RISC-V Platform Management Interface Specification (RPMI)

Version v1.0, 2025-07-16: Ratified

Table of Contents

Preamble

This document is in the Ratified state

No changes are allowed. Any necessary or desired modifications must be addressed through a follow-on extension. Ratified extensions are never revised.

Copyright and license information

Contributors

This RISC-V specification has been contributed to directly or indirectly by:

Anup Patel <apatel@ventanamicro.com>
Himanshu Chauhan <hchauhan@ventanamicro.com>
Joshua Yeong <joshua.yeong@starfivetech.com>
Ley Foon Tan <leyfoon.tan@starfivetech.com>
Rahul Pathak <rpathak@ventanamicro.com>
Samuel Holland <samuel.holland@sifive.com>
Subrahmanya Lingappa <slingappa@ventanamicro.com>
Yong Li <yong.li@intel.com>

Changelog

Version 1.0

- The RPMI specification version 1.0 with the foundations for the RPMI Message Protocol, RPMI Transport (shared memory based) and RPMI Service Groups for system control and management.

Terms and Abbreviations

Term	Meaning
A2P	Application Processor to Platform Microcontroller
ACPI	Advanced Configuration and Power Interface Specification
APEI	ACPI Platform Error Interfaces
AP	Application Processor
CPPC	Collaborative Processor Performance Control
GHES	Generic Hardware Error Source
MSI	Message Signaled Interrupt
P2A	Platform Microcontroller to Application Processor
PMA	Physical Memory Attributes
PMP	Physical Memory Protection
PuC	Platform Microcontroller
RAS	Reliability, Availability, and Serviceability
SBI	Supervisor Binary Interface

Chapter 1. Introduction

Today's platforms pose challenges in terms of manageability and controllability, where the operating system (OS) needs to support a variety of hardware with a variety of connected devices. The extra complexity and demand to manage and control the platform along with executing sophisticated workloads is a challenge for the application processors (APs) running a general purpose OS. To address this challenge, platforms today contain one or more microcontrollers which can offload various platform management and control tasks.

This document describes the RISC-V Platform Management Interface (RPMI), which is an OS-agnostic, firmware-agnostic, scalable and extensible interface for platform management and control from dedicated microcontrollers (also referred to as platform microcontroller or PuC).

The RPMI defines a message based communication between multiple application processors (APs) and platform microcontrollers (PuCs) for system management and control. This message based communication can also be virtualized by the machine-mode firmware or hypervisors using the SBI MPXY extension [1].

All RPMI capabilities and services provided by platform microcontroller are discoverable at runtime which allows adding new capabilities and services in the future.

1.1. RPMI Abstractions

RPMI Transport: An RPMI transport represents the mechanism by which the messages are exchanged between the application processor and the platform microcontroller. An RPMI transport instance is associated to a particular RISC-V privilege level of the application processors and it must be accessed only by that RISC-V privilege level.

RPMI Messaging Protocol: The RPMI messaging protocol defines different types of RPMI messages and the format for each RPMI message type.

RPMI Service Groups: The services provided by the platform microcontrollers to the application processors are grouped into RPMI service groups based on functionality. Each RPMI service group specifies the RISC-V privilege levels from which the application processor can access it. Platform vendors can implement custom RPMI service groups.

RPMI Client: An RPMI client is a software or a driver running on the application processor which is capable of sending and receiving RPMI messages.

RPMI Context: An RPMI context consists of an RPMI transport instance, RPMI message protocol layer, a mandatory RPMI BASE service group and other optional RPMI service groups. An RPMI context is associated with a RISC-V privilege level which matches the included RPMI transport instance. The RPMI service groups included in an RPMI context must be accessible from the RISC-V privilege level associated with that RPMI context.

The RPMI is designed to work with a single or multi-tenant topology as shown in the Figure 1 below whereas the high-level architecture is shown in the Figure 2 below.

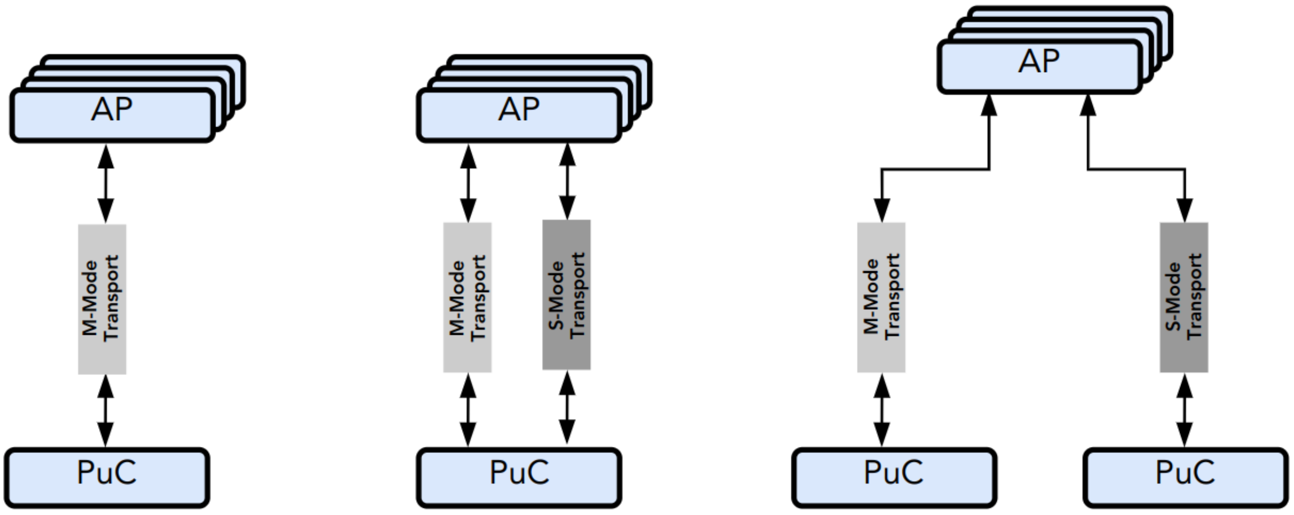

Figure 1. Transport for M-Mode and S-Mode

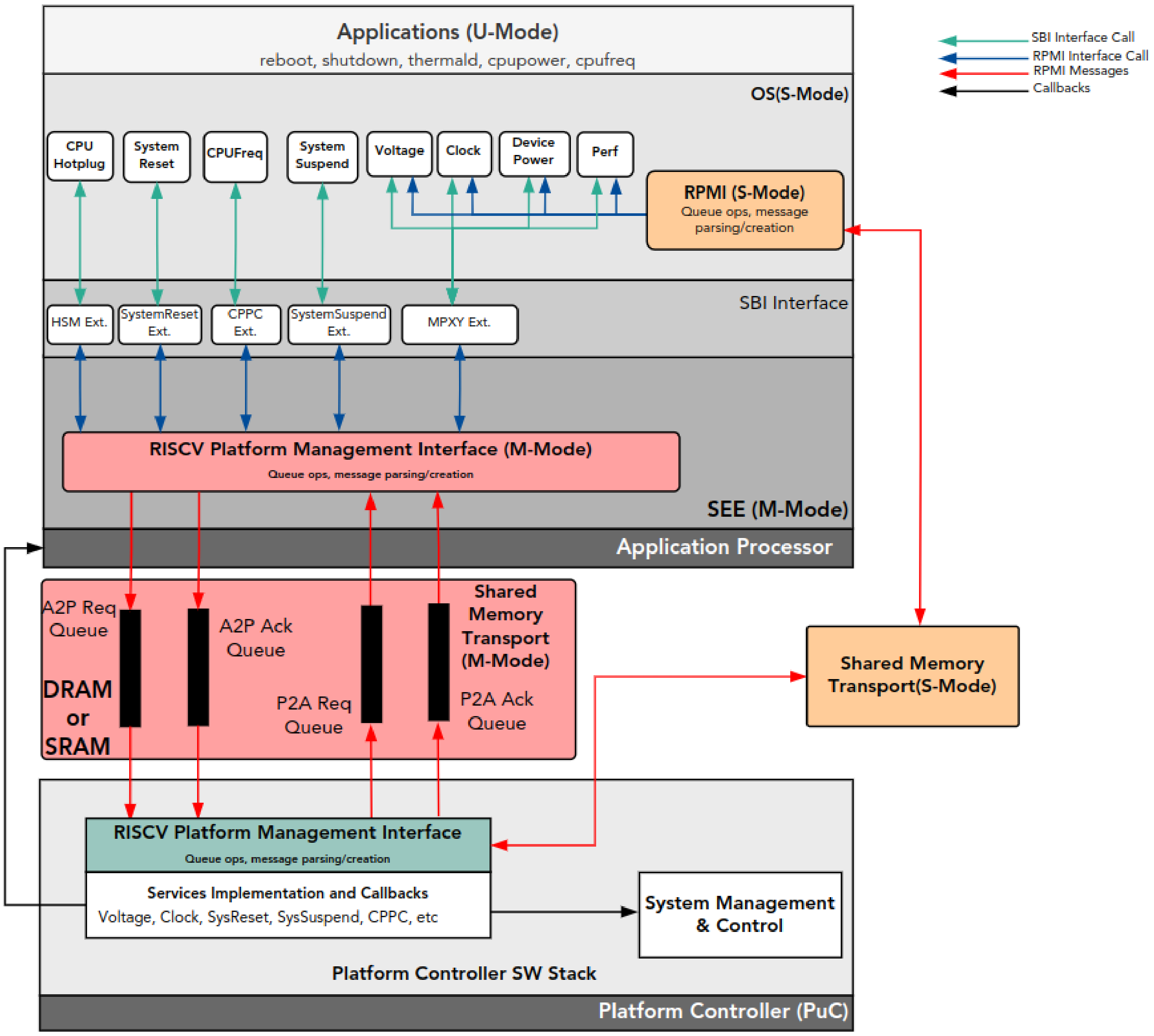
SBI Interface Call
RPMI Interface Call
RPMI Messages
Callbacks
Applications (U-Mode)
reboot, shutdown, thermald, cpupower, cpufreq
OS(S-Mode)
CPU Hotplug
System Reset
CPUFreq
System Suspend
Voltage
Clock
Device Power
Perf
RPMI (S-Mode)
Queue ops, message parsing/creation
SBI Interface
HSM Ext.
SystemReset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

Figure 2. High Level Architecture

Chapter 2. Transport

An RPMI transport is an abstraction over a physical medium used to send and receive messages between the application processors (APs) and the platform microcontroller (PuC). It provides bi-directional communication between a RISC-V privilege-level of application processors and a platform microcontroller. The application processors can have multiple RPMI transport instances with a platform microcontroller. Also, a platform can also have multiple microcontrollers each with its own RPMI transport instance as shown in the Figure 1 below.

An RPMI transport instance consists of two logical bi-directional channels for message delivery as shown in the Figure 3 below. Each channel is capable of transferring messages in request-response pairs. A channel which transfers a request message from the application processors (APs) to the platform microcontroller (PuC) and response/acknowledgement back in opposite direction is called an A2P channel. Similarly, the channel for request messages from the platform microcontroller (PuC) to the application processors (APs) is called a P2A channel. The P2A channel also transfers notification messages to the application processors.

An RPMI transport instance must implement the A2P channel but the P2A channel is optional. Platforms which do not require requests and notification messages from the platform microcontroller can avoid implementing the P2A channel.

The current RPMI specification only defines a shared memory based transport but other transport types can be added in the future.

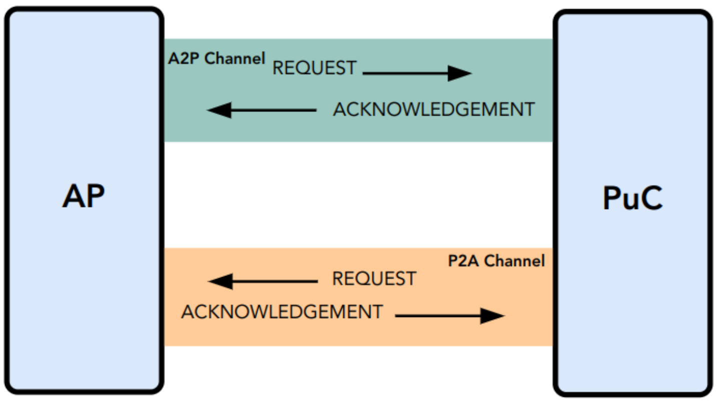

Figure 3. Bi-directional Communication

2.1. Shared Memory Transport

The RPMI shared memory transport defines a mechanism to exchange messages via shared memory which can be on-chip SRAM or a reserved portion of DRAM or some device memory. The RPMI shared memory transport does not specify where the shared memory resides in a platform, but it must be accessible from both the application processors and the platform microcontroller.

At a minimum, the platform must configure the physical memory attribute of the RPMI transport shared memory as an I/O type with read/write access that is coherent and non-cacheable for both the application processor and the platform microcontroller. The mechanism used by the platform to implement such memory attributes is implementation-defined.

Such physical memory attributes are required to avoid the caching side-effects because it is possible that the application processor and the platform microcontroller are not cache-coherent, and using the shared memory may lead to caching side-effects such as data

> *inconsistency between the platform microcontroller and the application processor, write propagation delays and other issues that can lead to race conditions.*

All data sent or received through the RPMI shared memory transport must follow little-endian byte-order.

The Figure 4 below shows the high-level architecture of the RPMI shared memory transport. The layout and attributes of a RPMI shared memory transport may be static for the platform microcontroller but must be discoverable by the application processors through hardware description mechanisms such as device tree or ACPI.

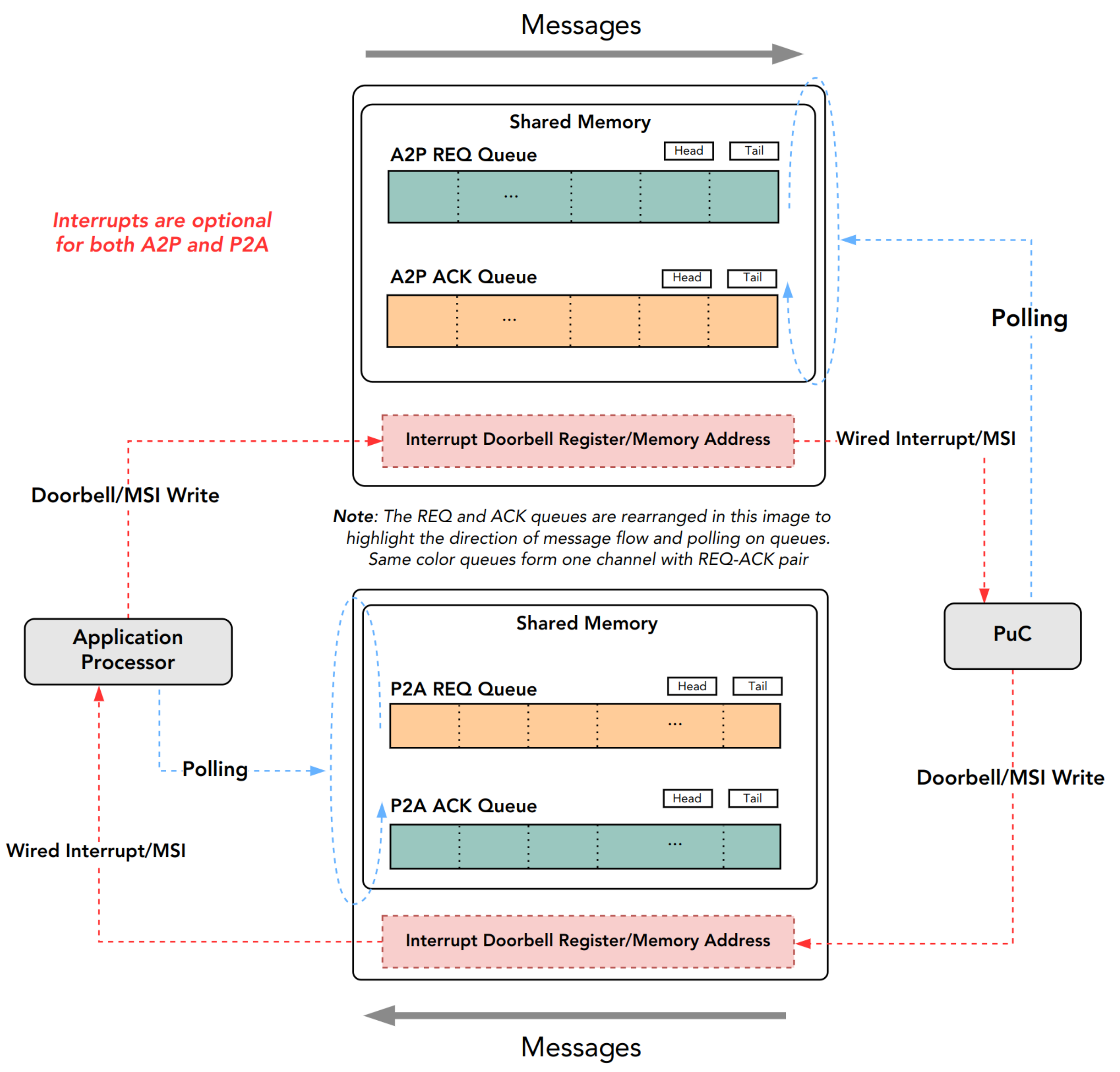
Messages
Shared Memory
A2P REQ Queue
Head
Tail
...
A2P ACK Queue
Head
Tail
...
Interrupts are optional for both A2P and P2A
Polling
Interrupt Doorbell Register/Memory Address
Wired Interrupt/MSI
Doorbell/MSI Write
Note: The REQ and ACK queues are rearranged in this image to highlight the direction of message flow and polling on queues. Same color queues form one channel with REQ-ACK pair
Application Processor
PuC
Shared Memory
P2A REQ Queue
Head
Tail
...
Polling
P2A ACK Queue
Head
Tail
...
Doorbell/MSI Write
Wired Interrupt/MSI
Interrupt Doorbell Register/Memory Address
Messages

Figure 4. Shared Memory Transport Architecture

2.1.1. Queue Types

The RPMI shared memory transport consists of four unidirectional queues. The type of messages and the direction of message delivery is fixed for each RPMI shared memory transport queue. The Table 1 below provides a more detailed description of all RPMI shared memory transport queues.

Table 1. Shared Memory Transport Queues

Name	Message Type	Description
A2P REQ	REQUEST	The request message queue from the application processor to the platform microcontroller.
P2A ACK	ACKNOWLEDGEMENT	The acknowledgement message queue from the platform microcontroller to the application processor.
P2A REQ	REQUEST & NOTIFICATION	The request message queue from the platform microcontroller to the application processor. This queue is also used for sending the notification messages.
A2P ACK	ACKNOWLEDGEMENT	The acknowledgement message queue from the application processor to the platform microcontroller.

The A2P REQ queue is paired with P2A ACK queue to form the A2P channel of the RPMI shared memory transport. Similarly, the P2A REQ queue is paired with the A2P ACK queue to form the P2A channel of the RPMI shared memory transport. The Figure 5 below shows the high-level flow of messages in a RPMI shared memory transport.

