MPAM Firmware-backed (Fb) Profile 0.1

Platform Design Document

Non-confidential

ALP0

This is an engineering draft of the specification. It is meant to obtain feedback from Arm partners and internally within Arm. It is subject to change based on this feedback.

Document number: DEN0144

Contents

Release information

Date	Version	Changes
2023/Dec/06	0.1	• First public Alpha release

About this document

Terms and abbreviations

Term	Meaning
ACPI	Advanced Configuration and Power Interface specification
MPAM	Memory System Resource Partitioning And Monitoring
MPAM-Fb	MPAM Firmware-backed
MSC	Memory System Component
PMG	Performance Monitoring Group
SCMI	System Control and Management Interface
SMMU	Arm System Memory Management Unit
TLB	Translation Look-aside Buffer

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 A-profile architecture, Arm DDI 0598*. Arm Limited.

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

[3] *Arm® System Control and Management Interface, Platform Design Document, Arm DEN 0056*. Arm Limited.

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

[5] *ACPI for Memory System Resource Partitioning and Monitoring, Platform Design Document, Arm DEN 0065*. Arm Limited.

Feedback

Arm welcomes feedback on its documentation.

If you have comments or suggestions for additions and improvements, please create a ticket at https://support.developer.arm.com. As part of the ticket please include:

- The title (MPAM Firmware-backed (Fb) Profile).
- The document ID and version (DEN0144 0.1).
- The section name to which your comments refer.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

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

Inclusive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive terms. If you find offensive terms in this document, please contact terms@arm.com.

1 Introduction

This document specifies the Firmware-backed profile of the Arm Memory System-Resource Partitioning and Monitoring extension, MPAM [1].

1.1 Overview of MPAM

MPAM provides mechanisms to limit and to monitor the usage of memory system resources among competing software entities. Memory access requests from requesters like a Processing Element (PE) are handled by Memory-system Components or MSCs. System MMUs, caches, TLBs and memory controllers are examples of an MSC. An MPAM compliant MSC provides partitioning and monitoring controls for managing system memory resources, thereby affecting the performance of memory transactions.

MPAM describes:

- A mechanism for attaching Partition Identifiers (PARTID) and Performance Monitoring Groups (PMG) to software payloads executing on an Arm PE through PE specific System Registers.
- Memory Mapped Interfaces to MSCs for configuring memory resource controls and for monitoring the usage of controlled resources.

2 MPAM Firmware-backed (Fb) Profile

In some systems, the memory mapped interface of an MSC might not be accessible in the physical address spaces of the PE due to implementation or other constraints. The software resident on the PE does not have direct access to such an MSC. Figure 1 shows an example system which consists of a mix of MSCs. Some of the MSCs are memory mapped into the PE physical address space, while some of the MSCs, like MSC 1 and MSC 2, are not.

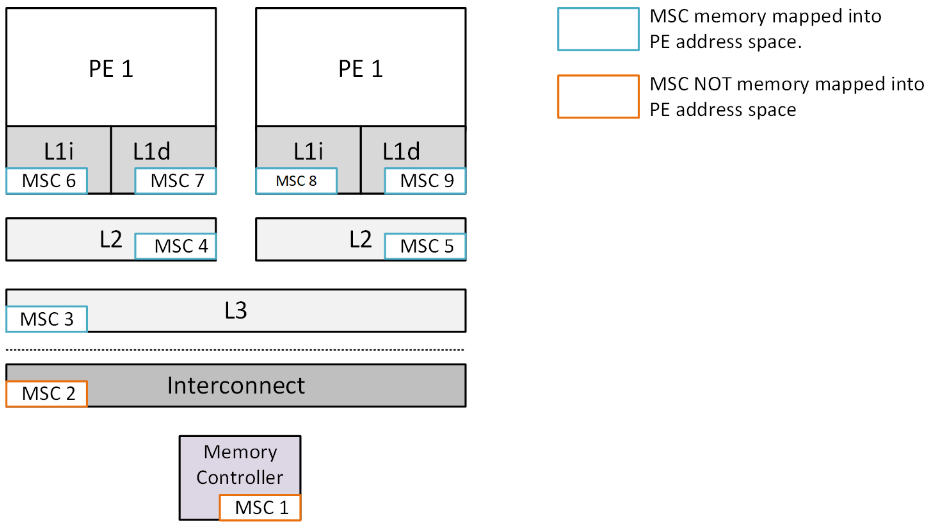

Figure 1: An example system with MSCs not memory mapped into the PE physical address space

The MSCs which are not memory mapped into the PE physical address space are controlled by an entity called the *MPAM Manager* which may not be resident on the PE.

