RISC-V Platform Management Interface (RPMI) Proposal

RISC-V RPMI Task Group

Version 0.1, 5/2023: This document is in development. Assume everything can change. See http://riscv.org/spec-state for details.

Contributors to all versions of this specification in alphabetical order (please contact the editor to suggest corrections):

Andrew Jones, Anup Patel, Chinliang See, JeeHeng Sia, Himanshu Chauhan, Leyfoon Tan, Rahul Pathak, Subrahmanya Lingappa and Sunil V L.

Table of Contents

Preamble

This document is in the Development state
Assume everything can change. This draft specification will change before being accepted as standard, so implementations made to this draft specification will likely not conform to the future standard.

Copyright and license information

This RISC-V Platform Management Interface (RPMI) specification is

© 2023 RISC-V International

This specification is licensed under the Creative Commons Attribution 4.0 International License (CC-BY 4.0). The full license text is available at creativecommons.org/licenses/by/4.0/.

Change Log

Version 0.1

- Initial version

Terms and abbreviations

This specification uses the following terms and abbreviations:

Term	**Meaning**
ACPI	Advanced Configuration and Power Interface
AP	Application Processor, can be a Hart or Cluster of Harts which runs a OS
Client	Software component that utilizes the services provided by the PuC via RPMI interface. The client is responsible for sending requests, receiving responses, and processing notifications and data from the PuC. An example of a client can be an operating system running on the AP.
DT	Devicetree
HART	RISC-V Hardware thread
MSI	Message Signaled Interrupts
PuC	Platform Microcontroller
RPMI	RISC-V Platform Management Interface
WORD	32-bit word.
MM	Management Mode. Generic term used to describe a secure execution environment provided by the CPU.

1. Introduction

This document describes RISC-V Platform Management Interface (RPMI), which is an extensible interface to manage and control the system using a dedicated microcontroller. Systems today pose challenges in terms of manageability and controllability where the OS may have to support a variety of hardware which can be different in design and devices connected. The extra complexity and demand to manage and control the system along with executing compute workloads is challenging for Application Processors running the OS. To mitigate this, systems today contain one or more Platform Microcontrollers which abstract various platform specific system management and control related tasks. RPMI enables the communication between the application processors and the Platform micro-controllers. It abstracts the system complexity and provides a message based interface for system management and control.

RPMI is not limited to a single Application Processor and Platform Microcontroller. It can support multiple Application Processors and multiple Platform Microcontrollers.

The Platform Micro-Controller (PuC) serves as an external embedded processor that abstracts low-level platform control management from the application processor system. This design allows RPMI to provide an OS-agnostic, generic interface that can support various System-on-Chip (SoC) hardware or platforms.

Furthermore, RPMI offers a scalable and extensible interface that can support the addition of new service groups over time, allowing for the implementation of new features and functions. This feature simplifies the integration of new components and technologies into existing systems, making it easier for system designers to keep up with changing requirements and adapt to evolving industry standards.

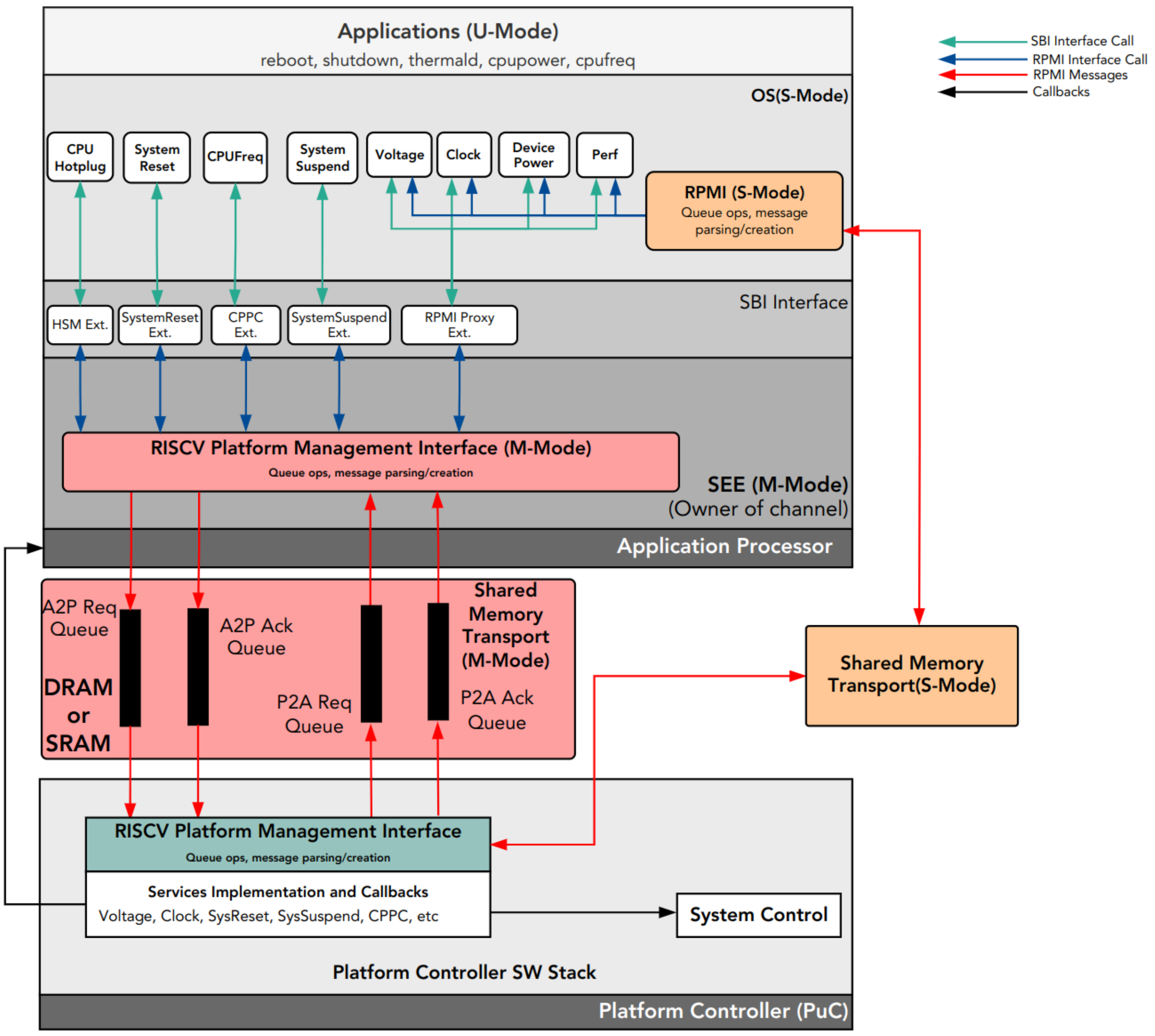
RPMI provides two types of abstractions:

Transport: Describes the mechanism by which the messages are exchanged between the Application Processors and Platform Microcontrollers.

Messaging Protocol: Provides Messaging Interface between Application Processors and Platform Microcontrollers to communicate with each other via messages to make requests for various services supported by the hardware platform. This is accomplished by grouping each management interface into service groups. With each service group implementing several individual services within.

RPMI currently provides message interfaces to manage power, voltage, performance, idle states etc, along with extensibility for vendors to add their own interfaces.

Application Processor and A.P are used interchangeably in this document. Similarly, Platform Microcontroller and PuC are used interchangeably.



Applications (U-Mode)
reboot, shutdown, thermald, cpupower, cpufreq
OS(S-Mode)
CPU Hotplug
System Reset
CPUFreq
System Suspend
Voltage
Clock
Device Power
Perf
RPMI (S-Mode)
Queue ops, message parsing/creation
SBI Interface
HSM Ext.
SystemReset Ext.
CPPC Ext.
SystemSuspend Ext.
RPMI Proxy Ext.
RISCV Platform Management Interface (M-Mode)
Queue ops, message parsing/creation
SEE (M-Mode)
(Owner of channel)
Application Processor
A2P Req Queue
A2P Ack Queue
P2A Req Queue
P2A Ack Queue
Shared Memory Transport (M-Mode)
DRAM or SRAM
Shared Memory Transport(S-Mode)
RISCV Platform Management Interface
Queue ops, message parsing/creation
Services Implementation and Callbacks
Voltage, Clock, SysReset, SysSuspend, CPPC, etc
System Control
Platform Controller SW Stack
Platform Controller (PuC)
SBI Interface Call
RPMI Interface Call
RPMI Messages
Callbacks

Figure 1.1. High Level RPMI based System Architecture

An RPMI agent is an implementation instance of RPMI Messaging Protocol and Transport. It is necessary to implement the AP side RPMI agent for most RPMI services in M-mode. While it may be possible to implement the AP side RPMI agent in S-mode for some RPMI services, an M-mode RPMI agent also enhances security.

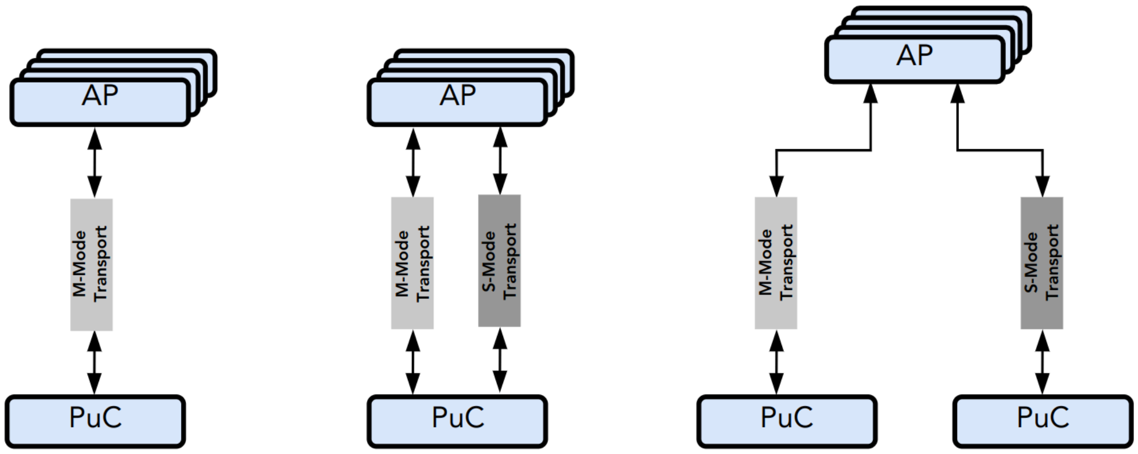


Figure 1.2*. Transport in different system topologies*

RPMI is designed to work with a single or multi-tenant topology as depicted above.

NOTE: The discovery of the transport itself is out of scope for this document. Which can either be described in firmware through DT or ACPI.

2. Transport

RPMI Transport is the abstraction over a physical medium used to send and receive messages between an Application Processor and Platform Microcontroller enabling bidirectional communication.

A RPMI transport instance provides a bidirectional communication channel between a privilege level of a set of APs and one PuC. There can be multiple RPMI transport instances between a set of APs and one PuC. For example: there can be a separate transport instance between each privilege level of APs and one PuC. A platform can also have multiple PuCs with each having its own set of RPMI transport to communicate with APs. The **Figure 1.2** above shows different topologies of RPMI transport.

The physical medium of a RPMI transport can be shared memory or I2C or SPI or some other medium. This specification only defines a shared memory based RPMI transport but a platform can define its own RPMI transport.

2.1 Doorbell Interrupt

A RPMI transport may also provide doorbell interrupts for either Application Processors or Platform Microcontrollers or both to signal new messages. The doorbell mechanism is optional for a RPMI transport and implementations can always use a polling mechanism for checking the arrival of messages.

The doorbell interrupt from APs to PuC can be a MSI or a wired interrupt with a RPMI transport specific way to trigger the interrupt.

The doorbell interrupt from PuC to APs can be either a wired interrupt or message signaled interrupt (MSI). If the doorbell interrupt from PuC to APs is a wired interrupt then RPMI transport must define a way to trigger the interrupt. If the doorbell interrupt from PuC to APs is a MSI then RPMI messages can be used by APs to configure the MSI.

2.2 Shared Memory Transport

The physical memory of the RPMI shared memory transport can be either a device memory or dedicated SRAM or some portion of DRAM. The RPMI shared memory transport does not specify where the shared memory resides, but it should be accessible from both Application Processor and Platform Microcontroller and all necessary configuration needs to be done to make sure that side effects related to caching or anything else do not happen.

NOTE: To avoid the caching side effects, the platform can configure the shared memory as IO or non-cacheable memory for both APs and PuC.

All data sent/received through RPMI shared memory MUST follow Little-Endian byte ordering, unless, optionally, the byte-ordering is discovered during the transport discovery phase.

The doorbell interrupt from APs to PuC is optional for RPMI shared memory transport and if available then it MUST be supported through a read-modify-write sequence to a memory mapped register. This read-modify-write mechanism can be discovered by APs via DT or ACPI using properties such as register physical address, mask, and value.

The doorbell interrupt from PuC to APs for RPMI shared memory transport is platform specific.

The subsequent sections describe the layout and attributes of shared memory which should be consistent across both Application Processor and Platform Microcontroller. These attributes can be static for the Platform Microcontroller and discoverable via DT or ACPI for the Application Processor.

2.2.1 Attributes of Shared Memory

- (Base Address, Size) for Total Shared Memory

    - Base address is 4KB aligned
    - Size is multiple of 4KB.
- Slot Size ≥ 64-Bytes

$N\ Bytes$	*Total Shared Memory Size*
$M = (N/4) Bytes$	*Single Queue Shared Memory Size*
$(M - 8)/SlotSize$	*No. of Slots in single Queue*

where $SlotSize \geq$ *Maximum Message Size*
and
8 = *No. of Bytes for Head and Tail*

2.2.2 Layout of Shared Memory

The shared memory RPMI Transport defines four shared memory based queues for bidirectional synchronous/asynchronous communication between an AP and PuC. The detailed layout of shared memory and queues is shown in **Figure 2.1.** All queues in shared memory are contiguous and of equal size. Sequence of queues is arranged such that queues which enable Request-Acknowledgement for a side either AP to PuC or vice versa are together.