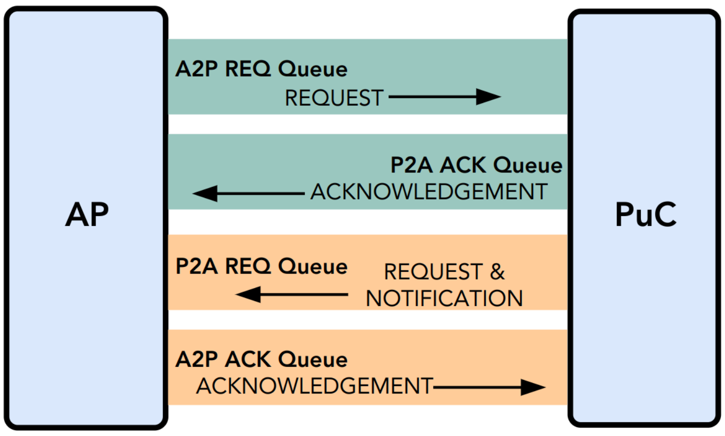

Figure 5. Shared Memory Transport Message Flow

2.1.2. Queue Layout

An RPMI shared memory queue is divided into `M` contiguous slots of equal size which are used to form a circular queue. The size of each slot (or slot size) must be a `power-of-2` and must be at least `64 bytes`. The slot size is same across all RPMI shared memory queues and the physical address of each slot must be aligned at slot size boundary.

The slot size should match with the maximum cache block size used in a platform. The requirement of `power-of-2` *slot size with minimum value of* `64 bytes` *is because usual CPU cache block size is* `64 bytes` *or some* `power-of-2` *value.*

The slots of the RPMI shared memory queue are assigned with sequentially increasing indices starting from `0`. The slot at index `0` is referred to as the **`head`** slot and the slot at index `1` is referred to as the **`tail`** slot. The remaining `(M - 2)` slots of the RPMI shared memory queue are message slots. The first `4 bytes` of the **`head`** slot is used as the head of the circular queue which contains a `(slot index - 2)` value pointing to the message slot from where the next message is dequeued. The first `4 bytes` of the **`tail`** slot is used as the tail of the circular queue which contains a `(slot index - 2)` value pointing to the message slot from where the next message is enqueued. The pictorial view of the RPMI shared memory queue internals is shown in the Figure 6 below.

In the total `M` *slots only the* `(M - 2)` *slots are used as an queue having RPMI messages stored* *as data. The* `(slot index - 2)` *index value represents that from all slots perspective in a queue shared memory which also includes the* `head` *and* `tail` *slots, the* `head` *and* `tail` *stores the indices of the message slots which effectively starts from* `slot index - 2`*.*

The requirement of keeping `head` *and* `tail` *in separate slots is to prevent both* `head` *and* `tail` *using the same cache block so that cache maintenance such as using cache flush and invalidate operations can be done separately for both* `head` *and* `tail`*.*

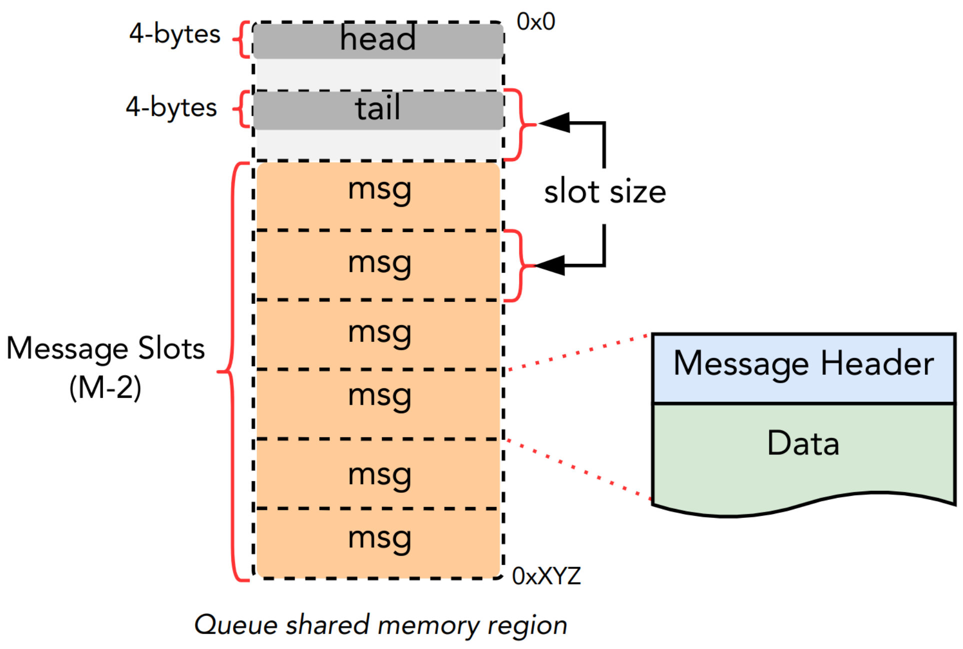

Figure 6. Shared Memory Queue Internals

A message consumer dequeues pending message from the message slot pointed by the `head` of the RPMI shared memory queue whereas a message producer enqueues new message at the message slot pointed by the `tail` of the RPMI shared memory queue. If there are no messages in the RPMI shared memory queue then message consumer must wait for messages to be available. If all message slots in the RPMI shared memory queue are occupied then message producer must wait for messages to be consumed. The ownership of `head` and `tail` is mutually exclusive where only the message consumer should update the `head` and only the message producer should update `tail` of the RPMI shared memory queue.

For example, only application processors should enqueue new messages and update `head` *of the A2P REQ queue whereas only platform microcontroller should dequeue messages and update* `tail` *of the A2P REQ queue.*

2.1.3. Queue Placement

The RPMI shared memory transport divides the underlying shared memory region into two parts where one part belongs to the A2P channel and other belongs to the P2A channel. The shared memory region sizes of the A2P and P2A channel can be different. For each channel (A2P or P2A), the corresponding REQ and ACK queues must be of the same size hence equal number of slots (or queue capacity). The size of each RPMI shared queue must be a multiple of the slot size.

A platform should provide sufficient shared memory for all RPMI shared memory queues so that the number of slots (queue capacity) does not become a bottleneck in message communication. It is recommended that the number of slots in queues belonging to A2P channel should be proportional to the number of application processors accessing the A2P channel.

The RPMI shared memory queues can be placed anywhere in the underlying shared memory region but there must be no overlap among the queues. The Figure 7 below shows a recommended way of placing RPMI queues in shared memory.

A platform may allocate separate non-contiguous shared memory regions for queues which may require platform to configure and manage memory attributes separately for each region. Instead, the platform can allocate contiguous regions for all four queues. For example, the platform may allocate `4096 bytes` *of shared memory for all four queues and memory attributes can be configured once only for single contiguous region.*

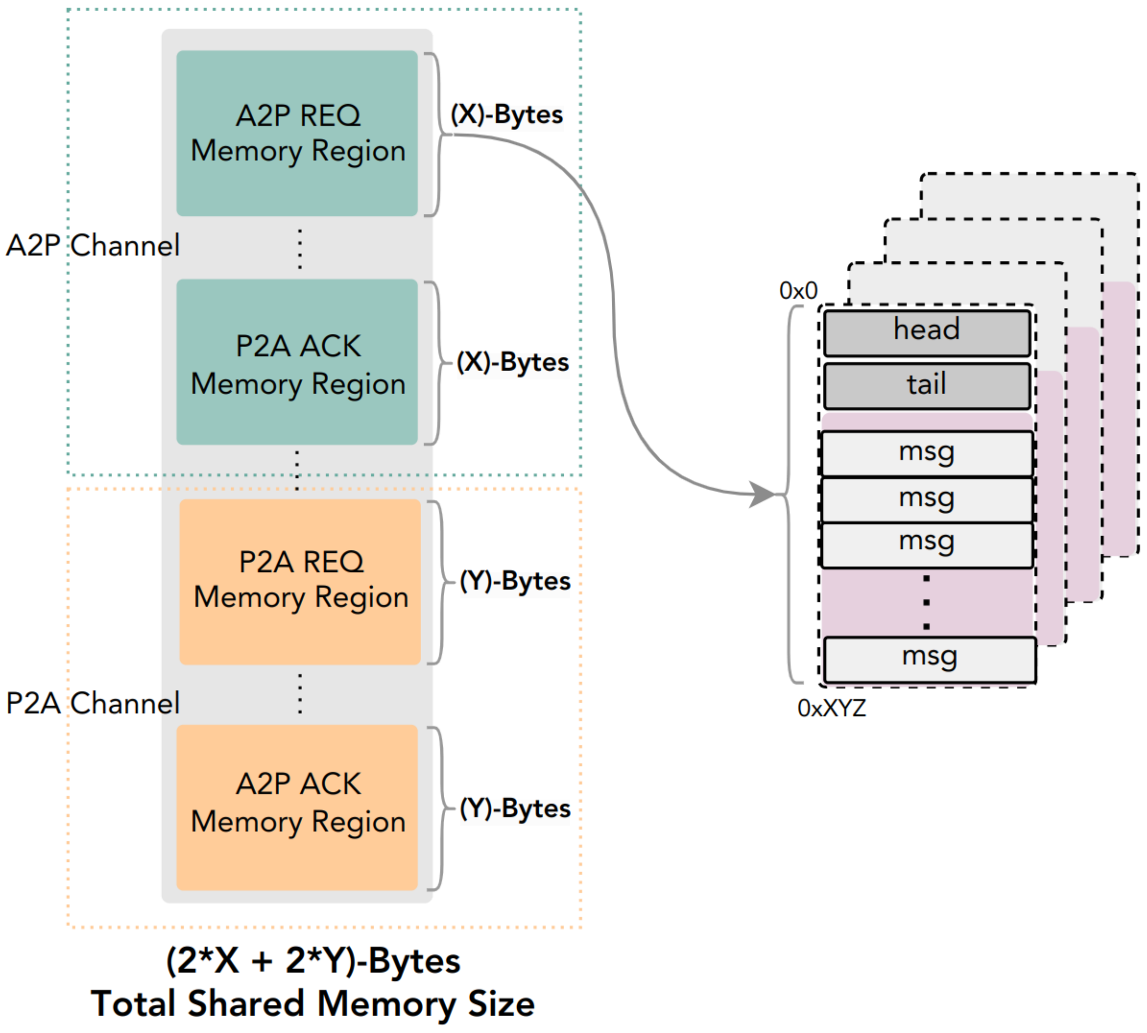

Figure 7. Recommended Placement of Queues in Shared Memory

2.1.4. Queue Implementation in Software

Queue Discovery

The shared memory for the queues including the `head` and `tail` slots is initialized by the platform microcontroller and the details of the shared memory queues are provided to the application processors.

The physical base address and size of each RPMI shared memory queue may be fixed for the platform microcontroller but the application processors must discover it through hardware description mechanisms such as device tree or ACPI.

The slot size of the RPMI shared memory queues may be fixed for the platform microcontroller but the application processors must discover it through hardware description mechanisms such as device tree or ACPI.

The total number of slots in each RPMI shared memory queue can be easily calculated by dividing the queue size by the slot size.

```
Example calculation

X bytes : Queue shared memory size.
M = (X / slot-size) : Total slot count in a queue
(M-2) : Message slot count (2 slots less for `head` and `tail`)
```

Queue Operation

In a queue, the `head` is used to dequeue the message and the `tail` is used to enqueue the message.

In an implementation, a queue is empty if the `head` == `tail` and a queue is full if `((tail + 1) % (M - 2)) == head`.

The queues and queue states are shared between application processors, and due to mechanisms such as kexec and others that can spawn another OS/firmware from the currently running OS/firmware, notifications or response messages may be delivered that are not intended for the newly spawned OS/firmware, and such messages may be ignored.

2.1.5. Doorbell Interrupt

An RPMI shared memory transport may also provide optional doorbell interrupts for application processors and/or the platform microcontroller to signal the arrival of new messages. This doorbell interrupt can be either a message-signaled interrupt (MSI) or a wired interrupt. The RPMI implementations may ignore the doorbell mechanism of RPMI shared memory transport and always use a polling mechanism to check the arrival of new messages.

A2P Doorbell

The A2P doorbell is a signal for new messages from the application processors (APs) to the platform microcontroller (PuC).

The platform must support A2P doorbell interrupt triggering from application processors through 32-bit memory-mapped register with write access, which can be discovered by the application processors using hardware description mechanisms such as device tree or ACPI.

P2A Doorbell

The P2A doorbell is a signal for new messages from the platform microcontroller (PuC) to the application processors (APs).

If the P2A doorbell is a wired interrupt then the platform must provide a way to the platform microcontroller to trigger the interrupt and application processors must discover it using standard hardware description mechanisms such as device tree or ACPI.

If the P2A doorbell is a MSI then the application processors must configure the P2A doorbell MSI on the platform microcontroller side using RPMI services defined by the `SYSTEM_MSI` service group.

> *If the platform supports PLIC, the platform need to provide a MMIO register to inject an edge-triggered interrupt.*

> *The doorbell attribute contains a doorbell write value which must be written to the doorbell memory mapped register to trigger the interrupt. The write value may also contains other set bits which must persist on every write to the doorbell register.*

2.1.6. Integration with RPMI Message Protocol

If the doorbell interrupts are supported and enabled, the shared memory transport uses `FLAGS[3]` bit in the message header of RPMI normal request as a **doorbell interrupt request** flag. This flag represents if the doorbell interrupt is requested to notify about the response of a request message.

The sender of the RPMI normal request type message can set the `FLAGS[3]` bit to `1` to inform the service provider to ring the doorbell after sending the response message back. If the `FLAGS[3]` bit is `0`, it means that the sender is going to poll for the response message in the queue and the service provider does not need to ring the doorbell.

If the sender of an RPMI normal request message sets the `FLAGS[3]` bit to `1` without supporting or enabling the doorbell interrupt, the behavior is undefined.

> *The `FLAGS[3]` bit can be used for a particular RPMI normal request message or for the entire life-cycle of RPMI message communication. For example, if the P2A doorbell is MSI and the application processor has configured MSI target details via `SYSTEM_MSI` service group, then `FLAGS[3]` bit can always be set to `1` so that the platform microcontroller will always send the MSI for every response. The application processor can also selectively disable it for a request message so that the platform microcontroller does not trigger the doorbell for the response message.*

2.2. Shared Memory based Fast-channels

A fast-channel is a unidirectional shared memory channel with a dedicated RPMI service type. The data transmitted over a fast-channel is without any message header and its layout is defined by the service which is dedicated to that fast-channel. Unlike normal RPMI transport, which can be shared by multiple service groups and services, a fast-channel is exclusive to a service in a service group which allows faster exchange of the data. A fast-channel can be used in scenarios that require lower latency and faster processing of requests between the application processors and the platform microcontroller.

> *Because of fixed data format and type associated with a fast-channel, the requests made over a fast-channel can be processed quickly, but the time required by the platform microcontroller to complete the requests may not be less than the time required for completion of requests made over the normal RPMI transport The request completion time depends on the platform implementation.*

A service group that supports fast-channels for services:

- May only enable some services to be used over fast-channels.
- Must provide physical address and other attributes (such as optional fast-channel doorbell) of the fast-channels via a services defined by the service group.

The layout and data format of a fast-channel are RPMI service specific in a service group and defined in the respective service group sections.

At a minimum, the platform must configure the physical memory attribute of the fast-channels shared memory as an I/O type with read/write access that is coherent and non-cacheable for both the application processor and the platform microcontroller. The mechanism used by the platform to implement such memory attributes is implementation-defined.

Such physical memory attributes are required to avoid the caching side-effects because it is possible that the application processor and the platform microcontroller are not cache-coherent, and using the shared memory may lead to caching side-effects such as data inconsistency between the platform microcontroller and the application processor, write propagation delays and other issues that can lead to race conditions.

Chapter 3. Messaging Protocol

The RPMI messaging protocol includes all the RPMI messages exchanged over a RPMI transport channel.

3.1. Message Types

The RPMI messaging protocol defines three types of RPMI messages namely: **REQUEST**, **ACKNOWLEDGEMENT** and **NOTIFICATION**. The Table 2 below summarize all RPMI message types.

An **RPMI request message** represents a specific control and management task which needs to be performed and it is also referred to as an **RPMI service**. Multiple related RPMI services are grouped logically into an **RPMI service group** such as Clock, Voltage, Performance, etc. Depending on the RPMI service, a RPMI request message may carry data required to perform the control and management task. An RPMI request message may have an associated response which is sent back as an **RPMI acknowledgement message** on the same RPMI transport channel. The RPMI acknowledgement message carries the status and optional response data from an RPMI request after it has been processed.

An RPMI request message which has an associated RPMI acknowledgement message is referred to as a **NORMAL REQUEST** otherwise it is referred to as a **POSTED REQUEST**.

An **RPMI notification message** is an asynchronous message from the platform microcontroller to the application processors which is used to inform the later about certain events that have occurred in the system. There is no response required for an RPMI notification message from the application processors.

Table 2. RPMI Message Types

Message Type	Message Subtypes	Description
REQUEST	NORMAL REQUEST Request with Acknowledgement. POSTED REQUEST Request without Acknowledgement.	Messages for requesting a service from the platform microcontroller.
ACKNOWLEDGEMENT	*Not applicable*	Response message corresponding to a NORMAL REQUEST message.
NOTIFICATION	*Not applicable*	Asynchronous messages from the platform microcontroller representing system events.

3.2. Message Format

An RPMI message consists of a fixed `8-byte` message header followed by a variable sized optional message data as show in the Figure 8 below. The byte ordering of an RPMI message is defined by the underlying RPMI transport.

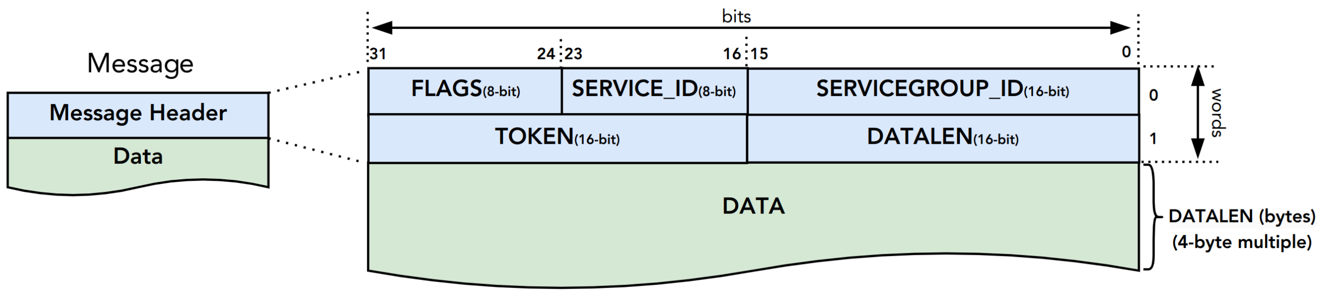

Figure 8. RPMI Message Format

3.2.1. Message Layout Tables

The RPMI message header and message data are split into multiple **words**, where each word is `4-byte` wide and indexed starting from `0`. The RPMI message layout is presented throughout the RPMI specification in the form of tables as shown in the Table 3 below. Some of the columns listed below may be omitted in the layout tables if not required.

Table 3. Message Layout Table Example

Word	Name	Type	Description
Index of the 4-byte word at which the field starts in the message header or message data.	Name of the field. Name may be omitted if not required.	Type of field, eg: int32 or uint32, etc.	Description and interpretation of the field.

3.2.2. Message Header

The layout of the `8-byte` wide RPMI message header is shown in the Table 4 below. The RPMI message header provide a unique identity to the corresponding RPMI message withing an RPMI context.

Table 4. RPMI Message Header

Word	Description		
	Bits	Name	Description

0	[31:24]	FLAGS	Message flags.
			`FLAGS[7:4]: Reserved and must be 0.`
			`FLAGS[3]: Reserved for RPMI transport.`
			`Refer the corresponding RPMI transport chapter for more details.`
			`FLAGS[2:0]: Message Type.`
			`0b000: NORMAL_REQUEST.`
			`0b001: POSTED_REQUEST.`
			`0b010: ACKNOWLEDGEMENT.`
			`0b011: NOTIFICATION.`
			`0b100 - 0b111: Reserved for future use.`

	[23:16]	SERVICE_ID	Service ID. 8-bit wide identifier representing a RPMI service. This identifier is unique within a given RPMI service group.
	[15:0]	SERVICEGROUP_ID	Service group ID. 16-bit wide unique identifier representing a RPMI service group.

