ACPI for Memory System Resource Partitioning and Monitoring 1.0

Platform Design Document

Non-confidential

Document number: DEN0065

Contents

Release information

Date	Version	Changes
2021/Sep/30	1.0	• External release

About this document

Terms and abbreviations

Term	Meaning
ACPI	Advanced Configuration and Power Interface specification
DSDT	Differentiated System Description Table
GIC	The Arm Generic Interrupt Controller
GSIV	Global System Interrupt Vector
HID	ACPI Hardware Identifier
HMAT	Heterogeneous Memory Attributes Table
IORT	I/O Remapping Table
LLC	Last-level Cache
MPAM	Memory System Resource Partitioning And Monitoring
MSC	Memory System Component
MSI	Message Signaled Interrupt
NUMA	Non-uniform Memory Architecture
OSPM	Operating System Power Management
PMG	Performance Monitoring Group
PPI	Processor Private Interrupt
PPTT	Processor Properties Topology Table
PSCI	Power State Coordination Interface
RIS	Resource Instance Selection
SMMU	Arm System Memory Management Unit
SRAT	ACPI System Resource Affinity Table
TLB	Translation Look-aside Buffer
UID	ACPI Unique Identifier
UUID	Universally Unique Identifier

References

This section lists publications by Arm and by third parties.

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

[1] *Arm® Architecture Reference Manual Supplement Memory System Resource Partitioning and Monitoring(MPAM), for Armv8, DDI 0598A.a, ID103018A*. Arm Limited.

[2] *Advanced Configuration and Power Interface Specification 6.3*. UEFI Forum.

[3] *Arm® System Memory Management Unit Architecture Specification SMMU architecture versions 3.0, 3.1*

and 3.2, IHI 0070C.a. Arm Limited.

[4] *DEN0049E IO Remapping Table*. Arm Limited.

[5] *DEN0093 ACPI for Arm Components 1.0*. Arm Limited.

[6] *Arm® Architecture Reference Manual ARMv8, for the ARMv8-A architecture profile, Arm DDI 0487*. Arm Limited.

[7] *_DSD Implementation Guide v2.0*. UEFI Forum.

Feedback

Arm welcomes feedback on its documentation.

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

- The title (ACPI for Memory System Resource Partitioning and Monitoring).
- The document ID and version (DEN0065 1.0).
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

1 Introduction

This document describes the specification for the ACPI description of the Arm Memory System-Resource Partitioning and Monitoring feature, MPAM [1].

2 The ACPI MPAM table

The MPAM capabilities in a system are described by the ACPI MPAM table. Memory requests from requesters are handled by Memory-system Components or MSCs. SMMUs, caches, TLBs and memory channel controllers are examples of an MSC. An MSC provides partitioning and monitoring controls for managing system memory resources that provide performance to memory transactions.

The MPAM table describes the properties of MSCs in the system. The table also describes the topological arrangement of MSCs and resources in the system. The topology is expressed in terms of the location of resources within the system and the relation between an MSC and the resources that it is manages.

When MPAM is enabled, all MSCs within the MPAM topology must be programmed and enabled to support partitioning and monitoring of memory requests that are associated with a given range of PARTIDs. Different MSCs might support different PARTID and PMG ranges, and hence the smallest common value is required to be programmed into each MSC in the system to allow MPAM to operate correctly at a system level.

All MSCs in the system must be described to allow the OS to discover this common value.

The table format is decribed in Table 3.

Table 3: The MPAM table

Field	Byte length	Byte offset	Description
Header			Standard ACPI format for header.
Signature	4	0	‘MPAM’, Memory Resource Partitioning and Monitoring table.
Length	4	4	Length of this table in bytes.
Revision	1	8	Must be 0 for version 1.0 of this specification.
Checksum	1	9	The entire table must sum to zero.
OEM ID	6	10	OEM ID.
OEM Table ID	8	16	The table ID is the manufacture model ID.
OEM Revision	4	24	OEM revision of the MPAM table for the supplied OEM Table ID.
Creator ID	4	28	The vendor ID of the utility that created the table.
Creator Revision	4	32	The revision of the utility that created the table.
Body			
Array of MPAM MSC node structures	–	36	An array of MPAM node structures that describes MSCs in the system.

2.1 MPAM MSC node

The MPAM MSC node describes an MSC. The format of the MSC node is specified below in Table 4.

Table 4: MSC node body

Field	Byte length	Byte offset	Description
Length	2	0	Length of MPAM node in bytes. The length equals the size of the entire node, and includes the resource node list.
Reserved	2	2	Reserved, must be zero.
Identifier	4	4	A unique identifier for this node that can be used to reference the node.
Base address	8	8	Base address for memory-mapped MPAM control registers. The format of these registers is described in [1].
MMIO size	4	16	Size of the memory-mapped memory region that holds the registers of the MSC.
Overflow interrupt	4	20	Identifier for the overflow interrupt if the MSC supports monitoring capabilities as specified in [1]. For wired interrupts, this field represents the GSIV of the interrupt. If the MSC supports MSI, as indicated by the MPAMF_MSMON_IDR.HAS_OFLW_MSI bit, then this field must be set to 0 and ignored by the OS. If this MSC does not support overflow interrupts or monitors, this field must be set to 0.
Overflow interrupt flags	4	24	See Table 5.
Reserved1	4	28	Reserved, must be zero.
Overflow interrupt affinity	4	32	This field specifies the ACPI _UID of the processor or processor container that the overflow interrupt is routed to. The affinity type field in the interrupt flags specifies whether the affinity is with a processor or a processor container. This field is valid only if the Affinity valid flag is set to 1.
Error interrupt	4	36	Identifier for the error interrupt if the MSC supports error interrupts. For wired interrupts, this field represents the GSIV of the interrupt. If the MSC supports MSI, as indicated by the MPAMF_IDR.HAS_ERR_MSI bit, then this field must be set to 0 and ignored by the OS. If this MSC does not support error interrupts, this field must be set to 0.
Error interrupt flags	4	40	See Table 5.
Reserved2	4	44	Reserved, must be zero.

Field	Byte length	Byte offset	Description
Error interrupt affinity	4	48	This field specifies the ACPI _UID of the processor or processor container that the error interrupt is routed to. The affinity type field in the interrupt flags specifies whether the affinity is with a processor or processor container. This field is valid only if the Affinity valid flag is set to 1.
MAX_NRDY_USEC	4	52	For MSCs that support monitoring capabilities, the maximum time in microseconds that the *Not ready signal* can remain asserted following a configuration change.
Hardware ID of linked device	8	56	Describes the linked device for this MSC. This field is used for power management of the MSC. Linked devices are described in Section 2.8.1. This field must match the ACPI _HID of the linked device. This field must be set to zero if there is no linked device for this MSC.
Instance ID of linked device	4	64	The ACPI _UID of the linked device. This field must be set to zero if there is no linked device for this MSC.
Number of resource nodes	4	68	Number of entries in the list of MPAM resource nodes.
List of resource nodes	–	72	List of MPAM resource descriptor nodes. Resource nodes are described in Section 2.2.

2.1.1 Interrupt Flags

The interrupt flags are for both overflow and error interrupts, and are described in Table 5.

