**Document Identifier: DSP0282**

**Date: 2024-10-09**

**Version: 1.0.1**

# Memory-Mapped Buffer Interface (MMBI) Specification

**Supersedes: 1.0.0**

**Document Class: Normative**

**Document Status: Published**

**Document Language: en-US**

# CONTENTS

# Figures

# Tables

# Foreword

The *Memory-Mapped Buffer Interface (MMBI) Specification* (DSP0282) was prepared by DMTF's PMCI Working Group.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. For information about DMTF, visit dmtf.org.

This version supersedes version 1.0.0. For a list of changes, see the change log in ANNEX B.

## Acknowledgments

DMTF acknowledges the following individuals for their contributions to this document:

Editors:

- Janusz Jurski – Intel Corporation
- Richard Marian Thomaiyar – Intel Corporation

DMTF Contributors:

- Rama Bisa – Dell Inc.
- Patrick Caporale – Lenovo
- Samer El-Haj-Mahmoud – ARM Inc.
- Ted Emerson – Hewlett Packard Enterprise
- John Guan – Inspur
- Ramesha He – Dell Inc.
- Tiffany Kasanicky – Intel Corporation
- Mahesh Natu – Intel Corporation
- Chandra Nelogal – Dell Inc.
- Edward Newman – Hewlett Packard Enterprise
- Scott Phuong – Cisco
- Derek Roberts – Xilinx Inc.
- William Scherer III – Hewlett Packard Enterprise
- Hemal Shah – Broadcom Inc.
- Bob Stevens – Dell Inc.

# Introduction

The *Memory-Mapped Buffer Interface (MMBI) Specification* defines the mechanisms facilitating communication between platform components, typically host software and a Management Controller (usually a Baseboard Management Controller). Using the shared memory concept, this document defines the MMBI protocol that allows packet exchanges between communicating devices. The described memory mapping makes it possible for both boot code (such as UEFI firmware), as well as OS-level software (such as OS kernel or drivers) to establish efficient communication with a (Baseboard) Management Controller at bandwidth and latency limited by the underlying memory mapping mechanisms. MMBI can also be used to enable communication between other types of platform components, not just host software and a Management Controller (MC) or a Baseboard Management Controller (BMC).

# 1 Scope

This document provides the specifications for the Memory-Mapped Buffer Interface (MMBI). MMBI assumes an underlying memory mapping capability, such as PCIe MMIO/BAR, allowing host software to efficiently access data stored in (B)MC memory. MMBI defines generic packet-based communication mechanism (based on circular buffers), and specific protocols, such as MCTP, should be covered in other documents.

# 2 Normative references

The following referenced documents are indispensable for the application of this document. For dated or versioned references, only the edition cited (including any corrigenda or DMTF update versions) applies. For references without a date or version, the latest published edition of the referenced document (including any corrigenda or DMTF update versions) applies.

DMTF, DSP0236, *Management Component Transport Protocol (MCTP) Base Specification 1.3*, https://www.dmtf.org/standards/published_documents/DSP0236_1.3.pdf

DMTF, DSP0239, *Management Component Transport Protocol (MCTP) IDs and Codes 1.10*, https://www.dmtf.org/standards/published_documents/DSP0239_1.10.pdf

DMTF, DSP0276, *Secured Messages using SPDM over MCTP Binding Specification 1.1.0*, https://www.dmtf.org/standards/published_documents/DSP0276_1.1.0.pdf

DMTF, DSP0284, *Management Component Transport Protocol (MCTP) Memory-Mapped Buffer Interface (MMBI) Transport Binding Specification 1.0*, https://www.dmtf.org/standards/published_documents/DSP0284_1.0.pdf

IANA, *Internet Assigned Numbers Authority – Private Enterprise Numbers (PEN)*, https://www.iana.org/assignments/enterprise-numbers

# 3 Terms and definitions

In this document, some terms have a specific meaning beyond the normal English meaning. Those terms are defined in this clause.

The terms “shall” (“required”), “shall not”, “should” (“recommended”), “should not” (“not recommended”), “may”, “need not” (“not required”), “can” and “cannot” in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 7. The terms in parentheses are alternatives for the preceding term, for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that ISO/IEC Directives, Part 2, Clause 7 specifies additional alternatives. Occurrences of such additional alternatives shall be interpreted in their normal English meaning.