Word	Description		
1	Bits	Name	Description
	[31:16]	TOKEN	Message token. 16-bit number for a RPMI message.
	[15:0]	DATALEN	Message data length. Stores the size of the message data in bytes. The value stored in this field must be a multiple of `4-byte` or `0` if no message data is present.

For an RPMI normal request message, the `TOKEN`, `SERVICEGROUP_ID`, and `SERVICE_ID` fields of the RPMI acknowledgement message must have the same values as corresponding fields in the RPMI request message. The `DATALEN` field of the RPMI acknowledgement message must be set according to the data carried by this acknowledgement.

The message token will help the application processors to keep track of the origin of the request when it receives a response. This is useful when the multiple application processors are sharing the same queues. For example, two different application processors may send the same type of request message with the same SERVICEGROUP_ID and SERVICE_ID. When the response messages for both requests are received from the platform microcontroller, the token helps distinguish which response belongs to which request. For other message types such as RPMI posted request and RPMI notification messages, the implementations may use the token for debugging or logging purposes.

The RPMI specification recommends monotonically increasing token numbers and the token number can be initialized from any value without any constraints.

For an RPMI notification message, the platform microcontroller will set appropriate values for the `TOKEN`, `SERVICEGROUP_ID`, and `DATALEN` fields whereas the `SERVICE_ID` field must be always set to `0x0`.

3.2.3. Message Data

The message data of an RPMI message is optional and variable sized. The maximum message data size of an RPMI message depends on the underlying RPMI transport implementation.

The message data carries different information based on the RPMI message type:

- An RPMI request message carries data required to perform the control and management task.
- An RPMI acknowledgement message carries the status and optional response data.
- An RPMI notification message carries an array of RPMI events.

The message data format for RPMI request message and RPMI acknowledgement message is defined separately for each RPMI service. The message data format for RPMI notification message is defined in the Section 3.3.

An RPMI acknowledgement message must have a signed `STATUS` field as the first 4-byte word of the message data containing an error code defined in the Section 3.4. An RPMI service where the response data exceeds the maximum message data size can use multipart RPMI acknowledgement messages.

If a physical address is passed in the message data of any message type, then it refers to the physical address space of the application processor.

3.3. Notifications

The platform microcontroller can use RPMI notification message to notify application processors about system events which are also referred to as **RPMI events**. An RPMI notification message has no associated response or acknowledgement from application processors. Multiple RPMI events can be packed into a single RPMI notification message depending on the space available in the message data. Each RPMI event may have additional data associated with it based on the type of RPMI event. Any action required for handling an RPMI event depends on the application processors. The format of an RPMI notification message in shown in the Figure 9 below.

The RPMI events are defined separately for each RPMI service group. An RPMI service group must have a `ENABLE_NOTIFICATION` service with a fixed `SERVICE_ID=0x01` which can be used by the application processors to enable or disable notification messages for a particular RPMI event defined by the RPMI service groups. By default, notifications are disabled for all RPMI events of an RPMI service group. The platform microcontroller only sends RPMI notification messages for RPMI events which are enabled by the application processors. If multiple RPMI events are supported by an RPMI service group then the application processors must enable to each RPMI event individually.

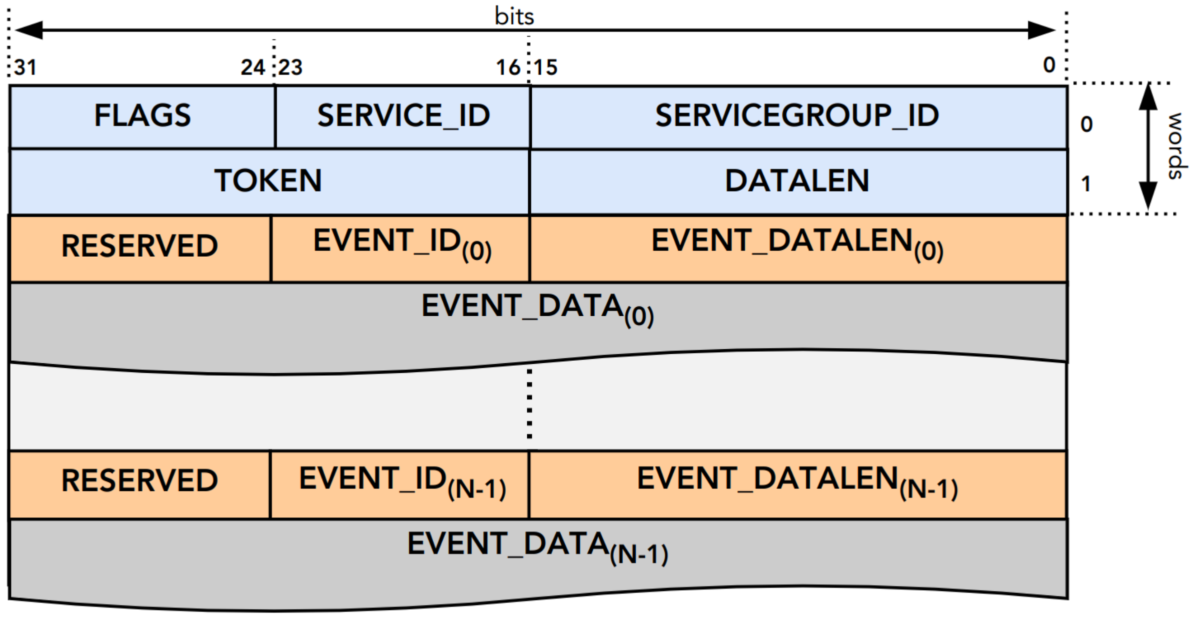

Figure 9. RPMI Notification Message Format

3.3.1. Events

An RPMI event consists of a header containing two fields: `EVENT_ID (8-bit)` and `EVENT_DATALEN (16-bit)`. An RPMI event may have associated data whose size is specified in the `EVENT_DATALEN` field of the header and this data size must be a multiple of `4-byte`.

The number of RPMI events that can be stored in a single RPMI notification message depends on the maximum RPMI message data size. The `DATALEN` field in the RPMI message header represents the aggregate size of all RPMI events included in RPMI message data.

The Table 5 below defines the format of an RPMI event whereas the Figure 10 below shows a pictorial view of an RPMI event. The format of the event data for each RPMI event is defined separately by an RPMI service group. If multiple RPMI events are packed into a single RPMI notification message then the ordering of RPMI events within the RPMI notification message is implementation defined.

The platform microcontroller is not required to include every occurrence of an event of the same type in a notification message. Instead, the platform microcontroller must only include the most recently occurred event of the same type.

Table 5. Event Format

Word	Name	Description
0	EVENT_HDR	32-bit field represents a single event. **Bits** | **Name** | **Description** [31:24] | *Reserved* | *Reserved* and must be `0`. [23:16] | EVENT_ID | Unique identifier for an event in a service group. [15:0] | EVENT_DATALEN | 16-bit field to store event data size in bytes.
1	EVENT_DATA	Event data whose size is specified by `EVENT_DATALEN`.

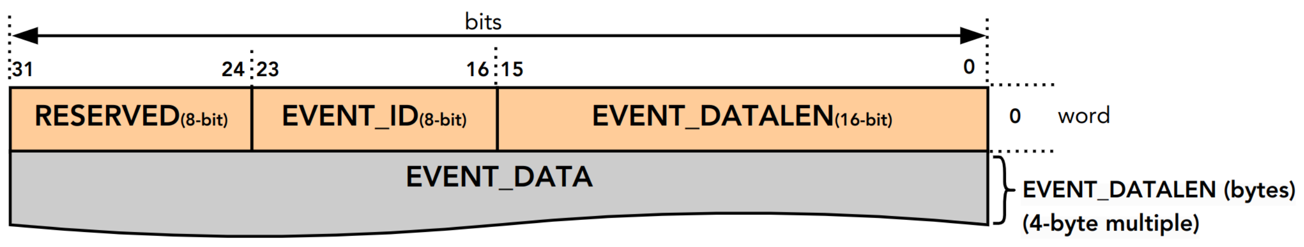

Figure 10. Event Header

3.4. Possible Error Codes

The Table 6 below lists the error codes which can be returned by an RPMI service in the `STATUS` field of the RPMI acknowledgement message.

Table 6. RPMI Error Codes

Name	Error Code	Description
RPMI_SUCCESS	0	Service has been completed successfully.
RPMI_ERR_FAILED	-1	Failed due to general error.
RPMI_ERR_NOT_SUPPORTED	-2	Service or feature is not supported.
RPMI_ERR_INVALID_PARAM	-3	One or more parameters passed are invalid.
RPMI_ERR_DENIED	-4	Requested operation denied due to insufficient permissions or failed dependency check.
RPMI_ERR_INVALID_ADDR	-5	One or more addresses are invalid.
RPMI_ERR_ALREADY	-6	Operation already in progress or state changed already for which the operation was performed.
RPMI_ERR_EXTENSION	-7	Error in extension implementation that violates the extension specification or the extension version mismatch.
RPMI_ERR_HW_FAULT	-8	Failed due to hardware fault.
RPMI_ERR_BUSY	-9	Service cannot be completed due to system or device is busy.
RPMI_ERR_INVALID_STATE	-10	Invalid state.
RPMI_ERR_BAD_RANGE	-11	Bad or invalid range.
RPMI_ERR_TIMEOUT	-12	Failed due to timeout.
RPMI_ERR_IO	-13	Input/Output error.
RPMI_ERR_NO_DATA	-14	Data not available.

Name	Error Code	Description
	-15 to -127	*Reserved.*
	< -127	*Vendor or Implementation specific.*

Chapter 4. Service Groups

An RPMI service group is a collection of RPMI services that are logically grouped based on functionality. For example, all the voltage related services are grouped into a voltage service group. The functionality implemented by certain RPMI service groups may impact the architectural state of application processors due to this each RPMI service group specifies the RISC-V privilege levels of the application processor which can be access it. For example, the clock service groups can be accessed from M-mode and S-mode but the HSM service group can be only accessed from M-mode.

All RPMI service groups except the BASE service group are optional. If the `BASE_PROBE_SERVICE_GROUP` service indicates that a service group is implemented then the RPMI service group version must conform to the RPMI specification version returned by the `BASE_GET_SPEC_VERSION` service. All implemented RPMI service groups must satisfy the following requirements:

1. The RPMI service group must be accessible from the RISC-V privilege level associated with the RPMI context which includes it.
2. All RPMI services of the RPMI service groups must be supported except the dedicated notification service (`SERVICE_ID = 0x00`) which is reserved for RPMI notification messages. A RPMI service group may implement its RPMI services partially only if it also defines a mechanism to discover supported RPMI services.
3. The RPMI service group must implement a dedicated RPMI service with `SERVICE_ID = 0x01` to subscribe for event notifications.

The RPMI services listed within each RPMI service group do not have a specific order. Additionally, the sequence in which services are defined in the specification does not necessarily reflect the order in which they should be invoked.

This specification defines standard RPMI service groups and RPMI services with the provision to add more service groups as required in the future. The RPMI specification also provides experimental service group IDs space for development of service group until a standard service group ID is allocated. The platform vendors can provide implementation specific RPMI service groups. The Table 7 table below lists all standard RPMI service groups defined by this specification.

Table 7. RPMI Service Groups

Service Group ID	RPMI Version (Major:Minor)	Service Group Name	Allowed RISC-V Privilege Levels on Application Processors
0x0001	1.0	BASE	M-mode, S-mode
0x0002	1.0	SYSTEM_MSI	M-mode, S-mode
0x0003	1.0	SYSTEM_RESET	M-mode
0x0004	1.0	SYSTEM_SUSPEND	M-mode
0x0005	1.0	HART_STATE_MANAGEMENT	M-mode
0x0006	1.0	CPPC	M-mode, S-mode
0x0007	1.0	VOLTAGE	M-mode, S-mode
0x0008	1.0	CLOCK	M-mode, S-mode
0x0009	1.0	DEVICE_POWER	M-mode, S-mode
0x000A	1.0	PERFORMANCE	M-mode, S-mode
0x000B	1.0	MANAGEMENT_MODE	M-mode, S-mode
0x000C	1.0	RAS_AGENT	M-mode, S-mode

Service Group ID	RPMI Version (Major:Minor)	Service Group Name	Allowed RISC-V Privilege Levels on Application Processors
0x000D	1.0	REQUEST_FORWARD	M-mode, S-mode
0x000E - 0x7BFF		*Reserved for Future Use*	
0x7C00 - 0x7FFF		*Experimental Service Groups*	
0x8000 - 0xFFFF		*Implementation Specific Service Groups*	

4.1. Service Group: BASE (SERVICEGROUP_ID: 0x0001)

The BASE service group is mandatory and provides the following services:

- Initial handshaking between the application processor and the platform microcontroller.
- Discovering the RPMI implementation version information.
- Discovering the implementation of a service group.
- Discovering platform specific information.

The following table lists the services in the BASE service group:

Table 8. BASE Services

Service ID	Service Name	Request Type
0x01	BASE_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	BASE_GET_IMPLEMENTATION_VERSION	NORMAL_REQUEST
0x03	BASE_GET_IMPLEMENTATION_ID	NORMAL_REQUEST
0x04	BASE_GET_SPEC_VERSION	NORMAL_REQUEST
0x05	BASE_GET_PLATFORM_INFO	NORMAL_REQUEST
0x06	BASE_PROBE_SERVICE_GROUP	NORMAL_REQUEST
0x07	BASE_GET_ATTRIBUTES	NORMAL_REQUEST

4.1.1. RPMI Implementation IDs

The RPMI specification defines space for standard implementation IDs and for experimental implementation IDs. The experimental implementation IDs can be used by the implementations until a standard implementation ID is assigned to it.

The RPMI implementations that have been assigned a standard implementation ID are listed in the table below.

Table 9. RPMI Implementation IDs

Implementation ID	Name
0x00000000	libRPMI [2]
0x00000001 - 0x7FFFFFFF	*Reserved for Future Use*
0x80000000 - 0xFFFFFFFF	*Experimental Implementation IDs*

4.1.2. Notifications

This service is used by the platform microcontroller to send the asynchronous message of type notification to the application processor. The message transfers the events defined by this service group. The events defined are listed in the below table.

Table 10. BASE Service Group Events

Event ID	Event Name	Event Data	Description
0x01	REQUEST_HANDLE_ERROR	NA	This event indicates that the platform microcontroller is unable to serve the message requests and acknowledgements are not guaranteed.

4.1.3. Service: BASE_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `BASE` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.1.2.

Table 11. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 12. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. **Error Code** — **Description** RPMI_SUCCESS — Event is subscribed successfully. RPMI_ERR_INVALID_PARAM — `EVENT_ID` or `REQ_STATE` is invalid. RPMI_ERR_NOT_SUPPORTED — Notification for the `EVENT_ID` is not supported. • Other errors Table 6.

Word	Name	Type	Description

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.1.4. Service: BASE_GET_IMPLEMENTATION_VERSION (SERVICE_ID: 0x02)

This service is used to get the RPMI implementation version of the platform microcontroller. The version returned is a 32-bit composite number containing the `MAJOR` and `MINOR` version numbers.

Table 13. Request Data

NA

Table 14. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. <table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_SUCCESS</td><td>RPMI implementation version returned successfully.</td></tr></table> ● Other errors Table 6.
1	IMPL_VERSION	uint32	Implementation version. <table><tr><th>Bits</th><th>Description</th></tr><tr><td>[31:16]</td><td>`MAJOR` number.</td></tr><tr><td>[15:0]</td><td>`MINOR` number.</td></tr></table>

4.1.5. Service: BASE_GET_IMPLEMENTATION_ID (SERVICE_ID: 0x03)

This service is used to get a 32-bit RPMI implementation ID assigned to the software that implements the RPMI specification. Every implementation ID is unique and listed in the Table 9.

Table 15. Request Data

NA

Table 16. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | RPMI implementation ID returned successfully. ● Other errors Table 6.
1	IMPL_ID	uint32	Implementation ID.

4.1.6. Service: BASE_GET_SPEC_VERSION (SERVICE_ID: 0x04)

This service is used to get the implemented RPMI specification version. The version returned is a 32-bit composite number containing the `MAJOR` and `MINOR` version numbers.

Table 17. Request Data

NA

Table 18. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | RPMI specification version returned successfully. ● Other errors Table 6
1	SPEC_VERSION	uint32	RPMI specification version. Bits | Description [31:16] | `MAJOR` number. [15:0] | `MINOR` number.

4.1.7. Service: BASE_GET_PLATFORM_INFO (SERVICE_ID: 0x05)

This service is used to get additional platform information if available.

Table 19. Request Data

NA

Table 20. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | Platform information returned successfully. ● Other errors Table 6.

Word	Name	Type	Description
1	PLATFORM_ID_LEN	uint32	Platform Identifier field length in bytes.
2	PLATFORM_ID	uint8[`PLATFORM_ID_LEN`]	Platform Identifier. Up to `PLATFORM_ID_LEN` bytes NULL terminated ASCII string. The use and interpretation of this field is implementation-defined. It can be used to convey details such as the vendor ID, vendor name, specific product model, revision, or configuration of the hardware.

4.1.8. Service: BASE_PROBE_SERVICE_GROUP (SERVICE_ID: 0x06)

This service is used to probe the implementation of a service group and to obtain the implemented service group version. The service group version is a 32-bit composite number containing the `MAJOR` and `MINOR` numbers.

If the service group is successfully probed then the implemented service group version is returned in the `SERVICE_GROUP_VERSION` field. Otherwise it returns `0`.

Table 21. Request Data

Word	Name	Type	Description
0	SERVICEGROUP_ID	uint32	Service group ID.

Table 22. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | Service probed successfully. ● Other errors Table 6
1	SERVICE_GROUP_VERSION	uint32	Service group version. Bits | Description [31:16] | `MAJOR` number. [15:0] | `MINOR` number.

4.1.9. Service: BASE_GET_ATTRIBUTES (SERVICE_ID: 0x07)

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

Table 23. Request Data

NA

Table 24. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code: RPMI_SUCCESS — Attributes returned successfully. ● Other errors Table 6.
1	FLAGS0	uint32	Bits [31:2]: *Reserved* and must be `0`. Bits [1]: RPMI context privilege level. `0b1: M-mode.` `0b0: S-mode.` Bits [0]: Event notification support in platform. `0b1: Supported.` `0b0: Not supported.`
2	FLAGS1	uint32	*Reserved* and must be `0`.
3	FLAGS2	uint32	*Reserved* and must be `0`.
4	FLAGS3	uint32	*Reserved* and must be `0`.

4.2. Service Group - SYSTEM_MSI (SERVICEGROUP_ID: 0x0002)

The SYSTEM_MSI service group defines services to allow application processors to receive MSIs upon system events such as P2A doorbell, graceful shutdown/reboot request, CPU hotplug event, memory hotplug event, etc.

The number of system MSIs supported by this service group is fixed and referred to as `SYS_NUM_MSI`. Each system MSI is associated with a unique index which is also referred to as `SYS_MSI_INDEX` where `0 <= SYS_MSI_INDEX < SYS_NUM_MSI`.

The association between system events and system MSI index (aka `SYS_MSI_INDEX`) is platform specific and must be discovered using hardware description mechanisms such as device tree or ACPI.

