to the base address, size, and target, the LAW includes a Coherency Sub-domain Identifier (CSD_ID). This parameter identifies a group made of one or more cores and other caching devices that represent a coherency sub-domain.

For a given transaction hitting a LAW, the CSD_ID will instruct CoreNet to direct (multi-cast) this transaction to the relevant caching devices for snooping.

- CSD_ID = 0 is the default group at Reset configuration, which includes all caching devices on the SoC.

The relationship between a CSD_ID and sub-domain caching devices is encompassed by another set of internal system registers called Coherency Sub-domain Mapping Registers (CIDMRn). There are 32 coherency subdomains and, therefore, 32 CIDMRn registers supported in the P4080. CSD_ID in LAW is used as an index to select one of the 32 CIDMRn registers. Each bit in the CIDMRn registers then selects (or excludes) each of the SoC caching agents from that given sub-domain.

### 3.3.1 Additional comments on coherency sub-domains

In the P4080, cores and PAMU are the caching devices that are selected to define coherency sub-domains. Effectively, as PAMU includes some internal memory for caching PAACEs, those entries may potentially be part of some coherency sub-domains.

For DMA-initiated transactions, in the PAACEs there is a similar mechanism for defining those caching devices that are involved in the CoreNet transaction. The associated field in the PAACE is called SnpID, which works as an index in associated system-level configuration registers (SIDMRn) for selecting the caching agents that snoop the transaction.

DMA-initiated transactions normally go through:

1. PAMU
2. CoreNet
3. LAWRn

In this case, the coherency sub-domain is defined by the LAW, so there is no need for the PAACE[SnpID] feature.

However, in a stashing transaction -- which does not systematically go through the LAWs -- coherency sub-domains that are implemented by PAMU through PAACE[SnpID] may be of interest to limit the coherency traffic overhead. A stashing transaction is a specific type of DMA-initiated transaction that explicitly targets a caching device, either a core/cache node or the shared CPC, for storing data in a cache before the core references this data. Stashing is a key optimization feature in all I/O and data-path intensive applications.

There is no direct link nor consistency enforced by hardware between the coherency sub-domains and logical partitions, as defined through the core-level LPIDR, because both notions are not strictly aligned. If all the cores involved in a logical partition (running a single OS) need to ensure coherency over their shared memory, some system use cases may call for the following optimizations:

- Restrict a coherency sub-domain for a given memory range to less than the logical partition.
- Extend a coherency sub-domain over two or more logical partition (inter-partition shared region).

The core-level LPID-based partitioning mechanism and the CoreNet-level CSD_ID-based coherency sub-domain mechanism must be considered as optional and complementary mechanisms working at two different levels, providing different functions.

A multicore system can be fully partitioned by appropriate configuration of MMUs and PAMUs, which both play the essential role of enforcing private and shared resources without the need to explicitly define logical partitions and/or coherency sub-domains. That type of architectural decision is dependent on how the designers want to use the features and optimizations provided by the architecture.

## 3.4 Shared CPC partitioning

The P4080 integrates a shared CPC. This cache is a 1MB, 32-way set associative, 64-byte cache line and works in-line between CoreNet and the DDR controller. As there are two DDR controllers in the P4080, there are also two CPCs, which provides a total of 2MB.

This internal SRAM resource can support multiple, flexible usages:

- Cache
- Static SRAM
- Partitioning

These options are configurable on a per-way granularity, each way providing 32KB.

When used as a cache resource, the CPC can cache data initially located in DDR main memory and can stash data coming from DMA-based peripherals.

When configured as an internal static SRAM resource, the CPC is mapped to a configurable fixed address range. Once the address range is configured, one or more ways can be configured to work as SRAM to a specific logical partition through the appropriate MMU and PAMU configuration.

When configured as a cache, the CPC can be partitioned. CPC partitioning is achieved through specific configuration control registers. The overall mapping mechanism is as follows:

1. A transaction matches a LAW whose target ID is the memory complex 1 or 2 (DDR/CPC sub-module).
2. LAW[CSD_ID] identifies the coherency sub-domain, which is described through CIDMRn registers.
3. The selected CSD_ID is searched through CPC partitioning control registers (CPCPIRn, CPCPARn, CPCPWRn).
   – Note: the result of this search determines which of the 32 physical ways the CPC should be considered if this transaction leads to a new allocation in the CPC.

CPC partition filtering only applies to transactions that lead to allocating a new line in the cache (such as CPC cache misses). For a transaction that hits the CPC, the related cache line is selected for appropriate action, independent of the per-way partitioning mechanism. Restricting the usage of CPC per-way partition to both allocate and further related hits requires an appropriate MMU/PAMU/CPC configuration.