Table 5: Interrupt flags

Field	Bit length	Bit offset	Description
Interrupt mode	1	0	• 0 if interrupt is level-triggered. • 1 if interrupt is edge-triggered. This field is valid only for wired interrupts.
Interrupt type	2	1	• 00b if this is a wired interrupt. Other values are reserved.
Affinity type	1	3	• 0 if this interrupt is associated with a processor. • 1 if this interrupt is associated with a processor container. This field is valid only if the Affinity valid field is set to 1.

Field	Bit length	Bit offset	Description
Affinity valid	1	4	• 0 if the affinity field is not valid. • 1 if the affinity field is valid. This field must be set to 0 if the interrupt is not a PPI.
Reserved	27	5	Must be zero.

2.2 MPAM resource node

From a software viewpoint, an MSC is a container of resources. The MPAM resource node specifies the properties of a resource.

An MSC has resource partitioning controls that operate on resources.

The MPAM specification [1] describes the following standard resources:

- Cache storage
- Memory bandwidth
- Priority
- Implementation-specific

The MPAM specification also describes standard resource partitioning controls, or simply, controls:

- Cache-portion partitioning (CPOR).
- Cache maximum-capacity partitioning (CCAP).
- Memory-bandwidth portion partitioning (MBW).
- Priority partitioning (PRI).
- Implementation-specific partitioning (IMPL).

Controls are associated with specific resource types as follows:

Table 6: Resource and control types

Resource type	Control type
Cache storage	CPOR, CCAP
Memory bandwidth	MBW
Priority	PRI
Implementation- specific	IMPL

This specification assumes that all controls of a particular control type in an MSC shall operate on *one and only one* resource instance of the corresponding type as described in Table 6. If a component has multiple resource instances of the same type, then it must either implement an MSC that supports the RIS feature or separate MSCs for each resource instance of the same resource type. For example, if a cache unit implements multiple caches as distinct resources, then the cache unit must implement one MSC per cache if the MSCs do not support RIS. Alternatively, the cache unit must implement a single MSC that applies to all caches and uses the RIS feature to provide independent controls to each cache.

This specification currently only supports discovery of cache storage and memory bandwidth resources. That is because these resources can be readily identified using existing ACPI constructs that software can understand and use. Other resource types are declared as unknown resources.

See Appendix (Section 3) for an example system that illustrates these points.

The OS must check the MPAMF_IDR.HAS_RIS field of the MSC to determine whether it supports the RIS feature.

The location of the resource is useful for the OS to uniquely identify the resource in a system. For example, if the resource is cache storage, then the location identifies which physical cache contains the cache storage. This information enables resource management software frameworks to utilize the MPAM capabilities of the system efficiently.

Table 7: Resource node

Field	Byte length	Byte offset	Description
Identifier	4	0	Identifier for this resource node. Each resource in the system must be assigned an identifer that is globally unique among all resources in the system.
RIS Index	1	4	Index of this resource if RIS is supported. This field must be set to a value that is between 0 and MPAMF_IDR.RIS_MAX. This field must be set to 0 if the parent MSC doesn't support RIS. All resource controls of the associated control type in Table 6 that the parent MSC provides are then considered to be operating on this resource only.
Reserved1	2	5	Reserved, must be zero.
Locator type	1	7	Identifies the location of this resource. Location types are defined in Section 2.3.1.
Locator	12	8	Locator structure that is used for determining where this resource is situated. Locators are described in Section 2.3.
Number of functional dependencies	4	20	Number of functional dependency descriptors. Must be zero if this resource has no functional dependency.
Functional dependency list	N	24	List of functional dependency descriptors for this resource.

2.2.1 Functional dependencies between MSCs

If an MSC has a resource partitioning control whose operation is dependent on or is influenced by resource in another MSC, then that dependency must be described using the functional dependency descriptor. MSC dependencies are explained in Section 2.5.

Table 8: Functional dependency descriptor

Field	Byte length	Byte offset	Description
Producer	4	0	Describes the producer resource that influences the operation of this resource. This field must be set to the Identifier field of the producer resource node.

Field	Byte length	Byte offset	Description
Reserved	4	4	Reserved, must be zero.

2.2.2 Empty MSC node

An empty MSC node has no resource nodes, and its number of resource nodes is set to 0.

If an MSC is described using an empty MSC node, its resources are still discoverable, but without a firmware description the OS is expected to program the controls to be unrestricted (e.g. all 1s for a CPOR bitmap) for any PARTID that it is using.

MSCs whose resources are not located on known components must be described using the empty MSC node. This allows the OS to discover the supported range of PARTID and PMG on that MSC and program all controls to be unrestricted.

2.3 Locators

Locators are a generic construct that specify where a component is situated in the system.

The Location field of an MPAM resource node specifies its location. The resource node is described in Table 7.

The format of the Location fields is based on the Locator structure, described in Section 2.3.2.

The location is expressed in terms of the ACPI identifier of the component that the resource instance is associated with.

2.3.1 Location types

Table 9: Location types

Type	Description
0x00	Processor cache
0x01	Memory
0x02	SMMU
0x03	Memory-side cache
0x04	ACPI device
0x05-0xFE	Reserved for future use by this specification.
0xFF	Unknown

2.3.2 Locator structure

Table 10: Locator structure

Field	Byte length	Byte offset	Description
Descriptor1	8	0	Primary ACPI description of the location, detailed in Section 2.3.3.
Descriptor2	4	8	Secondary ACPI description of the location, also detailed in Section 2.3.3.

2.3.3 Location descriptors

2.3.3.1 Processor cache locator descriptor

The processor cache locator descriptor are associated with a Location type value of 0x00.

For a resource node, this descriptor points to the processor cache that contains the cache storage being used as the MPAM resource.

The format of this locator is outlined in Table 11.

The cache locator is described in terms of the cache's topological position within the processor hierarchy. The position is derived from the ACPI PPTT table [2] , and is expressed as a reference to the cache (Type 1) structure within the PPTT table that describes the cache.

This locator works with PPTT tables of revisions 3 and above.

Table 11: Processor cache locator descriptor

Field	Byte length	Byte offset	Description
Cache reference	8	0	This field must match the Identifier field of the PPTT Type 1 structure that describes this cache.
Reserved	4	8	Reserved, must be zero.

2.3.3.2 Memory locator descriptor

The memory locator descriptor is associated with a Location type value of 0x01.

Memory bandwidth associated with memory regions can be managed as an MPAM resource. Memory bandwidth resources are located using the memory locator.

The format of the memory locator is outlined in Table 12.

The locator is expressed in the form of the proximity domain associated with the memory region.

It is assumed that the memory bandwidth applies to all memory locations within the proximity domain.

The SRAT table is a prerequisite and must be present.

Table 12: Memory locator descriptor

Field	Byte length	Byte offset	Description
Proximity domain	8	0	Proximity domain associated with the memory region.

Field	Byte length	Byte offset	Description
Reserved	4	8	Reserved, must be zero.

2.3.3.3 SMMU locator descriptor

The SMMU locator descriptor is associated with a Location type value of 0x02.

An SMMU [3] implementation can support MPAM through built-in MSCs that manage MPAM resources within the SMMU.

Examples of internal units in an SMMU that can be managed as MPAM resources are IO TLBs and translation caches in the SMMU itself, e.g., in the control unit of the SMMU.

For resource nodes, the SMMU locator describes the location of MPAM resources within the SMMU.

The format of this locator is outlined in Table 13.

The location information is provided in the form of a reference to the IORT[4] node that describes the internal unit in the SMMU.