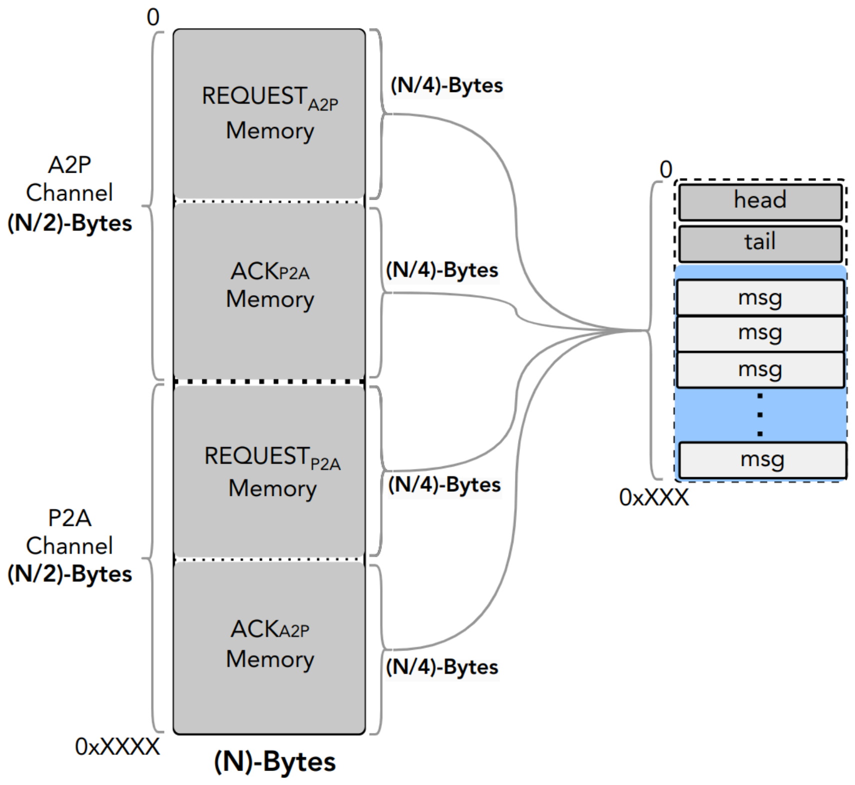


***Figure 2.1**. Memory Layout of Shared Memory*

2.2.3 Shared Memory Queues

2.2.3.1 AP to PuC Request (A2P REQ)

This queue is to transmit REQUEST messages from AP to PuC.

2.2.3.2 PuC to AP Acknowledgement (P2A ACK)

This queue is to transmit the ACKNOWLEDGEMENT messages from PuC to AP for the request messages received by PuC on A2P REQ Queue.

2.2.3.3 PuC to AP Request (P2A REQ)

This queue is to transmit REQUEST messages from PuC to AP.

2.2.3.4 AP to PuC Acknowledgement (A2P ACK)

This queue is to transmit the ACKNOWLEDGEMENT messages from AP to PuC for the request messages received by the AP on P2A REQ Queue.

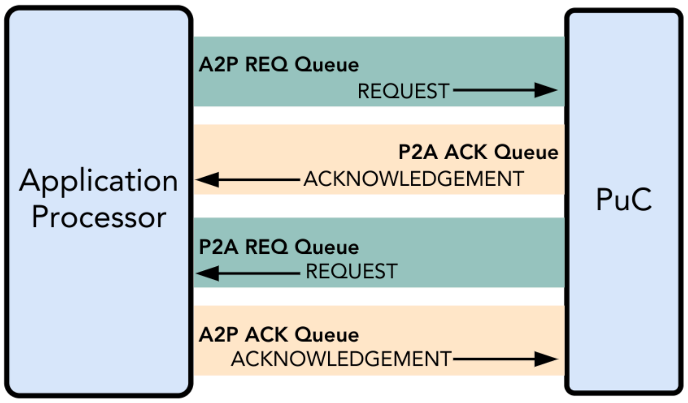


***Figure 2.2**. Transport Queues - High Level*

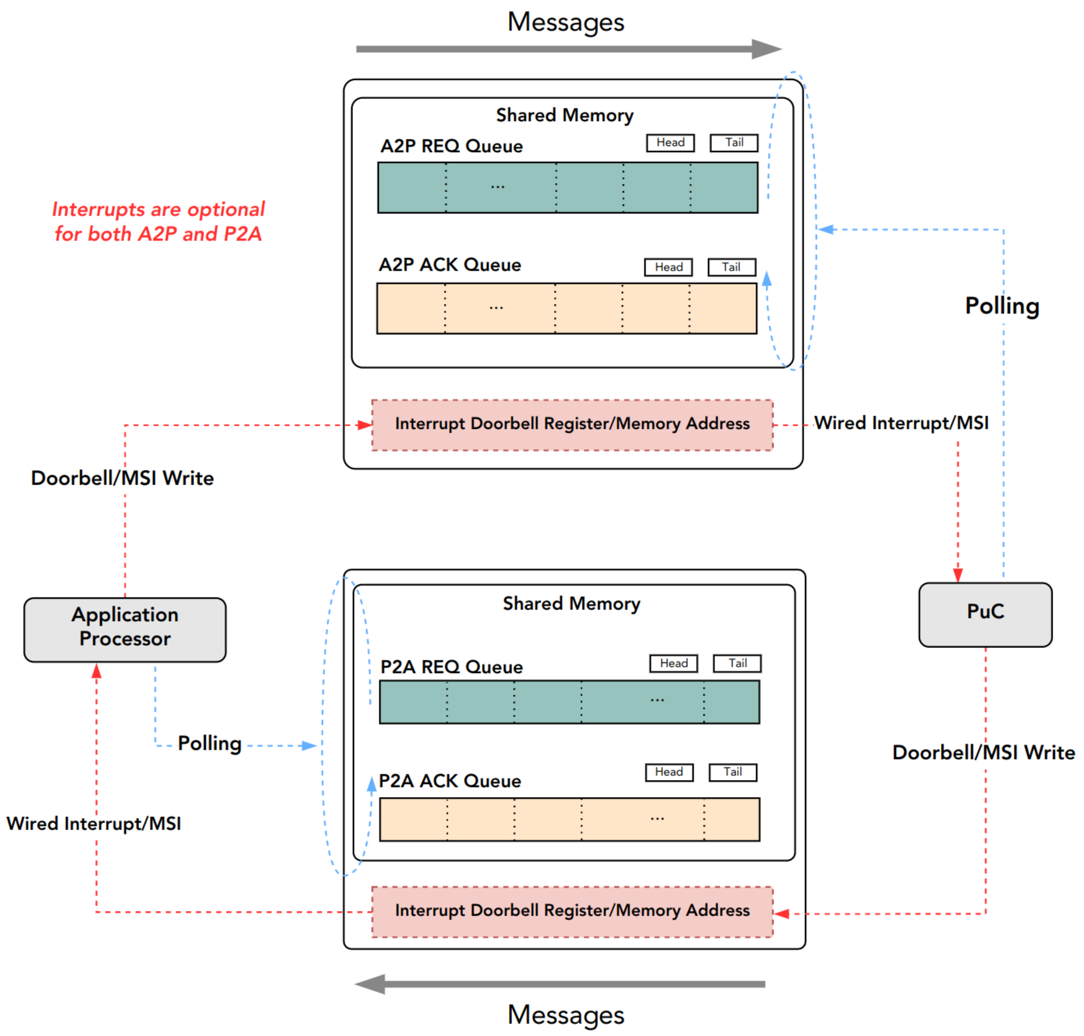


Figure 2.3. *Transport - Details*

Each queue contains **M** number of slots and each slot stores a single message. Slot size must be sufficient to store the biggest message in the framework. Shared memory also contains the head and tail for the enqueuing and dequeuing of the messages for each queue. The RPMI specification expects a minimum size of **64-Bytes** for each slot but bigger slots may also work depending on the implementation.

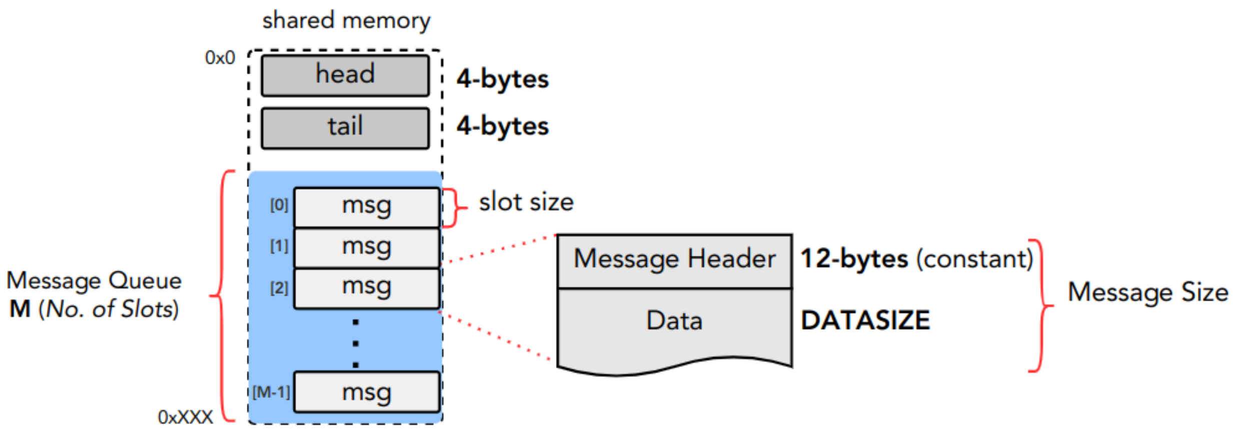


Figure 2.4. *A Queue - Internals*

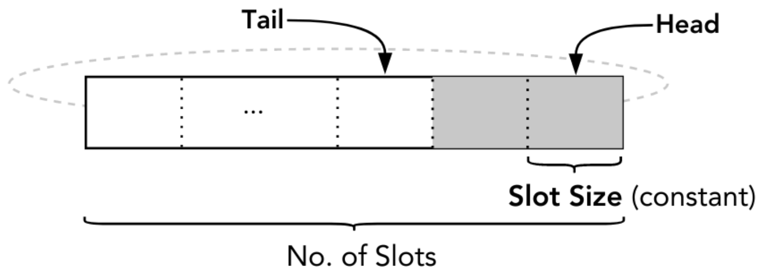
Slots can be accessed using head and tail which will store the indices. **Head will be used to dequeue** the message and **Tail will enqueue**.

Head and Tail will be owned and incremented by only a single entity depending on the role of that entity, whether that entity is enqueuing or dequeuing. For example, on the A2P channel, Application Processor will enqueue the message so it will own and increment the Tail, similarly, Platform Microcontroller will own the head to dequeue the messages and only Platform Microcontroller will increment the head.

Once the reader dequeues a message from the slot, it has to mark that slot to be usable by the writer to enqueue further messages into that slot. Message header flags are used to mark a message as invalid which makes that slot free to use.

Like a normal circular queue, it can either be empty, full or have valid messages.
Enqueue operation will check if the queue is not full by checking if the head is equal to the tail and the slot referenced by the current tail has a valid message. Similarly, the dequeue operation will check for the empty state by validating if the slot referenced by the current head has an invalid message.

Messages which are not consumed yet should not be overwritten and the sender must block until the slot is available for the sending messages.



$$No.\ of\ Slots = Size(shared\, memory)/slot\ size$$

where $slot\ size \geq Maximum\ Message\ Size$

***Figure 2.5**. Slots and Size of Slots in a Queue*

3. Messaging Protocol

Message Protocol supports a variety of system management and control tasks and provides a message interface for such tasks. Each message which does a specific task is called **Service.** Multiple messages or Services are grouped into **Service Groups** like Clock, Voltage, Power management, etc. A service call results in a message request being sent to the Platform micro-controller for a specific purpose. Each service may have an associated response message which enables the Platform micro-controller to provide the status or information for that request.

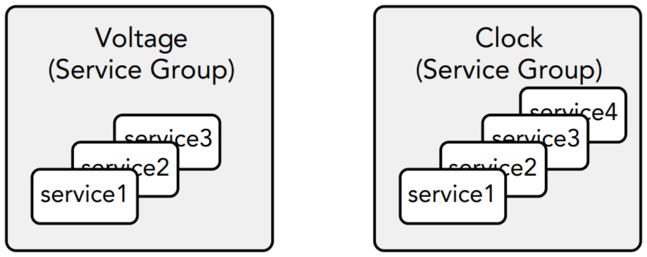


***Figure 3.1**. Service Groups and Services*

RISC-V S-Mode/M-Mode clients are responsible for the packing of the messages as per the message binary format requirements described later and sharing the message via the transport layer. The transport layer abstracts the mechanism of how the message is physically delivered between Application Processor and Platform Microcontroller or vice versa.

Application Processor and Platform Microcontroller communicate through messages which are sent to each other via queues which are unidirectional and implemented in the transport. Each queue may only have a sender at one end and a receiver at the other which makes it one-directional at any moment.

When a request message is sent from Application Processor to Platform Microcontroller then upon the completion of that request Platform Microcontroller optionally can acknowledge the status of the request or respond with requested data which requires bidirectional communication. To enable bidirectional communication for each request queue from AP to Platform Microcontroller and from Platform Microcontroller to AP there are acknowledgement queues from both sides.

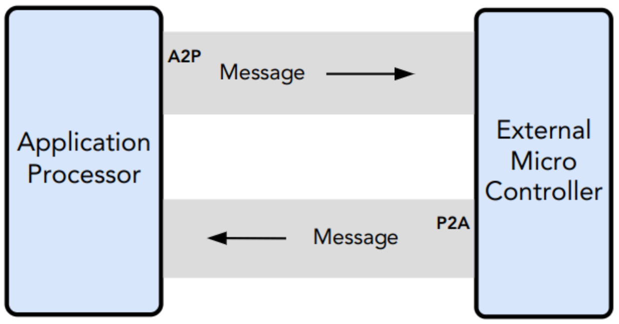


Figure 3.2. *Bidirectional Message Communication*

3.1 Types of Messages

A message can either be a request or an acknowledgement in response to a request received by an entity/client.