The PE resident software entity responsible for configuring MPAM is referred to as the *agent* in this specification.

The MPAM Firmware-backed (Fb) profile provides a command-response mechanism over a transport interface for an agent to communicate with the MPAM Manager to access the registers of those MSCs which are not memory mapped into the PE physical address space. The architecture governing the use of the MSC registers and their interpretation is specified in the MPAM specification [1].

MPAM Fb profile support is intended to be discovered through firmware decription mechanisms like Advanced Configuration and Power Interface (ACPI) or Flattened Device Tree (FDT).

Note

This specification retrospectively waives the requirement for an MPAM compliant MSC to be mapped into the PE physical address space through memory mapped registers. This waiver will be expressly stated in the MPAM architecture [1] in future revisions.

2.1 Agents

Agents are characterised by the following rules:

- An agent interacts with the MPAM Manager to configure any MSC which is not memory mapped into the PE physical address space.

- An agent cannot span across two different Security states or Exception levels. As an example, an agent resident in the Secure Security state is a distinctly separate entity from an agent resident in the Non-secure Security state. Similarly an agent resident in EL1 is a distinctly separate entity from an agent resident in EL2.
- Each agent has its own dedicated interface to communicate with the MPAM Manager. The same transport interface cannot be shared between two agents.
- An agent should not use the transport interface dedicated for another agent even if it has access to the physical address space (PAS) that the transport interface is mapped into as per the access rules specified in the Arm® Architecture Reference Manual for A-profile architecture [2]. Adherence to this rule is necessary for the MPAM Manager to track which agent is communicating with it.

Determining if an agent is allowed to communicate with the MPAM Manager is an implementation specific decision and is out of the scope of this specification.

2.2 Partitioning memory-system resources across Security states

A memory-system resource can be utilized by memory requests from different Security states. Some systems might require partitioning of the resource between memory requests originating from different Security states. This partitioning can be achieved either with or without co-ordination between agents across Security states. If there is no co-ordination between agents for partitioning the memory-system resource across different Security states, the MPAM-Manager can choose to partition the resource using an implementation defined policy.

2.3 Protocol and Transport

The MPAM Firmware-backed profile provides two levels of abstraction:

- **Protocol** which defines a group of messages to perform various functions. Message format and details are described in Section 3.
- **Transport** which describes the interface for communication between two entities using the protocol messages.

The MPAM Fb Protocol provides commands to:

- Describe the protocol version.
- Discover implementation attributes.
- Read and write to a selected MSC.

It is intended that the protocol and the transport are developed independently.

The exact architecture and definition of the transport is outside the scope of this specification. However any transport mechanism must have the following properties to be compatible with the protocol:

- The transport should be capable of accommodating and transmitting the largest command parameter and return value payload described in this specification.
- The transport should provide a separate communication interface dedicated to each agent which needs to communicate with the MPAM Manager.
- The transport should incorporate an explicit or implicit mechanism to identify the agents and their Security states. This allows the MPAM Manager to identify the agent communicating over the transport interface. An example of an implicit mechanism of identification is a transport interface which is mapped into a single physical address space, and hence only accessible from a specific Security state. This enables the MPAM Manager to pre-determine the agent which is using the interface, along with its possible Security state.

- The transport should be able to provide adequate guarantees on availability and message integrity of any message under transit. It should also specify any recovery mechanisms available in case the guarantees are breached. The exact measure of what constitutes *adequate* is determined by the threat-model of the implementation.

The physical address space (PAS) which the transport interface is mapped into determines the MPAM feature page and the PARTID space accessed in the MPAM Manager, as described in Table 3.

Table 3: PARTID space and MPAM feature page accessed by an agent

Transport Interface PAS	PARTID space	MPAM Feature Page
Non-secure	Non-secure	Non-secure
Secure	Secure	Secure
Realm	Realm	Realm
Root	Root	Root

2.4 Considerations for RME systems

In a system which supports the Arm Realm Management Extension (RME) [2], MSCs might support either a two space region or a four space region.

If any MSC which is not memory mapped into the PE physical address spaces supports a four space region, each Security state should have its own dedicated transport interface. This allows access to all four PARTID spaces and MPAM Feature pages as specified in Table 3.