Table 13: SMMU locator descriptor

Field	Byte length	Byte offset	Description
SMMU interface	8	0	The identifer field of the ACPI IORT table node that describes the SMMU or SMMU interface. This node only supports IORT tables of revisions 3 and above.
Reserved	4	8	Reserved, must be zero.

2.3.3.4 Memory-side cache locator descriptor

The Memory-side cache locator descriptor is associated with a Location type value of 0x03.

Memory-side caches operate as the front-end for memory devices. These caches are distinct from processor caches. Memory-side caches and their properties are defined in the ACPI HMAT table. The HMAT table is mandatory for this locator descriptor.

Memory-side caches in a system can function as MPAM resources that are managed by MSCs.

Memory-side caches are located by the memory-side cache locator.

The format of the locator is outlined in Table 14.

Table 14: Memory-side cache locator descriptor

Field	Byte length	Byte offset	Description
Reserved	8	0	Reserved, must be zero.
Reference	4	8	If the HMAT table is present, then this field points to the HMAT Type 2 structure that describes this memory-side cache.

2.3.3.5 ACPI device locator descriptor

The ACPI device locator descriptor is associated with a Location type value of 0x04.

A system might have various other components that lie in the path of memory transactions. Such components might support MPAM in the form of implementation-defined resources that are managed by MSCs.

Implementation-defined resources must be managed using the implementation-specific controls described in this section. Implementation-specific resources are managed by specific software drivers. The resources must be described as an ACPI device to assist these drivers to bind to their respective resources.

The ACPI device locator is used for describing such resources.

The format of the locator is outlined in Table 15.

Table 15: ACPI device locator descriptor

Field	Byte length	Byte offset	Description
ACPI Hardware ID	8	0	ACPI _HID of the device object in ACPI namespace that describes the component that is associated with this resource.
ACPI Unique ID	4	8	ACPI _UID of the device object in ACPI namespace that describes the component that is associated with this resource.

2.3.3.6 Unknown locator descriptor

The Unknown locator is associated with a Location type of 0xFF. This locator is useful for describing resources that are located in components that cannot be described in ACPI. An MPAM-enabled buffer in the SoC interconnect that contains an MSC, is an example of a component that cannot be described in ACPI. The buffer must hence be declared as an unknown resource.

2.4 MSC groups

Some MSCs in a system might be organized as a logical group. All MSCs of a logical group must be configured identically because they affect the same, common underlying component or its operation. OS can recognize the presence of a logical group through examination of the Location field of the resource nodes of the MSC nodes in the group. Each MSC node in the group must have at least one resource node that has its Location field set to the same common location.

Note

MSC groups and MSC functional dependencies are different, mutually independent concepts. Functional dependencies are explicitly specified in the functional dependency list of resource nodes and involve two differently-located MSCs. Secondly, functional dependencies are directional in nature: from the producer to the dependent. OSPM is required to understand these aspects and configure the MSCs accordingly.

2.5 MSC functional dependencies

MSCs in a system have implicit dependencies that follow the natural flow of memory transactions in a system. For example, the control action of an MSC in an L2 cache implicitly affects the monitored data or control action in a downstream L3 cache. Since these inter-dependencies are implicit, they need not be described in ACPI.

It is possible for MSCs to have explicit functional dependencies on other MSCs that are against the natural dataflow. Such dependencies might be of the form:

- the control policy applied on a first MSC, called the dependent MSC, might be influenced by the output of a monitor on a second MSC, called the producer MSC.
- the control policy applied on a dependent MSC might be influenced by the control policy settings of a producer MSC.

In both cases, the dependent MSC must not be the immediate downstream neighbour of the producer MSC, or in a position that falls in the natural flow of memory transactions from the producer MSC to the dependent MSC.

An example of a functional dependency would be a system that has an L3 cache that is an inclusive cache. Policies applied to the L3 cache, e.g., on eviction rates, could thus influence policies applied on the upstream L2 cache. The L2 cache is then said to be *functionally dependent* on the L3 cache.

A functional dependency is expressed as the tuple: {Producer MSC, Producer unit, Dependent MSC, Dependent unit} where:

- the Producer unit is a resource control in the producer MSC
- the Dependent unit is a resource control in the dependent MSC

MSC dependencies are expressed in resource nodes. See Section 2.2. The ACPI description of explicit dependencies helps the OS to make judicious decisions while programming the MSCs for MPAM functionality.

2.6 Representation of MSCs as ACPI devices

The MSC is a logic block or unit that has its own memory-mapped register space and interrupts. These characteristics make the MSC appear as a device to software, even though the MSC hardware might be a part of another device.

The MSC can thus be described optionally as a device object in ACPI namespace. The device model is useful for declaring requirements and operations for an MSC in addition to those that are presented through the MSC node in the MPAM table. Specifically, the device model is useful for supporting power management of the MSC, where the power state dependencies of the MSC can be expressed in ACPI, and additional requirements and programming relevant to power state transitions can be supplied through ACPI _PSx methods. See [2] for more information on these methods.

Note

MSCs must mandatorily be described in the MPAM table as the MSC node provides the canonical properties of the MSC. The device object representation should be considered as an optional auxilliary extension to the MSC node.

2.6.1 ACPI Hardware Identifiers

MSC devices in ACPI are identified using the _HID of “ARMHAA5C”. If present, the _CID object of the device must also be set to the same value.

Note that the MSC might be implemented as a sub-component within another component. The device view of the MSC has no relation to the container component which might be modeled as a separate ACPI device with its own _HID.

2.6.2 Unique Identifier

Each MSC device object must be declared with a distinct _UID value that must be set to the same value as the Identifier field of the MSC node in the MPAM table that describes the MSC.

2.6.3 Example MSC device

The following ASL code describes the power management requirements for an example MSC.

```
Device(MSC0) { // MSC device

      Name(_HID, "ARMHAA5C")
      Name(_CID, "ARMHAA5C")
      Name(_UID, 0)
      Name(_STR, Unicode("MSC0"))
}
```

2.7 MSI support

If the MSC supports MSI generation for its overflow or error interrupt, then it must be described as a device object in the DSDT. This device object can then be presented as being a part of the IO topology of the system. The OS can then walk the IO topology to determine the target ITS unit or group of units that the MSIs from the MSC are routed to. The OS can also use this information to program the MSI registers for this MSC. More information on IO topologies is available in [4].

2.8 MSC Power Management

If an MSC has platform-specific programming requirements for power management, one approach to describing those requirements would be to represent the MSC in ACPI namespace as a device object. The device object description supplements the information in the MSC node that describes the MSC, as explained in Section 2.6.

The device object allows ACPI methods to be declared for specifying runtime power management properties of the MSC.

The following ASL code snippet illustrates how power management requirements for an example MSC is declared in DSDT.

```
Device(MSC0) { // MSC device

      Name(_HID, "ARMHAA5C")
      Name(_CID, "ARMHAA5C")
      Name(_UID, 0)
      Name(_STR, Unicode("MSC0"))

      Method (_PS3, 0, Serialized) {  // _PS3: Power State 3
            // Turn MSC off
      }

      Method (_PS0, 0, Serialized) {  // _PS0: Power State 0
            // Turn MSC on
      }
}
```

2.8.1 MSC power state dependencies and accessibility

In this specification, the device where the MSC is located, or that the MSC is associated with, is called the linked device.

The MSC and its linked device might have power state dependencies. For example, the MSC and its linked device might share power resources. As a result, the MSC might enter low-power state and become inaccessible if the linked device enters low-power state.