The terms “clause”, “subclause”, “paragraph”, and “annex” in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 6.

The terms “normative” and “informative” in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 3. In this document, clauses, subclauses, or annexes labeled “(informative)” do not contain normative content. Notes and examples are always informative elements.

Refer to *Management Component Transport Protocol (MCTP) Base Specification* for the terms and definitions that are used across the MCTP specifications.

For the purposes of this document, the following terms and definitions apply.

### 3.1
**ACK**

Acknowledge

### 3.2
**B2H**

BMC-to-Host

### 3.3
**BAR**

Base Address Register

### 3.4
**(B)MC**

Baseboard Management Controller – term used interchangeably with Management Controller

### 3.5
**CCT**

Control Command Type

### 3.6
**Destination Device**

Device receiving the MCTP packet over MMBI

### 3.7
**H2B**

Host-to-BMC

### 3.8
**MMBI**

Memory-Mapped Buffer Interface

### 3.9
**MMIO**

Memory-Mapped Input/Output

### 3.10
**NACK**

Not acknowledge

### 3.11
**ROS**

Read-Only Structure

### 3.12
**RWS**

Read-Write Structure

### 3.13
**Source Device**

Device sending the MCTP packet over MMBI

**3.14**
**SPDM**
Security Protocol and Data Model

**3.15**
**VPSCB**
Variable Packet Size Circular Buffer

# 4 Conventions

The conventions described in the following clauses apply to this specification.

## 4.1 Reserved and unassigned values

Unless otherwise specified, any reserved, unspecified, or unassigned values in enumerations or other numeric ranges are reserved for future definition by DMTF.

Unless otherwise specified, numeric or bit fields that are designated as reserved shall be written as 0 (zero) and ignored when read.

## 4.2 Byte ordering

Unless otherwise specified, byte ordering of multi-byte numeric fields or bit fields is “Big Endian” (that is, the lower byte offset holds the most significant byte, and higher offsets hold less-significant bytes).

# 5 Assumptions

## 5.1 Underlying Memory Mapping

The fundamental assumption in this specification is that there exists an underlying platform mechanism allowing efficient memory sharing between the communicating entities (such as a host and a management controller). PCIe MMIO is an example of such a mechanism. This specification defines the packet transfer protocol on top of this assumed memory mapping layer.

Assumptions about the underlying layer are:

1) Memory mapping shall guarantee an error-free lossless channel.
2) The size of atomic operations is at least 4 bytes.
3) The order of operations must be preserved: writes must be visible to the other party in the order they were executed by the sender; reads cannot be prefetched/cached; if interrupts are used, they must also obey the order of operations.

## 5.2 Multiple Instances

This specification has been designed with the assumption that a single MMBI instance will serve communication between the two communicating entities only (typically host software and management controller firmware components) and so the interface is not shared between multiple communicating entities.

Multiple components in the system, e.g., multiple host tenant / software agents communicating to a (B)MC, can be supported using a plurality of MMBI interfaces (each being an independent instance of the

interface), located in different memory locations. Such MMBI instances shall operate independently as shown in Figure 1:

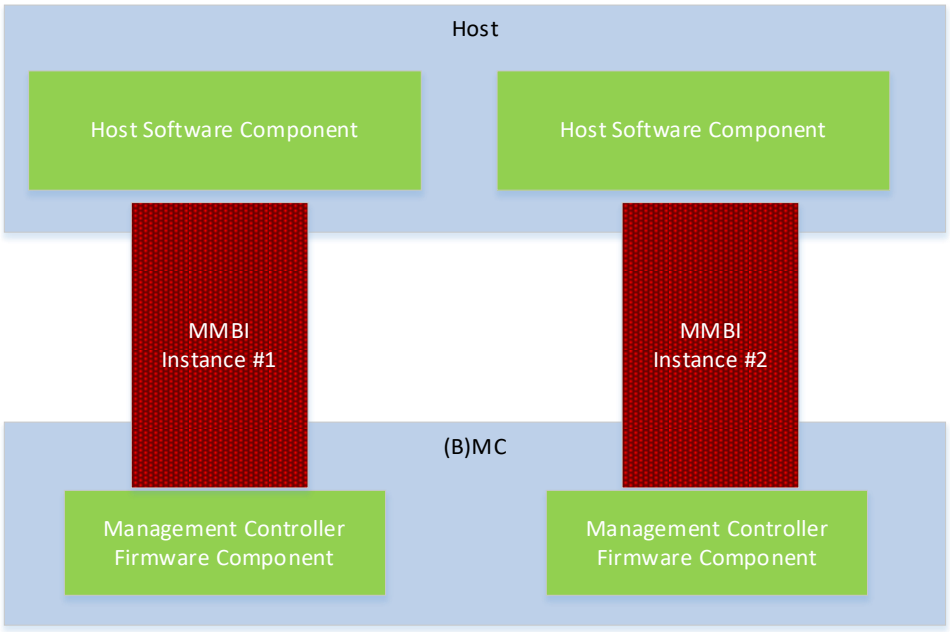



**Figure 1 – Multiple MMBI Instances**

## 5.3 Resets and Errors

MMBI allows lossless communication as well as graceful reset/initialization on request from a communicating party (in case of a reset of a software entity). However, MMBI does not provide guaranteed delivery in case of ungraceful resets of the communicating parties. Applications that care about data loss in such situations shall employ an ACK packet scheme to verify data reception by the other party and handle the error if ACK is not received.

## 5.4 Notifications (Interrupts)

MMBI is designed to execute in both interrupt and polling mode.

The memory sharing capability may be accompanied by the ability to receive interrupts by the communicating software entities. MMBI enables discovery and enables use of the optional interrupt mechanism for efficient data exchange between communicating entities. If interrupts are used, it is assumed that the interrupt delivery mechanism is reliable.

If interrupts are not available, a polling mode can be used. Platform designers can choose polling or interrupt mode, based on their needs.

## 5.5 Packet Sizes, Types, and Packet Flow

MMBI allows variable packet sizes, with the maximum size dependent on the underlying physical layer's memory mapping capabilities. MMBI provides a discovery method allowing the communicating parties to define and discover the circular buffer sizes, which limit the maximum packet sizes that can be transmitted (fragmentation/reassembly is not supported by this version of MMBI protocol). The upper

layers must adhere to the discovered limits and, if necessary, handle fragmentation/reassembly accordingly.

MMBI allows multiple packets (datagrams) to be in-flight. That is, the sender can place more than one packet in the memory buffer even before they are consumed by the receiver. This enables asynchronous operation of the communicating entities. Regardless of the number of packets in-flight, they are guaranteed to arrive to the receiver in the FIFO order (note: upper layer can elect to process in same order or in different order, which will not be guaranteed by the MMBI layer). Note that if multiple instances of MMBI are in the system, they operate independently and no packet ordering guarantees exist between them.

MMBI enables and defines discovery mechanisms to support the exchange of a variety of packet protocol types, such as MCTP. Binding of these protocols to MMBI is defined in separate documents, such as *Management Component Transport Protocol (MCTP) Memory-Mapped Buffer Interface (MMBI) Transport Binding Specification*.

## 5.6 Security

MMBI does not provide any security guarantees. Any authentication, integrity protection, and/or encryption is to be implemented by the other layers of the protocol stack. For example, for secure implementation of communication between the host and (B)MC using MMBI, *Secured Messages using SPDM over MCTP Binding Specification* can be used. Another alternative can be host-based memory protection mechanisms.

# 6 Basic Architecture Concept

The host and the (B)MC use circular buffers to exchange data. One buffer is used to send data from the host to the (B)MC and is referred to as H2B (Host-to-BMC). The other buffer is used for communication in the opposite direction and is referred to as B2H (BMC-to-Host). The buffers are used to store packet data, and they are accompanied by a descriptor structure. The descriptor is a data structure in the shared memory used to store important capabilities and control information. These data structures are shown in Figure 2 and are defined in detail in section 7.