If an MSC which is not memory mapped into the PE physical address spaces supports a two space region, then:

- If the MSC does not perform MPAM monitoring of Root and Realm memory access requests, the MSC cannot be accessed through the transport interfaces dedicated to the Root or Realm Security states. The MPAM Manager is only required to present those MSCs which are accessible to the agent through the agent specific transport interface. Any accesses to unavailable MSC through the transport interface should return the *NOT_FOUND* status code from the MPAM Manager. Status codes are listed in Section 2.5.1.
- Else, the PARTID space and the MPAM feature page accessed through the transport interface dedicated to the Root and Realm Security state is determined by the four-space to two-space reduction scheme implemented, as specified in the MPAM specification [1].

2.5 Protocol Details

All messages specified by the MPAM Fb protocol must start with a 32-bit message header as described in Table 4.

Table 4: Message Header

Field	Name	Description
Bits[31:28]	Reserved	Must be zero.
Bits[27:18]	Token	Caller defined value.

Field	Name	Description
Bits[17:10]	Protocol ID	Must be **0x1A** to indicate MPAM Fb Protocol. All other values are reserved.
Bits[09:08]	Message Type	Must be **0x0** for commands. All other values are reserved.
Bits[07:00]	Message ID	Message ID as specified in each command.

All commands have a *Message Type* of 0x0.

The usage of the *Token* field is at the discretion of the caller. For example, the token field can be used to track several commands in-flight. When a command returns, the token field is returned unmodified by the MPAM Manager.

Commands to the MPAM Manager comprises of the message header followed by command specific input parameters as shown in Table 5. Not all commands have associated input parameters.

Table 5: Command

Field	Byte length	Description
Message Header	4	Message header as specified in Table 4.
Parameters	–	The parameters of the command.

Command responses from the MPAM Manager comprises of the unmodified command message header followed by the return status code and data as shown in Table 6.

Table 6: Response

Field	Byte length	Description
Message Header	4	Message header as specified in Table 4.
Return values	–	The return values from the command's response.

The offset of the message header, the parameters and the return value field is determined by the transport.

All parameters, message headers, and return values are expressed in the little-endian format. The endian-ness rule does not apply to strings.

MPAM-Fb protocol is intended to be compliant with System Control and Management Interface (SCMI) [3], and can be used over any transport specified by SCMI. MPAM-Fb protocol can also utilize ACPI Platform Communication Channel Type 3 and 4 [4] as transport [5].

2.5.1 Status Codes

Messages return status codes to the sender. Negative 32-bit integers are used to return error status codes.

The status codes and their meaning is shown in Table 7. These status codes are applicable to all commands in this specification, unless specified otherwise.

Table 7: Status Codes

Status Code	Description	Meaning
0	SUCCESS	Command completed successfully.
-1	NOT_SUPPORTED	The command or feature is not supported.
-2	INVALID_PARAMETERS	One or more of the parameters are invalid.
-3	DENIED	The caller is not permitted to perform the specified action.
-4	NOT_FOUND	The resource being accessed does not exist.
-5	OUT_OF_RANGE	Reqeusted settings are not within the legal range under current operating conditions or state.
-6	BUSY	The platform is out of resources and is unable to process the command. The caller can proceed to activate recovery mechanims.
-7	COMMS_ERROR	The message could not be correctly transmitted. This error is caused due to issues in the transport. Possible causes could be HW buffer overflows, or other reasons.
-8	GENERIC_ERROR	The command failed due to an unspecified fault.
-9	HW_ERROR	A hardware error occured in the platform during command execution.
-10	PROTOCOL_ERROR	The caller has violated the protocol specification.
-11	IN_USE	The resource is currently is use by another caller and cannot be operated upon.
< -11	Reserved	

3 Commands

The MPAM Fb protocol uses an integer identifier to specify the MSC that a command refers to. The MSC identifiers are sequential and start from 0 for each agent which uses this protocol.

The same MSC identifier can map to a different MSC in the system when used in the context of different agents. The mapping of an MSC identifier to an MSC in the system is intended to be discovered through firmware decription mechanisms like Advanced Configuration and Power Interface (ACPI) or Flattened Device Tree (FDT).

This specification allows the following commands.

3.1 PROTOCOL_VERSION

On success, this command returns the version of the MPAM Fb Profile implemented by the MPAM Manager as specified in Table 8.

It is mandatory to implement this command.

Protocol versioning uses a 32-bit unsigned integer, where the upper 16 bits are the major revision, and the lower 16 bits are the minor revision. The following rules apply to the version numbering:

- Higher numbers denote newer versions.
- Different major revision values indicate possibly incompatible messages. For two protocol versions, A and B, which differ in major revision, and where B is higher than A, the following might be true:
  - B can remove messages that were present in A.
  - B can add new messages that were not present A.
  - B can modify the behavior or parameters of messages that are also present in A.
- Minor revisions allow extensions but must retain compatibility. For two protocol versions, A and B, that differ only in the minor revision, and where B is higher than A, the following must hold:
  - Every message in A must also be present in B, and work with compatible effect.
  - It is possible for revision B to have a higher message count than revision A.