If there is a power state inter-dependency between the MSC and its linked device, then:

- when transitioning to low-power state, the MSC and its linked device must enter low-power state together
- when transitioning into ON state, the MSC must enter running state *before* the linked device

If the MSC is modeled as an ACPI device, the OSPM can orchestrate power management of the MSC and its linked device, provided the firmware description includes an ACPI device representation for the MSC with the following:

- standard ACPI methods, _PSx, for managing power states as specified in Section 2.8.
- the link between the MSC and the linked device. See Section 2.8.2.

This version of the specification focuses only on device linkage between MSCs and processors or processor clusters. Section 2.8.3 provides more details.

2.8.2 Describing MSC device linkage

The linked device is specified in the MSC node in the form of the ACPI identifiers for the linked device.

2.8.3 MSCs associated with PEs and CPU affinity

MSCs that are associated with PEs, such as MSCs on processor caches or in CPU clusters can be placed in low-power state when the associated PE or cluster transitions to low-power state. PEs or processor clusters enter low-power state in response to PSCI requests from the OS, notably CPU_SUSPEND. As such, low-power state entry of PEs is strictly the responsibility of the PSCI firmware and the OS has no control over these state transitions.

Hence, Arm strongly recommends that MSCs associated with PEs are power managed entirely by PSCI firmware.

MSCs on PEs or clusters are accessible only if the associated PE or processor cluster is in the running state. CPU affinity is specified in the MSC node. The OS must use this affinity information to determine the PE or set of PEs that needs to be woken up before the MSC can be accessed.

3 Appendix

3.1 Example System with MPAM

Figure 1 is an example system with MPAM support implemented in the form of MSCs distributed across various system memory components.

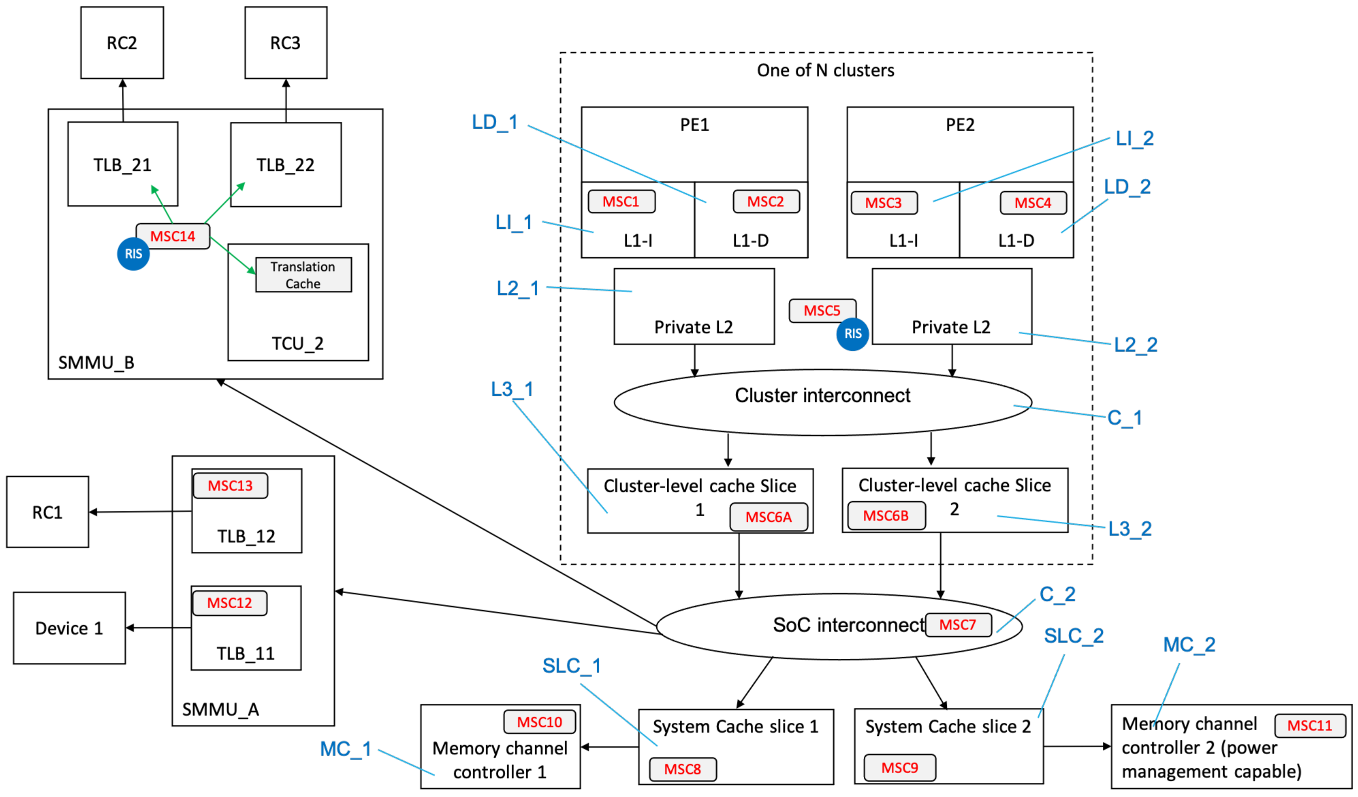

Figure 1: An example system with MPAM support

The example system is intended to illustrate how the MPAM description of a system might be provided in ACPI. It showcases the use of location of MSCs for power management, the association of MSCs to resources, how resources are defined and located, and special cases that highlight use of features such as RIS.

Note

This example system is intended to provide an illustration of how a reasonably complex SoC with MPAM support could be represented in ACPI. It must not be considered as a reference design for building MPAM support in a system.

The ACPI description of the MPAM properties of this example system is outlined in the sections that follow.

3.1.1 Notations

Single alphabetical letters in italics are used to represent numeric values assigned to ACPI identifiers: *a* to *z* and *A* to *Z*.

3.1.2 Prerequisites

3.1.2.1 The ACPI PPTT table

All processor caches in the example system must be described in the ACPI PPTT table.

PPTT Table
Header
...
Cache structures
...

Cache structure for LI_1	
...	...
Cache ID	*m*

Cache structure for LD_1	
...	...
Cache ID	*n*

Cache structure for LI_2	
...	...
Cache ID	*p*

Cache structure for LD_2	
...	...
Cache ID	*q*

Cache structure for L2_1	
...	...
Cache ID	*r*

Cache structure for L2_2	
...	...
Cache ID	*s*

Cache structure for L3_1 and L3_2	
...	...
Cache ID	*t*

The two slices of the L3 cache appear to software as a single monolithic L3 cache in this particular example. Hence, they are described using a single, common Type 1 cache structure in the PPTT as above.

3.1.2.2 The ACPI SRAT table

SRAT	
...	...
MC_1 memory range	Proximity Domain = *P*
MC_2 memory range	Proximity Domain = *Q*

3.1.2.3 The ACPI HMAT table

HMAT	
SLC_1	Proximity Domain = *P*
SLC_2	Proximity Domain = *Q*

3.1.2.4 IORT table

The SMMUs, Device 1, and the root complexes are described using the ACPI IORT table.

IORT Table
Header
SMMU node 1
SMMU node 2
Root complex node 1
Root complex node 2
Root complex node 3
Named component node 1
...

SMMU node 1

The SMMU node 1 describes SMMU_A in the example system.