3.1.1 REQUEST

Messages which convey a command to the other entity/client, usually from the AP from either S-Mode/M-mode client to Platform Microcontroller (it may also send REQUESTs to AP, but there are no use cases yet). Each command will have a certain action that the Platform Microcontroller will take and it may acknowledge the AP with another message in return which may contain some data or just the status code.

Not every request needs to be acknowledged because the action taken by the Platform Microcontroller may leave the AP in a state where it is not able to wait for the acknowledgment. For example, in the case when the AP asks the Platform Microcontroller to perform a Reset, where an acknowledgment would not be meaningful.

- **Normal Requests**: Requests with Acknowledgement
- **Posted Requests:** Requests without Acknowledgement

3.1.2 ACKNOWLEDGEMENT

Messages sent from Platform Microcontroller to AP (or vice-versa) in a response to any previous message request. Acknowledgment is dependent on the service and each service dictates if acknowledgment is mandatory or not.

3.1.3 NOTIFICATIONS

Notifications are messages which are sent asynchronously from PuC to AP to notify about the events happening in the system related to various service groups. Notifications are covered in detail in below section

3.2 Message Format

On the AP side, each call to a service results in a message request to the PuC. The type of message depends upon the type of service and to which group that particular service belongs. Each message, whether its Request or Acknowledgement, consists of two main parts - Message Header and Data.

Size of message header is fixed, but size of data is implementation defined, based on the transport queues layout. All fields in the request and acknowledgment must follow byte ordering defined by the RPMI transport.

3.2.1 Message Header

Message header has a fixed size of **12-Bytes** which further contains three fields each of **4-Bytes**. Each message gets a unique identity in a RPMI instance through its header.

Both **REQUEST** and **ACKNOWLEDGEMENT** messages have the same message format.

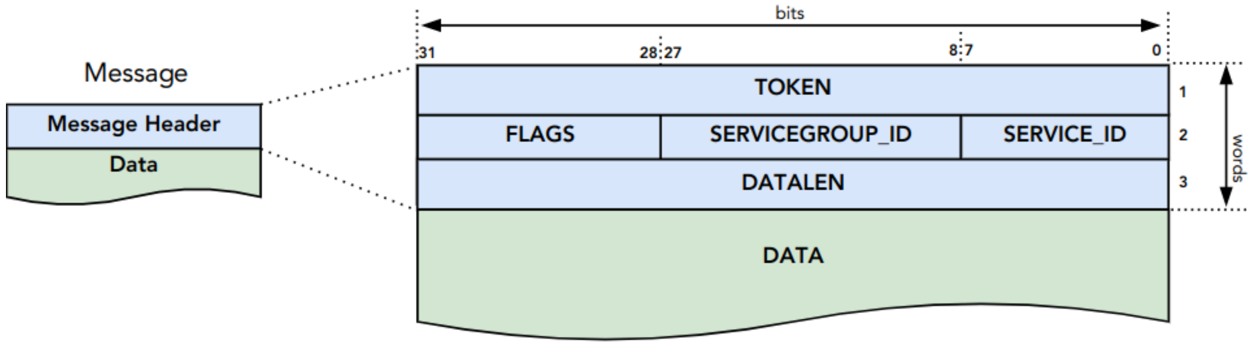


Figure 3.3. *Message Format*

Word	Description
0	**TOKEN**: Message identifier. Unique to each REQUEST - ACKNOWLEDGEMENT transaction for a RPMI implementation instance. In case of Notifications, SERVICE_ID and MESSAGE_TYPE for notification messages are fixed which will contribute to the unique identity of the message in that RPMI instance implementation.

1	**MESSAGE_DESC**: 32-Bit field

Bits	Description
31:28	**FLAGS:**
27:8	**SERVICEGROUP_ID:** Services alike are grouped into Service Groups and each group is identified by SERVICEGROUP_ID which is a 20-bit identifier
7:0	**SERVICE_ID:** which is an **8-bit identifier** from LSB in the message identifier word**,** Services are the functions which, when called, result in different message requests for different control and management tasks meant for Platform Microcontroller. Each service is identified by a **SERVICE_ID**

Bits	Description
[3]	Reserved
[2]	**DOORBELL** 0: Doorbell interrupt is enabled. 1: Doorbell interrupt disabled. PuC will not ring the doorbell to AP. This can be used by AP software in case of doorbell interrupts causing spurious interrupts while it is also being polled.
[1:0]	**MESSAGE_TYPE**: **0b00**: NORMAL_REQUEST **0b01**: POSTED_REQUEST **0b10**: ACKNOWLEDGEMENT **0b11**: NOTIFICATION

2	**DATALEN:** This field will encode the size of the data in the message, data will also be in 32-Bit chunks. If there is no data, then it must be initialized to zero.

Once a message request has been serviced and that service is of type Normal Request which requires an Acknowledgement, PuC must preserve the **TOKEN**, **SERVICEGROUP_ID**, **SERVICE_ID** from the **NORMAL REQUEST** message header and use these fields in **ACKNOWLEDGEMENT** message header. PuC must mark the message type in the **FLAGS** and also according to the data expected in the **ACKNOWLEDGEMENT** it may change **DATALEN** in the message header. In case of Notifications**,** PuC will generate the **TOKEN** and set the **SERVICEGROUP_ID** and fixed **SERVICE_ID=0x00** assigned for each notification message in every service group and set the **FLAGS** with **NOTIFICATION** message type marked. Notification messages do not require any acknowledgement and how data from notification messages is utilized is dependent on the implementation.

3.2.2 Message Data

Request message data format and Acknowledgement data format depends on each service and details are present with each service section below in their respective service groups. Size of data each message can accommodate depends on the transport queues slot size. This specification already defines the data layout for each size. For few services where the data exceeds the size which a single message can accommodate, multipart messages are used.

Message data formats in this specification are tabulated with a list of 32-bit wide Word in each of the service groups section as depicted below.

Word	**Name**	**Type**	**Description**
Word index in message DATA field	Name of the field	Type of the field, eg: int32, uint32, etc	Description of the fields and interpretation

The data in acknowledgment at least should contain a **32-Bit STATUS** code. Apart from status code, an acknowledgement may encode more data as response to the request message and details are subsequently present in each service section below.

3.3 Notifications

Notifications are the messages from Platform Microcontroller to Application Processor to notify about events taking place in the system. Notifications are posted messages which do not require acknowledgement from the recipient. Events can include the system states, power states, errors/faults in the system etc. Action taken on behalf of any event notification is completely dependent on the AP and they can be ignored. Platform Microcontroller may combine multiple events into a single message depending on the available space in the message data. Individual events may also have additional data associated. **Figure 3.4** shows the notification message format.

Each service group will have a notification service with fixed **SERVICE_ID = 0x00** across every service group. Notifications are sent from PuC to AP with events and associated data if any. AP has to subscribe to the supported events in each service group to receive these notification messages. A notification message may have one or more events with their associated data.
This service with service_id=0x00 is reserved in each service group even if the service group does not support notifications or need to support any events. These notifications will only be sent for those events only for which the AP subscribes. When there are multiple events supported in each service group, AP has to subscribe to each event and has to make multiple calls to notification enable service.
Notifications enable service is also present in each service group even if that service group does not support notifications or implement any event support.

Every event will have an Event Header which consists of two fields to identify an event - **EVENT_ID** (12-bit) and **EVENT_DATALEN** (20-bit). Events may have data associated, if present, must be multiple of 4 bytes.

The number of events which can be accommodated in the message data depends on the message data field size. The **DATALEN** field in the message header will encode how much size data is present in the message which is the aggregate of all events. Then AP must parse each event and its data according to the Event header.

Event data and its format depend on the service group and details are present in respective service group sections.

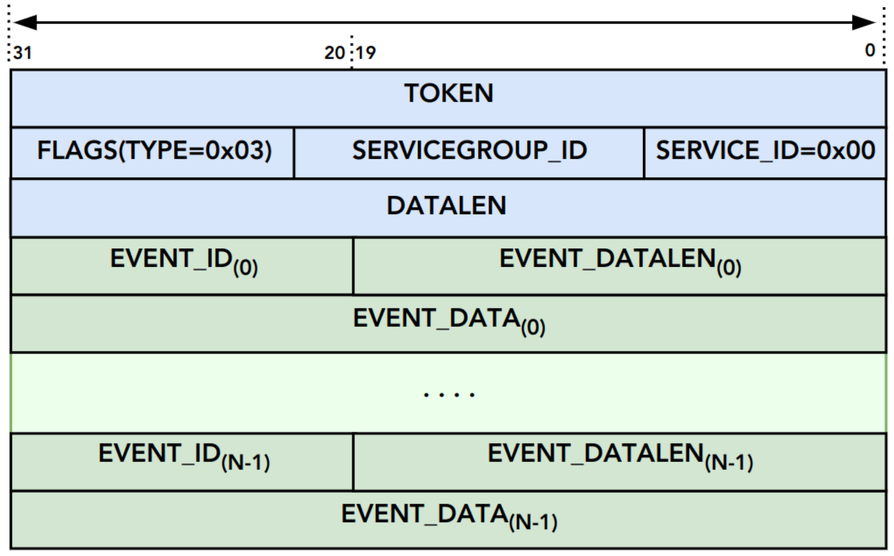


Figure 3.4. *Notification Message*

3.3.1 Notification Message Data Format

Word	Name	Description
0	**EVENT_HDR**	Event Header 32-Bit field: **Bits** / **Description** [31:20] / **EVENT_ID:** Unique identifier to identify an event among a service group.

		[19:0] — **EVENT_DATALEN:** Event data size is a 20-Bit field which encodes the number of 4-byte words of the trailing event data.
1:(EVENT_DATALEN -1)	**EVENT_DATA**	Event Data

Above table represents the format for one event with its data. Subsequent events will be packed in the same manner. This spec does not define any ordering of packing of multiple events and its implementation defined.

3.4 Return Status Codes

Below table lists all the error codes which can be returned by any service. Few probable error codes for each service are also provided in each service response message format. AP must check for each error code and action based on that is dependent on the AP.

Name	STATUS Code	Description
RPMI_SUCCESS	0	The operation was successful.
RPMI_ERROR_FAILED	-1	The operation has failed due to general error
RPMI_ERROR_NOT_SUPPORTED	-2	Service/feature not supported
RPMI_ERROR_INVALID_PARAMETER	-3	One or more of the parameters passed to the operation were invalid.
RPMI_ERROR_DENIED	-4	The requested operation was denied due to insufficient permissions.
RPMI_ERROR_NOT_FOUND	-5	The requested resource was not found.
RPMI_ERROR_OUT_OF_RANGE	-6	Index out of range
RPMI_ERROR_OUT_OF_RESOURCE	-7	Resource limit reached
RPMI_ERROR_HW_FAULT	-8	Operation failed due to hardware issues.
RPMI_ERROR_BUSY	-9	The system is currently busy and cannot respond to the request.
RPMI_ERROR_TIMEOUT	-10	The system did not respond to the request within the specified time limit.
RPMI_ERROR_COMMS	-11	Error in communication, such as buffer overflow.

RPMI_ERROR_ALREADY	-12	Operation failed as it was already in progress or the state has changed already for which the operation was carried out.
RPMI_ERROR_EXTENSION	-13	Error in extension implementation that violates the extension specification or extension version mismatch.
	-14 to -127	Reserved
	> -127	Vendor specific

4. Service Groups

A service group is a collection of services which are logically grouped according to the functionality they provide. For example, all voltage related services/messages are collectively referred to as Voltage Service Group.

This specification defines standard service groups and services with provision to add more service groups as per requirement in the future.

Following table lists the service group

Service Group ID	Service Group Name
0x00001	BASE
0x00002	SYSTEM_RESET
0x00003	SYSTEM_SUSPEND
0x00004	HART_STATE_MANAGEMENT
0x00005	CPPC
0x00006	VOLTAGE
0x00007	CLOCK
0x00008	DEVICE_POWER
0x00009	PERFORMANCE
0x0000A	MM_SERVICE

0x0000B	RAS_AGENT
0x0000B - 0x7FFFF	*Reserved for Future Use*
0x80000 - 0xFFFFF	*Implementation specific service groups*

NOTE: The services listed in each service group do not follow any sequence and are not defined in any specific order. It's possible that any particular service in any service group inherently requires to be called first before other defined services before that.

4.1. Service Group - **BASE** (servicegroup_id: 0x00001)

Base service group is mandatory and provides services for -

- Initial handshaking between the Application processor and Platform micro-controller.
- Querying the message framework implementation version information.
- Discovering the implementation and version of a specific service group or any specific service in any service group.
- Gathering allowed system information from the firmware.
- Status of the transport and if the Platform Microcontroller is ready for accepting messages.

Below table lists the services in this group:

Service ID	Service Name	Request type
0x01	ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	GET_IMPLEMENTATION_VERSION	NORMAL_REQUEST
0x03	GET_IMPLEMENTATION_ID	NORMAL_REQUEST
0x04	GET_SPEC_VERSION	NORMAL_REQUEST
0x05	GET_HW_INFO	NORMAL_REQUEST
0x06	PROBE_SERVICE_GROUP	NORMAL_REQUEST
0x07	GET_BASE_ATTRIBUTES	NORMAL_REQUEST
0x08	SET_MSI	NORMAL_REQUEST

4.1.1 Base Notifications

Platform Microcontroller can send asynchronous notifications to AP via this service. There can be multiple types of events classified in the Base service group which can be combined into a single message as depicted in **Figure 3.5**. If Platform Microcontroller has multiple events for the same type, Platform Microcontroller can send the single instance of that event that was received last.

Base Service Group Events:

Event ID	Name	Event Data	Description
0x001	REQUEST_HANDLE_ERROR	NA	This event indicates that the Platform Microcontroller is unable to serve the message requests anymore and this event may send the list of messages which the Platform Microcontroller was processing when the failure took place. No event data is necessary for this notification

4.1.2 Service: **ENABLE_NOTIFICATION**

This service allows AP to subscribe to Base service group notifications.
Platform can optionally support notifications of errors which might occur in the platform. PuC can send these notification messages to AP if they are implemented and AP has subscribed to these. Events supported are described above in Base Notifications.

Request data

Word	Name	Type	Description
0	**EVENT_ID**	**uint32**	Event to be subscribed for notification.