For a transaction that leads to allocating a new line in the CPC, the per-way partitioning configuration scheme provides some flexibility to choose (and restrict) the types of transactions that are allowed to allocate a new line.

The complete description of CPC modes and configuration is covered in the P4080 Reference Manual CPC chapter.

## 3.5 SoC-level features for inter-partition communication

In addition to core-level mechanisms covered in section 2.5, the P4080 provides other means at SoC level for inter-core and inter-partition communication.

### 3.5.1 QMan

Queue Manager (QMan) is one component of the Data Path Acceleration Architecture (DPAA), which is a set of hardware accelerators whose main purpose is to offload pre- and post-processing of data-flows. Beyond its key role in managing the exchanges between the high-speed ports(1G/10G Ethernet), the cores, and the SEC/PME hardware accelerators, QMan can be used as an engine for inter-core/inter-partition communication.

A queue, as managed by QMan, is a FIFO structure where consumers and producers can en-queue and de-queue descriptors, which are16-byte objects whose main component is a pointer to a buffer in memory. The contents of a buffer are not interpreted by QMan; so, a buffer can contain a protocol frame or packet, a message, a task description, or any kind of application-specific object.

Figure13 illustrates how producers and consumers can be the cores and the various DPAA off-loading engines.

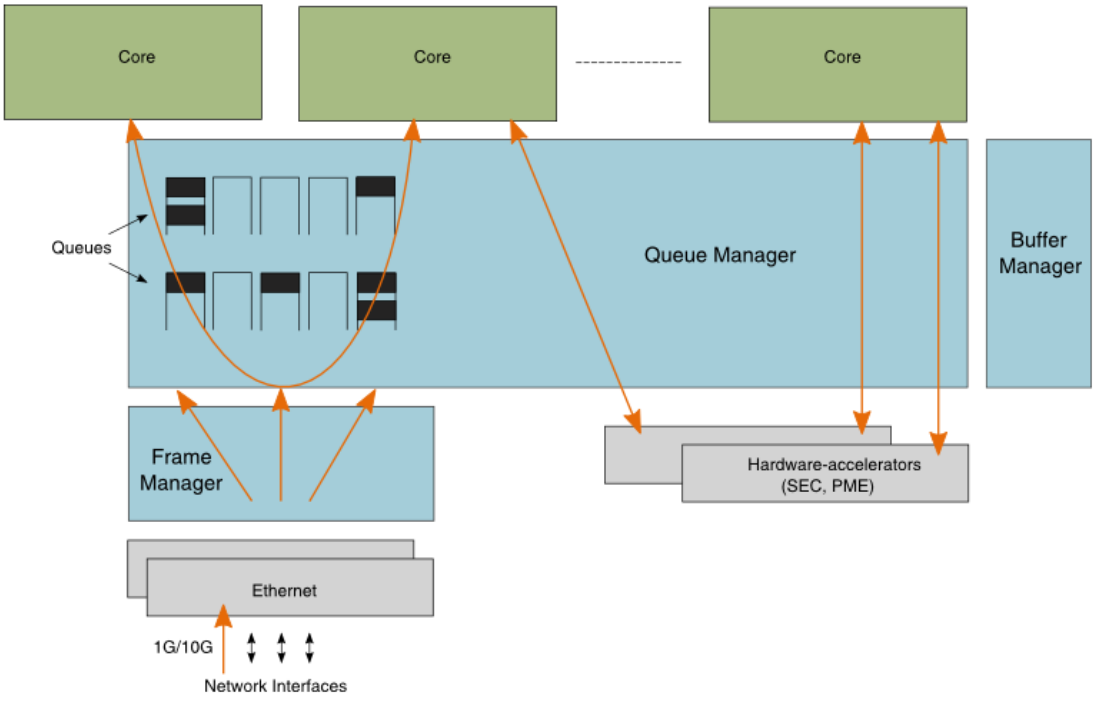


**Figure 13. QMan as general SoC inter-communication engine**

The key role of QMan is to realize a complete hardware abstraction by providing the producers and consumers a simple interface to utilize the queues: by means of en-queue and de-queue commands and interrupt/polling signaling mechanisms.

QMan supports a nearly unlimited number of queues (16M max). Each queue is created to perform a particular task, such as inter-core exchange, which provides flexible synchronization and messaging services.

All the intricacies related to synchronization and atomicity between producers and consumers sharing a given queue are fully abstracted by the hardware and are, therefore, transparent in the software.