The system MSI state is 32-bit word also referred to as `SYS_MSI_STATE` which includes whether the system MSI is enabled/disabled and whether system MSI is currently pending at the platform microcontroller. The Table 25 below shows the encoding of `SYS_MSI_STATE`.

A system MSI can be pending for several reasons. For example, if the MSI target address and data are not configured, or if the configured MSI target address is not valid.

Table 25. System MSI State

Bits	Permission	Description
[31:2]	NA	*Reserved* and must be `0`.

Bits	Permission	Description
[1]	Read-Only	MSI pending state. 0b1: MSI is pending. 0b0: MSI is not pending.
[0]	Read-Write	MSI enable state. 0b1: MSI enabled. 0b0: MSI disabled.

The platform microcontroller can only send a pending system MSI if it is enabled and the configured with a valid MSI target address. The system MSI can be enabled/disabled using the `SYSMSI_SET_MSI_STATE` service whereas the system MSI target configuration can be done using the `SYSMSI_SET_MSI_TARGET` service.

If the system MSI target address is IMSIC, then the application processors will directly receive the system MSI whereas if the system MSI target address is `setipnum` *register of a APLIC domain then the application processors receive it as wired interrupt.*

The Table 26 below lists the services in the SYSTEM_MSI service group:

Table 26. SYSTEM_MSI Services

Service ID	Service Name	Request Type
0x01	SYSMSI_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	SYSMSI_GET_ATTRIBUTES	NORMAL_REQUEST
0x03	SYSMSI_GET_MSI_ATTRIBUTES	NORMAL_REQUEST
0x04	SYSMSI_SET_MSI_STATE	NORMAL_REQUEST
0x05	SYSMSI_GET_MSI_STATE	NORMAL_REQUEST
0x06	SYSMSI_SET_MSI_TARGET	NORMAL_REQUEST
0x07	SYSMSI_GET_MSI_TARGET	NORMAL_REQUEST

4.2.1. Notifications

This service group does not support any events for notification.

4.2.2. Service: SYSMSI_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `SYSTEM_MSI` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.2.1.

Table 27. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

Word	Name	Type	Description

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 28. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. <table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_SUCCESS</td><td>Event is subscribed successfully.</td></tr><tr><td>RPMI_ERR_INVALID_PARAM</td><td>`EVENT_ID` or `REQ_STATE` is invalid.</td></tr><tr><td>RPMI_ERR_NOT_SUPPORTED</td><td>Notification for the `EVENT_ID` is not supported.</td></tr></table>• Other errors Table 6.

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.2.3. Service: SYSMSI_GET_ATTRIBUTES (SERVICE_ID: 0x02)

This service is used to discover attributes of the system MSI service group.

Table 29. Request Data

NA

Table 30. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. | Error Code | Description | |---|---| | RPMI_SUCCESS | Service completed successfully. | • Other errors Table 6.
1	SYS_NUM_MSI	uint32	Number of system MSIs.
2	FLAGS0	uint32	*Reserved* and must be `0`.
3	FLAGS1	uint32	*Reserved* and must be `0`.

4.2.4. Service: SYSMSI_GET_MSI_ATTRIBUTES (SERVICE_ID: 0x03)

This service is used to discover attributes of a particular system MSI.

Table 31. Request Data

Word	Name	Type	Description
0	SYS_MSI_INDEX	uint32	Index of the system MSI.

Table 32. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. | Error Code | Description | |---|---| | RPMI_SUCCESS | Service completed successfully. | | RPMI_ERR_INVALID_PARAM | `SYS_MSI_INDEX` value is greater than `SYS_NUM_MSI`. | • Other errors Table 6.
1	FLAGS0	uint32	| Bits | Description | |---|---| | [31:1] | *Reserved* and must be `0`. | | [0] | Preferred privilege level for MSI handling. `0b1: M-mode.` `0b0: M-mode or S-mode.` |
2	FLAGS1	uint32	*Reserved* and must be `0`.
3:6	SYS_MSI_NAME	uint8[16]	System MSI name, a NULL-terminated ASCII string up to 16-bytes.

4.2.5. Service: SYSMSI_SET_MSI_STATE (SERVICE_ID: 0x04)

This service is used to update the state of a system MSI. Specifically, it allows application processors to enable or disable a system MSI. The read-only bits of the system MSI state are not updated by this service.

Table 33. Request Data

Word	Name	Type	Description
0	SYS_MSI_INDEX	uint32	Index of the system MSI.
1	SYS_MSI_STATE	uint32	System MSI state as defined in Table 25.

Table 34. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | MSI is enabled or disabled successfully. RPMI_ERR_INVALID_PARAM | `SYS_MSI_INDEX` value is greater than `SYS_NUM_MSI` or `SYS_MSI_STATE` value is reserved or invalid. ● Other errors Table 6.

4.2.6. Service: SYSMSI_GET_MSI_STATE (SERVICE_ID: 0x05)

This service is used to get the current state of a system MSI.

Table 35. Request Data

Word	Name	Type	Description
0	SYS_MSI_INDEX	uint32	Index of the system MSI.

Table 36. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | MSI state is returned successfully. RPMI_ERR_INVALID_PARAM | `SYS_MSI_INDEX` value is greater than `SYS_NUM_MSI`. ● Other errors Table 6.
1	SYS_MSI_STATE	uint32	System MSI state as defined in Table 25.

4.2.7. Service: SYSMSI_SET_MSI_TARGET (SERVICE_ID: 0x06)

This service is used to configure the target address and data of a system MSI.

Table 37. Request Data

Word	Name	Type	Description
0	SYS_MSI_INDEX	uint32	Index of the system MSI.
1	SYS_MSI_ADDRESS_LOW	uint32	Lower 32-bit of the MSI address.
2	SYS_MSI_ADDRESS_HIGH	uint32	Upper 32-bit of the MSI address.

Word	Name	Type	Description
3	SYS_MSI_DATA	uint32	32-bit MSI data.

Table 38. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | MSI address and data are configured successfully. RPMI_ERR_INVALID_PARAM | `SYS_MSI_INDEX` value is greater than `SYS_NUM_MSI`. RPMI_ERR_INVALID_ADDR | MSI target address is invalid or it is not `4-byte` aligned. ● Other errors Table 6.

4.2.8. Service: SYSMSI_GET_MSI_TARGET (SERVICE_ID: 0x07)

This service is used to get the current target address and data of a system MSI.

Table 39. Request Data

Word	Name	Type	Description
0	SYS_MSI_INDEX	uint32	Index of the system MSI.

Table 40. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | MSI target details returned successfully. RPMI_ERR_INVALID_PARAM | `SYS_MSI_INDEX` value is greater than `SYS_NUM_MSI`. ● Other errors Table 6.
1	SYS_MSI_ADDRESS_LOW	uint32	Lower 32-bit of the MSI address.
2	SYS_MSI_ADDRESS_HIGH	uint32	Upper 32-bit of the MSI address.
3	SYS_MSI_DATA	uint32	32-bit MSI data.

4.3. Service Group - SYSTEM_RESET (SERVICEGROUP_ID: 0x0003)

This service group defines services for system-level reset or shutdown.

The architectural reset types that are supported by default are System Cold Reset and System Shutdown.

System Cold Reset, also known as power-on-reset, involves power cycling the entire system. Upon a successful system cold reset, all devices undergo power cycling in an implementation-defined sequence similar to the initial power-on sequence of the system.

System Shutdown results in all components/devices in the system losing power. Currently, the application processor is the only entity that can request the system shutdown, which means that for the platform microcontroller, it is not necessary to categorize it as a graceful or forceful shutdown. In the case of a shutdown request, it is implicit for the platform microcontroller that the application processor has prepared itself for a successful shutdown.

The following table lists the services in the SYSTEM_RESET service group:

Table 41. SYSTEM_RESET Services

Service ID	Service Name	Request Type
0x01	SYSRST_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	SYSRST_GET_ATTRIBUTES	NORMAL_REQUEST
0x03	SYSRST_RESET	POSTED_REQUEST

4.3.1. Reset Types

RPMI supports reset types and their values as defined by SBI specification. Refer to SRST System Reset Types in the RISC-V SBI Specification [1] for the `RESET_TYPE`.

4.3.2. Notifications

This service group does not support any events for notification.

4.3.3. Service: SYSRST_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `SYSTEM_RESET` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.3.2.

Table 42. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 43. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. | Error Code | Description | |---|---| | RPMI_SUCCESS | Event is subscribed successfully. | | RPMI_ERR_INVALID_PARAM | `EVENT_ID` or `REQ_STATE` is invalid. | | RPMI_ERR_NOT_SUPPORTED | Notification for the `EVENT_ID` is not supported. | ● Other errors Table 6.

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.3.4. Service: SYSRST_GET_ATTRIBUTES (SERVICE_ID: 0x02)

This service is used to discover the attributes of a reset type. The attribute flags indicates if a `RESET_TYPE` is supported or not apart from the System Shutdown and System Cold Reset which are mandatory and supported by default. System Warm Reset support can be discovered with this service.

Table 44. Request Data

Word	Name	Type	Description
0	RESET_TYPE	uint32	Reset type. Refer Section 4.3.1 for more details.

Table 45. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. | Error Code | Description | |---|---| | RPMI_SUCCESS | Attributes returned successfully. | ● Other errors Table 6.

<table>
<tr><th>Word</th><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>1</td><td>FLAGS</td><td>uint32</td><td>Reset type attributes.
<table><tr><th>Bits</th><th>Description</th></tr><tr><td>[31:1]</td><td><i>Reserved</i> and must be <code>0</code>.</td></tr><tr><td>[0]</td><td>Reset type support.<pre>0b1: Supported.
0b0: Not supported.</pre></td></tr></table></td></tr>
</table>

4.3.5. Service: SYSRST_RESET (SERVICE_ID: 0x03)

This service is used to initiate the system reset or system shutdown. The application processor must only request supported reset types, discovered using the `SYSRST_GET_ATTRIBUTES` service except for System Shutdown and System Cold Reset which are supported by default. This service does not return response message in case of successful reset.

Table 46. Request Data

Word	Name	Type	Description
0	RESET_TYPE	uint32	Reset type. Refer Section 4.3.1 for more details.

Table 47. Response Data

<table>
<tr><th>Word</th><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>0</td><td>STATUS</td><td>int32</td><td>Return error code.
<table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_ERR_INVALID_PARAM</td><td>Reset type is not supported or invalid.</td></tr></table><ul><li>Other errors Table 6.</li></ul></td></tr>
</table>

4.4. Service Group - SYSTEM_SUSPEND (SERVICEGROUP_ID: 0x0004)

This service group defines services used to request platform microcontroller to transition the system into a suspend state, also called a sleep state. The suspend state `SUSPEND_TO_RAM` is supported by default by the platform and if the application processor requests for `SUSPEND_TO_RAM`, it's implicit for the platform microcontroller that all the application processors except the one requesting are in `STOPPED` state and necessary state saving in the RAM has been complete.

The following table lists the services in the SYSTEM_SUSPEND service group:

Table 48. SYSTEM_SUSPEND Services

Service ID	Service Name	Request Type
0x01	SYSSUSP_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	SYSSUSP_GET_ATTRIBUTES	NORMAL_REQUEST

Service ID	Service Name	Request Type
0x03	SYSSUSP_SUSPEND	NORMAL_REQUEST

4.4.1. Suspend Types

RPMI supports suspend types and their values as defined by SBI specification. Refer to SBI System Sleep Types in the RISC-V SBI Specification [1] for the `SUSPEND_TYPE` definition.

4.4.2. Notifications

This service group does not support any events for notification.

4.4.3. Service: SYSSUSP_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `SYSTEM_SUSPEND` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.4.2.

Table 49. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 50. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. **Error Code** / **Description**: RPMI_SUCCESS: Event is subscribed successfully. RPMI_ERR_INVALID_PARAM: `EVENT_ID` or `REQ_STATE` is invalid. RPMI_ERR_NOT_SUPPORTED: Notification for the `EVENT_ID` is not supported. ● Other errors Table 6.

Word	Name	Type	Description

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.4.4. Service: SYSSUSP_GET_ATTRIBUTES (SERVICE_ID: 0x02)

This service is used to discover the attributes of a suspend type. The attribute flags for a suspend type indicate whether a `SUSPEND_TYPE` is supported. Additionally, the flags specify whether a `SUSPEND_TYPE` supports a resume address.

Table 51. Request Data

Word	Name	Type	Description
0	SUSPEND_TYPE	uint32	Suspend type. Refer Section 4.4.1 for more details.

Table 52. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. <table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_SUCCESS</td><td>Attributes returned successfully.</td></tr></table>● Other errors Table 6.
1	FLAGS	uint32	Suspend type attributes. <table><tr><th>Bits</th><th>Description</th></tr><tr><td>[31:2]</td><td>*Reserved* and must be `0`.</td></tr><tr><td>[1]</td><td>Resume Address Support. If a `SUSPEND_TYPE` supports custom resume address which platform must configure for the resuming application processor. `0b1: Supported.` `0b0: Not supported.`</td></tr><tr><td>[0]</td><td>Suspend type Support. `0b1: Supported.` `0b0: Not supported.`</td></tr></table>

4.4.5. Service: SYSSUSP_SUSPEND (SERVICE_ID: 0x03)

This service is used to request the platform microcontroller to transition the system in a suspend state. This service returns successfully when the platform microcontroller accepts the system suspend request. The application processor which called this service must then enter into a quiesced state such as WFI. The platform microcontroller will transition the system to the requested `SUSPEND_TYPE` upon the successful transition of the application processor into the supported quiesced state. The mechanism for detecting the quiesced state of the application processor is platform specific.

The application processor must only request supported suspend types, discovered using the `SYSSUSP_GET_ATTRIBUTES` service.

If a suspend type does not support the custom resume address that the application processor can discover through the `SYSSUSP_GET_ATTRIBUTES` service then the `RESUME_ADDR_LOW` and `RESUME_ADDR_HIGH` will be ignored and the application processor will resume from the `pc` (program counter) after the instruction that put the application processor in the quiesced state, such as the `WFI` instruction.

Table 53. Request Data

Word	Name	Type	Description
0	HART_ID	uint32	Hart ID of the calling hart.
1	SUSPEND_TYPE	uint32	Suspend type. Refer Section 4.4.1 for more details.
2	RESUME_ADDR_LOW	uint32	Lower 32-bit address.
3	RESUME_ADDR_HIGH	uint32	Upper 32-bit address.

Table 54. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.

Error Code	Description
RPMI_SUCCESS	Service completed successfully. Suspend request has been accepted.
RPMI_ERR_INVALID_PARAM	`HART_ID` or `SUSPEND_TYPE` is invalid.
RPMI_ERR_INVALID_ADDR	Resume address is invalid.

- Other errors Table 6.

4.5. Service Group - HART_STATE_MANAGEMENT (SERVICEGROUP_ID: 0x0005)

This service group defines services to control and manage the application processor (hart) power states. Hart power states include power on, power off, suspend modes, etc. A hart is identified by a 32-bit identifier called `HART_ID`.

In a platform, depending on the sharing of power controls and common resources, the harts can be grouped in a hierarchical topology to form cores, clusters, nodes, etc. In such cases the power state change for a hart can affect the entire hierarchical group in which the hart is located, requiring coordination for the power state change. RPMI supports the coordination mechanisms and hart power states defined by the RISC-V SBI Specification [1].

The following table lists the services in the HART_STATE_MANAGEMENT service group:

Table 55. HART_STATE_MANAGEMENT Services

Service ID	Service Name	Request Type
0x01	HSM_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	HSM_GET_HART_STATUS	NORMAL_REQUEST
0x03	HSM_GET_HART_LIST	NORMAL_REQUEST
0x04	HSM_GET_SUSPEND_TYPES	NORMAL_REQUEST
0x05	HSM_GET_SUSPEND_INFO	NORMAL_REQUEST
0x06	HSM_HART_START	NORMAL_REQUEST
0x07	HSM_HART_STOP	NORMAL_REQUEST
0x08	HSM_HART_SUSPEND	NORMAL_REQUEST

4.5.1. Hart States

Hart HSM states and the HSM state machine supported by the RPMI are defined in the RISC-V SBI Specification [1]. Refer to HSM States.

From a hart perspective a start state means hart has started execution of instructions and stop state means that hart is not executing the instructions. The platform can implement the stop state either by powering down the hart or just putting the hart in a platform supported low-power state.

4.5.2. Hart Suspend Types

The RPMI supports the hart suspend types encoding as defined in RISC-V SBI Specification [1]. Refer to HSM Suspend Types. The values for the platform supported suspend types are discovered through a service defined in this service group.

4.5.3. Notifications

This service group does not support any events for notification.

4.5.4. Service: HSM_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `HART_STATE_MANAGEMENT` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.5.3.

Table 56. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

Word	Name	Type	Description

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 57. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. **Error Code** — **Description** RPMI_SUCCESS — Event is subscribed successfully. RPMI_ERR_INVALID_PARAM — `EVENT_ID` or `REQ_STATE` is invalid. RPMI_ERR_NOT_SUPPORTED — Notification for the `EVENT_ID` is not supported. • Other errors Table 6.

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.5.5. Service: HSM_GET_HART_STATUS (SERVICE_ID: 0x02)

This service returns the current HSM state of a hart. If a hart is in an invalid state that is not a defined HSM state, an error code will be set in the `STATUS` field.

Table 58. Request Data

Word	Name	Type	Description
0	HART_ID	uint32	Hart ID.

Table 59. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code / Description: RPMI_SUCCESS: Service completed successfully. RPMI_ERR_INVALID_PARAM: `HART_ID` is invalid. RPMI_ERR_INVALID_STATE: Hart is in invalid state. ● Other errors Table 6.
1	HART_STATE	uint32	Hart state. Refer Section 4.5.1 for more details.

4.5.6. Service: HSM_GET_HART_LIST (SERVICE_ID: 0x03)

This service retrieves the list of Hart IDs managed by this service group.

If the number of words required for all available Hart IDs exceeds the number of words that can be returned in one acknowledgement message then the platform microcontroller will set the `REMAINING` and `RETURNED` fields accordingly and only return the Hart IDs which can be accommodated. The application processor may need to call this service again with the appropriate `START_INDEX` until the `REMAINING` field returns `0`.

Table 60. Request Data

Word	Name	Type	Description
0	START_INDEX	uint32	Start index of the Hart ID.

Table 61. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code / Description: RPMI_SUCCESS: Service completed successfully. RPMI_ERR_INVALID_PARAM: `START_INDEX` is invalid. ● Other errors Table 6.
1	REMAINING	uint32	Remaining number of Hart IDs to be returned.
2	RETURNED	uint32	Number of Hart IDs returned in this request.
3	HART_ID[N]	uint32	Hart IDs list.

4.5.7. Service: HSM_GET_SUSPEND_TYPES (SERVICE_ID: 0x04)

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

If the number of words required for all available suspend types exceeds the number of words that can be returned in one acknowledgement message then the platform microcontroller will set the `REMAINING` and `RETURNED` fields accordingly and only return the suspend types which can be accommodated. The application processor may need to call this service again with the appropriate `START_INDEX` until the

`REMAINING` field returns `0`.

The attributes and details of each suspend type can be discovered using the `HSM_GET_SUSPEND_INFO` service.

Table 62. Request Data

Word	Name	Type	Description
0	START_INDEX	uint32	Start index of the Hart ID. `0` for the first call, subsequent calls will use the next index of the remaining items.