Response data

Word	Name	Type	Description	
0	**STATUS**	**int32**	Return Status Code	
			Error Code	**Description**
			RPMI_SUCCESS	when the notifications are subscribed successfully for EVENT_ID.
			RPMI_ERROR_NOT_FOUND	Event in EVENT_ID is not supported or invalid.
			RPMI_ERROR_NOT_SUPPORTED	Notifications not supported in this service group.

			● Other errors in section 3.4 Return Status Code

4.1.3 Service: **GET_IMPLEMENTATION_VERSION**

Get the implementation version of the message framework from Platform Microcontroller. Versioning is also done for the spec but it's possible that for a single version of spec there may be multiple implementation versions and based on that any lowest supported implementation version can be mandated to ensure the stability and support. Version returned using this service is a 32-Bit composite number which contains both Major and Minor numbers.

Request data

- NA

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** / **Description** RPMI_SUCCESS: when the implementation version is returned successfully. ● Other errors in section 3.4 Return Status Code
1	**VERSION**	**uint32**	**Bits** / **Description** [31:16]: Major number [15:0]: Minor number

4.1.4 Service: **GET_IMPLEMENTATION_ID**

Get the RPMI Implementation ID assigned to the Operating system or Firmware or any other software host which implements the RPMI specification.

Request data

- NA

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** | **Description** RPMI_SUCCESS | when the implementation id is returned successfully. ● Other errors in section 3.4 Return Status Code
1	**IMPL_ID**	**uint32**	Implementation ID

4.1.5 Service: **GET_SPEC_VERSION**

Get the version of the implemented RPMI specification.

Request data

- NA

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** | **Description** RPMI_SUCCESS | when the spec version is returned successfully. ● Other errors in section 3.4 Return Status Code
1	**VERSION**	**uint32**	**Bits** | **Description** [31:16] | Major number [15:0] | Minor number

4.1.6 Service: **GET_HW_INFO**

This service is used to retrieve the Vendor ID and Name of the Vendor having a RPMI implementation on PuC. Each vendor will be assigned a unique Vendor ID.

Request data
- NA

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** / **Description** RPMI_SUCCESS / when the vendor info is returned successfu ● Other errors in section [3.4 Return Status Code](#)
1	**VENDOR_ID**	**uint32**	Vendor Identification **Bits** / **Description** [31:16] / **SUB_VENDOR_ID** (optional) **0**: Not Supported. It is an additional numeric value used to further differentiate between different sub-vendors or pr lines within the same hardware vendor. [15:0] / **VENDOR_ID**. Hardware Vendor ID is a numeric value that uniqu identifies the manufacturer or vendor of the hard platform or device.
2	**HW_ID_LEN**	**uint32**	**HW_ID** field length in bytes.
3:	**HW_ID**	**uint8[HW_ID_LEN]**	Hardware identifier String Up to **HW_ID_LEN** bytes NULL terminated ASCII string. It can be used to convey details such as the specific product model, revision, or configuration of the hardware.

4.1.7 Service: **PROBE_SERVICE_GROUP**

Probe the implementation of any service group by its service group id. Except BASE, rest of the service groups are optional but if a service group is implemented then it has to be implemented completely with all services in that group. The notifications in that service group are still optional which will be implemented by the PuC.

Request data

Word	Name	Type	Description
0	**SERVICEGROUP_ID**	**uint32**	Service group id: A **24-Bit** ID assigned to each service group.

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** | **Description** RPMI_SUCCESS | when the service group is probed successfully. If service group represented by **SERVICEGROUP_ID** preset on not must be identified by the service_group_status field ● Other errors in section 3.4 Return Status Code
1	**SERVICE_GROUP_ STATUS**	**uint32**	**0**: Service group not implemented by platform **1:** Service group is implemented by platform

4.1.8 Service: **GET_BASE_ATTRIBUTES**

This service is used to discover additional features supported by the base service group.

Request Data

- NA

Response Data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** | **Description** RPMI_SUCCESS | when the base attributes are returned successfully. ● Other errors in section 3.4 Return Status Code

1	**FLAGS0**	**uint32**	**Bits** / **Description** [31] / **EVENT_NOTIFICATION** **0b0**: Notification are not supported **0b1**: Notifications are supported [30] / **MSI** **0b0**: Not supported **0b1**: Supported [29:0] / Reserved
2	**FLAGS1**	**uint32**	Reserved, must be initialized to zero
3	**FLAGS2**	**uint32**	Reserved, must be initialized to zero
4	**FLAGS3**	**uint32**	Reserved, must be initialized to zero

4.1.9 Service: **SET_MSI**

Configure the MSI address and data which the Platform Microcontroller can use as a doorbell to AP.

The PuC to AP MSI can be used for both sending MSI or injecting wired interrupts. If the MSI target address is IMSIC then AP will take MSI whereas if the MSI target address is "setipnum" of APLIC then AP will take wired interrupt.

In case of platforms with PLIC, the platform need to provide a MMIO register to inject a edge-triggered interrupt

Request data

Word	Name	Type	Description
0	**MSI_ADDRESS_LOW**	**uint32**	Lower 32 bits of MSI address
1	**MSI_ADDRESS_HIGH**	**uint32**	Higher 32 bits of MSI address
2	**MSI_DATA**	**uint32**	32 bit MSI data

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	Description
RPMI_SUCCESS	MSI address and data are configured successfully.
RPMI_ERROR_NOT_SUPPORTED	when MSI is not supported. Implementation must use base attributes to discover this capability and then use this service.

- Other errors in section 3.4 Return Status Code

4.2 Service Group - **SYSTEM_RESET** (servicegroup_id: 0x00002)

This service group provides services to reset or shutdown the system. *System Shutdown* or *System Cold Reset* are Architectural System Reset types.

System Shutdown causes every component/device in the system to lose power. Currently, AP is the only entity to request the system shutdown, which means for Platform Microcontroller it is not important to categorize it as graceful or forceful shutdown. In case of shutdown request it is implicit for the Platform micro-controller that AP has prepared itself for the successful shutdown.

System Cold Reset also called Power-On-Reset is the power cycling of the complete system. On the successful system cold reset, all devices will be power cycled in a implementation defined sequence similar to the first power on of the system.

Below table lists the services in this group:

Service ID	Service Name	Request type
0x01	ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	GET_SYSTEM_RESET_ATTRIBUTES	NORMAL_REQUEST
0x03	SYSTEM_RESET	POSTED_REQUEST

4.2.1 System Reset Notifications

This service group does not support any event for notification

4.2.2 Service: **ENABLE_NOTIFICATION**

This service allows AP to subscribe to system reset service group notifications.

Platform can optionally support notifications of events which might occur in the platform. PuC can send these notification messages to AP if they are implemented and AP has subscribed to these. Events supported are described above in System Reset Notifications.

Request data

Word	Name	Type	Description
0	**EVENT_ID**	**uint32**	Event to be subscribed for notification.

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	Description
RPMI_SUCCESS	when the notifications are subscribed successfully for EVENT_ID.
RPMI_ERROR_NOT_FOUND	Event in EVENT_ID is not supported or invalid
RPMI_ERROR_NOT_SUPPORTED	Notifications not supported in this service group

- Other errors in section 3.4 Return Status Code

4.2.3 Service: **GET_SYSTEM_RESET_ATTRIBUTES**

This service is used to discover system reset types supported by the platform. *System Shutdown* and *System Cold Reset* are mandatory and assumed to be supported. *System Warm Reset* is a qualified reset type which can be discovered using this service. Space is reserved for more reset types to be added later as required.

Request Data

Word	Name	Type	Description
0	**RESET_TYPE**	**uint32**	

Value	Description
0x0	System Shutdown
0x1	System Cold Reset

			0x2	System Warm Reset
			0x3-0xEFFFFFFF	Reserved for future use
			0xF0000000-0xFFFFFFFF	Implementation specific reset types

Response Data

Word	**Name**	**Type**	**Description**	
0	**STATUS**	**int32**	Return Status Code	
			Error Code	**Description**
			RPMI_SUCCESS	when attributes are returned successfully.
			RPMI_ERROR_NOT_SUPPORTED	when reset_type is not supported.
			RPMI_ERROR_INVALID_PARAMETER	when the reset_type in request is invalid.
			● Other errors in section 3.4 Return Status Code	
1	**FLAGS**	**uint32**	**Bits**	**Description**
			[31]	Reset type supported if set
			[30:0]	Reserved, must be initialized to zero

4.2.4 Service: **SYSTEM_RESET**

This service is used to initiate the system shutdown or reset the system with the provided type. *System Shutdown* and *System Cold Reset* are supported implicitly but *System Warm Reset* and other implementation specific reset types are optional and discoverable.

AP must only request for supported reset types which are discovered using the attributes service. If an unsupported or invalid state is requested the system might enter into non-functional state

Request Data

Word	Name	Type	Description
0	**RESET_TYPE**	**uint32**	

Value	**Description**
0x0	System Shutdown
0x1	System Cold Reset
0x2	System Warm Reset
0x3-0xEFFFFFFF	Reserved for future use
0xF0000000-0xFFFFFFFF	Implementation specific reset types

Response Data

- NA

4.3 Service Group - **SYSTEM_SUSPEND** (servicegroup_id: 0x00003)

This service group provides services to put the entire system in a low power suspend state. All HARTs except the HART doing system suspend need to be in the STOPPED state.

Below table lists the services in this group:

Service ID	Service Name	Request type
0x01	ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	GET_SYSTEM_SUSPEND_ATTRIBUTES	NORMAL_REQUEST
0x03	SYSTEM_SUSPEND	NORMAL_REQUEST

4.3.1 System Suspend Notifications

This service group does not support any event for notification

4.3.2 Service: **ENABLE_NOTIFICATION**

This service allows AP to subscribe to system suspend service group notifications.

Platform can optionally support notifications of events which might occur in the platform. PuC can send these notification messages to AP if they are implemented and AP has subscribed to these. Events supported are described above in System Suspend Notifications.

Request data

Word	Name	Type	Description
0	**EVENT_ID**	**uint32**	Event to be subscribed for notification.

Response data

Word	Name	Type	Description	
0	**STATUS**	**int32**	Return Status Code	
			Error Code	**Description**
			RPMI_SUCCESS	when the notifications are subscribed successfully for EVENT_ID.
			RPMI_ERROR_NOT_FOUND	Event in EVENT_ID is not supported or invalid
			RPMI_ERROR_NOT_SUPPORTED	Notifications not supported in this service group
			● Other errors in section 3.4 Return Status Code	

4.3.3 Service: **GET_SYSTEM_SUSPEND_ATTRIBUTES**

This service helps querying supported suspend types.

Request data:

Word	Name	Type	Description	
0	**SUSPEND_TYPE**	**uint32**	**Value**	**Description**
			0x0	Suspend to RAM
			0x1-0xFFFFFFFF	Reserved

Return Values:

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code | **Error Code** | **Description** | | RPMI_SUCCESS | Attributes are returned successfully. | | RPMI_ERROR_NOT_SUPPORTED | **SUSPEND_TYPE** is not supported. | ● Other errors in section 3.4 Return Status Code
1	**FLAGS**	**uint32**	| **Bits** | **Description** | | [31] | **0b0**: Custom resume address not supported **0b1**: Custom resume address supported | | [30] | **0b0**: Suspend not supported **0b1**: Suspend supported | | [29:0] | Reserved, must be initialized to zero |

4.3.4 Service: **SYSTEM_SUSPEND**

This service puts the system into low power state based on the specified suspend type. The service requires all harts except the calling hart to be in STOPPED state as defined by the Hart State Management service group. The system goes into low power state only after the HART which requested system suspend, executes WFI instruction.

Request data

Word	Name	Type	Description
0	**HART_ID**	**uint32**	Hart ID of the calling HART
1	**SUSPEND_TYPE**	**uint32**	| **Value** | **Description** | | 0x0 | Suspend to RAM | | 0x1-0xFFFFFFFF | Reserved |
2	**RESUME_ADDR_LOW**	**uint32**	Lower 32 bit address

3	**RESUME_ADDR_HIGH**	**uint32**	Higher 32 bit address

Return Values:

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** / **Description** RPMI_ERROR_NOT_SUPPORTED: when suspend_type is not supported. AP must use attributes service to discover the suspend_types and request the supported suspend_type only. RPMI_ERROR_INVALID_PARAMETER: when the suspend_type in request is invalid. ● Other errors in section 3.4 Return Status Code

4.4. Service Group - **HART_STATE_MANAGEMENT**(servicegroup_id: 0x00004)

The Hart State Management (HSM) service group defines a set of Hart states and functionality equivalent to the RISC-V SBI HSM extension.

Ref: SBI SPEC: Chapter 9. Hart State Management Extension (EID #0x48534D "HSM")

State ID	State Name	Description
0	STARTED	The hart is physically powered-up and executing normally.
1	STOPPED	The hart is not executing in supervisor-mode or any lower privilege mode. It is probably powered-down by the SBI implementation if the underlying platform has a mechanism to physically power-down harts.
2	START_PENDING	Some other hart has requested to start (or power-up) the hart from the STOPPED state and the SBI implementation is still working to get the hart in the STARTED state.
3	STOP_PENDING	The hart has requested to stop (or power-down) itself from the STARTED state and the SBI implementation is still working to get the hart in the STOPPED state.
4	SUSPENDED	This hart is in a platform specific suspend (or low power) state.

5	SUSPEND_PENDING	The hart has requested to put itself in a platform specific low power state from the STARTED state and the SBI implementation is still working to get the hart in the platform specific SUSPENDED state.
6	RESUME_PENDING	An interrupt or platform specific hardware event has caused the hart to resume normal execution from the SUSPENDED state and the SBI implementation is still working to get the hart in the STARTED state.

Below table lists the services in this group:

Service ID	Service Name	Request type
0x01	ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	HART_START	NORMAL_REQUEST
0x03	HART_STOP	NORMAL_REQUEST
0x04	HART_SUSPEND	NORMAL_REQUEST
0x05	GET_HART_STATUS	NORMAL_REQUEST
0x06	GET_HART_LIST	NORMAL_REQUEST
0x07	GET_SUSPEND_TYPES	NORMAL_REQUEST
0x08	GET_SUSPEND_INFO	NORMAL_REQUEST

4.4.1 Hart State Management Notifications

This service group does not support any event for notification

4.4.2 Service: **ENABLE_NOTIFICATION**

This service allows AP to subscribe to hart state management service group notifications.
Platform can optionally support notifications of events which might occur in the platform. PuC can send these notification messages to AP if they are implemented and AP has subscribed to these. Events supported are described above in Hart State Management Notifications.