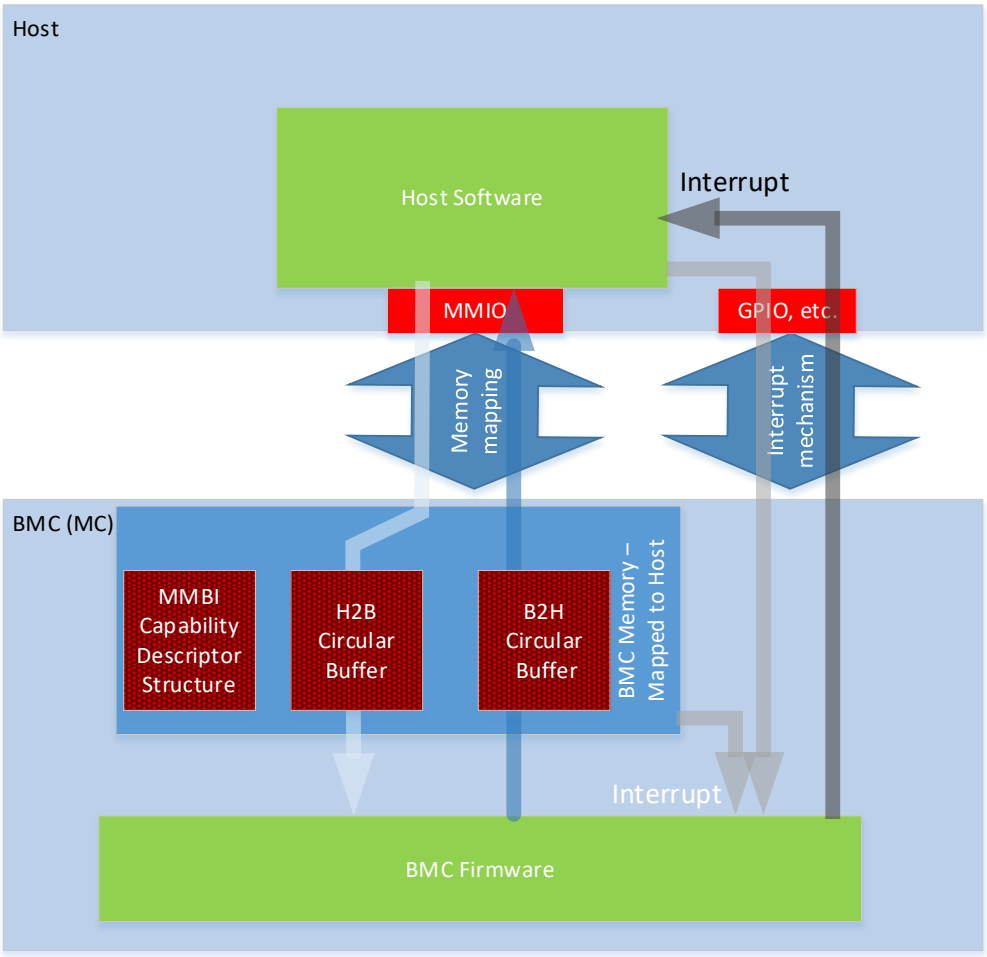


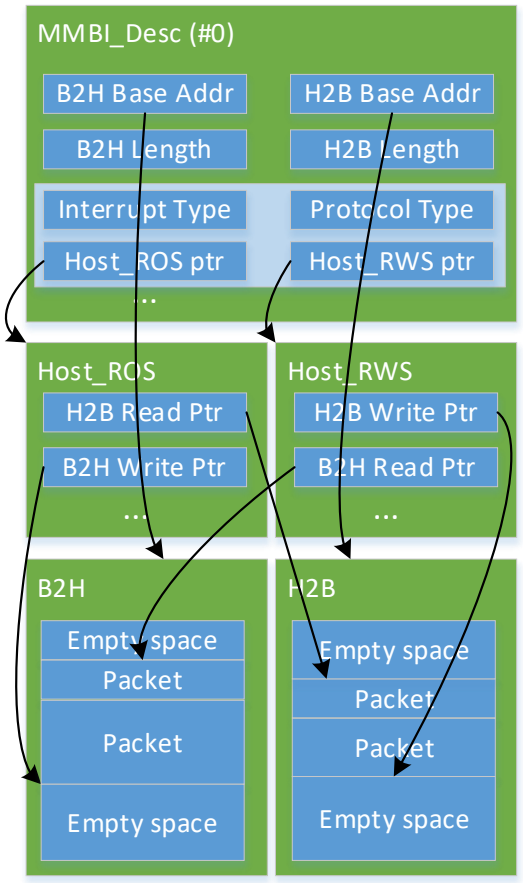

**Figure 2 – MMBI Interface Concept Overview**

# 7 MMBI Data Structures

Each instance of the MMBI interface is divided into sections as defined below:

- "BMC-to-Host" (B2H) region with substructure as follows:
  - *MMBI Capability Descriptor* (*MMBI_Desc* Structure) — see section 7.1 for details
  - *Host_ROS* (Host Read-Only Structure) — see section 7.2.3 for details
  - BMC-to-Host Circular buffer (B2H Circular buffer) — see section 8 for details
- "Host-to-BMC" (H2B) region with substructure as follows:
  - *Host_RWS* (Host Read-Write Structure) — see section 7.2.2 for details
  - Host-to-BMC circular buffer (H2B Circular buffer) — see section 8 for details

The format of the H2B and B2H circular buffers is a sequence of packets, and this format is referred to as Variable Packet Size Circular Buffer (VPSCB). For VPSCB, the relationships between these data structures and their main pointers are as presented in Figure 3.




307

308 **Figure 3 – MMBI Data Structure Relationships**

309 Details of these data structures are presented in the following subsections. Note that the data structures
310 maintain 4-byte alignment for fields that need to be updated atomically. Packets in the circular buffers are
311 also aligned to 4-byte boundaries.

## 312 7.1 MMBI Capability Descriptor

313 *MMBI Capability Descriptor* is used to define the MMBI interface details. (B)MC updates this data
314 structure during initialization. Other than that, the (B)MC and host are not allowed to update it. The host
315 only reads this descriptor to understand the format of the MMBI data structures in memory and shall
316 never write to this data structure. The layout of the structure is presented in Figure 4 and described in
317 Table 1. See also section 8.1.