Table 63. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code: RPMI_SUCCESS — Description: Service completed successfully. Error Code: RPMI_ERR_INVALID_PARAM — Description: `START_INDEX` is invalid. • Other errors Table 6.
1	REMAINING	uint32	Remaining number of suspend types to be returned.
2	RETURNED	uint32	Number of suspend types returned in this request.
3	SUSPEND_TYPE[N]	uint32	Suspend types. Refer Section 4.5.2 for more details.

Nested table in STATUS description:

Error Code	Description
RPMI_SUCCESS	Service completed successfully.
RPMI_ERR_INVALID_PARAM	`START_INDEX` is invalid.

4.5.8. Service: HSM_GET_SUSPEND_INFO (SERVICE_ID: 0x05)

This service is used to get the attributes of a suspend type. The attributes of a suspend type include various associated latencies. The entry latency for a suspend type is the maximum amount of time that the hart requires to transition from the execution state to the low-power suspend state when the hart invokes an quiesced state entry mechanism such as WFI. The exit latency is the time required by the hart to transition from the suspend state to execution state after the wakeup event.

For each suspend state there is a point of no return after which the suspend state transition cannot be reversed. The wakeup latency is the maximum time required by the hart to transition from the point of no return to the execution state. If the platform returns `0` in `WAKEUP_LATENCY` then the application processor can use the `(ENTRY_LATENCY + EXIT_LATENCY)` as the wakeup latency.

The minimum residency of a suspend state is the minimum time the application processor must remain in that suspend state to become energy efficient compared to the shallower suspend state.

The energy is also consumed while entering and exiting a suspend state. The application processor must spend time equal to or more than minimum residency to justify the energy cost of entering and exiting that suspend state.

The application processor entering into a deeper suspend state with a high minimum residency will incur a longer wakeup latency due to more time required to exit that suspend state. If the predicted idle time by the application processor is less than the minimum residency of a suspend state, it should select the shallower suspend state to minimize the wakeup latency to achieve energy savings.

Table 64. Request Data

Word	Name	Type	Description
0	SUSPEND_TYPE	uint32	Suspend type. Refer Section 4.5.2 for more details.

Table 65. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | Service completed successfully. RPMI_ERR_INVALID_PARAM | `SUSPEND_TYPE` is invalid. ● Other errors Table 6.

1	FLAGS	uint32	Bits | Description
			[31: 1] | *Reserved* and must be `0`.
			[0] | Local timer running status.
			`0b1: Local timer stops when the hart is suspended.`
			`0b0: Local timer does not stop when hart is suspended.`

2	ENTRY_LATENCY	uint32	Entry latency in microseconds.
3	EXIT_LATENCY	uint32	Exit latency in microseconds.
4	WAKEUP_LATENCY	uint32	Wakeup latency in microseconds.
5	MIN_RESIDENCY	uint32	Minimum residency time in microseconds.

4.5.9. Service: HSM_HART_START (SERVICE_ID: 0x06)

This service is used to start the execution on a hart identified by `HART_ID`. This service requires a start address which is the physical address from which the target hart will start execution. Successful completion of this service means that the hart has started execution from the specified start address.

Table 66. Request Data

Word	Name	Type	Description
0	HART_ID	uint32	Hart ID of the target hart to be started.
1	START_ADDR_LOW	uint32	Lower 32-bit of the start address.
2	START_ADDR_HIGH	uint32	Upper 32-bit of the start address.

Table 67. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code: RPMI_SUCCESS — Service completed successfully and hart has started. RPMI_ERR_INVALID_PARAM — `HART_ID` or start address is invalid. RPMI_ERR_ALREADY — Hart is already in transition to start state or has already started. RPMI_ERR_DENIED — Hart is not in stopped state. RPMI_ERR_HW_FAULT — Failed due to hardware fault. ● Other errors Table 6.

Error Code	Description
RPMI_SUCCESS	Service completed successfully and hart has started.
RPMI_ERR_INVALID_PARAM	`HART_ID` or start address is invalid.
RPMI_ERR_ALREADY	Hart is already in transition to start state or has already started.
RPMI_ERR_DENIED	Hart is not in stopped state.
RPMI_ERR_HW_FAULT	Failed due to hardware fault.

4.5.10. Service: HSM_HART_STOP (SERVICE_ID: 0x07)

This service stops the execution on the calling hart. The mechanism for stopping the hart is platform specific. The hart can be powered down, if supported, or put into the deepest available sleep state.

This service returns successful if the platform microcontroller has successfully acknowledged that the target hart can be stopped. The hart upon successful acknowledgement can perform the final context saving if required and must enter into a quiesced state such as WFI which can be detected and allow the platform microcontroller to proceed to stop the hart. The mechanism to detect the hart quiesced state by the platform microcontroller is platform specific.

Once the hart is stopped, it can only be restarted by explicitly invoking the `HSM_HART_START` service call explicitly by any other hart.

Table 68. Request Data

Word	Name	Type	Description
0	HART_ID	uint32	Hart ID of the calling hart.

Table 69. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. ● Other errors Table 6.

Error Code	Description
RPMI_SUCCESS	Service completed successfully and hart is stopped.
RPMI_ERR_ALREADY	Hart is already in transition to stop state or has already stopped.
RPMI_ERR_DENIED	Hart is not in start state.
RPMI_ERR_HW_FAULT	Failed due to hardware failure.

4.5.11. Service: HSM_HART_SUSPEND (SERVICE_ID: 0x08)

This service is used to put a hart in a low-power suspend state supported by the platform. Each suspend type is a 32-bit value which is discovered through the `HSM_GET_SUSPEND_TYPES` service.

This service returns successful if the platform microcontroller has successfully acknowledged that the target hart can be put into the requested `SUSPEND_TYPE` state. The target hart after the successful acknowledgement must enter into a quiesced state such as WFI which can be detected and allow the platform microcontroller complete the suspend state transition. The mechanism to detect the hart quiesced state by the platform microcontroller is platform specific.

For non-retentive suspend state the hart will resume its execution from the provided resume address.

Table 70. Request Data

Word	Name	Type	Description
0	HART_ID	uint32	Hart ID of the calling hart.
1	SUSPEND_TYPE	uint32	Suspend type. Refer Section 4.5.2 for more details.
2	RESUME_ADDR_LOW	uint32	Lower 32-bit of the resume address. Only used for non-retentive suspend types.
3	RESUME_ADDR_HIGH	uint32	Upper 32-bit of the resume address. Only used for non-retentive suspend types.

Table 71. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.

Error Code	Description
RPMI_SUCCESS	Service completed successfully.
RPMI_ERR_INVALID_PARAM	`HART_ID` or `SUSPEND_TYPE` is invalid.

- Other errors Table 6.

4.6. Service Group - CPPC (SERVICEGROUP_ID: 0x0006)

This service group defines the services to control application processor performance by managing a set of registers per application processor that are used for performance management and control. The ACPI CPPC (Collaborative Processor Performance Control) is an abstract and flexible mechanism that allows application processor to collaborate with the platform microcontroller to control the performance.

The CPPC extension defined in the RISC-V SBI specification [1] defines the register IDs for the standard CPPC registers, along with additional registers also required by the application processor.

The ACPI CPPC specification [3] provides the details of the CPPC registers and also provides details on the performance control mechanism through CPPC.

This service group works with the abstract performance scale defined by the ACPI CPPC and is managed by the platform which is responsible for the conversion between the abstract performance level and the internal performance operating point.

The platform may have multiple application processors that share the actual performance controls like clock, voltage regulator and others depending on the platform. In such cases a performance level change for one application processor will affect the entire the group sharing the controls. Its the responsibility of the power and performance management software running on the application processor and the platform to coordinate and manage the group level performance changes.

The following table lists the services in the CPPC service group:

Table 72. CPPC Services

Service ID	Service Name	Request Type
0x01	CPPC_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	CPPC_PROBE_REG	NORMAL_REQUEST
0x03	CPPC_READ_REG	NORMAL_REQUEST
0x04	CPPC_WRITE_REG	NORMAL_REQUEST
0x05	CPPC_GET_FAST_CHANNEL_REGION	NORMAL_REQUEST
0x06	CPPC_GET_FAST_CHANNEL_OFFSET	NORMAL_REQUEST
0x07	CPPC_GET_HART_LIST	NORMAL_REQUEST

4.6.1. CPPC Fast-channel

The CPPC service group defines the fast-channels to be used by the application processor to request performance changes and to obtain performance change feedback for an application processor from the platform microcontroller.

A fast-channel shared memory layout is specific to the CPPC service group. The data written in a fast-channel do not follow the conventional RPMI message format. The simple data format supported by the fast-channel allows faster processing of the performance change requests made through a fast-channel and faster read of the performance feedback values supported over the fast-channel.

The CPPC service group defines two types of fast-channel for each application processor. If fast-channels are supported then each application processor must be assigned both types fast-channel.

Performance Request Fast-channel

In this fast-channel the application processor will either write the desired performance level in case of normal mode or the minimum and maximum performance level in case of Autonomous (CPPC2) mode in the fast-channel. Otherwise the application processor can call the service **`CPPC_WRITE_REG`** for the `DesiredPerformanceRegister` or `MinimumPerformanceRegister` and `MaximumPerformanceRegister`.

The supported values in this fast-channel which depends on the CPPC mode, either normal or autonomous mode is discoverable through `CPPC_GET_FAST_CHANNEL_REGION` service.

The size of this fast-channel type is `8 bytes`.

Table 73. CPPC Performance Request Fast-channel Layout

CPPC Mode	Layout	
	Offset	**Value (32-bit)**
Normal Mode	0x0	Desired performance level.
	0x4	*Reserved* and must be `0`.

CPPC Mode	Layout	
	Offset	Value (32-bit)
Autonomous (CPPC2) mode	0x0	Minimum performance level.
	0x4	Maximum performance level.

Performance Feedback Fast-channel

In this fast-channel the application processor will read the supported value for estimating the delivered performance as performance feedback for an application processor. The application processor current frequency (Hz) is used for performance feedback in this fast-channel. The platform microcontroller must write the frequency of an application processor in the fast-channel whenever it changes.

The size of this fast-channel type is `8 bytes`.

Table 74. CPPC Performance Feedback Fast-channel Layout

Offset	Value (32-bit)
0x0	Current frequency low 32-bit (Hz).
0x4	Current frequency high 32-bit (Hz).

CPPC Fast-channel Shared Memory Region

The size of the shared memory region containing all the fast-channels for all the managed application processors must be a `power-of-2`. The `base-address` and `size` (bytes) of this shared memory region can be discovered using the service `CPPC_GET_FAST_CHANNEL_REGION`. The `base-address` of the shared memory region must be aligned to `8 bytes` which is maximum size of a fast-channel in both the types.

The offsets of fast-channels of both types for an application processor are aligned to `8 bytes`. The offset of both fast-channel types in the shared memory region can be discovered through service `CPPC_GET_FAST_CHANNEL_OFFSET`. The offsets discovered can be added to the `base-address` of the shared memory region to form the address of Performance Request fast-channel and Performance Feedback fast-channel for an application processor.

Performance Request Fast-channel Doorbell

A doorbell can also be supported for this fast-channel type which is shared between all the application processors. The doorbell, if supported must be a memory mapped register with write access. The doorbell details and attributes such as doorbell register address, doorbell write value can be discovered by the application processor through the `CPPC_GET_FAST_CHANNEL_REGION` service.

The doorbell register address is the physical address of the register. The doorbell write value is the value which must be written in the doorbell register to trigger the doorbell interrupt.

The width of the doorbell write value must be equal to the doorbell register width.

The write value may also contains other set bits which must persist on every write to the doorbell register.

4.6.2. Notifications

This service group does not support any events for notification.

4.6.3. Service: CPPC_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `CPPC` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.6.2.

Table 75. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 76. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. <table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_SUCCESS</td><td>Event is subscribed successfully.</td></tr><tr><td>RPMI_ERR_INVALID_PARAM</td><td>`EVENT_ID` or `REQ_STATE` is invalid.</td></tr><tr><td>RPMI_ERR_NOT_SUPPORTED</td><td>Notification for the `EVENT_ID` is not supported.</td></tr></table>● Other errors Table 6.

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.6.4. Service: CPPC_PROBE_REG (SERVICE_ID: 0x02)

This service is used to probe a CPPC register implementation status for a application processor. If the CPPC register `reg_id` is implemented then the length in bits is returned in `REG_LENGTH` field. If the register is not supported or invalid then the `REG_LENGTH` will be `0`.

Table 77. Request Data

Word	Name	Type	Description
0	REG_ID	uint32	CPPC register ID.
1	HART_ID	uint32	Hart ID.

Table 78. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.
1	REG_LENGTH	uint32	Register length (bits).

Error Code	Description
RPMI_SUCCESS	CPPC register probed successfully.
RPMI_ERR_INVALID_PARAM	`HART_ID` or `REG_ID` is invalid.
RPMI_ERR_NOT_SUPPORTED	`REG_ID` is not supported.

- Other errors Table 6.

4.6.5. Service: CPPC_READ_REG (SERVICE_ID: 0x03)

This service is used to read a CPPC register. If the fast-channels are supported, a read of the `DesiredPerformanceRegister` or `MinimumPerformanceRegister` and `MaximumPerformanceRegister` through this service will return the current desired performance level or minimum and maximum performance level limit depending on the CPPC mode from the fast-channel of a application processor.

Table 79. Request Data

Word	Name	Type	Description
0	REG_ID	uint32	CPPC register ID.
1	HART_ID	uint32	Hart ID.

Table 80. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.
1	DATA_LOW	uint32	Lower 32-bit of the data.
2	DATA_HIGH	uint32	Upper 32-bit of data. This will be `0` if the register is of 32-bit length.

Error Code	Description
RPMI_SUCCESS	Service completed successfully.
RPMI_ERR_INVALID_PARAM	`HART_ID` or `REG_ID` is invalid.
RPMI_ERR_NOT_SUPPORTED	`REG_ID` is not supported.

- Other errors Table 6.

4.6.6. Service: CPPC_WRITE_REG (SERVICE_ID: 0x04)

This service is used to write a CPPC register.

If the fast-channels are supported the application processor must only write desired performance level in the fast-channel instead of writing into the `DesiredPerformanceRegister` through this service. Similarly, in case of the autonomous mode the application processor must write minimum and maximum limit levels into the fast-channel instead of calling this service for `MinimumPerformanceRegister` and `MaximumPerformanceRegister`. Otherwise the writes to these registers may be ignored.

Table 81. Request Data

Word	Name	Type	Description
0	REG_ID	uint32	CPPC register ID.
1	HART_ID	uint32	Hart ID.
2	DATA_LOW	uint32	Lower 32-bit of data.
3	DATA_HIGH	uint32	Upper 32-bit of data. This is ignored if the register is of 32-bit length.

Table 82. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.

Error Code	Description
RPMI_SUCCESS	Service completed successfully.
RPMI_ERR_INVALID_PARAM	`HART_ID` or `REG_ID` is invalid.
RPMI_ERR_NOT_SUPPORTED	`REG_ID` is not supported.
RPMI_ERR_DENIED	`REG_ID` is read only.

- Other errors Table 6.

4.6.7. Service: CPPC_GET_FAST_CHANNEL_REGION (SERVICE_ID: 0x05)

This service is used to get the details of the shared memory region containing all the fast-channels, attributes of the fast-channel and the details of the doorbell if supported.

The doorbell details are unspecified and considered invalid if the Performance Request fast-channel doorbell (`FLAGS[0] = 0`) is not supported and must not be used.

Table 83. Request Data

NA

Table 84. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.

Error Code	Description
RPMI_SUCCESS	Service completed successfully.
RPMI_ERR_NOT_SUPPORTED	Fast-channels not supported.

- Other errors Table 6.

Word	Name	Type	Description

1	FLAGS	uint32	**Bits** / **Description**
			[31:5] — *Reserved* and must be `0`.
			[4:3] — CPPC mode.
			`0b00: Normal mode. Desired performance level for performance change.`
			`0b01: Autonomous mode. Performance limit change. Platform can choose the level in the requested limit.`
			`0b10 - 0b11: Reserved.`
			[2:1] — Performance Request fast-channel doorbell register width.
			`0b00: 8-bit.`
			`0b01: 16-bit.`
			`0b10: 32-bit.`
			`0b11: Reserved.`
			[0] — Performance Request fast-channel doorbell support.
			`0b1: Supported.`
			`0b0: Not supported.`

2	REGION_ADDR_LOW	uint32	Lower 32-bit of the fast-channels shared memory region physical address.
3	REGION_ADDR_HIGH	uint32	Upper 32-bit of the fast-channels shared memory region physical address.
4	REGION_SIZE_LOW	uint32	Lower 32-bit of the fast-channels shared memory region size.
5	REGION_SIZE_HIGH	uint32	Upper 32-bit of the fast-channels shared memory region size.
6	DB_ADDR_LOW	uint32	Lower 32-bit of doorbell register address for Performance Request fast-channel.
7	DB_ADDR_HIGH	uint32	Upper 32-bit of doorbell register address for Performance Request fast-channel.

Word	Name	Type	Description
8	DB_WRITE_VALUE	uint32	32-bit doorbell write value for Performance Request fast-channel. If the doorbell register width is less than 32-bit, the lower bits in this field equal to the doorbell register width must be used as write value.

4.6.8. Service: CPPC_GET_FAST_CHANNEL_OFFSET (SERVICE_ID: 0x06)

This service is used to get the offsets of Performance Request fast-channel and Performance Feedback fast-channel for an application processor in the shared memory region containing all the fast-channels.

Table 85. Request Data

Word	Name	Type	Description
0	HART_ID	uint32	Hart ID.

Table 86. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. **Error Code** | **Description** RPMI_SUCCESS | Service completed successfully. RPMI_ERR_INVALID_PARAM | `HART_ID` is invalid. RPMI_ERR_NOT_SUPPORTED | Fast-channels not supported. • Other errors Table 6.
1	PERF_REQUEST_OFFSET_LOW	uint32	Lower 32-bit of a Performance Request fast-channel offset.
2	PERF_REQUEST_OFFSET_HIGH	uint32	Upper 32-bit of a Performance Request fast-channel offset.
3	PERF_FEEDBACK_OFFSET_LOW	uint32	Lower 32-bit of a Performance Feedback fast-channel offset.
4	PERF_FEEDBACK_OFFSET_HIGH	uint32	Upper 32-bit of a Performance Feedback fast-channel offset.

4.6.9. Service: CPPC_GET_HART_LIST (SERVICE_ID: 0x07)

This service retrieves the list of Hart IDs managed by this service group for performance control.

If the number of words required for all available Hart IDs exceeds the number of words that can be returned in one acknowledgement message then the platform microcontroller will set the `REMAINING` and `RETURNED` fields accordingly and only return the Hart IDs which can be accommodated. The application processor may need to call this service again with the appropriate `START_INDEX` until the `REMAINING` field returns `0`.

Table 87. Request Data

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

Table 88. Response Data

<table>
<tr><th>Word</th><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>0</td><td>STATUS</td><td>int32</td><td>Return error code.
<table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_SUCCESS</td><td>Service completed successfully.</td></tr><tr><td>RPMI_ERR_INVALID_PARAM</td><td><code>START_INDEX</code> is invalid.</td></tr></table>
● Other errors Table 6.</td></tr>
<tr><td>1</td><td>REMAINING</td><td>uint32</td><td>Remaining number of Hart IDs to be returned.</td></tr>
<tr><td>2</td><td>RETURNED</td><td>uint32</td><td>Number of Hart IDs returned in this request.</td></tr>
<tr><td>3</td><td>HART_ID[N]</td><td>uint32</td><td>Hart IDs.</td></tr>
</table>

4.7. Service Group - VOLTAGE (SERVICEGROUP_ID: 0x0007)

This service group is used to control the voltage level of the voltage domains. A voltage domain is the logical grouping of one or more devices powered by a single controllable voltage source. A system may have multiple voltage domains and the services defined in this service group are used to manage and control the voltage levels of these voltage domains. Each voltage domain is identified by `DOMAIN_ID` which is a 32-bit integer starting from `0`.