Request data

Word	Name	Type	Description
0	**EVENT_ID**	**uint32**	Event to be subscribed for notification.

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** / **Description** RPMI_SUCCESS: when the notifications are subscribed successfully for EVENT_ID. RPMI_ERROR_NOT_FOUND: Event in EVENT_ID is not supported or invalid RPMI_ERROR_NOT_SUPPORTED: Notifications not supported in this service group ● Other errors in section 3.4 Return Status Code

4.4.3. Service: **HART_START**

This service helps start (or power up) a hart with a specified hart-id. Successful completion of this service means that hart with specified hart-id has started execution from the provided start address. The previous state of the hart before this service was called is platform specific. It's possible that hart was already started or hart with specified hart-id does not exist. Implementation should return proper error code in the status field accordingly.

Request data

Word	Name	Type	Description
0	**HART_ID**	**uint32**	Hart id of the HART to be started
1	**START_ADDR_LOW**	**uint32**	Lower 32 bits of start address
2	**START_ADDR_HIGH**	**uint32**	Higher 32 bits of start address

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code **Error Code** / **Description** RPMI_SUCCESS: No error

			RPMI_ERROR_INVALID_PARAMETER — Invalid HART_ID ● Other errors in section 3.4 Return Status Code

4.4.4. Service: **HART_STOP**

This service stops the calling hart. Mechanism to stop the hart is platform specific. Hart can be powered down if supported or it can be put into a deepest sleep state supported. Platform must acknowledge that hart can be stopped and return success. Application Processor must execute WFI upon successful acknowledgement. Platform then proceeds to stop the hart. Detecting the WFI state from PuC is implementation defined. Once the hart is stopped it can only be started using an explicit HART_START service call from the application processor.

Request data

Word	Name	Type	Description
0	**HART_ID**	**uint32**	Hart ID fo the calling HART

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code **Error Code** — **Description** RPMI_SUCCESS — No error RPMI_ERROR_DENIED — Not allowed due to current hart state which is platform specific ● Other errors in section 3.4 Return Status Code

4.4.5. Service: **HART_SUSPEND**

This service puts the calling hart in suspended (or low power) state. Upon success, the calling hart is suspended only after it executes WFI instruction.

Request data

Word	Name	Type	Description

0	**HART_ID**	**uint32**	Hart ID fo the calling HART
1	**SUSPEND_TYPE**	**uint32**	Hart suspend type as defined by the RISC-V SBI HSM extension
2	**RESUME_ADDR_LOW**	**uint32**	*NOTE: Only used for non-retentive suspend types*
3	**RESUME_ADDR_HIGH**	**uint32**	*NOTE: Only used for non-retentive suspend types*

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code **Error Code** / **Description** RPMI_SUCCESS / No error RPMI_ERROR_INVALID_PARAMETER / Invalid Suspend type ● Other errors in section 3.4 Return Status Code

4.4.6. Service: **GET_HART_STATUS**

This service gets the status of a hart with a specified hart ID.

Request data

Word	Name	Type	Description
0	**HART_ID**	**uint32**	Hart ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code **Error Code** / **Description** RPMI_SUCCESS / No error RPMI_ERROR_INVALID_PARAMETER / Invalid HART ID ● Other errors in section 3.4 Return Status Code

1	**HART_STATUS**	**uint32**	Upon failure, 0xffffffff is returned Upon success, one of the following Hart state values is returned

Value	**Description**
0x0	STARTED
0x1	STOPPED
0x2	START_PENDING
0x3	STOP_PENDING
0x4	SUSPENDED
0x5	RESUME_PENDING
0x6-0xFFFFFFFF	Reserved

4.4.7. Service: **GET_HART_LIST**

This service gets the list of a hart with a specified hart ID start index

Request data

Word	**Name**	**Type**	**Description**
0	**START_INDEX**	**uint32**	Starting index of Hart ID

Response data

Word	**Name**	**Type**	**Description**
0	**STATUS**	**int32**	Return Status code

Error Code	**Description**
RPMI_SUCCESS	No error
RPMI_ERROR_INVALID_PARAMETER	Invalid start index

- Other errors in section 3.4 Return Status Code

Word	**Name**	**Type**	**Description**
1	**REMAINING**	**uint32**	Total number of remaining items to be returned
2	**RETURNED**	**uint32**	Total number of items returned

3:	**HART_ID[0]**	**uint32**	Hart ID
	HART_ID[1]	**uint32**	Hart ID
…	**HART_ID[N-1]**	**uint32**	Hart ID

4.4.8. Service: **GET_SUSPEND_TYPES**

This service gets a list of all supported suspend types. The system types in the list must be ordered based on increasing power savings.

Request data

Word	**Name**	**Type**	**Description**
0	**START_INDEX**	**uint32**	Starting index of the item, 0 for the first call, subsequent calls will use the next index of the remaining items.

Response data

Word	**Name**	**Type**	**Description**
0	**STATUS**	**int32**	Return Status code **Error Code** / **Description** RPMI_SUCCESS / No error ● Other errors in section 3.4 Return Status Code
1	**REMAINING**	**uint32**	Total number of remaining items to be returned
2	**RETURNED**	**uint32**	Total number of items returned
3:	**SUSPEND_TYPE[0]**	**uint32**	Suspend Type
	SUSPEND_TYPE[1]	**uint32**	Suspend Type
…	**SUSPEND_TYPE[N-1]**	**uint32**	Suspend Type

4.4.9 Service: **GET_SUSPEND_INFO**

This service gets more details of a specific suspend type

Request data

Word	Name	Type	Description
0	**SUSPEND_TYPE**	**uint32**	Suspend Type

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code **Error Code** / **Description** RPMI_SUCCESS / No error RPMI_ERROR_INVALID_PARAMETER / Invalid Suspend Type ● Other errors in section 3.4 Return Status Code
1	**FLAGS**	**uint32**	**Bits** / **Description** [31] / **0b0**: Counter doesn't stop if this bit is cleared **0b1**: Local timer stops when the hart is suspended if this bit is set. [30:0] / Reserved, must be zero
2	**ENTRY_LATENCY_US**	**uint32**	Entry latency in microsecond(us)
3	**EXIT_LATENCY_US**	**uint32**	Exit latency in microsecond(us)
4	**WAKEUP_LATENCY_US**	**uint32**	Wakeup latency in microsecond(us)
5	**MIN_RESIDENCY_US**	**uint32**	Minimum residency latency in microsecond(us)

4.5 Service Group - **CPPC** (servicegroup_id: 0x00005)

This service group defines the services to control CPU performance by managing a set of registers and a dedicated physical memory block, which will define fast channel access to each hart in the system. Hart writes a specific 32-Bit desired performance number in fast channel memory. PuC decodes the request and performs the corresponding CPPC operations.

CPPC physical memory will have enough space to accommodate all available harts. Hart can send an RPMI message GET_CPPC_PERF_CHAN_ADDR to query the physical address of that particular hart.

Once Hart receives the physical address of the REQ-ACK registers, Hart clears the acknowledgement register and writes a performance counter value in the request register and a sequence number in the sequence number register. On observing this request, PuC changes the performance state and writes the sequence number value of the original request in the Acknowledgement register. That completes the request.

PROBE_CPPC_REG service can be used to check if a particular CPPC register is implemented. The service READ_CPPC_REG can be used to read all implemented CPPC register values and WRITE_CPPC_REG can be used to write the registers.

For extra debugging, Hart can send a GET_CPPC_PERF_POKE message to exclusively request PuC to process the pending CPPC requests, if any.

For more information on CPPC, refer ACPI Specification and CPPC Extension in SBI Specification

CPPC fast channel REQ-ACK memory entry per hart:

Word	Description	Entity which writes this word
0	Request Sequence number register	AP
1	Request register	AP
2	Acknowledgement register	PuC
3	Reserved	None

Below table lists the services in this group:

Service ID	Service Name	Request type
0x01	ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	PROBE_REG	NORMAL_REQUEST
0x03	READ_REG	NORMAL_REQUEST
0x04	WRITE_REG	NORMAL_REQUEST
0x05	GET_FAST_CHANNEL_ADDR	NORMAL_REQUEST
0x06	POKE_FAST_CHANNEL	NORMAL_REQUEST
0x07	GET_HART_LIST	NORMAL_REQUEST

4.5.1 CPPC Notifications

This service group does not support any event for notification

4.5.2 Service: **ENABLE_NOTIFICATION**

This service allows AP to subscribe to CPPC service group notifications.
Platform can optionally support notifications of events which might occur in the platform. PuC can send these notification messages to AP if they are implemented and AP has subscribed to these. Events supported are described above in CPPC Notifications.

Request data

Word	Name	Type	Description
0	**EVENT_ID**	**uint32**	Event to be subscribed for notification.

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** / **Description** RPMI_SUCCESS — when the notifications are subscribed successfully for EVENT_ID. RPMI_ERROR_NOT_FOUND — Event in EVENT_ID is not supported or invalid RPMI_ERROR_NOT_SUPPORTED — Notifications not supported in this service group ● Other errors in section 3.4 Return Status Code

Error Code	Description
RPMI_SUCCESS	when the notifications are subscribed successfully for EVENT_ID.
RPMI_ERROR_NOT_FOUND	Event in EVENT_ID is not supported or invalid
RPMI_ERROR_NOT_SUPPORTED	Notifications not supported in this service group

- Other errors in section 3.4 Return Status Code

4.5.3. Service: **PROBE_REG**

Probe for CPPC register and get its length in bytes. If the register is not implemented, reg_length returned will be zero.

Request data

Word	Name	Type	Description
0	**HART_ID**	**uint32**	Hart ID
1	**REG_ID**	**uint32**	Register ID (more details in CPPC Section of SBI Spec)

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code
1	**REG_LENGTH**	**uint32**	Register length

Error Code	**Description**
RPMI_SUCCESS	CPPC register probed successfully
RPMI_ERROR_NOT_FOUND	Hart id in request not found

- Other errors in section [3.4 Return Status Code]

4.5.4. Service: **READ_REG**

Read CPPC register value.

Request data

Word	Name	Type	Description
0	**HART_ID**	**uint32**	Hart ID
1	**REG_ID**	**uint32**	Register ID

Response data

Word	Name	Type	Description
0	**STATUS**	**uint32**	Return Status code

Error Code	**Description**
RPMI_SUCCESS	CPPC register read successfully
RPMI_ERROR_NOT_FOUND	Hart id in request not found
RPMI_ERROR_INVALID_PARAMETER	Invalid Register ID or Not implemented. AP must probe for register id using probe service before reading.

- Other errors in section [3.4 Return Status Code]

1	**DATA_LOW**	**uint32**	Lower 32 bits of data
2	**DATA_HIGH**	**uint32**	Higher 32 bits of data

4.5.5. Service: **WRITE_REG**

Write a CPPC register

Request data

Word	**Name**	**Type**	**Description**
0	**HART_ID**	**uint32**	Hart ID
1	**REG_ID**	**uint32**	Register ID
2	**DATA_LOW**	**uint32**	Lower 32 bits of data
3	**DATA_HIGH**	**uint32**	Higher 32 bits of data

Response data

Word	**Name**	**Type**	**Description**
0	**STATUS**	**int32**	Return Status code **Error Code** / **Description** RPMI_SUCCESS / CPPC register write successfully RPMI_ERROR_NOT_FOUND / Hart id in request not found RPMI_ERROR_INVALID_PARAMETER / Invalid Register ID or Not implemented. AP must probe for register id using probe service before writing. ● Other errors in section 3.4 Return Status Code

4.5.6. Service: **GET_FAST_CHANNEL_ADDR**

Request for physical address of CPPC fast channel for the hart ID specified, this physical address shall be used to write the value of CPPC desired performance register.

Request data

Word	**Name**	**Type**	**Description**
0	**HART_ID**	**uint32**	Hart ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code <table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_SUCCESS</td><td>Service processed successfully</td></tr><tr><td>RPMI_ERR_NOT_FOUND</td><td>Invalid HART ID</td></tr><tr><td>RPMI_ERR_NOT_SUPPORTED</td><td>Operation not supported</td></tr></table>● Other errors in section 3.4 Return Status Code
1	**FLAGS**	**uint32**	<table><tr><th>Bits</th><th>Description</th></tr><tr><td>[31:3]</td><td>Reserved</td></tr><tr><td>[2:1]</td><td>Doorbell register width **0b00**: Doorbell register is 8 bits wide. **0b01**: Doorbell register is 16 bits wide. **0b10**: Doorbell register is 32 bits wide. **0b11**: Doorbell register is 64 bits wide.</td></tr><tr><td>[0]</td><td>**0b0**: Doorbell is unsupported **0b1**: Doorbell is supported</td></tr></table>
2	**PHYS_ADDR_LOW**	**uint32**	Lower 32-bits of Physical address
3	**PHYS_ADDR_HIGH**	**uint32**	Higher 32-bits of Physical address
4	**DB_ADDR_LOW**	**uint32**	Lower 32-bits of Doorbell address
5	**DB_ADDR_HIGH**	**uint32**	Higher 32-bits if Doorbell address
6	**DB_ID_LOW**	**uint32**	Lower 32-bits of Doorbell ID
7	**DB_ID_HIGH**	**uint32**	Higher 32-bits if Doorbell ID

4.5.7. Service: **POKE_FAST_CHANNEL**

Debug request message to poke the PuC FW to process the pending CPPC messages if any.