Table 8: PROTOCOL_VERSION

Message ID	**0x0**
Return Values	
Name	**Description**
int32 status	See Table 7 for status code definitions.
uint32 version	For this version of the specification, this value must be 0x10000

3.2 PROTOCOL_ATTRIBUTES

This command returns the implementation details associated with this protocol, like the number of MSCs supported, as specified in Table 9.

It is mandatory to implement this command.

Table 9: PROTOCOL_ATTRIBUTES

Message ID	**0x1**
Return Values	
Name	**Description**
int32 status	See Table 7 for status code definitions.
uint32 attributes	Bits[31:16] Reserved, must be zero. Bits[15:00] Number of MSCs present.

3.3 PROTOCOL_MESSAGE_ATTRIBUTES

This command returns the implementation details associated with a specific message in this protocol, as specified in table Table 10.

This command can be used to discover if a command has been implemented.

It is mandatory to implement this command.

Table 10: PROTOCOL_MESSAGE_ATTRIBUTES

Message ID	**0x2**
Parameters	
Name	**Description**
uint32 msg_id	message_id of the message
Return Values	
Name	**Description**
int32 status	One of the following: • SUCCESS: In case the message is implemented and available to use. • NOT_FOUND: If the message is not implemented by the platform. See Table 7 for status code definitions.
uint32 attributes	Reserved, must be zero.

3.4 MPAM_MSC_ATTRIBUTES

This command returns the attributes associated with the MSC specified in this command. The format of this command is specified in Table 11.

It is mandatory to implement this command.

Table 11: MPAM_MSC_ATTRIBUTES

Message ID	**0x3**
Parameters	
Name	**Description**
uint32 msc_id	Identifier of the MSC
uint32 flags	Reserved, must be zero.
Return Values	
Name	**Description**
int32 status	One of the following: • SUCCESS: If valid attributes are returned. • NOT_FOUND: If the msc_id is non-existent. See Table 7 for status code definitions.
uint32 attributes	Reserved for future use and must be zero.

3.5 MPAM_MSC_READ

This command can be used to read the contents of the MSC registers. The format and layout of the MSC registers is described in the MPAM specification[1].

This command operates on registers using 32-bit access width. Register address boundaries must be 32-bit aligned. A 64 bit register is read as a combination of two 32-bit registers.

The format of this command is specified in Table 12.

It is mandatory to implement this command.

Table 12: MPAM_MSC_READ

Message ID	**0x4**
Parameters	
Name	**Description**
uint32 msc_id	Identifier of the MSC
uint32 flags	Reserved, must be zero.
uint32 offset	The offset of the register to read from. The offset is specified from the base address of the MPAM MSC node.
Return Values	
Name	**Description**

Message ID	**0x4**
int32 status	One of the following: • SUCCESS • NOT_FOUND: If the msc_id is non-existent. • INVALID_PARAMETERS: If any of the input parameters are invalid. • OUT_OF_RANGE: If the offset specified is out of range. • NOT_SUPPORTED: If the request is not supported. See Table 7 for status code definitions.
uint32 val	The value read from the register. This field must be ignored if the status is not SUCCESS.

3.6 MPAM_MSC_WRITE

This command can be used to write into the MSC registers. The format and layout of the MSC registers is described in the MPAM specification[1].

The MPAM Manager should verify the value requested to be written before executing the actual write to the MSC register.

This command operates on registers using 32-bit access width. Register address boundaries must be 32-bit aligned. A 64 bit register is written sequentially as a combination of two 32-bit registers.

The format of this command is specified in Table 13.

It is mandatory to implement this command.

Table 13: MPAM_MSC_WRITE

Message ID	**0x5**
Parameters	
Name	**Description**
uint32 msc_id	Identifier of the MSC
uint32 flags	Reserved, must be zero.
{uint32 offset, uint32 val} buf	A tuple containing the value to be written to the MSC register. • offset: The offset of the register measured from the base address of the MPAM MSC node. • val : The value to write into the register.
Return Values	
Name	**Description**

Message ID	**0x5**
int32 status	One of the following: • SUCCESS • NOT_FOUND: If the msc_id is non-existent. • INVALID_PARAMETERS: If any of the input parameters are invalid. • OUT_OF_RANGE: If the offset specified is out of range. • NOT_SUPPORTED: If the request is not supported. See Table 7 for status code definitions.