The following table lists the services in the VOLTAGE service group:

Table 89. VOLTAGE Services

Service ID	Service Name	Request Type
0x01	VOLT_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	VOLT_GET_NUM_DOMAINS	NORMAL_REQUEST
0x03	VOLT_GET_ATTRIBUTES	NORMAL_REQUEST
0x04	VOLT_GET_SUPPORTED_LEVELS	NORMAL_REQUEST
0x05	VOLT_SET_CONFIG	NORMAL_REQUEST
0x06	VOLT_GET_CONFIG	NORMAL_REQUEST
0x07	VOLT_SET_LEVEL	NORMAL_REQUEST
0x08	VOLT_GET_LEVEL	NORMAL_REQUEST

4.7.1. Voltage Level Format

There are two types of voltage level formats supported in the VOLTAGE service group. The voltage levels are represented as a group.

Discrete Format

A set of discrete voltage levels arranged in a sequence, starting from the lowest value at the lowest index and increasing sequentially to higher levels. The following table shows the structure of the discrete format.

```
[voltage0, voltage1, voltage2, ... , voltage(N-1)]

where:
voltage0 < voltage1 < voltage2 < ... < voltage(N-1)
```

Word	Name	Description
0	VOLTAGE	Discrete voltage level in microvolts (uV).

Linear Range Format

A linear range of voltage levels with a constant step size. The following table shows the structure of the linear range voltage format.

```
[voltage_min, voltage_max, voltage_step]
```

Multi-linear range format can be supported by having multiple `linear range` tuple arranged in an continuous array format.

```
[voltage_min0, voltage_max0, voltage_step0],
[voltage_min1, voltage_max1, voltage_step1],
...
[voltage_min(N-1), voltage_max(N-1), voltage_step(N-1)],
```

The format must be packed sequentially such that `voltage_max0 < voltage_min1, voltage_max1 < voltage_min2` and so on.

Each linear range is considered as a single voltage level.

Word	Name	Description
0	VOLTAGE_MIN	Lower boundary of voltage level in microvolts (uV).
1	VOLTAGE_MAX	Upper boundary of voltage level in microvolts (uV).
2	STEP	Step size in microvolts (uV).

4.7.2. Notifications

This service group does not support any events for notification.

4.7.3. Service: VOLT_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `VOLTAGE` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.7.2.

Table 90. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 91. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code / Description: RPMI_SUCCESS: Event is subscribed successfully. RPMI_ERR_INVALID_PARAM: `EVENT_ID` or `REQ_STATE` is invalid. RPMI_ERR_NOT_SUPPORTED: Notification for the `EVENT_ID` is not supported. ● Other errors Table 6.

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.7.4. Service: VOLT_GET_NUM_DOMAINS (SERVICE_ID: 0x02)

This service is used to query the number of voltage domains available in the system.

Table 92. Request Data

NA

Table 93. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.

Error Code	Description
RPMI_SUCCESS	Service completed successfully.

- Other errors Table 6.

1	NUM_DOMAINS	uint32	Number of voltage domains.

4.7.5. Service: VOLT_GET_ATTRIBUTES (SERVICE_ID: 0x03)

Each domain may support multiple voltage levels, which are permitted by the domain for operation. The number of levels indicates the total count of voltage levels supported within a voltage domain. Transition latency denotes the maximum time required for the voltage to stabilize upon a change in the regulator. The `FLAGS` field encodes the voltage format supported by the hardware, including discrete and linear range formats." The `NUM_LEVELS` field returns the number of discrete voltage in case discrete format and number of linear range tuple in linear range voltage format. Each domain can support only one voltage level format. Additional voltage formats can be accommodated in the future if required.

Table 94. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Voltage domain ID.

Table 95. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.

Error Code	Description
RPMI_SUCCESS	Service completed successfully.
RPMI_ERR_INVALID_PARAM	`DOMAIN_ID` is invalid.

- Other errors Table 6.

Word	Name	Type	Description

1	FLAGS	uint32	**Bits** | **Description**
			[31:4] | *Reserved* and must be `0`.
			[3:1] | Voltage format.
			Refer to Section 4.7.1 for more details.
			`0b000: Discrete format.`
			`0b001: Linear range format.`
			`0b010 - 0b111: Reserved.`
			[0] | Voltage domain control support.
			`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	The number of voltage levels (number of arrays) supported by the domain. If the voltage level format is a linear range, then each linear range is considered a single voltage level.
3	TRANS_LATENCY	uint32	Transition latency, in microsecond (us).
4:7	DOMAIN_NAME	uint8[16]	Voltage domain name, a NULL-terminated ASCII string up to 16-bytes.

4.7.6. Service: VOLT_GET_SUPPORTED_LEVELS (SERVICE_ID: 0x04)

Each domain may support multiple voltage levels which are allowed by the domain to operate. The number of voltage levels returned depends on the format of the voltage level.

The total number of words required to represent the voltage levels in one message cannot exceed the total words available in one message `DATA` field. If the number of levels exceeds this limit, the platform microcontroller will return the maximum number of levels that can be accommodated in one message and adjust the `REMAINING` field accordingly. When the `REMAINING` field is not zero, the application processor must make subsequent service calls with the appropriate `VOLTAGE_LEVEL_INDEX` set to retrieve the remaining voltage levels. It is possible that multiple service calls may be necessary to retrieve all the voltage levels.

Table 96. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Voltage domain ID.
1	VOLTAGE_LEVEL_INDEX	uint32	The index of discrete voltage if the format is discrete, or index of linear range tuple if the format is linear range.

Table 97. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code / Description: RPMI_SUCCESS: Service completed successfully and voltage levels returned. RPMI_ERR_INVALID_PARAM: Voltage `DOMAIN_ID` is invalid. RPMI_ERR_INVALID_PARAM: `VOLTAGE_LEVEL_INDEX` is invalid. • Other errors Table 6.
1	FLAGS	uint32	*Reserved* and must be `0`.
2	REMAINING	uint32	The remaining number of discrete voltage if the format is discrete type, or number of linear range tuple if the format is linear range.
3	RETURNED	uint32	The number of discrete voltage levels returned if the format is discrete type, or the number of linear range tuple if the format is linear range.
4	VOLTAGE_LEVEL[]	uint32[]	Voltage levels. The voltage format data structure and its packing is according to the supported format. Refer to Section 4.7.1 for more details.

4.7.7. Service: VOLT_SET_CONFIG (SERVICE_ID: 0x05)

This service is used to configure a voltage domain.

Table 98. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Voltage domain ID.

1	CONFIG	uint32	Voltage domain config.
			
			Bits / Description:
			[31:1]: *Reserved* and must be `0`.
			[0]: Voltage supply control.
			`0b1: Enable voltage supply.`
			`0b0: Disable voltage supply.`

Table 99. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code: RPMI_SUCCESS — Description: Service completed successfully. Error Code: RPMI_ERR_INVALID_PARAM — Description: Voltage `DOMAIN_ID` or `CONFIG` is invalid. ● Other errors Table 6.

4.7.8. Service: VOLT_GET_CONFIG (SERVICE_ID: 0x06)

This service is used to get the configuration of a voltage domain.

Table 100. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Voltage domain ID.

Table 101. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code: RPMI_SUCCESS — Description: Service completed successfully. Error Code: RPMI_ERR_INVALID_PARAM — Description: Voltage `DOMAIN_ID` not found. ● Other errors Table 6.

1	CONFIG	uint32	Voltage domain config.
			
			Bits [31:1] — Description: *Reserved* and must be `0`.
			Bits [0] — Description: Voltage supply state.
			`0b1: Voltage supply is enabled.`
			`0b0: Voltage supply is disabled.`

4.7.9. Service: VOLT_SET_LEVEL (SERVICE_ID: 0x07)

This service is used to set the voltage level in microvolts of a voltage domain.

Table 102. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Voltage domain ID.
1	VOLTAGE_LEVEL	int32	Voltage level, in microvolts.

Table 103. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | Service completed successfully. RPMI_ERR_INVALID_PARAM | Voltage `DOMAIN_ID` or `VOLTAGE_LEVEL` is invalid. ● Other errors Table 6.

4.7.10. Service: VOLT_GET_LEVEL (SERVICE_ID: 0x08)

This service is used to get the current voltage level in microvolts of a voltage domain.

Table 104. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Voltage domain ID.

Table 105. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | Service completed successfully. RPMI_ERR_INVALID_PARAM | Voltage `DOMAIN_ID` not found. ● Other errors Table 6.
1	VOLTAGE_LEVEL	int32	Voltage level, in microvolts.

4.8. Service Group - CLOCK (SERVICEGROUP_ID: 0x0008)

This service group is used to manage system clocks. Services defined in this group are used to enable or disable clocks, set or get clock rates, etc.

Each clock in the system is identified by a clock ID, which is a 32-bit integer identifier assigned to each clock. The mapping of CLOCK_ID to clock is known to both the application processor and the platform microcontroller. Clock IDs are sequential and start from `0`.

A device or a group of devices sharing the same clock source form a single clock domain identified by the CLOCK_ID. Any change to the clock source affects the entire domain, which may include multiple devices.

The topology of the devices and the clock source depends on the system design and is implementation specific. The operating system can discover this topology through supported hardware description mechanisms.

The following table lists the services in the CLOCK service group:

Table 106. CLOCK Services

Service ID	Service Name	Request Type
0x01	CLK_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	CLK_GET_NUM_CLOCKS	NORMAL_REQUEST
0x03	CLK_GET_ATTRIBUTES	NORMAL_REQUEST
0x04	CLK_GET_SUPPORTED_RATES	NORMAL_REQUEST
0x05	CLK_SET_CONFIG	NORMAL_REQUEST
0x06	CLK_GET_CONFIG	NORMAL_REQUEST
0x07	CLK_SET_RATE	NORMAL_REQUEST
0x08	CLK_GET_RATE	NORMAL_REQUEST

4.8.1. Clock Rate Format

Each clock rate is a array 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`.

Discrete Format

A set of discrete clock rates arranged in a sequence, starting from the lowest value at the lowest index and increasing sequentially to higher clock rate. The following table shows the structure of the discrete clock rate format.

```
[clock_rate0, clock_rate1, clock_rate2, ... , clock_rate(N-1)]

where:
clock_rate0 < clock_rate1 < clock_rate2 < ... < clock_rate(N-1)
```

Table 107. Discrete Clock Format Structure

Word	Name	Description
0	CLOCK_RATE_LOW	Lower 32-bit clock rate in Hz.
1	CLOCK_RATE_HIGH	Upper 32-bit clock rate in Hz.

Linear Range Format

A linear range of clock rates represented by minimum and maximum clock rate and a constant step size. The following table shows the fixed structure of the linear range format for clock rates.

```
[clock_rate_min, clock_rate_max, clock_step]
```

Table 108. Linear Range Format Structure

Word	Name	Description
0	CLOCK_MIN_RATE_LOW	Lower 32-bit of the lowest clock rate in Hz.
1	CLOCK_MIN_RATE_HIGH	Upper 32-bit of the lowest clock rate in Hz.

Word	Name	Description
2	CLOCK_MAX_RATE_LOW	Lower 32-bit of the highest clock rate in Hz.
3	CLOCK_MAX_RATE_HIGH	Upper 32-bit of the highest clock rate in Hz.
4	CLOCK_STEP_LOW	Lower 32-bit of the step between two successive rates in Hz.
5	CLOCK_STEP_HIGH	Upper 32-bit of the step between two successive rates in Hz.

A clock may also support clock rates which can be represented by multiple linear ranges. For example,

```
[clock_rate_min0, clock_rate_max0, clock_rate_step0],
[clock_rate_min1, clock_rate_max1, clock_rate_step1],
 ...,
[clock_rate_min(N-1), clock_rate_max(N-1), clock_rate_step(N-1)]
```

The linear ranges must be packed sequentially such that `clock_rate_max0 < clock_rate_min1`, `clock_rate_max1 < clock_rate_min2` and so on.

4.8.2. Notifications

This service group does not support any events for notification.

4.8.3. Service: CLK_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `CLOCK` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.8.2.

Table 109. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 110. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code: RPMI_SUCCESS — Description: Event is subscribed successfully. Error Code: RPMI_ERR_INVALID_PARAM — Description: `EVENT_ID` or `REQ_STATE` is invalid. Error Code: RPMI_ERR_NOT_SUPPORTED — Description: Notification for the `EVENT_ID` is not supported. ● Other errors Table 6

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.8.4. Service: CLK_GET_NUM_CLOCKS (SERVICE_ID: 0x02)

This service is used to query the number of clocks available in the system. All supported clocks in the system are designated by an integer identifier called `CLOCK_ID`.

Table 111. Request Data

NA

Table 112. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code: RPMI_SUCCESS — Description: Service completed successfully. ● Other errors Table 6.
1	NUM_CLOCKS	uint32	Number of clocks.

4.8.5. Service: CLK_GET_ATTRIBUTES (SERVICE_ID: 0x03)

This service provides detailed attributes of a clock, including its name, represented as a 16-byte array of ASCII strings. It also specifies the transition latency, which denotes the maximum time for the clock to stabilize after a configuration change. The `FLAGS` field encodes the clock formats supported by the clock. When the format is of the discrete type, the `NUM_RATES` field returns the number of discrete clock rates supported by the clock. In the case of linear range format the `NUM_RATES` will return the number of linear ranges supported.

Table 113. Request Data

Word	Name	Type	Description
0	CLOCK_ID	uint32	Clock ID.

Table 114. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code: RPMI_SUCCESS — Description: Service completed successfully. Error Code: RPMI_ERR_INVALID_PARAM — Description: `CLOCK_ID` is invalid. ● Other errors Table 6.

1	FLAGS	uint32	Bits [31:2]: *Reserved* and must be `0`.
			Bits [1:0]: Clock format.
			
			Refer to Section 4.8.1 for more details.
			
			`0b00: Discrete format.`
			`0b01: Linear range format.`
			`0b10 - 0b11: Reserved.`

2	NUM_RATES	uint32	The number of discrete clock rates if the format is of discrete type, or the number of linear ranges if the format is linear range.
3	TRANSITION_LATENCY	uint32	Transition latency, in microseconds (us).
4:7	CLOCK_NAME	uint8[16]	Clock name, a NULL-terminated ASCII string up to 16-bytes.

4.8.6. Service: CLK_GET_SUPPORTED_RATES (SERVICE_ID: 0x04)

This service is used to get the supported clock rates. The clock rate data returned by this service depends on the format supported by the clock.

If the format is discrete, the message can pass the `CLOCK_RATE_INDEX` which is the index to the first rate value to be described in the returned rate array. If all supported rate values are required then this index value can be `0`. Similarly, if the format is linear range, then the `CLOCK_RATE_INDEX` is the index of the first linear range to be described in the returned clock rate linear ranges. If all the supported linear ranges are needed then this index value can be `0`.

The total number of words required for the number of discrete clock rates or linear ranges according to the format in one message must not exceed the total words available in a message DATA field. If the format is linear range and a clock supports multiple linear ranges, then only complete linear ranges must be returned as per the data format of the linear range described in Section 4.8.1.2.

If the total number of words required to store all supported discrete clock rates or the linear ranges exceed the available words in message DATA field then `REMAINING` and `RETURNED` must be set accordingly. In such condition, if the format is discrete, the platform microcontroller will return the discrete clock rates which can be accommodated in one message and set the `RETURNED` field to number of discrete clock rates returned and `REMAINING` field is set to the remaining number of discrete clock rates. Similarly if the format is linear, the linear ranges which can be accommodated in one message are returned with `RETURNED` field set to the number of linear ranges returned and `REMAINING` field is set to the remaining number of linear ranges.

The application processor, when `REMAINING` field is not `0` must call this service again with appropriate `CLOCK_RATE_INDEX` set to get the remaining discrete clock rates or linear ranges.

Table 115. Request Data

Word	Name	Type	Description
0	CLOCK_ID	uint32	Clock ID.
1	CLOCK_RATE_INDEX	uint32	Clock rate index.

Table 116. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | Service completed successfully. RPMI_ERR_INVALID_PARAM | `CLOCK_ID` or `CLOCK_RATE_INDEX` is invalid. ● Other errors Table 6.
1	FLAGS	uint32	*Reserved* and must be `0`.
2	REMAINING	uint32	The remaining number of discrete clock rates if the format is discrete type, or the remaining number of linear ranges if the format is linear range.
3	RETURNED	uint32	The number of discrete clock rates returned if the format is discrete type, or the number of linear ranges returned if the format is linear range.
4	CLOCK_RATE[ ]	uint32[2]	Clock rates. The clock rate data structure and its packing is according to the supported format. Refer to Section 4.8.1 for more details.

4.8.7. Service: CLK_SET_CONFIG (SERVICE_ID: 0x05)

This service is used to configure a clock domain.

Table 117. Request Data

Word	Name	Type	Description
0	CLOCK_ID	uint32	Clock ID.

Word	Name	Type	Description
1	CONFIG	uint32	Clock config.

Bits	Description
[31:1]	*Reserved* and must be `0`.

[0]	Clock control.
	`0b0: Disable clock.`
	`0b1: Enable clock.`

Table 118. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.

Error Code	Description
RPMI_SUCCESS	Service completed successfully.
RPMI_ERR_INVALID_PARAM	`CLOCK_ID` or `CONFIG` is invalid.

- Other errors Table 6.

4.8.8. Service: CLK_GET_CONFIG (SERVICE_ID: 0x06)

This service is used to get the configuration of a clock domain.

Table 119. Request Data

Word	Name	Type	Description
0	CLOCK_ID	uint32	Clock ID.

Table 120. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.

Error Code	Description
RPMI_SUCCESS	Service completed successfully.
RPMI_ERR_INVALID_PARAM	`CLOCK_ID` is invalid.

- Other errors Table 6

Word	Name	Type	Description

1	CONFIG	uint32	Clock config.
			
			Bits: [31:1] — *Reserved* and must be `0`.
			Bits: [0] — Clock state.
			`0b0: Clock is disabled.`
			`0b1: Clock is enabled.`

4.8.9. Service: CLK_SET_RATE (SERVICE_ID: 0x07)

This service is used to set the clock rate of a specific clock.

Table 121. Request Data

Word	Name	Type	Description
0	CLOCK_ID	uint32	Clock ID.

1	FLAGS	uint32	Bits: [31:2] — *Reserved* and must be `0`.
			Bits: [1:0] — Clock rate rounding mode.
			`0b00: Round down.`
			`0b01: Round up.`
			`0b10: Auto.`
			`0b11: Reserved.`
			
			`In Auto mode the platform can autonomously chooses a supported rate closest to the requested rate.`

2	CLOCK_RATE_LOW	uint32	Lower 32-bit of the clock rate in Hertz.
3	CLOCK_RATE_HIGH	uint32	Upper 32-bit of the clock rate in Hertz.

Table 122. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code: RPMI_SUCCESS — Service completed successfully. Error Code: RPMI_ERR_INVALID_PARAM — `CLOCK_ID` or clock rate is invalid or the flags passed are invalid or reserved. ● Other errors Table 6.

4.8.10. Service: CLK_GET_RATE (SERVICE_ID: 0x08)

This service is used to get the current clock rate.

Table 123. Request Data

Word	Name	Type	Description
0	CLOCK_ID	uint32	Clock ID.