Request data

- NA

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code

Error Code	Description
RPMI_SUCCESS	Fast channel poked successfully

- Other errors in section 3.4 Return Status Code

4.9.8. Service: **GET_HART_LIST**

This service gets the list of a hart with a specified hart ID start index

Request data

Word	Name	Type	Description
0	**START_INDEX**	**uint32**	Starting index of Hart ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code
1	**REMAINING**	**uint32**	Total number of remaining items to be returned
2	**RETURNED**	**uint32**	Total number of items returned
3	**HART_ID[0]**	**uint32**	Hart ID
4	**HART_ID[1]**	**uint32**	Hart ID

Error Code	Description
RPMI_SUCCESS	No error
RPMI_ERROR_INVALID_PARAMETER	Invalid start index

- Other errors in section 3.4 Return Status Code

...	**HART_ID[N]**	**uint32**	Hart ID

4.6 Service Group - **VOLTAGE** (servicegroup_id: 0x00006)

The complete system can be classified into multiple domains for voltage control. Each domain is the logical group of one or more devices that have a single voltage source. The action corresponding to the messages in this group controls the voltage source which affects the device or the group of devices together since those devices share the same voltage source. Each domain is identified by domain_id which is a 32-Bit integer starting from 0.

Below table lists the services in this group:

Service ID	Service Name	Request type
0x01	ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	GET_NUM_DOMAINS	NORMAL_REQUEST
0x03	GET_DOMAIN_ATTRIBUTES	NORMAL_REQUEST
0x04	GET_DOMAIN_LEVELS	NORMAL_REQUEST
0x05	SET_DOMAIN_CONFIG	NORMAL_REQUEST
0x06	GET_DOMAIN_CONFIG	NORMAL_REQUEST
0x07	SET_DOMAIN_LEVEL	NORMAL_REQUEST
0x08	GET_DOMAIN_LEVEL	NORMAL_REQUEST

4.6.1 Voltage Notifications

This service group does not support any event for notification

4.6.2 Service: **ENABLE_NOTIFICATION**

This service allows AP to subscribe to voltage service group notifications.
Platform can optionally support notifications of events which might occur in the platform. PuC can send these notification messages to AP if they are implemented and AP has subscribed to these. Events supported are described above in Voltage Notifications.

Request data

Word	Name	Type	Description
0	**EVENT_ID**	**uint32**	Event to be subscribed for notification.

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	Description
RPMI_SUCCESS	when the notifications are subscribed successfully for EVENT_ID.
RPMI_ERROR_NOT_FOUND	Event in EVENT_ID is not supported or invalid
RPMI_ERROR_NOT_SUPPORTED	Notifications not supported in this service group

- Other errors in section 3.4 Return Status Code

4.6.3 Service: **GET_NUM_DOMAINS**

Request for number of domains available in the system

Request data

- NA

Response dat

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code
1	**NUM_DOMAINS**	**uint32**	Number of domains

Error Code	Description
RPMI_SUCCESS	Voltage domains returned successfully

- Other errors in section 3.4 Return Status Code

4.6.4 Service: **GET_DOMAIN_ATTRIBUTES**

Each domain may have supported multiple voltage levels which are allowed by the domain to operate. This message for each domain returns a domain name which is a null terminated ASCII string of 16-bytes. Number of levels represents the total number of voltage levels supported by a voltage domain. Transition latency is the max time taken for voltage to stabilize when changed on the regulator. Voltage levels depending on the hardware can be of different formats and this service currently supports Simple Linear, Multi-Linear and Discrete range. More voltage formats can be supported in future if required.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Voltage domain Id

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	Description
RPMI_SUCCESS	Voltage domain attributes returned successfully
RPMI_ERROR_NOT_FOUND	Voltage domain id not found

- Other errors in section 3.4 Return Status Code

Word	Name	Type	Description
1	**FLAGS**	**uint32**	

Bits	Description
[31:30]	**VOLTAGE_FORMAT** **0b00**: Fixed Voltage **0b01**: Simple Linear Range containing single triplet (min_uV, max_uV, step_uV). **0b10**: Multi-Linear range containing multiple linear ranges of type 0x1 where each range contains (min_uV, min_sel, max_sel, step_uV). **0b11**: Discrete range.
[29:1]	**RESERVED**

			[0] — **ALWAYS_ON** **0b0**: Voltage domain can be enabled/disabled **0b1**: Voltage domain is always-on, voltage value can be changed in the supported voltage range.
2	**NUM_LEVELS**	**uint32**	Number of voltage levels supported by domain. Some values are dependent on the VOLTAGE_FORMAT **Value** — **scription** 1 — when VOLTAGE_FORMAT=0x0 3 — when VOLTAGE_FORMAT=0x1 N — when VOLTAGE_FORMAT= 0x2 *or* 0x3. Based on the format here each item can be a single voltage value or tuple of values. Check VOLTAGE_FORMAT field in FLAGS
3	**TRANSITION_LATENCY**	**uint32**	Transition latency
4:7	**VOLTAGE_DOMAIN_NAME**	**uint8[16]**	Voltage domain name

4.6.5 Service: **GET_DOMAIN_LEVELS**

Each domain may support multiple voltage levels which are allowed by the domain to operate. Depending on the Power supply/Voltage Regulator the domain may support voltage levels which can be either discrete or stepwise range. Discrete voltage range will be in sequence starting from lower voltage value at the lowest index to higher voltage level with increasing index. Number of voltage levels returned depends on the format of the voltage level. Total words required for the number levels according to the format in one message cannot exceed the total words available in one message DATA field. If they exceed then PuC will return the number of levels which can be accommodated in one message and set the REMAINING field accordingly. AP, when REMAINING field is not 0 must call this service again with appropriate VOLTAGE_LEVEL_INDEX set to get the remaining voltage levels. It's possible that multiple service calls may be required to get all the voltage levels.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Domain ID
1	**VOLTAGE_LEVEL_INDEX**	**uint32**	Level index

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** — **Description** RPMI_SUCCESS — Voltage domain levels returned successfully RPMI_ERROR_NOT_FOUND — Voltage domain id not found RPMI_ERR_INVALID_PARAM — voltage_level_index is not in valid range ● Other errors in section 3.4 Return Status Code
1	**FLAGS**	**uint32**	RESERVED and must be 0
2	**REMAINING**	**uint32**	Remaining number of items to be returned
3	**RETURNED**	**uint32**	Number of items returned
4:	**VOLTAGE[0]**	**int32**	Voltage array where each entry in the array is voltage level in microvolts(uV), N is specified by the FLAGS[11:0]. Voltage represented in microVolt (uV). If the bits in attribute service FLAGS[31:30] are set to 0, VOLTAGE[0] contains a fixed voltage level in the array. *VOLTAGE[0]: volt_uV* If the bits in attribute service FLAGS[31:30] are set to 1, it means that the voltage array contains three entries as below: *VOLTAGE[0]: min_uV* *VOLTAGE[1]: max_uV* *VOLTAGE[2]: step_uV* If the bits in attribute service FLAGS[31:30] are set to 2, it indicates that the voltage array contains multiple groups of four entries. Each group represent a linear voltage range and consists of the following entries: *VOLTAGE[0] = min_uV* *VOLTAGE[1] = min_sel*

			VOLTAGE[2] = max_sel *VOLTAGE[3] = step_uV* If the bits in [attribute service] FLAGS[31:30] are set to 3, it means that the entry array contains discrete voltage levels listed in ascending numeric order. *VOLTAGE[0]: Voltage #0 (min_uV)* *VOLTAGE[1]: Voltage #1* *VOLTAGE[2]: Voltage #2* *VOLTAGE[N - 1]: Voltage #(N – 1) (max_uV)*
	VOLTAGE[1]	**int32**	
…	**VOLTAGE[N-1]**	**int32**	

4.6.6 Service: **SET_DOMAIN_CONFIG**

Set voltage config message enable or disable any domain. Enabling the voltage means applying the domain with the voltage level to operate normally. AP can enable the voltage to any domain without knowing the actual voltage levels.
Disabling the voltage level means disabling the voltage supply to the domain.

uint32 config encodes these discrete settings which do not require AP to know the voltage level

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Domain ID
1	**CONFIG**	**uint32**	

Bits	Description
[31:1]	Reserved
[0]	**0b0**: Disable Voltage for the domain. **0b1**: Enable Voltage for domain to operate normally

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	Description
RPMI_SUCCESS	Voltage configuration is set successfully
RPMI_ERROR_NOT_FOUND	Voltage domain id not found
RPMI_ERROR_INVALID_PARAMETER	Voltage config is not supported by the specified voltage domain id

- Other errors in section 3.4 Return Status Code

4.6.7 Service: **GET_DOMAIN_CONFIG**

Get voltage config message request for the configuration of the voltage domain currently set. uint32 config encodes these modes.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Domain ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** | **Description** RPMI_SUCCESS | Voltage configuration is returned successfully RPMI_ERROR_NOT_FOUND | Voltage domain id not found ● Other errors in section 3.4 Return Status Code
1	**CONFIG**	**uint32**	**Value** | **Description** 0x0 | Disabled

			<table><tr><td>0x1</td><td>Enabled</td></tr></table>

4.6.8 Service: **SET_DOMAIN_LEVEL**

Set the voltage level in microvolts(uV) of a voltage domain

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Domain ID
1	**VOLTAGE_LEVEL**	**int32**	Voltage level

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code <table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_SUCCESS</td><td>Voltage level is set successfully</td></tr><tr><td>RPMI_ERROR_NOT_FOUND</td><td>Voltage domain id not found</td></tr><tr><td>RPMI_ERROR_INVALID_PARAMETER</td><td>Voltage level is not supported by the specified voltage domain id</td></tr></table>● Other errors in section 3.4 Return Status Code

4.6.9 Service: **GET_DOMAIN_LEVEL**

Get the current voltage level in microvolts(uV) of a voltage domain.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Domain ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	Description
RPMI_SUCCESS	Voltage level is returned successfully
RPMI_ERROR_NOT_FOUND	Voltage domain id not found

- Other errors in section 3.4 Return Status Code

1	**VOLTAGE_LEVEL**	**int32**	Voltage Level

4.7 Service Group - **CLOCK** (servicegroup_id: 0x00007)

This service group is for the management of system clocks. Services defined in this group are used to enable or disable clocks, and to set/get clock rates.
Each clock in the system is identified by the clock id which is an integer identifier assigned to each clock. The mapping of clock_id and clock is known to both AP and Platform Microcontroller. Clock ID identifiers are sequential and start from 0.
The device or the group of devices which share the same clock source becomes a single clock domain, which is identified by the clock_id. Any change in the clock source affects the whole domain which can contain multiple devices. This topology of devices and clock source is dependent on how the system is designed and implementation specific. OS can discover this topology through firmware tables (DT/ACPI).

4.7.1 Clock Rate Value Format

Each clock rate value is a tuple of two 32-bit values(uint32, uint32) represented as (clock_rate_low, clock_rate_high) and packed in the same order where clock_rate_low is at the lower index than the clock_rate_high

Below table lists the services in this group:

Service ID	Service Name	Request type
0x01	ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	GET_NUM_CLOCKS	NORMAL_REQUEST
0x03	GET_ATTRIBUTES	NORMAL_REQUEST
0x04	GET_SUPPORTED_RATES	NORMAL_REQUEST
0x05	SET_CONFIG	NORMAL_REQUEST
0x06	GET_CONFIG	NORMAL_REQUEST

0x07	SET_RATE	NORMAL_REQUEST
0x08	GET_RATE	NORMAL_REQUEST

4.7.2 Clock Notifications

This service group does not support any event for notification

4.7.3 Service: **ENABLE_NOTIFICATION**

This service allows AP to subscribe to clock service group notifications.
Platform can optionally support notifications of events which might occur in the platform. PuC can send these notification messages to AP if they are implemented and AP has subscribed to these. Events supported are described above in Clock Notifications.

Request data

Word	Name	Type	Description
0	**EVENT_ID**	**uint32**	Event to be subscribed for notification.

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	**Description**
RPMI_SUCCESS	when the notifications are subscribed successfully for EVENT_ID.
RPMI_ERROR_NOT_FOUND	Event in EVENT_ID is not supported or invalid
RPMI_ERROR_NOT_SUPPORTED	Notifications not supported in this service group

- Other errors in section 3.4 Return Status Code

4.7.4 Service: **GET_NUM_CLOCKS**

Request for number of clocks available in the system. All supported clocks in the system are designated by an integer identifier called clock_id. Clock_id are sequential starting from 0.

Request data

- NA

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** / **Description** RPMI_SUCCESS / Clocks number returned successfully • Other errors in section [3.4 Return Status Code]
1	**NUM_CLOCKS**	**uint32**	Number of Clocks

4.7.5 Service: **GET_ATTRIBUTES**

This service returns the attributes of a clock like name of the clock which is an array ASCII string of 16-Bytes. Transition latency which is the worst case latency for the clock to return to a stable state once clock configuration is changed. Number of Clock rates supported by the requested clock. Flags which encode the clock format supported by the clock.

Current supported clock formats are discrete, which is an array of discrete values where each value represents a clock rate. Another format is linear range which is represented by (min_Hz, max_Hz, step_Hz). In future more clock formats can be supported if required.

Request data

Word	Name	Type	Description
0	**CLOCK_ID**	**uint32**	Clock ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** / **Description** RPMI_SUCCESS / Clock attributes returned successfully

2			<table><tr><td>RPMI_ERROR_NOT_FOUND</td><td>Clock id not found</td></tr></table> • Other errors in section 3.4 Return Status Code
1	**FLAGS**	**uint32**	<table><tr><th>Bits</th><th>Description</th></tr><tr><td>[31:30]</td><td>**CLOCK_FORMAT** **0b00**: Discrete Format Each element in the **CLOCK_RATE** array is a supported discrete clock rate value packed in ascending order. Each rate is in Hertz. **0b01**: Linear Range The **CLOCK_RATE** array contains the triplet (min_Hz, max_Hz, step_Hz). Each item in the triplet is a clock rate value CLOCK_RATE[0] = min_Hz (*lowest physical rate that the clock can synthesize)* CLOCK_RATE[1] = max_Hz (*highest physical rate that the clock can synthesize)* CLOCK_RATE[2] = step_Hz (*Step between two successive rates*)</td></tr><tr><td>[29:0]</td><td>**RESERVED** and must be 0</td></tr></table>
2	**NUM_RATES**	**uint32**	Number of Clock rates of type depending on **CLOCK_FORMAT**
3	**TRANSITION_LATENCY**	**uint32**	Transition latency
4:7	**CLOCK_NAME**	**uint8[16]**	Clock name

4.7.6 Service: **GET_SUPPORTED_RATES**

Each domain may support multiple clock rate values which are allowed by the domain to operate. Message can also pass the clock_rate_index which is the index to the first rate value to be described in the return rate array. If all supported rate values are required then this index value can be 0. If the CLOCK_FORMAT is discrete then the clock rate in the received data is an array of supported discrete rate values packed in ascending order starting from the lower index in the CLOCK_RATE field. If the CLOCK_FORMAT is a linear range, then the CLOCK_RATE array contains a triplet of (min_Hz, max_Hz, step_Hz) where each item in the triplet is a clock rate value.