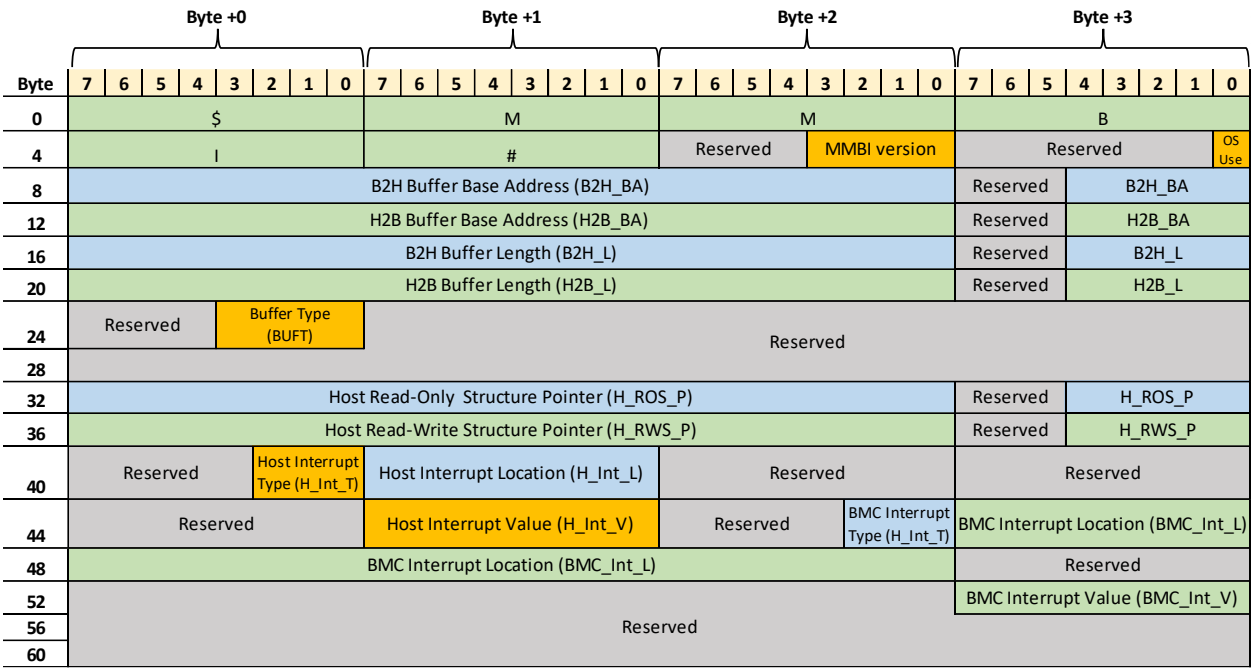



**Figure 4 – MMBI Capability Descriptor Layout**

**Table 1 – MMBI Capability Descriptor Structure (MMBI_Desc)**

| Byte(s) | Description |
|---|---|
| 0:5 | **MMBI Signature**<br>"#MMBI$" in ASCII. When this signature is not present, the host SW should assume the absence of MMBI. |
| 6 | [7:4] Reserved |
| | **[3:0] MMBI version**<br>0001b – Implementations of MMBI described in this document shall indicate version 1 of MMBI. |
| 7 | [7:1] Reserved |
| | [0] **OS Use**<br>Indicates if this MMBI interface is intended for OS use:<br>0b – OS should not use this MMBI interface as it is managed by other host software components (UEFI BIOS, ACPI ASL code, etc.).<br>1b – This MMBI interface is intended for OS use. |
| 8:11 | [31:29] – Reserved |
| | **[28:0] B2H Buffer Base Address (B2H_BA)**<br>B2H (BMC-to-Host) buffer base address expressed in 8-byte units as offset relative to the beginning of the descriptor |

| Byte(s) | Description |
|---|---|
| 12:15 | [31:29] – Reserved |
| | **[28:0] H2B Buffer Base Address (H2B_BA)**<br>H2B (Host-to-BMC) buffer base address expressed in 8-byte units as offset relative to the beginning of the descriptor |
| 16:19 | **B2H Buffer Length (B2H_L)**<br>The size of the B2H buffer (can represent up to 4GB) |
| 20:23 | **H2B Buffer Length (H2B_L)**<br>The size of the B2H buffer (can represent up to 4GB) |
| 24 | [7:4] Reserved |
| | **[3:0] Buffer Type (BUFT)**<br>Indicates the type of data structures in H2B and B2H buffers. The following values are defined:<br>0001b – MMBI Variable Packet Size Circular Buffers (VPSCB) v1 (see section 7.2)<br>Other values are reserved. |
| 25:31 | Reserved |
| 32:52 | **Buffer Type Dependent Descriptor**<br>The definition of this field is dependent on the BUFT field value:<br>If BUFT=0001b (VPSCB), Table 2 in section 7.2 defines the format of these bytes and the packet format in circular buffers is defined in section 9 |
| 56:63 | Reserved |

## 7.2 MMBI Circular Buffers—Variable Packet Size Circular Buffer

This section describes data structures used when the communication between (B)MC and host SW happens according to the VPSCB Buffer Type (BUFT=0001b).

### 7.2.1 Variable Packet Size Circular Buffer Descriptor

Variable Packet Size Circular Buffer Descriptor is part of the *MMBI_Desc* structure. Its access rules are the same as *MMBI_Desc*:

- The (B)MC updates this data structure during MMBI interface initialization.
- Neither the (B)MC nor the host are allowed to update it at any other time.

330 **Table 2 – Buffer Type Dependent Descriptor for BUFT=0001b (VPSCB Descriptor)**

| Byte(s) | Description |
|---|---|
| 0:3 | [31:29] – Reserved |
| | **[28:0] Host Read-Only Structure Pointer (H_ROS_P)**<br>Points to the *Host_ROS* structure. The base address is expressed in 8-byte units as the offset relative to beginning of the descriptor |
| 4:7 | [31:29] – Reserved |
| | **[28:0] Host Read-Write Structure Pointer (H_RWS_P)**<br>Points to the *Host_RWS* structure. The base address is expressed in 8-byte units as the offset relative to beginning of the descriptor |
| 8 | [7:3] – Reserved |
| | **[2:0] Host Interrupt Type (H_Int_T)**<br>Defines how the (B)MC interrupts the host. This is an informative field from the host's perspective with the intention to keep the (B)MC and host in sync.<br>0 – no interrupt / polling<br>1 – PCIe interrupt (bus specific)<br>2 – physical pin (GPIO)<br>3 – eSPI Virtual Wire<br>Other values are reserved |
| 9 | **Host Interrupt Location (H_Int_L)**<br>If H_Int_T = 0: reserved<br>If H_Int_T = 1: for PCIe, indicates the PCIe interrupt message number<br>If H_Int_T = 2: pin number<br>If H_Int_T = 3: eSPI Virtual Wire Index number<br>Reserved otherwise |
| 10:12 | Reserved |
| 13 | **Host Interrupt Value (H_Int_V)**<br>If H_Int_T = 3: eSPI Virtual Wire data value<br>Reserved otherwise |