SMMU node	
. . .	. . .
Identifier	*i*
. . .	. . .

SMMU node 2

The SMMU node 2 describes SMMU_B in the example system.

SMMU node	
. . .	. . .
Identifier	*j*
. . .	. . .

Root complex node 1

RC node	
. . .	. . .
Identifier	*A*
. . .	. . .

Root complex node 2

RC node	
. . .	. . .
Identifier	*B*
. . .	. . .

Root complex node 3

RC node	
. . .	. . .
Identifier	*C*
. . .	. . .

Named component node 1

This named component node describes the properties of Device 1 in the example system.

NC node	
. . .	. . .
Identifier	*D*
. . .	. . .

3.1.3 The ACPI description of the CPU topology

The CPU topology is described in the PPTT. However, to avoid using pointers to nodes in the PPTT, the MPAM table uses the ACPI description of processors and processor containers in DSDT to cross-reference the latter.

The following DSDT reference code provides the ACPI description of the CPU topology for the example system.

```
Device (SYST) { // (Top-) System-level processor container

   Name (_HID, "ACPI0010")
   Name (_UID, 0)

   Device (CLU0) { // CPU Cluster

      Name (_HID, "ACPI0010")
      Name (_UID, 1)

      Device ( CPU0 ) { // PE0
         Name (_HID, "ACPI0010")
         Name (_UID , 1)
         ...
      } // CPU0 description ends here

      Device ( CPU1 ) { // PE1
         Name (_HID, "ACPI0010")
         Name (_UID , 2)
         ...
      } // CPU1 description ends here
   }
}
```

3.1.4 The ACPI MPAM table

The MPAM table must describe every MSC in the system to aid software discovery of the MSCs.

MPAM Table	
Signature	“MPAM”
. . .	. . .
MSC node 1	. . .

MPAM Table	
MSC node 2	. . .
. . .	. . .
MSC node 14	. . .

In this table, MSC node *i* is the descriptor for MSC*i*.

3.1.5 The L1 caches

The example L1 caches in Figure 1 are intended to illustrate the simplest manner in which MPAM support might be built in a cache, and how controls in an MSC are associated with resources.

Figure 2 provides details of MSCs MSC1-4.

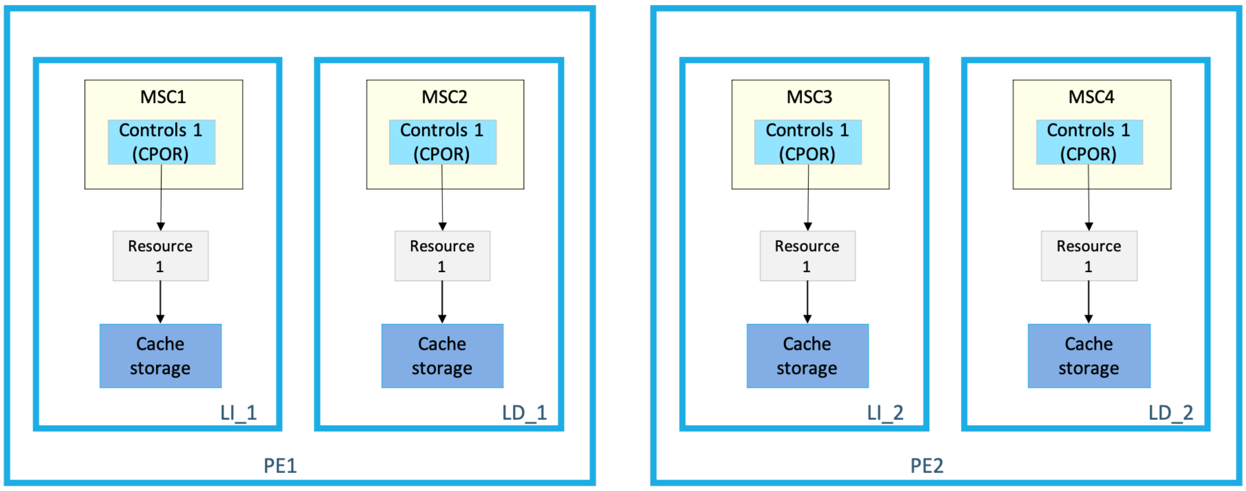

Figure 2: Internal organization of the L1 caches in the example system

The controls implemented by each MSC are considered to relate directly to the cache storage in the cache. Since the MSCs do not support RIS, there is a single resource instance only, and the RIS index field is set to 0 accordingly. Note that the MPAMF_IDR.HAS_RIS bit of each MSC must return 0 on read.

Accordingly, the MSC nodes for MSC1-4 have a single resource node each that describes the cache storage part of the cache as a resource instance.

In this particular case, the control and resource types both pertain to cache storage, which is based on the association outlined in Table 6.

MSC node 1	
Identifier	0
. . .	. . .
Linked device ID (_HID, _UID)	“ACPI0007”, 0x01 (CPU0)

MSC node 1	
Number of resource nodes	1

Resource node 11	
RIS Index	0 (Don’t care)
Locator descriptor	Type = 0x00 (Processor cache) Cache reference = *m*

MSC node 2	
Identifier	1
. . .	. . .
Linked device ID	“ACPI0007”, 0x01 (CPU0)
Number of resource nodes	1

Resource node 21	
RIS Index	0 (Don’t care)
Locator descriptor	Type = 0x00 (Processor cache) Cache reference = *n*

MSC node 3	
Identifier	2
. . .	. . .
Linked device ID	“ACPI0007”, 0x02 (CPU0)
Number of resource nodes	1

Resource node 31	
RIS Index	0 (Don’t care)
Locator descriptor	Type = 0x00 (Processor cache) Cache reference = *p*

MSC node 4	
Identifier	3
. . .	. . .
Linked device ID	“ACPI0007”, 0x02 (CPU0)
Number of resource nodes	1

Resource node 41	
RIS Index	0 (Don’t care)
Locator descriptor	Type = 0x00 (Processor cache) Cache reference = *q*

3.1.5.1 Power management

Power management of MSC1-4 must be performed by PSCI firmware as they are associated with PEs. The Locator fields of the MSC nodes describe the CPU affinities of the MSCs.

3.1.6 L2 caches on PE1 and PE2

Figure 3 provides an overview of MSC5, which operates on the L2 caches of both PEs, L2_1 and L2_2, respectively.

MSC5 is shown to have three partitioning controls: CPOR, CCAP and PRI. The CPOR and CCAP controls operate on cache storage resources, while the PRI controls operate on incoming DVM messages.

MSC5 is enabled with the RIS feature to enable it to support multiple controls of the same type that operate on different resource instances of the same resource type.