Total words required for the number of clock rates according to the format in one message cannot exceed the total words available in one message DATA field. If they exceed then PuC will return the number of clock rates which can be accommodated in one message and set the REMAINING field accordingly. AP, when REMAINING field is not 0 must call this service again with appropriate CLOCK_RATE_INDEX set to get the remaining clock rates. It's possible that multiple service calls may be required to get all the clock rates.
In case the CLOCK_FORMAT is a linear range the RETURNED field will be set to 3.

Request data

Word	Name	Type	Description
0	**CLOCK_ID**	**uint32**	Clock ID
1	**CLOCK_RATE_INDEX**	**uint32**	Rate index

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** / **Description** RPMI_SUCCESS / Clock supported rates returned successfully RPMI_ERROR_NOT_FOUND / Clock id not found RPMI_ERROR_OUT_OF_RANGE / clock_rate_index is not in valid range ● Other errors in section 3.4 Return Status Code
1	**FLAGS**	**uint32**	RESERVED and must be 0
2	**REMAINING**	**uint32**	Remaining number of items to be returned
3	**RETURNED**	**uint32**	Number of items returned
4:	**CLOCK_RATE[0]**	**(uint32, uint32)**	Clock rate value
	CLOCK_RATE[1]	**(uint32, uint32)**	Clock rate value
…	**CLOCK_RATE[N-1]**	**(uint32, uint32)**	Clock rate value

4.7.7 Service: **SET_CONFIG**

Request data

Word	Name	Type	Description
0	**CLOCK_ID**	**uint32**	Clock ID
1	**CONFIG**	**uint32**	<table><tr><th>Bits</th><th>Description</th></tr><tr><td>[31:1]</td><td>Reserved</td></tr><tr><td>[0]</td><td>0: Disable 1: Enable</td></tr></table>

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code <table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_SUCCESS</td><td>Clock configuration is set successfully</td></tr><tr><td>RPMI_ERROR_NOT_FOUND</td><td>Clock id not found</td></tr><tr><td>RPMI_ERROR_INVALID_PARAMETER</td><td>Clock config is not supported by the specified clock id</td></tr></table> ● Other errors in section 3.4 Return Status Code

4.7.8 Service: **GET_CONFIG**

Get the current status of a clock, if it's enabled or disabled.

Request data

Word	Name	Type	Description
0	**CLOCK_ID**	**uint32**	Clock ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	**Description**
RPMI_SUCCESS	Clock configuration is returned successfully
RPMI_ERROR_NOT_FOUND	Clock id not found

- Other errors in section 3.4 Return Status Code

Word	Name	Type	Description
1	**CONFIG**	**uint32**	

Value	**Description**
0x0	Disabled
0x1	Enabled

4.7.9 Service: **SET_RATE**

Set the clock rate.

Request data

Word	Name	Type	Description
0	**CLOCK_ID**	**uint32**	Clock ID
1	**FLAGS**	**uint32**	

Bits	**Description**
[31:30]	Clock Rate Round Up/Down: **0b00**: Platform rounds up/down autonomously to choose a physical rate closest to the requested rate. **0b01**: Round down. **0b10**: Round Up **0b11**: *Reserved*
[29:0]	Reserved, must be zero

2	**CLOCK_RATE_LOW**	**uint32**	
3	**CLOCK_RATE_HIGH**	**uint32**	

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	Description
RPMI_SUCCESS	Clock rate is set successfully
RPMI_ERROR_NOT_FOUND	Clock id not found
RPMI_ERROR_INVALID_PARAMETER	Clock rate is not supported by the specified clock id

- Other errors in section 3.4 Return Status Code

4.7.10 Service: **GET_RATE**

Get the current clock rate value.

Request data

Word	Name	Type	Description
0	**CLOCK_ID**	**uint32**	Clock ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	Description
RPMI_SUCCESS	Clock rate returned successfully
RPMI_ERROR_NOT_FOUND	Clock id not found

			• Other errors in section 3.4 Return Status Code
1	**CLOCK_RATE_LOW**	**uint32**	Lower 32-Bits of the physical rate in Hertz.
2	**CLOCK_RATE_HIGH**	**uint32**	Upper 32-Bits of the physical rate in Hertz.

4.8 Service Group - **DEVICE_POWER** (servicegroup_id: 0x00008)

This device power service group provides messages to manage the power states of a device power domain. This service group is only used for device power management since System and CPU power management is handled by already defined service groups like SYSTEM RESET, SYSTEM SUSPEND and HART STATE MANAGEMENT.
A domain can consist of one device if its power states can be controlled independently or it may also have multiple devices if they all share the same power control lines and power state can only be changed collectively.
Each domain must support ON and OFF states along with custom power states which are discoverable. Domains may also have power states which may preserve the context. Level of context preserved will depend on the level of power state.
Power states for domains will be discovered via DT or ACPI where the values for ON and OFF are already fixed and known. Power state encodes both the power state value and the context preserved or lost information corresponding to that state.

POWER_STATE (uint32)	Field	Description
POWER_STATE[31]	CONTEXT	0: Context is preserved 1: Context is lost
POWER_STATE[30:16]	Reserved	Reserved
POWER_STATE[15:0]	VALUE	**Value** / **Description**: 0x0000 — **ON** with POWER_STATE[31] = 1 0x0001 — Reserved 0x0002 — Reserved 0x0003 — **OFF** with POWER_STATE[31] = 0 0x0004 - 0x0FFF — Reserved 0x1000 - 0xFFFF — Vendor Specific States

Below table lists the services in this group:

Service ID	Service Name	Request type
0x01	ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	GET_DEVICE_POWER_DOMAINS	NORMAL_REQUEST
0x03	GET_DEVICE_POWER_DOMAIN_ATTRIBUTES	NORMAL_REQUEST
0x04	SET_DEVICE_POWER_STATE	NORMAL_REQUEST
0x05	GET_DEVICE_POWER_STATE	NORMAL_REQUEST

4.8.1 Device Power Notifications

This service group does not support any event for notification

4.8.2 Service: **ENABLE_NOTIFICATION**

This service allows AP to subscribe to device power service group notifications.
Platform can optionally support notifications of events which might occur in the platform. PuC can send these notification messages to AP if they are implemented and AP has subscribed to these. Events supported are described above in Device Power Notifications.

Request data

Word	Name	Type	Description
0	**EVENT_ID**	**uint32**	Event to be subscribed for notification.

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** / **Description** RPMI_SUCCESS: when the notifications are subscribed successfully for EVENT_ID. RPMI_ERROR_NOT_FOUND: Event in EVENT_ID is not supported or invalid

			<table><tr><td>RPMI_ERROR_NOT_SUPPORTED</td><td>Notifications not supported in this service group</td></tr></table> ● Other errors in section 3.4 Return Status Code

4.8.3. Service: **GET_DEVICE_POWER_DOMAINS**

This service is used to query the number of device power domains available which can be controlled by the client. The number of domains returned can be less than the actual number of domains present with the platform. The number of domains returned are allowed to be managed by the client.

Request data

- NA

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code <table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_SUCCESS</td><td>success</td></tr></table> ● Other errors in section 3.4 Return Status Code
1	**NUM_DOMAINS**	**uint32**	Number of domains

4.8.4. Service: **GET_DEVICE_POWER_DOMAIN_ATTRIBUTES**

This service is used to query the attributes of a device power domain.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Device power domain id

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	Description
RPMI_SUCCESS	Device power domain attributes returned successfully
RPMI_ERROR_NOT_FOUND	Device power domain id not found

- Other errors in section 3.4 Return Status Code

1	**FLAGS**	**uint32**	Reserved
2	**TRANSITION_LATENCY**	**uint32**	Worst case transition latency of domain from one power state to another
3:6	**DEVICE_POWER_DOMAIN_NAME**	**uint8[16]**	Device power domain name

4.8.5. Service: **SET_DEVICE_POWER_DOMAIN_STATE**

This service is used to change the power state of a device power domain.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Device Power Domain ID
1	**POWER_STATE**	**uint32**	Power state This field indicates the power state to which the power domain should transition. The specific power states and their meanings may vary depending on the implementation, but generally, they include values such as "ON", "OFF" and vendor specific power state. See Power States table in the service group description.

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	Description

			<table><tr><td>RPMI_SUCCESS</td><td>Device power domain attributes returned successfully</td></tr><tr><td>RPMI_ERROR_NOT_FOUND</td><td>Device power domain id not found</td></tr><tr><td>RPMI_ERROR_INVALID_PARAMETER</td><td>Invalid or Not supported POWER_STATE value</td></tr><tr><td>RPMI_ERROR_DENIED</td><td>Client does not have permissions to change the Device power domain power state.</td></tr><tr><td>RPMI_ERROR_HW_FAULT</td><td>Failed due to hardware error</td></tr></table> ● Other errors in section 3.4 Return Status Code

4.8.6. Service: **GET_DEVICE_POWER_DOMAIN_STATE**

This service is used to get the current power state of a device power domain.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Device power domain ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code <table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_SUCCESS</td><td>Device power domain attributes returned successfully</td></tr><tr><td>RPMI_ERROR_NOT_FOUND</td><td>Device power domain id not found</td></tr><tr><td>RPMI_ERROR_DENIED</td><td>Client does not have permissions to read the Device power domain power state.</td></tr></table>

			● Other errors in section 3.4 Return Status Code
1	**POWER_STATE**	**uint32**	This field indicates the current power state of the specified domain. The power state can be one of several predefined values, such as ON, OFF, or vendor specific implementation. SeePower States table in service group description

4.9 Service Group - **PERFORMANCE** (servicegroup_id: 0x00009)

This performance domain extension is designed for managing the performance of a group of devices or application processors that operate in the same performance domain. Unlike legacy performance control mechanisms where the OS was responsible to directly control the voltage and clocks this mechanism instead operates on an metric less integer performance scale. Each integer value on this scale represents a performance operating point. What this scale represents and the metric is completely platform dependent. Values on this scale are discrete and the platform is given complete control over mapping these performance operating points to performance states which eventually gets converted into hardware parameters like voltage and frequency, etc. The mapping between levels and frequencies can be as simple as using a multiplication factor of 1000.

CPPC service group is also meant for performance control but it's only for the application processors. This service group is primarily meant for devices like GPUs, Accelerators, etc but can also be used for application processors.

It is important to note that performance domains should not be confused with power domains. A performance domain is defined as a set of devices that must always run at the same performance level, while a power domain is defined as a set of devices that can be turned on or off together for power management purposes.

The Performance service groups provide a set of services for managing performance domains. These services include retrieving performance domain information, setting the performance level, obtaining the current performance level of a performance domain, and performing other related operations. These operations are described in more detail in the following section.

Below table lists the services in this group:

Service ID	Service Name	Request type
0x01	ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	GET_PERF_DOMAINS	NORMAL_REQUEST
0x03	GET_PERF_DOMAIN_ATTRIBUTES	NORMAL_REQUEST

0x04	GET_PERF_DOMAIN_LEVELS	NORMAL_REQUEST
0x05	GET_PERF_LEVEL	NORMAL_REQUEST
0x06	SET_PERF_LEVEL	NORMAL_REQUEST
0x07	GET_PERF_LIMIT	NORMAL_REQUEST
0x08	SET_PERF_LIMIT	NORMAL_REQUEST
0x09	GET_PERF_DOMAIN_FAST_CHANNEL_ADDR	NORMAL_REQUEST

4.9.1. Performance Notifications

When a client registers for performance change notifications, the platform will send notification to the client whenever there is a change in the performance level, performance limit or the performance power of a specific performance domain. This notification is typically sent by the platform control processor to inform clients in the system about changes in the performance domain.

Performance Service Group Events:

Event ID	Name	Event Data			Description
		Word	**Type**	**Description**	
0x001	**PERF_POWER_CHANGE**	0	uint32	Performance domain ID whose power changed	**Performance power changed notification.**
		1	uint32	New Power value(uW)	
0x002	**PERF_LIMIT_CHANGE**	0	uint32	Performance domain ID whose performance limit changed	**Performance limit changed notification.**
		1	uint32	New Max Perf Level	
		2	uint32	New Min Perf Level	

0x003	**PERF_LEVEL_CHANGE**		**Performance level changed notification**.

Word	Type	Description
0	uint32	Performance domain ID whose performance level changed
1	uint32	New Perf Level

4.9.2. Service: **ENABLE_NOTIFICATION**

This service is to enable or disable the performance changed notification event. This notification is sent from the PuC when the performance level, performance limit or performance power of a performance domain has changed. This allows the system to adjust its behavior in response to performance changes and ensure that it is operating within its desired performance level.

Request data

Word	Name	Type	Description
0	**EVENT_ID**	**uint32**	Event ID which needs to be subscribed

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code

Error Code	Description
RPMI_SUCCESS	Notification set successfully
RPMI_ERR_NOT_FOUND	Cant find domain ID
RPMI_ERR_NOT_SUPPORTED	This function is not supported.

- Other errors in section 3.4 Return Status Code

4.9.3. Service: **GET_PERF_DOMAIN**

Returns the number of performance domains supported by the system.

The number of performance domains can vary depending on the hardware platform and implementation. In general, performance domains are used to group related hardware components, such as CPUs, GPUs, memory, and peripherals, into separate domains that can be independently controlled and managed. This allows for more fine-grained control over the performance of specific components, which can be important for optimizing system performance and power consumption.