Table 124. Request Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code: RPMI_SUCCESS — Description: Service completed successfully. Error Code: RPMI_ERR_INVALID_PARAM — Description: `CLOCK_ID` is invalid. • Other errors Table 6.
1	CLOCK_RATE_LOW	uint32	Lower 32-bit of the clock rate in Hertz.
2	CLOCK_RATE_HIGH	uint32	Upper 32-bit of the clock rate in Hertz.

4.9. Service Group - DEVICE_POWER (SERVICEGROUP_ID: 0x0009)

This DEVICE_POWER service group provides messages to manage the power states of a device power domain. This service group is used only for device power management since system and CPU power management is handled by already defined service groups such as 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 states 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. The level of context preserved will depends on the level of power state.

Power states for domains will be discovered via supported hardware description mechanisms where the values for ON and OFF are already fixed and known. The power state encodes both the power state value and the context preserved or lost information corresponding to that state.

The DEVICE_POWER services take a 32-bit integer identifier known as `DOMAIN_ID` to specify the device power domain. These `DOMAIN_ID` identifiers are sequential and start from `0`.

The following table lists the services in the DEVICE_POWER service group:

Table 125. DEVICE_POWER Services

Service ID	Service Name	Request Type
0x01	DPWR_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	DPWR_GET_NUM_DOMAINS	NORMAL_REQUEST
0x03	DPWR_GET_ATTRIBUTES	NORMAL_REQUEST
0x04	DPWR_SET_STATE	NORMAL_REQUEST
0x05	DPWR_GET_STATE	NORMAL_REQUEST

4.9.1. Power State Format

The power state is represented as a 32-bit value. The following table shows the encoding for the power state.

Table 126. Power State Encoding

Bit	Name	Description
[31:17]	RESERVED	*Reserved* and must be `0`.

[16]	CONTEXT	`0b1: Context is lost.`
		`0b0: Context is preserved.`

[15:0]	VALUE	Value / Description: 0x0000: On. 0x0001: *Reserved* and must be `0`. 0x0002: *Reserved* and must be `0`. 0x0003: Off. 0x0004 - 0x0FFF: *Reserved* and must be `0`. 0x1000 - 0xFFFF: Vendor specific states.

4.9.2. Notifications

This service group does not support any events for notification.

4.9.3. Service: DPWR_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `DEVICE_POWER` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.9.2.

Table 127. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 128. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code — Description RPMI_SUCCESS — Event is subscribed successfully. RPMI_ERR_INVALID_PARAM — `EVENT_ID` or `REQ_STATE` is invalid. RPMI_ERR_NOT_SUPPORTED — Notification for the `EVENT_ID` is not supported. ● Other errors Table 6.

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.9.4. Service: DPWR_GET_NUM_DOMAINS (SERVICE_ID: 0x02)

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 may be less than the actual number of domains present on the platform.

Table 129. Request Data

NA

Table 130. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code — Description RPMI_SUCCESS — Service completed successfully. ● Other errors Table 6.
1	NUM_DOMAINS	uint32	Number of device power domains.

4.9.5. Service: DPWR_GET_ATTRIBUTES (SERVICE_ID: 0x03)

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

Table 131. Request Data

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

Table 132. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code / Description: RPMI_SUCCESS — Service completed successfully. RPMI_ERR_INVALID_PARAM — `DOMAIN_ID` is invalid. ● Other errors Table 6.
1	FLAGS	uint32	*Reserved* and must be `0`.
2	TRANSITION_LATENCY	uint32	Worst case transition latency of domain from one power state to another, in microseconds (us).
3:6	DOMAIN_NAME	uint8[16]	Device power domain name, a NULL-terminated ASCII string up to 16-bytes.

4.9.6. Service: DPWR_SET_STATE (SERVICE_ID: 0x04)

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

Table 133. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Device power domain ID.
1	POWER_STATE	uint32	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. Refer Section 4.9.1 for more details.

Table 134. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code / Description: RPMI_SUCCESS — Service completed successfully. RPMI_ERR_INVALID_PARAM — `DOMAIN_ID` or `POWER_STATE` is invalid. RPMI_ERR_DENIED — Denied due to no permission. RPMI_ERR_HW_FAULT — Failed due to hardware error. ● Other errors Table 6.

4.9.7. Service: DPWR_GET_STATE (SERVICE_ID: 0x05)

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

Table 135. Request Data

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

Table 136. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code: RPMI_SUCCESS — Description: Service completed successfully. Error Code: RPMI_ERR_INVALID_PARAM — Description: `DOMAIN_ID` is invalid. ● Other errors Table 6.
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. Refer Section 4.9.1 for more details.

4.10. Service Group - PERFORMANCE (SERVICEGROUP_ID: 0x000A)

This PERFORMANCE service group is used to manage the performance of a group of devices or application processors that operate in the same performance domain. Unlike traditional performance control mechanisms, where the OS is responsible for directly controlling voltages 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 entirely platform-dependent. Values on this scale are represented with `performance level index`, and the platform has complete control over mapping these performance operating points to performance states, which are eventually converted into hardware parameters such as voltage and frequency. The level index does not need to be contiguous or to be on a linear scale. For example, the mapping between levels index and frequencies can be as straightforward as using a multiplication factor of `1000` or ascending index number starting from `0`.

The CPPC service group is designed for performance control, but it is only intended for application processors. This service group is primarily meant for devices such as GPUs and accelerators, though it can also be used for application processors.

It is important to distinguish between performance domains and power domains. A performance domain refers to a set of devices that must always operate at the same performance level, whereas a power domain refers to a set of devices that can be turned on or off together for power management purposes.

The following table lists the services in the PERFORMANCE service group:

Table 137. PERFORMANCE Services

Service ID	Service Name	Request Type
0x01	PERF_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	PERF_GET_NUM_DOMAINS	NORMAL_REQUEST
0x03	PERF_GET_ATTRIBUTES	NORMAL_REQUEST
0x04	PERF_GET_SUPPORTED_LEVELS	NORMAL_REQUEST
0x05	PERF_GET_LEVEL	NORMAL_REQUEST
0x06	PERF_SET_LEVEL	NORMAL_REQUEST
0x07	PERF_GET_LIMIT	NORMAL_REQUEST

Service ID	Service Name	Request Type
0x08	PERF_SET_LIMIT	NORMAL_REQUEST
0x09	PERF_GET_FAST_CHANNEL_REGION	NORMAL_REQUEST
0x0A	PERF_GET_FAST_CHANNEL_ATTRIBUTES	NORMAL_REQUEST

4.10.1. Performance Level Attribute

The following table shows the structure of a single performance level and its' attribute.

Table 138. Performance Level Attributes Structure

Word	Name	Description
0	INDEX	Performance Level Index
1	CLOCK_FREQ	Clock frequency (kHz).
2	POWER_COST	Power cost (uW).
3	TRANSITION_LATENCY	Transition latency (us).

4.10.2. Performance Fast-channel

This section provides an overview of the properties associated with the fast-channel for PERFORMANCE service group.

Supported Services

The fast-channel currently supports only the following PERFORMANCE services:

- PERF_GET_LEVEL
- PERF_SET_LEVEL
- PERF_GET_LIMIT
- PERF_SET_LIMIT

Platform Dependency

- Not all performance domains or performance services are required to support fast-channel functionality.
- Support for fast-channel depends on the platform implementation.

Performance Fast-channel Shared Memory Region

- In the memory region designated by the platform for fast-channels, it is essential that the Performance fast-channels are organized in a continuous memory block.
- The shared memory region designated for fast-channels across performance service group must be a `power-of-two` in size. The base address and size (in bytes) of this shared memory region can be obtained through the service `PERF_GET_FAST_CHANNEL_REGION`. The base address of the shared memory region must be aligned to 8 bytes.

Discovering Fast-channel

- Fast-channels support are discoverable through PERFORMANCE service calls.
- To determine if a platform supports fast-channel for a specific performance domain, use the `PERF_GET_ATTRIBUTES` service call.
- If fast-channel support is available, retrieve fast-channel attributes for specific PERFORMANCE service call using the `PERF_GET_FAST_CHANNEL_ATTRIBUTES` service call.

- The `PERF_GET_FAST_CHANNEL_REGION` provides physical memory for Performance Service Group. The offset of the physical address retrieve in `PERF_GET_FAST_CHANNEL_ATTRIBUTES` of the 'Performance Domain / Service ID' paired is based on the starting address in `PERF_GET_FAST_CHANNEL_REGION` service.

Doorbell Support

- The doorbell, if supported must be a memory mapped register with write access.
- The doorbell details such as doorbell register address and write value can be discovered by the application processor through the `PERF_GET_FAST_CHANNEL_ATTRIBUTES` service.
- The doorbell register address is the physical address of the register. The doorbell write value is the value which must be written in the doorbell register to trigger the doorbell interrupt. The width of the doorbell write value must be equal to the doorbell register width.

```
NOTE: The write value may also contains other set bits which must persist
on every write to the doorbell register.
```

- Doorbell support is not available for `PERF_GET_LEVEL` and `PERF_GET_LIMIT` service calls.
- When fast-channels are implemented for `PERF_GET_LEVEL` and `PERF_GET_LIMIT` service calls, the last known valid performance level or performance limits are always accessible via the fast-channel without requiring a doorbell trigger.
- For other PERFORMANCE service calls that support fast-channel, doorbell support is optional.

Payload Requirements

- The payload of a fast-channel should exclusively include message specific parameters and exclude the `DOMAIN_ID`. Since a fast-channel is specific to both `DOMAIN_ID` and `SERVICE_ID`, there is no need to include `DOMAIN_ID` or any other channel specific and message specific headers when using a fast-channel. For instance, the payload of the `PERF_SET_LIMIT` message should consist of a 32-bit `MAX_PERF_LEVEL` and a 32-bit `MIN_PERF_LEVEL`.

4.10.3. 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 microcontroller to inform clients in the system about changes in the performance domain.

Table 139. Performance Notification Events

Event ID	Name	Event Data			Description
		Word	Type	Description	
0x01	PERF_POWER_CHANGE	0	uint32	Performance domain ID for which the power has changed.	Performance power changed notification.
		1	uint32	New power value (uW).	

Event ID	Name	Event Data			Description
		Word	Type	Description	
0x02	PERF_LIMIT_CHANGE	0	uint32	Performance domain ID for which the performance limit has changed.	Performance limit changed notification.
		1	uint32	New maximum performance level.	
		2	uint32	New minimum performance level.	
		Word	Type	Description	
0x03	PERF_LEVEL_CHANGE	0	uint32	Performance domain ID for which the performance level has changed.	Performance level changed notification.
		1	uint32	New performance level.	

4.10.4. Service: PERF_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `PERFORMANCE` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.10.3.

Table 140. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 141. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | Event is subscribed successfully. RPMI_ERR_INVALID_PARAM | `EVENT_ID` or `REQ_STATE` is invalid. RPMI_ERR_NOT_SUPPORTED | Notification for the `EVENT_ID` is not supported. ● Other errors Table 6.

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.10.5. Service: PERF_GET_NUM_DOMAINS (SERVICE_ID: 0x02)

This service returns the number of performance domains supported by the system. The number of performance domains may vary depending on the hardware platform and its 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.

Table 142. Request Data

NA

Table 143. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | Service completed successfully. ● Other errors Table 6.
1	NUM_DOMAINS	uint32	Number of performance domains.

4.10.6. Service: PERF_GET_ATTRIBUTES (SERVICE_ID: 0x03)

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.

Table 144. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Performance domain ID.

Table 145. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.

Error Code	Description
RPMI_SUCCESS	Service completed successfully.
RPMI_ERR_INVALID_PARAM	`DOMAIN_ID` is invalid.

- Other errors Table 6.

Word	Name	Type	Description

1	FLAGS	uint32	**Bits** / **Description**
			[31:3] — *Reserved* and must be `0`.
			[2] — Performance limit change support.
			This attribute indicates whether the platform allows software to set the performance limit for a specific performance domain.
			`0b0: Performance limit change is not allowed.`
			`0b1: Performance limit change is allowed.`
			[1] — Performance level change support.
			This attribute indicates whether the platform allows software to set the performance level for a specific performance domain.
			`0b0: Performance level change is not allowed.`
			`0b1: Performance level change is allowed.`
			[0] — Fast-channel support.
			This attribute indicates whether the platform supports fast-channel for a specific performance domain.
			`0b0: Fast-channel is not supported.`
			`0b1: Fast-channel is supported.`

2	NUM_LEVELS	uint32	The total number of supported performance levels.
3	TRANSITION_LATENCY	uint32	Minimum amount of time that needs to pass between two consecutive requests, in microseconds (us).
4:7	DOMAIN_NAME	uint8[16]	Performance domain name, a NULL-terminated ASCII string up to 16-bytes.