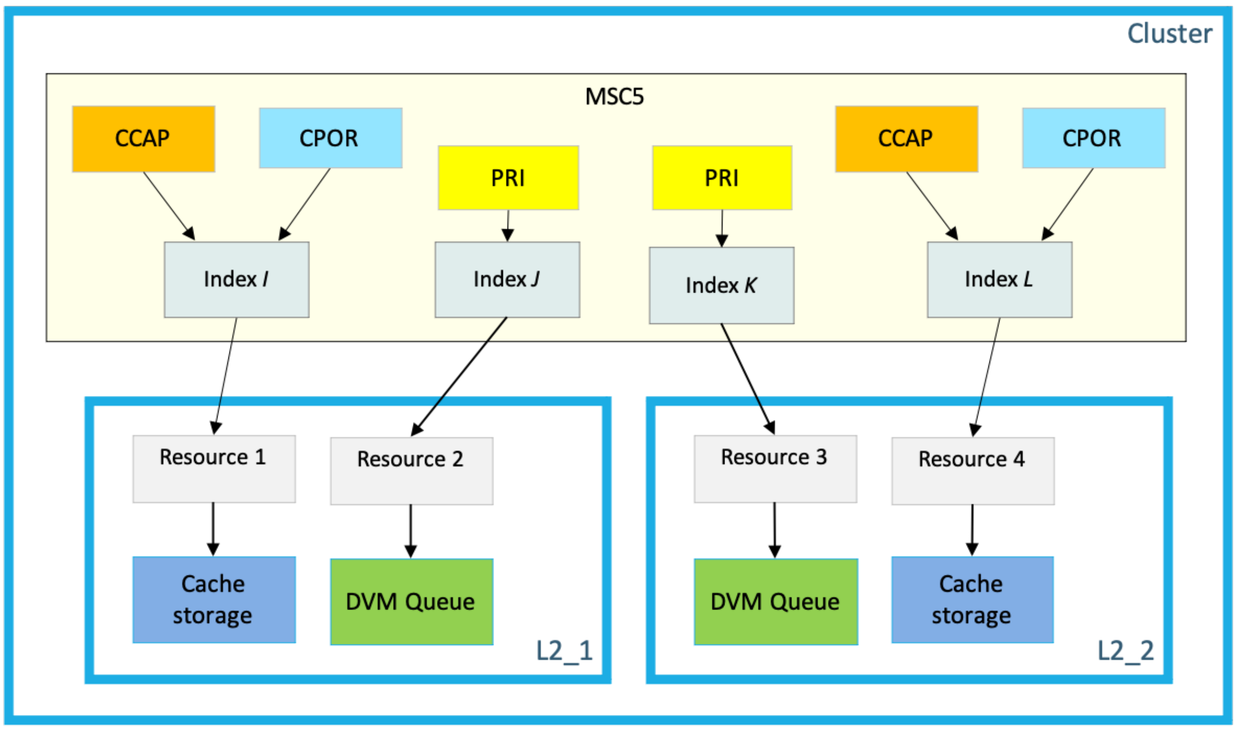

Figure 3: Organization of the common MSC for the L2 caches in the example system

MSC node 5	
Identifier	4
. . .	. . .
Linked device ID	“ACPI0010”, 0x01 (CLU0)
Number of resource nodes	4

Resource node 51	
RIS Index	*I*
Locator descriptor	Type = 0x00 (Processor cache) Cache reference = *r*

Resource node 52	
RIS Index	*J*

Resource node 52	

Locator descriptor	Type = 0xFF (Unknown)
	Reference = 0

Resource node 53	
RIS Index	*K*

Locator descriptor	Type = 0xFF (Unknown)
	Reference = 0

Resource node 54	
RIS Index	*L*
Locator descriptor	Type = 0x00 (Processor cache) Cache reference = *s*

3.1.6.1 Power management

Power management of MSC5 must be performed by PSCI firmware as it is associated with processor caches.

The CPU affinity of MSC5 is described by its Locator field.

3.1.7 L3 cache on cluster C_1

Figure 4 shows the internal layout of the L3 cache and its MPAM resources.

The example L3 cache is divided into two physical slices, L3_1 and L3_2 that appear as a monolithic single cache storage to the OS. Each cache slice is managed by a dedicated MSC.

Each slice has an MSC associated with it, as depicted in Figure 4. In accordance with the design rules assumed, both sets of controls, CCAP and CPOR, of each MSC, apply to the same resource instance - the cache storage part of the cache slice.

Since MSC6A and MSC6B are both operating on the same monolithic cache, both MSCs must be configured identically to allow uniform partitioning for all PARTIDs, and are considered as being part of an MSC group. MSC groups are explained in Section 2.4..

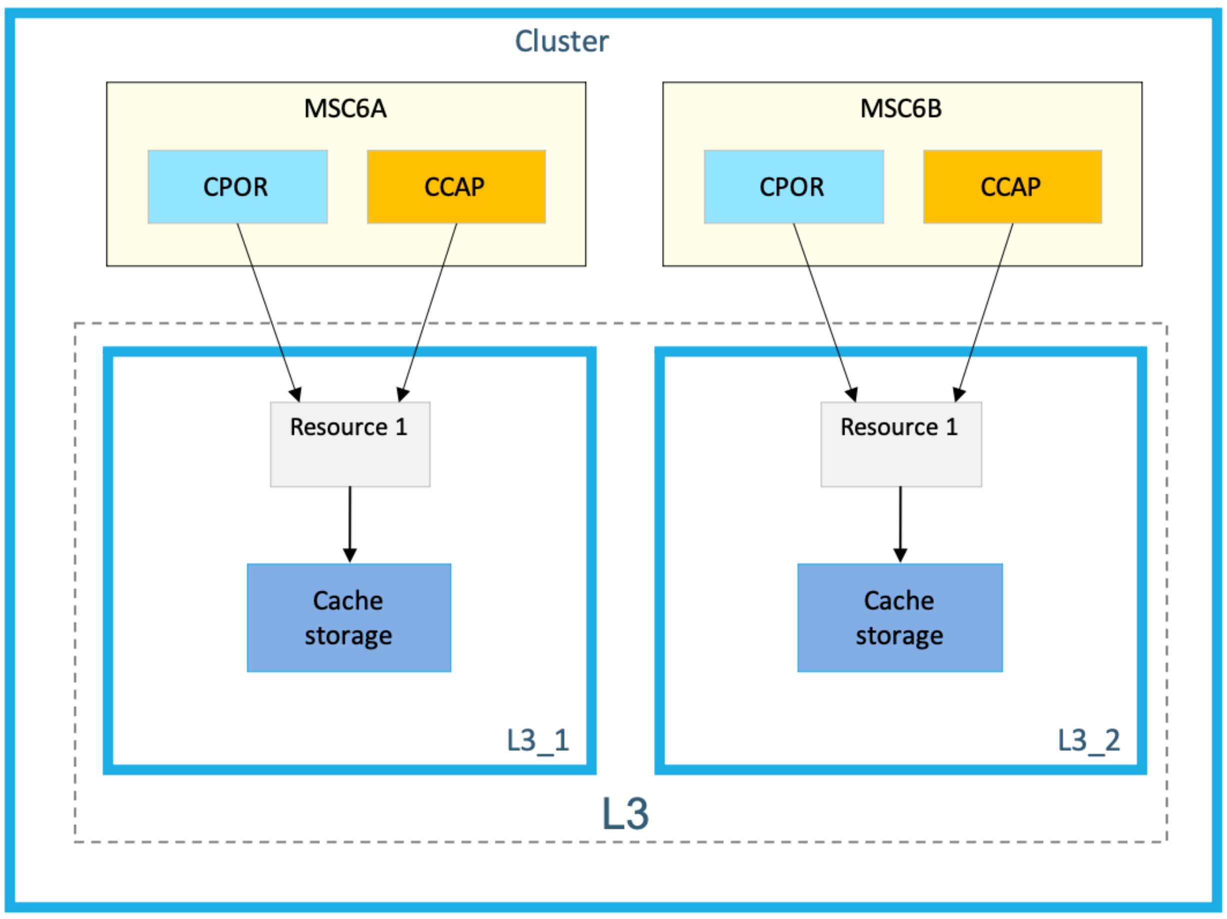

Figure 4: L3 cache in the example system

The combined cache storage of the L3 cache is located using the ACPI PPTT table, as explained in Section 2.3.3.