DPAA is described in the DPAA Reference Manual.

## 3.5.2 MPIC

MPIC also provides some capabilities for core-to-core communications by means of:

- Inter-processor interrupt channels
- Message interrupt channels

MPIC is described in P4080 Reference Manual specific chapters "Multicore Programmable Interrupt Controller" and "Interrupt assignments"

# 4 Other system-level considerations

## 4.1 Partitioning and determinism

Robust partitioning primarily addresses spatial partitioning, which provides segregation (in the sense of who can access what resource in the system), but it does not imply strict temporal partitioning.

Due to the internal, physical interactions at the interconnect fabric and when accessing shared resources, determining worst-case latencies from core and software perspectives in all potential scenarios for a multicore SoC is not realistic.

System designers should consider the following options when making architectural hardware and software decisions:

1. De-activate some of the optional mechanisms that improve performance to improve determinism.
2. Restrict hardware coherency and make use of several available capabilities (such as cache pre-loading and line-locking) at core level and L1/L2 cache levels to improve deterministic behavior.
3. Configure each partition to have private usage of each CPC+DDR controller subsystem to increase physical isolation between two logical partitions.
   - Such mapping (partition1/CPC-DDR1 and Partition2/CPC-DDR2) should be strictly enforced by the appropriate mechanisms (MMU, LAW).

**NOTE**

In this case, I/O DMA activities can still produce interactions with the partitions and core activities when accessing the CPC-DDR subsystem. The actual temporal behavior of a particular use case should be analyzed through measurement.

4. Disable the speculative ability to direct read accesses to the DDR before resolving if it is a hit or miss in the CPC. To disable, use CPCx_CPCHDBCR0[SPEC_DIS].
5. Restrict the interactions between cores and partitions by implementing time-management strategies from low-level software (RTOS/hypervisor).

## 4.2 Review of P4080 SE protection mechanisms

The P4080 integrates multiple protections against soft errors (hazardous bit errors) in internal and external memories. The main features include:

- L1-cache - parity on data and tag
- L2-cache - ECC or parity on data; parity on tag
  - For L2-caches that are allocated for data (data-only or unified), L2 supports "Write Shadow" mode where L2 content is always maintained as a super-set of L1. This mode brings

additional soft-error protection by automatically forcing L1-line invalidation and reload from L2 in the case of a L1 parity error on a modified line.

- CPC/L3-cache - ECC on data, tag, and status
- DDR controller - ECC on data
- Local Bus (eLBC) controller - parity on data; ECC on FCM (NAND flash controller)
- PAMU - ECC on both PAMU look-aside caches (Primary PAACT cache and Secondary PAACT cache)
- DPAA - ECC on QMan, BMan, FMan; and 10G-MAC internal memories

**NOTE**

Parity detects single-bit errors; ECC detects and corrects single-bit errors and detects double-bit errors.

Each SE protection mechanism for a given SoC block and its related control and status/report registers are described in the specific chapter of the reference manual for this block.

## 4.3 Configuration registers (CCSR, DCSR) protection

All P4080 Configuration Control and Status registers are accessible in a specific 16MB region of the local space. The base address of this local region is configurable through CCSRBARs registers

- The default address is 0_FE00_0000h

In a partitioned system, it is essential to restrict access to the CCSR region (or only part of it), depending on partitions and trustability. From core and software perspectives, this can be fully achieved with proper MMU settings. As it is mapped in the local space, CCSR can be a potential target for a peripheral resource, in which case protection can be enforced through PAMU.

In addition, in order to selectively allow direct access to some control/status registers (those attached to a specific service provided by an I/O, a hardware accelerator, or the interrupt controller), the CCSR is organized in 4KB sub-window granularity, which is optimized for MMU- and PAMU- based control. Each sub-window contains CCSR registers for a specific function. For example, DMA-controller1 registers are located in the 4KB region starting at offset 0x10_0000; DMA-controller2 registers are located in the next 4KB starting at offset 0x10_1000.

A similar local region of 4MB holds all the Debug and Performance Monitoring Control and Status Registers (DCSR). Mapping this region for access from the core/software is different than CCSR because an explicit LAW must be set with DCSR identified as the target. Apart from this, restriction to that region can be controlled with MMUs and PAMUs.

Detailed mapping of CCSR can be found in P4080-RM section "Configuration, Control, and Status Register (CCSR) Space".

## 4.4 What happens on partitioning violation

Robust spatial partitioning can be supported by strict enforcement in accessibility of any memory-mapped resource, enforcement by MMU for core-initiated transactions, or enforcement by PAMU for I/O-initiated

transactions. A memory-mapped resource can be an actual memory region, a peripheral (I/O), an internal hardware accelerator, or some internal configuration/status register.