Request data

- NA

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** | **Description** RPMI_SUCCESS | Voltage domains returned successfully ● Other errors in section 3.4 Return Status Code
1	**NUM_DOMAINS**	**uint32**	Number of domains

4.9.4. Service: **GET_PERF_DOMAIN_ATTRIBUTES**

This service is used to retrieve the attributes of a specific performance domain. These attributes provide information about the performance capabilities and constraints of the domain, such as the performance limit and performance level.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Performance domain ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code **Error Code** | **Description**

			<table><tr><td>RPMI_SUCCESS</td><td>The performance domain attributes are implemented and supported</td></tr><tr><td>RPMI_ERROR_NOT_FOUND</td><td>Can't find the performance domain ID</td></tr></table>● Other errors in section 3.4 Return Status Code
1	**FLAGS**	**uint32**	Return Status code<table><tr><td>**Bits**</td><td>**Description**</td></tr><tr><td>[31]</td><td>**PERF_LIMIT_SETTING** This attribute indicates whether the platform allows software to set the performance limit/range for a specific performance domain. **0b0**: Can't set performance limit **0b1**: Can set performance limit</td></tr><tr><td>[30]</td><td>**PERF_LEVEL_SETTING** This attribute indicates whether the platform allows software to set the performance level for a specific performance domain. **0b0**: Can't set performance level **0b1**: Can set performance level</td></tr><tr><td>[29]</td><td>**FAST_CHANNEL_SUPPORT** This attribute indicates whether the platform supports low latency communication channels for performance domain management. **0b0**: Not Supported **0b1**: Supported</td></tr><tr><td>[28:21]</td><td>**TOTAL_NUM_PERF_LEVELS** Total number of performance levels supported.</td></tr><tr><td>[20:0]</td><td>Reserved, must set to zero.</td></tr></table>
2	**RATE_LIMIT_US**	**uint32**	Minimum amount of time that needs to pass between two consecutive requests, in microsecond(us).
3:6	**PERF_DOMAIN _NAME**	**uint8[16]**	A NULL-terminated string for performance domain name. Up to 16-Bytes.

4.9.5. Service: **GET_PERF_DOMAIN_LEVELS**

This service provides a list of the available Performance levels or also called Operating performance points (OPPs) for a specific performance domain. These represent different performance levels that can be set for the components in the domain, and are defined by a combination of frequency, power cost and other parameters. By utilizing this information, the OS can choose the optimal performance level for the system workload and power constraints.

```
/* Pseudocode to retrieve the list of the supported OPP */

index = 0;
num = 0;
/* Allocate a buffer based on the value returned from the flags[28:21] */
total_num_levels = perf_domain_attributes.flags[28:21];

loop:
        list = get_domain_opp_list(index, domain_id);
        entry_num = 0;

        for (i = 0; i < list.returned; i++, num++) {
                opp[num].level = list.entry[entry_num++];
                opp[num].power = list.entry[entry_num++];
                opp[num].rate_limit = list.entry[entry_num++];
        }

        /* Check if there are remaining OPP to be read */
        if (list.remaining) {
                index += list.returned;
                goto loop;
        }
```

The pseudocode above demonstrates the process for retrieving the level information for a specific performance domain. First, the number of performance levels is determined by checking the **FLAGS[28:21]** parameter returned by the **GET_PERF_DOMAIN_ATTRIBUTES** service.

Total words required for the number of performance levels according to the format in one message cannot exceed the total words available in one message DATA field. If they exceed then PuC will return the number of levels which can be accommodated in one message and set the **REMAINING** field accordingly. AP, when **REMAINING** field is not 0 must call this service again with appropriate **PERF_LEVEL_INDEX** set to get the remaining levels. It's possible that multiple service calls may be required to get all the levels.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Performance domain ID. This field specifies the identifier of the performance domain whose OPPs are being described.
1	**PERF_LEVEL_INDEX**	**uint32**	Start array index to read. First index starts from zero.

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code **Error Code** | **Description** RPMI_SUCCESS | The performance domain OPP is implemented and supported RPMI_ERROR_NOT_FOUND | Can’t find the performance domain ID RPMI_ERROR_INVALID_PARAM | Invalid value in the index field ● Other errors in section 3.4 Return Status Code
1	**FLAGS**	**uint32**	Reserved, must set to zero.
2	**REMAINING**	**uint32**	Remaining number of items to be returned
3	**RETURNED**	**uint32**	The number of levels returned from this request. Each level comprises three 32-bit words.
4:	**LEVEL[0]**	**uint32**	Performance Level **Word** | **Description** 0 | OPP-level, a unique ID representing the performance level within the OPP table 1 | Power Cost in microwatt (uW). This is an optional parameter. Set to value of zero to indicate that power cost is not returned by the platform 2 | Transition latency in microsecond (us)
	LEVEL[1]	**uint32**	Performance Level

…	**LEVEL[N-1]**	**uint32**	Performance Level

4.9.6. Service: **GET_PERF_LEVEL**

This service is used to obtain the current performance level of a specific performance domain in the system.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Performance domain ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code **Error Code** / **Description**: RPMI_SUCCESS — The performance level is retrieved successfully RPMI_ERROR_NOT_FOUND — Can't find the performance domain ID ● Other errors in section 3.4 Return Status Code
1	**LEVEL**	**uint32**	Current performance level of the domain.

4.9.7. Service: **SET_PERF_LEVEL**

This service is used to set the current performance level of a specific performance domain in the system.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Performance domain ID
1	**LEVEL**	**uint32**	Performance level of the domain to set

Response data

Word	Name	Type	Description

0	**STATUS**	**int32**	Return Status code **Error Code** | **Description** RPMI_SUCCESS | The performance level is retrieved successfully RPMI_ERROR_NOT_FOUND | Can’t find the performance domain ID RPMI_ERROR_INVALID_PARAM | Invalid value in the input parameter. RPMI_ERROR_NOT_SUPPORTED | This function is not supported. RPMI_ERROR_DENIED | Client doesn’t have permission to perform this operation. RPMI_ERROR_HW | Operation failed due to hardware error ● Other errors in section 3.4 Return Status Code

4.9.8. Service: **GET_PERF_LIMIT**

This service is used to obtain the current performance limit of a specific performance domain in the system.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Performance domain ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code **Error Code** | **Description** RPMI_SUCCESS | The performance limit/range is retrieved successfully RPMI_ERR_NOT_FOUND | Can’t find the performance domain ID ● Other errors in section 3.4 Return Status Code
1	**MAX_PERF_LEVEL**	**uint32**	Maximum allowed performance level

2	**MIN_PERF_LEVEL**	**uint32**	Minimum allowed performance level

4.9.9. Service: **SET_PERF_LIMIT**

This service is used to set the current performance limit of a specific performance domain in the system.

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Performance domain ID
1	**MAX_PERF_LEVEL**	**uint32**	Maximum allowed performance level
2	**MIN_PERF_LEVEL**	**uint32**	Minimum allowed performance level

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code

Error Code	**Description**
RPMI_SUCCESS	The performance limit/range is set successfully
RPMI_ERROR_NOT_FOUND	Can't find the performance domain ID
RPMI_ERROR_INVALID_PARAM	Invalid value in the Max and Min parameters
RPMI_ERROR_NOT_SUPPORTED	Service not supported
RPMI_ERROR_DENIED	Client doesn't have permission to perform this operation
RPMI_ERROR_HW	Operation failed due to hardware error

- Other errors in section 3.4 Return Status Code

4.9.10. Service: **GET_PERF_DOMAIN_FAST_CHANNEL_ADDR**

This service allows clients to query attributes of the fast channel for the specific performance domain and the specific function

Request data

Word	Name	Type	Description
0	**DOMAIN_ID**	**uint32**	Performance Domain ID

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code <table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_SUCCESS</td><td>Service processed successfully</td></tr><tr><td>RPMI_ERR_NOT_FOUND</td><td>Can’t find the performance domain ID</td></tr><tr><td>RPMI_ERR_NOT_SUPPORTED</td><td>Operation not supported</td></tr></table>● Other errors in section 3.4 Return Status Code
1	**FLAGS**	**uint32**	<table><tr><th>Bits</th><th>Description</th></tr><tr><td>[31:3]</td><td>Reserved</td></tr><tr><td>[2:1]</td><td>Doorbell register width **0b00**: Doorbell register is 8 bits wide. **0b01**: Doorbell register is 16 bits wide. **0b10**: Doorbell register is 32 bits wide. **0b11**: Doorbell register is 64 bits wide.</td></tr><tr><td>[0]</td><td>**0b0**: Doorbell is Not Supported **0b1**: Doorbell is supported</td></tr></table>
2	**PHYS_ADDR_LOW**	**uint32**	Lower 32 bits of Physical address
3	**PHYS_ADDR_HIGH**	**uint32**	Higher 32 bits of Physical address
4	**DB_ADDR_LOW**	**uint32**	Lower 32 bits of Doorbell address
5	**DB_ADDR_HIGH**	**uint32**	Higher 32 bits if Doorbell address
6	**DB_ID_LOW**	**uint32**	Lower 32 bits of Doorbell ID

7	**DB_ID_HIGH**	**uint32**	Higher 32 bits if Doorbell ID
8	**DB_PRESERVED_LOW**	**uint32**	A lower 32 bits doorbell preserved mask to apply for this service before ring the doorbell. This field is unused if FLAGS[0] is zero.
9	**DB_PRESERVED_HIGH**	**uint32**	An upper 32 bit doorbell preserved mask to apply for this service before ring the doorbell. This field is only valid if the doorbell register width is 64 bits. This field is unused if FLAGS[0] is zero.

4.10 Service Group - **SECURE_MANAGEMENT_MODE** (servicegroup_id: 0x0000A)

This secure management mode service group is used for software invocation of Management Mode (MM) in a secure execution environment. Management Mode (MM) provides an environment for implementing OS agnostic services (MM services) like secure variable storage, and firmware updates in system firmware. The services can be invoked synchronously and asynchronously. This service group describes the interfaces for invoking MM services synchronously. For more information on Management Mode (MM), review the Platform Initialization (PI) specifications, Volume 4: Management Mode Core Interface.

Below table lists the services in this group:

Service ID	Service Name	Request type
0x01	ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	MM_VERSION	NORMAL_REQUEST
0x03	MM_COMMUNICATE	NORMAL_REQUEST
0x04	MM_COMPLETE	NORMAL_REQUEST
0x05	MM_INITIALIZE	NORMAL_REQUEST

4.10.1 Secure Management Mode Notifications

This service group does not support any event for notification

4.10.2 Service: **ENABLE_NOTIFICATION**

This service allows AP to subscribe to secure management mode service group notifications.

Platform can optionally support notifications of events which might occur in the platform. Management Mode firmware can send these notification messages to AP if they are implemented and AP has subscribed to these. Events supported are described above in Secure Management Mode Notifications.

Request data

Word	Name	Type	Description
0	**EVENT_ID**	**uint32**	Event to be subscribed for notification.

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	Description
RPMI_SUCCESS	when the notifications are subscribed successfully for EVENT_ID.
RPMI_ERROR_NOT_FOUND	Event in EVENT_ID is not supported or invalid
RPMI_ERROR_NOT_SUPPORTED	Notifications not supported in this service group

- Other errors in section 3.4 Return Status Code

4.10.3 Service: **MM_VERSION**

This service returns the version of a management mode service.

Request data

- NA

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code

Error Code	Description
RPMI_SUCCESS	MM communicate returned successfully

			RPMI_ERROR_DENIED: The requested operation was denied due to insufficient permissions. ● Other errors in section 3.4 Return Status Code
1	**MM_VERSION**	**uint32**	MM Version **Bits** / **Description** [31:16] Major Version [15:0] Minor Version

4.10.4 Service: **MM_COMMUNICATE**

Calling this **MM_COMMUNICATE** api invokes a MM service that is implemented in the secure execution environment. The **MM_COMM_BUFFER** contains data to identify and invoke the MM service. This synchronous call is returned by using **MM_COMPLETE.**

Request data

Word	Name	Type	Description
0:N	**MM_COMM_BUFFER**	**uint8**	MM data from non-secure to secure world

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** / **Description** RPMI_SUCCESS: MM communicate returned successfully RPMI_ERROR_DENIED: The requested operation was denied due to insufficient permissions. ● Other errors in section 3.4 Return Status Code

4.10.5 Service: **MM_COMPLETE**

Use this **MM_COMPLETE** as the "world-switch synchronous call" normally at the end of a synchronous **MM_COMMUNICATE** call to signal the readiness for handling the synchronous request. The **MM_COMM_BUFFER** contains the returned data of the MM service invoked.

Request data

Word	Name	Type	Description
0:N	**MM_COMM_BUFFER**	**uint8**	MM data from secure to non-secure world

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status Code **Error Code** / **Description** RPMI_SUCCESS: MM communicate returned successfully RPMI_ERROR_DENIED: The requested operation was denied due to insufficient permissions. ● Other errors in section 3.4 Return Status Code

4.10.6 Service: **MM_INITIALIZE**

This is an optional service. The MM modules may come in the firmware volume or FD files, loaded by the M-mode firmware like u-boot spl and initialized by the OpenSBI domain during the M-Mode firmware boot time. If so, this service api is not needed as default. But there is still case that the MM modules are requested to be loaded or initialized by the S-Mode firmware components, thus this service is used to launch the MM related modules as needed.

Request data

Word	Name	Type	Description
0	**HART_ID**	**uint8**	Hart id to launch
1	**DOMAIN_ID**	**uint8**	The secure domain id to be used to initialize the MM modules
2:3	**FLAGS**	**uint16**	

Bits	Description
[31:1]	Reserved
[0]	**0b0**: No payload information **0b1**: With payload information

4:5	**MM_PAYLOAD_BASE**	**uint64**	The base address of the MM payload loaded by the S-Mode firmware.
6:7	**MM_PAYLOAD_SIZE**	**uint64**	The MM payload size loaded by the S-Mode firmware.
8:263	**MM_PAYLOAD_SIGNATURE**	**uint8**	The MM payload signature loaded by the S-Mode firmware.

Response data

Word	Name	Type	Description
0	**STATUS**	**int32**	Return Status code

Error Code	Description
RPMI_SUCCESS	No error
PMI_ERROR_DENIED	The start operation is denied

- Other errors in section 3.4 Return Status Code

References

Reference	Description

[SBI]	RISC-V Supervisor Binary Interface Specification version 2.0 https://github.com/riscv-non-isa/riscv-sbi-doc/
[ACPI]	Advanced Control and Power Interface - version 6.4 https://uefi.org/htmlspecs/ACPI_Spec_6_4_html/index.html