MSC node 6A	
Identifier	0x6A
. . .	. . .
Linked device ID	"ACPI0010", 0x01 (CLU0)
Number of resource nodes	1

Resource node 6A1	
RIS Index	0 (Don't care)
Locator descriptor	Type = 0x00 (Processor cache) Cache reference = *t*

MSC node 6B	
Identifier	0x6B
. . .	. . .
Linked device ID	“ACPI0010”, 0x01 (CLU0)
Number of resource nodes	1

Resource node 6B1	
RIS Index	0 (Don’t care)
Locator descriptor	Type = 0x00 (Processor cache) Cache reference = *t*

3.1.7.1 Logical Grouping

MSC nodes for MSC6A and MSC6B have resources pointing to the same resource, the common cache storage that is part of the L3 cache. This is an indication for OSPM that the two MSCs are logically related and should be configured identically. MSC groups are explained in Section 2.4.

3.1.7.2 Power management considerations

Power management of MSC6A and MSC6B is aligned with that of the L3 cache, which in turn is managed by PSCI. The Locator fields of both MSC nodes point to cluster 0 to indicate that they are associated with all PEs that are part of this cluster.

3.1.8 SoC Interconnect

MSC7 in the SoC interconnect provides priority partitioning for requesters that interface with the interconnect.

Since the SoC interconnect is invisible to software, resources in MSC7 cannot be located. As a result, the MSC is described using an empty MSC node. This helps the OS discover the MSC and configure it with unrestricted access for all PARTIDs, as required in Section 2.2.2.

MSC node 7	
Identifier	6
. . .	. . .
Linked device ID	“ACPI0010”, 0x00 (CLU0)
Number of resource nodes	0

3.1.8.1 MSC Accessibility

MSC7 is present at the system level. Its CPU affinity can therefore be optionally described using its Locator field.

3.1.9 The SLC

Each SLC slice in Figure 1 has a dedicated MSC that allows it to be configured independently for MPAM. As described in Section 2.3.3.4, the slice is considered as a memory-side cache and is associated with the ACPI proximity domain of memory regions that use the SLC slice as a front-end cache. In this example system, SLC_1 acts as the memory-side cache for the proximity domain in which the memory ranges of MC_1 lie. Likewise, SLC_2 functions as the memory-side cache for the proximity domain associated with MC_2.

MSC node 8	
Identifier	7
. . .	. . .
Linked device ID	0, 0 (Not valid)
Number of resource nodes	1

Resource node 81	
RIS Index	0 (Don't care)
Locator descriptor	Type = 0x03 (Memory-side cache) HMAT reference = *P*

MSC node 9	
Identifier	8
. . .	. . .
Linked device ID	0, 0 (not valid)
Number of resource nodes	1

Resource node 91	
RIS Index	0 (Don't care)
Locator descriptor	Type = 0x03 (Memory-side cache) HMAT reference = *Q*

3.1.10 Memory channel controllers

Figure 5 shows how MSCs MSC10 and MSC11 are designed. In particular, both offer a single set of controls for partitioning memory bandwidth on the channels.

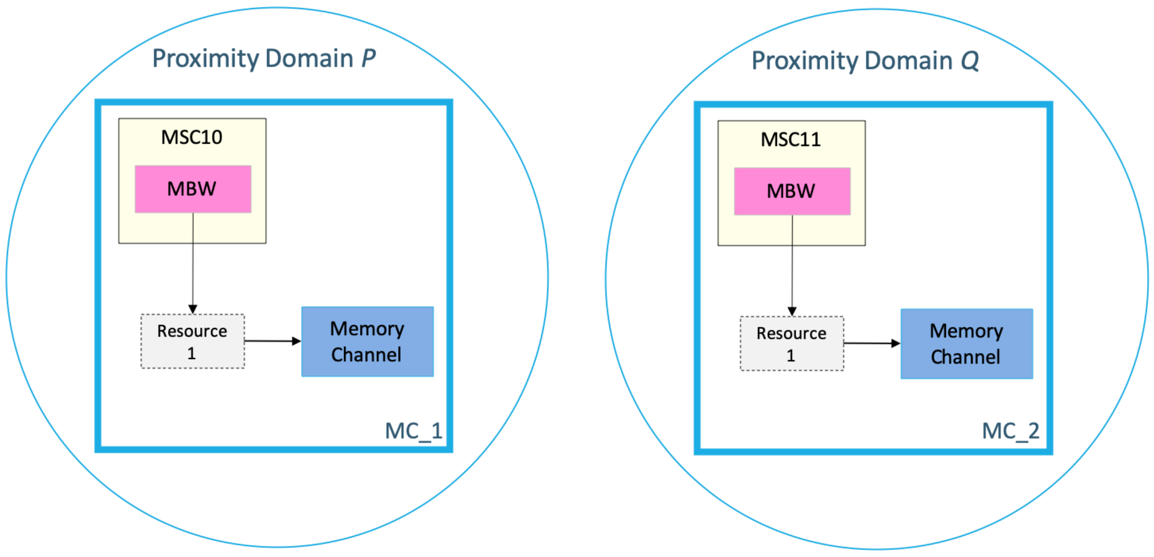

Figure 5: Memory channel controllers in the example system

The Locator reference for MC_1 and MC_2 is specified in the form of their proximity domains, which requires that the ACPI SRAT table must include a description of the memory ranges of these memory controllers. Section 3.1.2.2 provides a snapshot of the SRAT table to be used for the example system.

MSC node 10	
Identifier	9
. . .	. . .
Linked device ID	0, 0 (not valid)
Number of resource nodes	1

Resource node 101	
RIS Index	0 (Don't care)
Locator descriptor	Type = 0x01 (Memory) SRAT reference = *P*

MSC node 11	
Identifier	10
. . .	. . .
Linked device ID	“PNP0C80”, 17 (MEM2)
Number of resource nodes	1

Resource node 111	
RIS Index	0 (Don't care)
Locator descriptor	Type = 0x01 (Memory) SRAT reference = *Q*

3.1.10.1 Power management considerations

MSC11 has a power management dependency on memory controller MC_2, its linked device. MC_2 is capable of being offlined, and indicates support for it in its ACPI device object definition.

The reference ASL code below describes MC_2 as an ACPI device named MEM2, with a _UID of 17.

The MSC device object includes the _PSx methods that OSPM can invoke to perform power management of the MSC.

```
Device (MEM2) { // Memory controller 2, MC_2
    Name (_HID, "PNP0C80")
    Name (_UID, 17)

    Method (_PS3, 0, Serialized) {  // _PS3: Power State 3
          // Turn MC off
    }

    Method (_PS0, 0, Serialized) {  // _PS0: Power State 0
          // Turn MC on
    }
}

Device(MS11) {
    Name(_HID, "ARMHAA5C")
    Name(_CID, "ARMHAA5C")
    Name(_UID, 10) // _UID for MSC11, must be identical
                   // to the Identifier field of the MSC
                   // node.

    Name(_STR, Unicode("MSC11"))

    Method (_PS3, 0, Serialized) {  // _PS3: Power State 3
          // Turn MSC off
    }

    Method (_PS0, 0, Serialized) {  // _PS0: Power State 0
          // Turn MSC on
    }
}
```

3.1.11 SMMU_A

Figure 6 illustrates how SMMU_A is organized internally. shows an illustrative SMMU design that supports MPAM. The SMMU has two IO TLB units, TLB_11 and TLB_12, that service Device 1 and PCIe root complexes RC1, respectively.