Here are three typical situations in which access violations can occur:

1. Software (one core) attempts to access a resource whose virtual address (EA and context) is not successfully mapped in the current context by the MMU/TLB. In this case, an exception of type "Data TLB miss" (Interrupt vector #13) is generated that will direct control to either the supervisor- or hypervisor-level handler. Status information, including faulting EA, is available in several core registers ((G)DEAR, (G)ESR, MASn).
2. Software (one core) attempts to access a resource whose virtual address (EA and context) is successfully mapped by an entry in the MMU/TLB whose Virtualization Fault-bit is set. Typically, in a partitioned system with some level of virtualization, it is set by the hypervisor for pages associated with a device (or any service) for which the hypervisor is providing a virtual device through emulation. In this case, an exception of type "Data Storage - DSI" (Interrupt vector #2) is generated and the full status is reported.
3. An I/O (DMA) attempts to access a resource whose physical address is not successfully mapped by any of the PAMU entries configured for this I/O. In this case, an exception is generated by the PAMU logic to the MPIC (Interrupt vector #8). Status information on the PAMU access violation is available in a dedicated set of registers, Access Violation Registers.

For additional information, please see the following resources: P4080 RM, EREF and/or e500mc RM.

## 4.5 Secure-boot and platform assurance (TrustArchitecture)

A “trusted system” is a system that does what its builder (OEM) and users expect it to do and does not do what the OEM and users consider harmful. Trust architecture provides the tools to create a trusted system. OEMs who properly leverage the hardware hooks in the P4080 can trust that the software they loaded into the system during manufacturing or authorized software updating is the software that executes following system-boot. Once trusted software is in control, the OEM can leverage additional Trust architecture features to keep the trusted code in control of the system and defend against the extraction of system secrets or the introduction of malicious software.

Multicore SoCs with robust partitioning-support in hardware can leverage this capability as managed by RTOS or hypervisor to ensure the system is securely booted with trusted code and that the platform remains robust and secure during normal operation. Once protected, logical partitions are established and system designers can sandbox software applications and environments that are not trusted by leveraging all the mechanisms built to virtualize the hardware.

If the P4080 is configured to perform secure-boot, at Power-on Reset, CPU-core0 is released to begin execution from the internal bootROM containing Internal Secure Boot Code (ISBC), which is a non-modifiable code. The instructions executed from ISBC allow CPU-core0 to determine if the next code to execute outside the internal bootROM is safe by checking signed hash using key values contained in an internal fuse block. Starting from this secure boot, system developers can ensure further reasonable secure-boot chains of trust, allowing initial boot, hypervisor and/or RTOS, and further Guest-OSes running on multiple partitions to be validated before execution.

Another valuable feature provided by TrustArchitecture is the Run-Time Integrity Checking (RTIC). This checks, at regular intervals, that the content of a memory region has not been altered -- as compared to the initial image that has been signed. This feature is implemented as a background task by the SEC accelerator leveraging its cryptographic hashing capability and is able to manage four distinct memory regions. If the mechanism is primarily intended for security against malware, it may trigger alternate usage for fault-detection on critical code and data areas.

Trust architecture includes several other features, such as Secure Debug Access, Tamper Detection, Secret Protection, and so on. Detailed description can be found in P4080 Reference Manual and the White Paper "An Introduction to the QorIQ Platform's Trust Architecture".

## 4.6 Considerations for other P-Series QorIQ platforms that implement the P4080 multicore architecture

All the features related to partitioning described in this document apply to the SoC devices derived from "Multicore Architecture". These devices are:

- P2 platform - P2040, P2041
- P3 platform - P3041
- P4 platform - P4080, P4040
- P5 platform - P5020, P5010, P5040, P5021

Note that there are key differences between these devices:

- P2040, P2041, and P3041 only have one DDR/CPC subsystem and interface instead of two
- P2040 has L1-cache but not L2-cache
- All P50xx have the e5500 core, which is a 64-bit extension of the 32-bit e500mc. They all feature a bigger L2-cache (512KB instead of 128KB).
- P5020, P5040 and P5021 have two-to-four cores and two DDR/CPC sub-systems

More info on each P2, P3, P5 derivative, including product brief, block diagram and reference manual can be found in each product's section at www.freescale.com

***How to Reach Us:***

**Home Page:**
freescale.com

**Web Support:**
freescale.com/support



Document Number: QORIQHSRPWP
Rev. 0
05/2013