4.10.7. Service: PERF_GET_SUPPORTED_LEVELS (SERVICE_ID: 0x04)

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 using this information, the OS can select the optimal performance level based on the system's workload and power constraints using `performance level index` returned in this service.

```
/* Pseudocode to retrieve the list of the supported performance levels. */

index = 0;
num = 0;
/* Allocate a buffer based on the value returned from the NUM_LEVELS */
total_num_levels = perf_domain_attributes.num_levels;

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

    for (i = 0; i < list.returned; i++, num++) {
        opp[num].index = list.entry[entry_num++];
        opp[num].freq = list.entry[entry_num++];
        opp[num].power = list.entry[entry_num++];
        opp[num].transition_latency = 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 `NUM_LEVELS` parameter returned by the `PERF_GET_ATTRIBUTES` service.

The total number of performance levels included in one message must not exceed the available word count in the message's `DATA` field. If the performance levels exceed this limit, the platform microcontroller will return the number of levels that can be accommodated in one message and set the `REMAINING` field accordingly. When the `REMAINING` field is not zero, the application processor must call this service again with the appropriate `PERF_LEVEL_INDEX` to retrieve the remaining levels. Multiple service calls may be required to obtain all the levels.

Table 146. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Performance domain ID.
1	PERF_LEVEL_INDEX	uint32	Index of performance data array. The first index starts at zero.

Table 147. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code / Description: RPMI_SUCCESS: Service completed successfully. RPMI_ERR_INVALID_PARAM: `DOMAIN_ID` or `PERF_LEVEL_INDEX` is invalid. ● Other errors Table 6.
1	FLAGS	uint32	*Reserved* and must be `0`.
2	REMAINING	uint32	Remaining number of levels (number of arrays).
3	RETURNED	uint32	Number of levels returned (number of arrays).
4	LEVEL[]	uint32[4]	List of performance levels attributes. Refer to [section-perf-attribute] for the complete structure of performance level attributes.

4.10.8. Service: PERF_GET_LEVEL (SERVICE_ID: 0x05)

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

Table 148. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Performance domain ID.

Table 149. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code / Description: RPMI_SUCCESS: Service completed successfully. RPMI_ERR_INVALID_PARAM: `DOMAIN_ID` is invalid. ● Other errors Table 6.
1	LEVEL	uint32	Current performance level index of the domain.

4.10.9. Service: PERF_SET_LEVEL (SERVICE_ID: 0x06)

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

Table 150. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Performance domain ID.
1	LEVEL	uint32	Performance level index.

Table 151. Response Data

<table>
<tr><th>Word</th><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>0</td><td>STATUS</td><td>int32</td><td>Return error code.
<table>
<tr><th>Error Code</th><th>Description</th></tr>
<tr><td>RPMI_SUCCESS</td><td>Service completed successfully.</td></tr>
<tr><td>RPMI_ERR_INVALID_PARAM</td><td><code>DOMAIN_ID</code> or <code>LEVEL</code> is invalid.</td></tr>
<tr><td>RPMI_ERR_DENIED</td><td>Denied due to no permission.</td></tr>
<tr><td>RPMI_ERR_HW_FAULT</td><td>Operation failed due to hardware error.</td></tr>
</table>
<ul><li>Other errors Table 6.</li></ul></td></tr>
</table>

4.10.10. Service: PERF_GET_LIMIT (SERVICE_ID: 0x07)

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

Table 152. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Performance domain ID.

Table 153. Response Data

<table>
<tr><th>Word</th><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>0</td><td>STATUS</td><td>int32</td><td>Return error code.
<table>
<tr><th>Error Code</th><th>Description</th></tr>
<tr><td>RPMI_SUCCESS</td><td>Service completed successfully.</td></tr>
<tr><td>RPMI_ERR_INVALID_PARAM</td><td><code>DOMAIN_ID</code> is invalid.</td></tr>
</table>
<ul><li>Other errors Table 6.</li></ul></td></tr>
<tr><td>1</td><td>MAX_PERF_LEVEL</td><td>uint32</td><td>Maximum allowed performance level index.</td></tr>
<tr><td>2</td><td>MIN_PERF_LEVEL</td><td>uint32</td><td>Minimum allowed performance level index.</td></tr>
</table>

4.10.11. Service: PERF_SET_LIMIT (SERVICE_ID: 0x08)

This service is used to set the performance limit of a specific performance domain in the system. The platform must ensure that any subsequent calls to `PERF_SET_LEVEL` to adjust the performance level remain within the currently defined limits.

If the current performance level falls outside the newly defined minimum or maximum ranges, the platform will automatically adjust it to comply with the updated limits.

Examples:

- *If the current performance level is below the new minimum limit, the platform will set it to the new minimum limit.*

- *If the current performance level exceeds the new maximum limit, the platform will set it to the new maximum limit.*
- *No adjustment is required if the current performance level is within the new limits.*

If notifications are enabled, the platform will send an appropriate notification (e.g., `PERF_LEVEL_CHANGE`, `PERF_POWER_CHANGE`, etc.) to the application processor.

Table 154. Request Data

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

Table 155. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.

Error Code	Description
RPMI_SUCCESS	Service completed successfully.
RPMI_ERR_INVALID_PARAM	`DOMAIN_ID` or performance level is invalid.
RPMI_ERR_NOT_SUPPORTED	Performance limit change is not allowed.
RPMI_ERR_DENIED	Denied due to no permission.
RPMI_ERR_HW_FAULT	Operation failed due to hardware error.

- Other errors Table 6.

4.10.12. Service: PERF_GET_FAST_CHANNEL_REGION (SERVICE_ID: 0x09)

This service retrieves the physical address of the fast-channel region used in the performance service group. The fast-channel region is grouped in a continuous block of memory to ease the configuration of memory region protection.

Table 156. Request Data

NA

Table 157. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code

Error Code	Description
RPMI_SUCCESS	Service completed successfully.
RPMI_ERR_NOT_SUPPORTED	Fast-channel is not implemented.

- Other errors Table 6

1	REGION_PHYS_ADDR_LOW	uint32	Lower 32-bit of the fast-channels shared memory region physical address.
2	REGION_PHYS_ADDR_HIGH	uint32	Upper 32-bit of the fast-channels shared memory region physical address.
3	REGION_SIZE_LOW	uint32	Lower 32-bit of the fast-channels shared memory region size.
4	REGION_SIZE_HIGH	uint32	Upper 32-bit of the fast-channels shared memory region size.

4.10.13. Service: PERF_GET_FAST_CHANNEL_ATTRIBUTES (SERVICE_ID: 0x0A)

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

Table 158. Request Data

Word	Name	Type	Description
0	DOMAIN_ID	uint32	Performance domain ID.
1	SERVICE_ID	uint32	Performance Service ID. Refer service ID in Table 137.

Table 159. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.

Error Code	Description
RPMI_SUCCESS	Service completed successfully.
RPMI_ERR_INVALID_PARAM	`DOMAIN_ID` is invalid.
RPMI_ERR_NOT_SUPPORTED	Fast-channel is not implemented.

- Other errors Table 6.

Word	Name	Type	Description

1	FLAGS	uint32	**Bits** / **Description**
			[31:3]: *Reserved* and must be `0`.
			[2:1]: Doorbell register width. This field is unused if doorbell is not supported.
			`0b00: 8-bit.`
			`0b01: 16-bit.`
			`0b10: 32-bit.`
			`0b11: Reserved.`
			[0]: Doorbell support.
			`0b0: Doorbell is not supported.`
			`0b1: Doorbell is supported.`

2	FASTCHAN_OFFSET_LOW	uint32	Lower 32-bit offset of fast-channel physical address region.
3	FASTCHAN_OFFSET_HIGH	uint32	Upper 32-bit offset of fast-channel physical address region.
4	FASTCHAN_SIZE	uint32	The size of fast-channel physical address in bytes.
5	DB_ADDR_LOW	uint32	Lower 32-bit of doorbell register address for Performance Request fast-channel. This field is unused if the doorbell is not supported.
6	DB_ADDR_HIGH	uint32	Upper 32-bit of doorbell register address for Performance Request fast-channel. This field is unused if the doorbell is not supported.
7	DB_WRITE_VALUE	uint32	32-bit doorbell write value for Performance Request fast-channel. If the doorbell register width is less than 32-bit, the lower bits in this field equal to the doorbell register width must be used as write value.

4.11. Service Group - MANAGEMENT_MODE (SERVICEGROUP_ID: 0x000B)

This MANAGEMENT_MODE service group provides RPMI client a mechanism to invoke the Management Mode (MM) in a secure execution environment. For general background on Management Mode, refer to the Platform Initialization (PI) specifications [4], Volume 4: Management Mode Core Interface.

The Management Mode (MM) provides an environment for implementing OS agnostic MM services such as secure variable storage, and firmware updates in the platform firmware. The MANAGEMENT_MODE service group defines RPMI services for invoking an MM service synchronously where the `MM_COMMUNICATE` RPMI service is used as a synchronous call from the non-secure world to the secure world and the data exchanged with the MM service is passed via special Management Mode (MM) shared memory.

The following table lists the services in the MANAGEMENT_MODE service group:

Table 160. MANAGEMENT_MODE Services

Service ID	Service Name	Request Type
0x01	MM_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	MM_GET_ATTRIBUTES	NORMAL_REQUEST
0x03	MM_COMMUNICATE	NORMAL_REQUEST

4.11.1. Notifications

This service group does not support any events for notification.

4.11.2. Service: MM_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `MANAGEMENT_MODE` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.11.1.

Table 161. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 162. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. **Error Code** / **Description**: RPMI_SUCCESS — Event is subscribed successfully. RPMI_ERR_INVALID_PARAM — `EVENT_ID` or `REQ_STATE` is invalid. RPMI_ERR_NOT_SUPPORTED — Notification for the `EVENT_ID` is not supported. ● Other errors Table 6.

Word	Name	Type	Description

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.11.3. Service: MM_GET_ATTRIBUTES (SERVICE_ID: 0x02)

This RPMI service gets the attributes about Management Mode, including MM version, MM shared memory location, etc.

Table 163. Request Data

NA

Table 164. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code: RPMI_SUCCESS — Description: Service completed successfully. ● Other errors Table 6.
1	MM_VERSION	uint32	Management Mode version. Bits [31:16] — Major version. Bits [15:0] — Minor version.
2	MM_SHMEM_ADDR_LOW	uint32	Lower 32-bit of the MM shared memory physical address.
3	MM_SHMEM_ADDR_HIGH	uint32	Upper 32-bit of the MM shared memory physical address.
4	MM_SHMEM_SIZE	uint32	The size of MM shared memory in bytes.

4.11.4. Service: MM_COMMUNICATE (SERVICE_ID: 0x03)

The `MM_COMMUNICATE` service invokes an MM service implemented in the secure execution environment. The input data needed to identify and invoke the MM service is referred to as `MM_COMM_INPUT_DATA` whereas the output data returned by the MM service is referred to as `MM_COMM_OUTPUT_DATA`. The RPMI client in the non-secure execution environment provides the location of `MM_COMM_INPUT_DATA` and `MM_COMM_OUTPUT_DATA` in the MM shared memory as parameters of `MM_COMMUNICATE` service.

Table 165. Request Data

Word	Name	Type	Description
0	MM_COMM_INPUT_DATA_OFFSET	uint32	The offset in the MM shared memory where the input data is passed to the MM service.
1	MM_COMM_INPUT_DATA_SIZE	uint32	The size of the input data in the MM shared memory.
2	MM_COMM_OUTPUT_DATA_OFFSET	uint32	The offset in the MM shared memory where the output data will be written by the MM service.
3	MM_COMM_OUTPUT_DATA_SIZE	uint32	The maximum size of the output data which can be written by the MM service in the MM shared memory.

Table 166. Response Data

Word	Name	Type	Description	
0	STATUS	int32	Return error code.	
			Error Code	**Description**
			RPMI_SUCCESS	Service completed successfully.
			RPMI_ERR_INVALID_ADDR	Input data end (or Output data end) is outside MM shared memory.
			RPMI_ERR_DENIED	Denied due to no permission.
			• Other errors Table 6.	
1	MM_COMM_RETURN_DATA_SIZE	uint32	Actual size of the output data written by the MM service in the MM shared memory.	

4.12. Service Group - RAS_AGENT (SERVICEGROUP_ID: 0x000C)

The RAS_AGENT service group provides services to enumerate various error sources in a system and to retrieve their descriptors.

Each error source in a system is assigned a unique 32-bit identification number, referred to as `RAS_ERR_SRC_ID`.

The following table lists the services in the RAS_AGENT service group:

Table 167. RAS Agent Services

Service ID	Service Name	Request Type
0x01	RAS_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	RAS_GET_NUM_ERR_SRCS	NORMAL_REQUEST
0x03	RAS_GET_ERR_SRCS_ID_LIST	NORMAL_REQUEST
0x04	RAS_GET_ERR_SRC_DESC	NORMAL_REQUEST

4.12.1. Error Source Descriptor Format

ACPI Systems

For systems that support ACPI/APEI, the format of the error source descriptor is as defined in ACPI specification v6.4 or above, (GHESv2) [3]. If the value of `RAS_GET_ERR_SRC_DESC.FLAGS[3:0]` is `0`, it indicates that the error source descriptor format is GHESv2.

The RAS agent populates the error source descriptor fields according to the error source specified by `RAS_ERR_SRC_ID`.

The error source descriptor has an `error_status_structure` *field which is a generic address structure (*`GAS`*) as defined in ACPI v6.4 (GHESv2) [3]. This field specifies the location of a register that contains the physical address of a block of memory that holds the error status data for the specified error source. This block of memory is referred to as* `error_status_block`*. The allocation of* `error_status_block` *is platform dependent and is done by the RAS agent. The physical address of* `error_status_block` *is stored in the* `error_status_structure` *field of the error source descriptor being returned.*

Non-ACPI Systems

RAS is not standardized for non-ACPI systems. Such systems may define custom format for an error source descriptor. The type of custom error source descriptor format can be read from `RAS_GET_ERR_SRC_DESC.FLAGS[3:0]`.

4.12.2. Notifications

This service group does not support any events for notification.

4.12.3. Service: RAS_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `RAS_AGENT` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.12.2.

Table 168. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 169. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | Event is subscribed successfully. RPMI_ERR_INVALID_PARAM | `EVENT_ID` or `REQ_STATE` is invalid. RPMI_ERR_NOT_SUPPORTED | Notification for the `EVENT_ID` is not supported. ● Other errors Table 6.

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.12.4. Service: RAS_GET_NUM_ERR_SRCS (SERVICE_ID: 0x02)

This service queries number of error sources available in the system.

Table 170. Request Data

NA

Table 171. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code | Description RPMI_SUCCESS | Service completed successfully and number of error sources returned as `NUM_ERR_SRCS`. ● Other errors Table 6.
1	NUM_ERR_SRCS	uint32	Number of error sources.

4.12.5. Service: RAS_GET_ERR_SRCS_ID_LIST (SERVICE_ID: 0x03)

This service returns a list of `RAS_ERR_SRC_ID` for all error sources present in the system. The `RAS_ERR_SRC_ID` can be used to get the associated descriptor of the error source.

Table 172. Request Data

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

Table 173. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.
			Error Code | Description
			RPMI_SUCCESS | Service completed successfully and list of error sources returned.
			RPMI_ERR_INVALID_PARAM | `START_INDEX` is invalid.
			• Other errors Table 6.
1	FLAGS	uint32	*Reserved* and must be `0`.
2	REMAINING	uint32	Remaining number of error source IDs.
3	RETURNED	uint32	Number of error source IDs returned in this request.
4	RAS_ERR_SRC_ID[N]	uint32	An array of error source IDs where each entry in the array is a unique error source ID. N is equal to `RETURNED` number of error source IDs in this request.

4.12.6. Service: RAS_GET_ERR_SRC_DESC (SERVICE_ID: 0x04)

This service retrieves the error source descriptor of an error source specified by `RAS_ERR_SRC_ID`.

Word	Name	Type	Description
0	RAS_ERR_SRC_ID	uint32	Error source ID for which attributes are to be returned.
1	BYTE_OFFSET	uint32	Offset from which the descriptor is to be read. Offset `0` for the first call, subsequent byte offset of the remaining bytes.

Table 174. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code.
			Error Code | Description
			RPMI_SUCCESS | Service completed successfully and partial/complete error source descriptor returned.
			RPMI_ERR_INVALID_PARAM | `RAS_ERR_SRC_ID` or `BYTE_OFFSET` is invalid.
			• Other errors Table 6.

Word	Name	Type	Description
1	FLAGS	uint32	**Bits** / **Description**: [31:4] *Reserved* and must be `0`. [3:0] Format of the error source descriptor. `0b0000: GHESv2 format.` `0b0001 - 0b1110: Reserved.` `0b1111: Implementation specific.`
2	REMAINING	uint32	Remaining number of bytes to be read.
3	RETURNED	uint32	Number of bytes read in this request.
4	ERR_SRC_DESC[N]	uint8	Full or partial descriptor N is equal to the `RETURNED` bytes in this request.

4.13. Service Group - REQUEST_FORWARD (SERVICEGROUP_ID: 0x000D)

The REQUEST_FORWARD service group allows application processors to retrieve and process RPMI request messages which are forwarded by platform microcontroller from some other RPMI client. This service group also allows an SBI implementation to forward RPMI request messages from one system-level partition (or domain) to another using the SBI MPXY extension [1].

The platform microcontroller (or SBI implementation) should maintain a first-in first-out queue of forwarded RPMI request messages. The first (or oldest) forwarded RPMI request message in the queue is referred to as the current forwarded RPMI request message. The RPMI services defined by the REQUEST_FORWARD service group allow application processors to retrieve and process one forwarded RPMI request message at a time.

The Table 175 below lists the services defined by the REQUEST_FORWARD service group:

Table 175. Request Forward Services

Service ID	Service Name	Request Type
0x01	REQFWD_ENABLE_NOTIFICATION	NORMAL_REQUEST
0x02	REQFWD_RETRIEVE_CURRENT_MESSAGE	NORMAL_REQUEST
0x03	REQFWD_COMPLETE_CURRENT_MESSAGE	NORMAL_REQUEST

4.13.1. Notifications

The Table 176 below lists the notification events defined by the REQUEST_FORWARD service group.

Table 176. Request Forward Notification Events

Event ID	Name	Event Data	Description
0x01	REQFWD_NEW_MESSAGE	An array of `N` bytes representing the first `N` bytes of the current forwarded RPMI request message. The value `N` is specified by the `EVENT_DATALEN` field of the RPMI notification event as shown in Table 5.	This RPMI notification event represents the arrival of a new forwarded RPMI request message when there were no other pending forwarded RPMI request message in the queue.

4.13.2. Service: REQFWD_ENABLE_NOTIFICATION (SERVICE_ID: 0x01)

This service allows the application processor to subscribe to `REQUEST_FORWARD` service group notifications. The platform may optionally support notifications for events that may occur. The platform microcontroller can send these notification messages to the application processor if they are implemented and the application processor has subscribed to them. The supported events are described in Section 4.13.1.

Table 177. Request Data

Word	Name	Type	Description
0	EVENT_ID	uint32	The event to be subscribed for notification.

1	REQ_STATE	uint32	Requested event notification state.
			Change or query the current state of `EVENT_ID` notification.
			`0: Disable.`
			`1: Enable.`
			`2: Return current state.`
			Any other values of `REQ_STATE` field other than the defined ones are reserved for future use.

Table 178. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. <table><tr><th>Error Code</th><th>Description</th></tr><tr><td>RPMI_SUCCESS</td><td>Event is subscribed successfully.</td></tr><tr><td>RPMI_ERR_INVALID_PARAM</td><td><code>EVENT_ID</code> or <code>REQ_STATE</code> is invalid.</td></tr><tr><td>RPMI_ERR_NOT_SUPPORTED</td><td>Notification for the <code>EVENT_ID</code> is not supported.</td></tr></table>● Other errors Table 6.

1	CURRENT_STATE	uint32	Current `EVENT_ID` notification state.
			`0: Notification is disabled.`
			`1: Notification is enabled.`
			In case of `REQ_STATE = 0` or `1`, the `CURRENT_STATE` will return the requested state.
			In case of an error, the value of `CURRENT_STATE` is unspecified.

4.13.3. Service: REQFWD_RETRIEVE_CURRENT_MESSAGE (SERVICE_ID: 0x02)

This service allows application processors to retrieve the current forwarded RPMI request message. The current message may be the oldest forwarded RPMI request message in the platform microcontroller (or SBI implementation) queue.

Table 179. Request Data

Word	Name	Type	Description
0	START_INDEX	uint32	Starting index of first byte of the current forwarded RPMI request message. Use `0` for the first call, subsequent calls will use the next index of the remaining bytes.

Table 180. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. Error Code / Description: RPMI_SUCCESS — Service completed successfully. RPMI_ERR_INVALID_PARAM — `START_INDEX` is invalid. RPMI_ERR_NO_DATA — No forwarded RPMI request message available. • Other errors Table 6.
1	REMAINING	uint32	Remaining number of bytes in the current forwarded RPMI request message.
2	RETURNED	uint32	Number of bytes `N` of the current forwarded RPMI request message returned in this request.
3	REQUEST_MESSAGE[N]	uint8	An array of `N` bytes representing a part of the current forwarded RPMI request message at byte offset specified by `START_INDEX`.

4.13.4. Service: REQFWD_COMPLETE_CURRENT_MESSAGE (SERVICE_ID: 0x03)

This service allows the application processors to inform the platform microcontroller (or SBI implementation) that:

- The processing of the current forwarded RPMI request message has completed and RPMI response message must be sent to the original source of RPMI request message.
- The current forwarded RPMI request message must now point to the next forwarded RPMI request message if available.

If the service is called without retrieving the message, an error is returned. The service also returns `NUM_MESSAGES`, which is the number of forwarded messages available for retrieval from the platform microcontroller, excluding the current forwarded message.

Table 181. Request Data

Word	Name	Type	Description
0	RESPONSE_DATA[N]	uint8	An array of bytes representing the RPMI message data to be send as response data for the current forwarded RPMI request message. The `N` represents the total number of bytes in the response data which can be inferred by the platform microcontroller (or SBI implementation) from the overall size of the `REQFWD_COMPLETE_CURRENT_MESSAGE` service message.

Table 182. Response Data

Word	Name	Type	Description
0	STATUS	int32	Return error code. | Error Code | Description | |---|---| | RPMI_SUCCESS | Service completed successfully. | | RPMI_ERR_NO_DATA | No forwarded request message retrieved. | ● Other errors Table 6.
1	NUM_MESSAGES	uint32	Number of forwarded messages available for retrieval, excluding the current forwarded message.

Chapter 5. Integration with SBI MPXY Extension

A platform with a limited number of RPMI transport instances can share an M-mode RPMI transport instance with the supervisor software using the SBI MPXY extension [1]. An M-mode firmware or hypervisor can also virtualize RPMI message communication for the supervisor software using the SBI MPXY extension. As shown in the Figure 11 below, the SBI implementation acts as a **RPMI proxy** for the supervisor software when sending RPMI messages through an SBI MPXY channel.

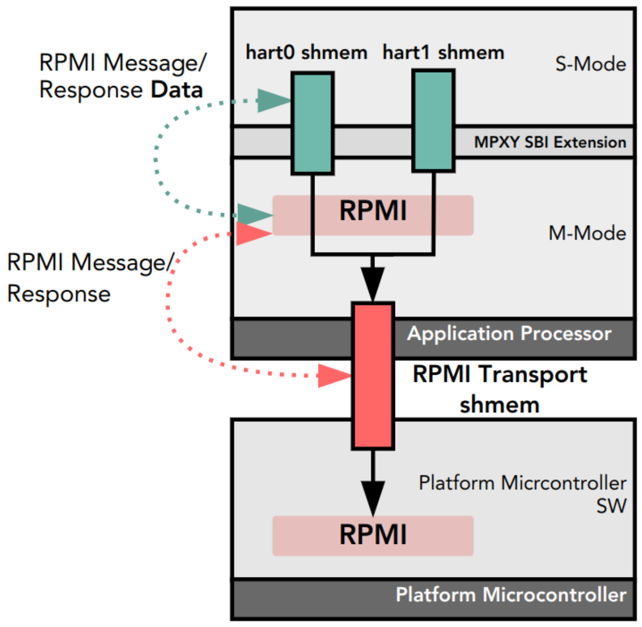

Figure 11. RPMI and SBI MPXY Integration

The RPMI communication via the SBI MPXY extension must satisfy the following requirements:

1. The SBI MPXY channel must correspond to a single RPMI service group which is allowed in S-mode except BASE and CPPC service groups. The Table 7 list the RPMI service groups allowed in S-mode.
2. The SBI MPXY channel corresponding to the RPMI SYSTEM_MSI service group must not support the P2A doorbell system MSI.
3. The `message_id` parameter passed to the `sbi_mpxy_send_message_with_response()` and `sbi_mpxy_send_message_without_response()` must represent the `SERVICE_ID` of an RPMI service belonging to the RPMI service group bound to the SBI MPXY channel.
4. The format of the message data passed via the SBI MPXY shared memory to the `sbi_mpxy_send_message_with_response()` and `sbi_mpxy_send_message_without_response()` must match the request data format of the RPMI service represented by the `message_id` parameter.
5. The format of the response message data returned in the SBI MPXY shared memory by the `sbi_mpxy_send_message_with_response()` must match the response data format of the RPMI service represented by the `message_id` parameter.
6. The format of the protocol-specific data returned in the SBI MPXY shared memory by the `sbi_mpxy_get_notification_events()` must match the RPMI notifications message data format.
7. The SBI MPXY channel must support message protocol attributes listed in the Table 183 below.

Table 183. RPMI Message Protocol Attributes of an SBI MPXY Channel

Attribute Name	Attribute ID	Access	Description
SERVICEGROUP_ID	0x80000000	RO	RPMI service group ID.
SERVICEGROUP_VERSION	0x80000001	RO	RPMI service group version.
IMPLEMENTATION_ID	0x80000002	RO	RPMI implementation ID.
IMPLEMENTATION_VERSION	0x80000003	RO	RPMI implementation version.

Bibliography

[1] “RISC-V Supervisor Binary Interface Specification v3.0.” [Online]. Available: github.com/riscv-non-isa/riscv-sbi-doc.

[2] “libRPMI.” [Online]. Available: github.com/riscv-software-src/librpmi.

[3] “Advanced Configuration and Power Interface Specification v6.6.” [Online]. Available: uefi.org/specifications.

[4] “UEFI Platform Initialization Specification.” [Online]. Available: uefi.org/specifications.