Each TLB has a dedicated MSC. The TLB is identified using the Identifier field of the component that it interfaces. TLB_11 is thus identified using the Identifier assigned to the IORT Named component node that describes Device 1.

TLB_12 is identified using the Identifier given to the IORT root complex node that describes RC. These nodes and their identifiers are explained in Section 3.1.2.4.

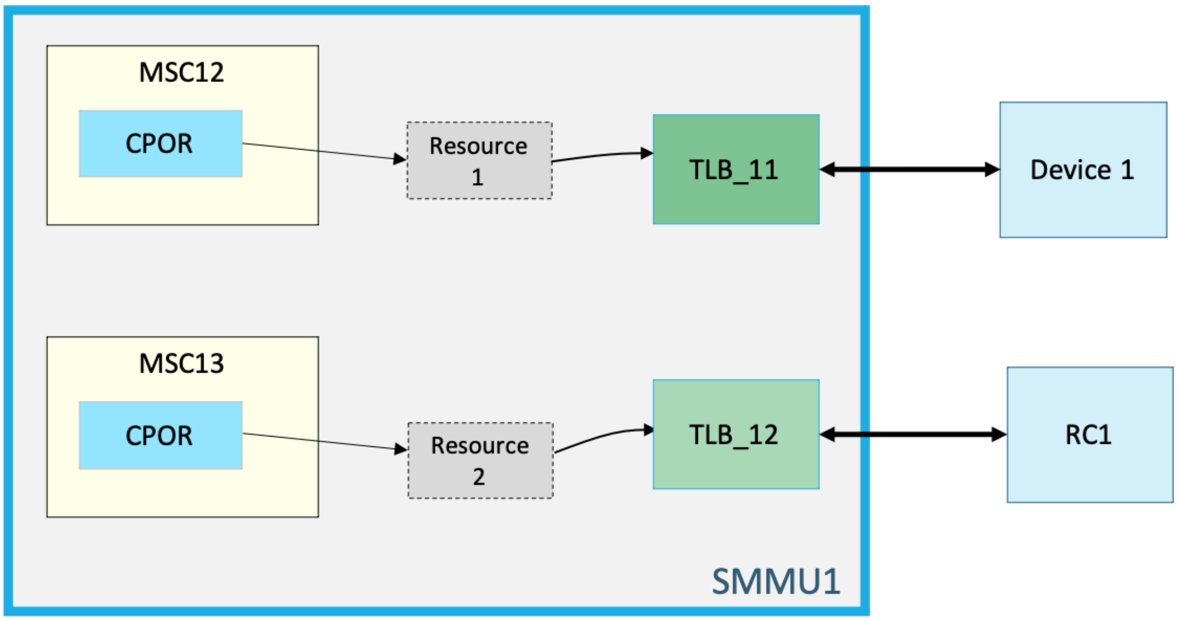

Figure 6: SMMU_A in the example system

The ACPI description of the MSCs in SMMU_A is as follows:

MSC node 12	
Identifier	11
. . .	. . .
Linked device ID	0, 0 (not valid)
Number of resource nodes	1

Resource node 121	
RIS Index	0 (Don't care)
Locator descriptor	Type = 0x02 (SMMU) IORT reference = *D*

MSC node 13	
Identifier	12
. . .	. . .

MSC node 13	
Linked device ID	0, 0 (not valid)
Number of resource nodes	1

Resource node 131	
RIS Index	0 (Don't care)
Locator descriptor	Type = 0x02 (SMMU) IORT reference = *A*

3.1.12 SMMU_B

Figure 7 illustrates the internal layout of the MSCs in SMMU_B.

SMMU_B differs from SMMU_A in that it has a single MSC, MSC14, that implements the RIS feature to provide support for the independent translation caches in TLB_21, TLB_22 and the TCU.

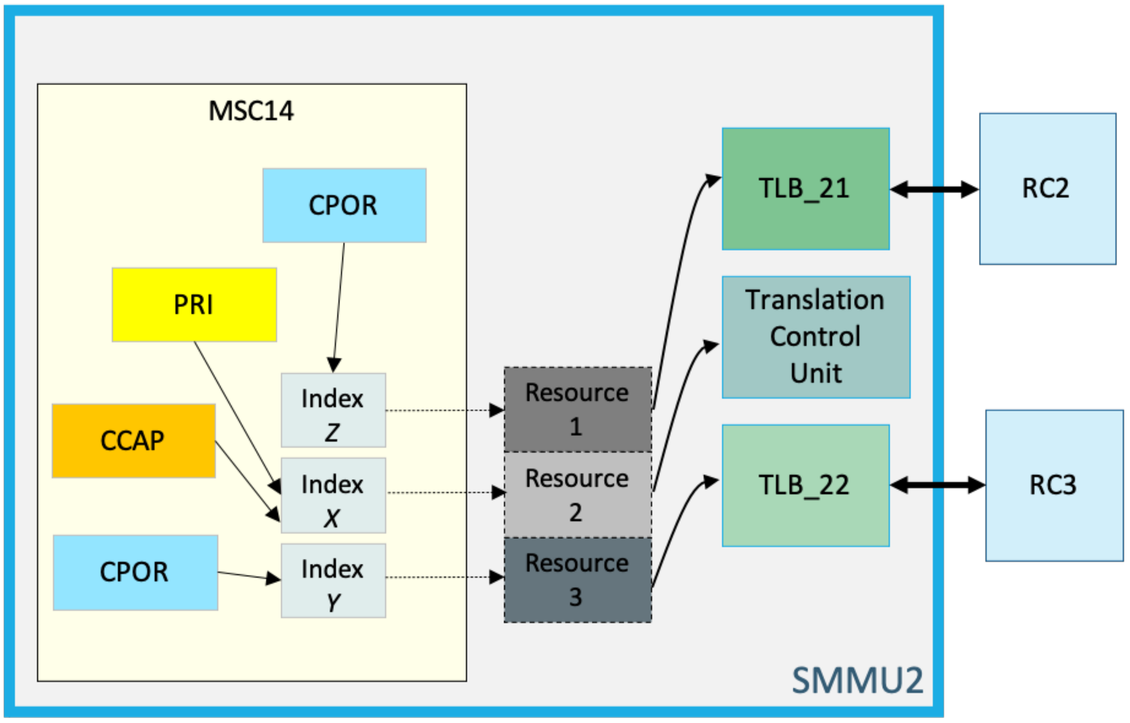

Figure 7: SMMU_B in the example system

The ACPI description of the MSC in this SMMU can now be performed as follows.

MSC node 14	
Identifier	13

MSC node 14	
...	...
Linked device ID	0, 0 (not valid)
Number of resource nodes	3

Resource node 141	
RIS Index	*Z*
Locator descriptor	Type = 0x02 (SMMU) IORT reference = *B*

Resource node 142	
RIS Index	*X*
Locator descriptor	Type = 0x02 (SMMU) IORT reference = *j*

Resource node 143	
RIS Index	*Y*
Locator descriptor	Type = 0x02 (SMMU) IORT reference = *C*