| Byte(s) | Description |
|---|---|
| 14 | [7:3] – Reserved |
| | **[2:0] (B)MC Interrupt Type (BMC_Int_T)**<br>Defines how the (B)MC wants to be interrupted:<br>0 – no interrupt triggering by the host<br>1 – relative memory space address (offset defined in the BMC_Int_L field)<br>2 – Inband interrupt (bus specific—such as PCIe MSI or virtual legacy wire)<br>Other values – reserved |
| 15:18 | **(B)MC Interrupt Location (BMC_Int_L)**<br>If BMC_Int_T = 1, memory address–offset relative to the beginning of the *MMBI Capability Descriptor* base address<br>Otherwise reserved |
| 19:22 | Reserved |
| 23 | **(B)MC Interrupt Value (BMC_Int_V)**<br>If BMC_Int_T = 1, this field indicates the value to be written at the given address to trigger an interrupt.<br>Otherwise reserved |

### 7.2.2 Host Read-Write Structure

The host's RW Structure Pointer in the above structure points to the *Host_RWS* structure, which is shown in Table 1. This structure is accessed as follows:

- It is initialized by the (B)MC to the default values.
- The host updates this structure during normal communication—it is read-writeable for the host.
- The (B)MC is not allowed to write to this structure during normal communication—it should treat this structure as read-only (any kind of hardware-based enforcement of the read-only behavior is out of scope of this specification).

339 **Table 3 – MMBI Host Read-Write Structure (Host_RWS)**

| Byte(s) | Description |
|---|---|
| 0:3 | **[31:2] H2B Write Pointer (H2B_WP)**<br>Bits [31:2] of the offset where the host can write the next data in the H2B circular buffer, counted from the beginning of the H2B buffer represented in 4-byte alignment.<br>Bits [1:0] of the offset are assumed to always be zero (for 4-byte alignment).<br>The (B)MC uses this pointer to determine how many bytes of valid data are present in the Circular Buffer (by comparing it with the H2B_RP offset).<br>The host shall advance the pointer once data is written to the Circular Buffer and shall update this pointer to mark the next available offset.<br>Note: The host shall not overwrite the data not read by the (B)MC, as indicated by the H2B_RP. |
| | **[1] Host Interface Up (H_UP)**<br>1 indicates that the host side of the interface is up and running, which means that the data structures can be used by the (B)MC. |
| | **[0] Host Reset Request (H_RST)**<br>Setting this flag to 1 will initiate a reset sequence to get the circular buffers into a known good state (see section 8.1 for more information). |
| 4:7 | **[31:2] B2H Read Pointer (B2H_RP)**<br>Bits [31:2] of the offset where the host reads data from the B2H circular buffer, counted from the beginning of the B2H buffer represented in 4-byte alignment.<br>Bits [1:0] of the offset are assumed to always be zero (for 4-byte alignment).<br>The (B)MC uses this pointer to determine how much of data is read by the host. Comparing this with the B2H Write Pointer (B2H_WP) will provide how much space is left to write the data.<br>The host shall only advance the pointer once the data available in B2H is read by the host. |
| | **[1] Reserved** |
| | **[0] Host Ready (H_RDY)**<br>0 indicates that the host is performing some tasks that keep it busy, and so it may be unresponsive. However, the (B)MC can use the data structures and, for example, put data into the buffers as long as H_UP = 1.<br>1 indicates that the host is ready to exchange data (see section 8.1 for more information). |

340 ### 7.2.3 Host Read-Only Structure

341 Host RO Structure Pointer points to *Host_ROS* structure. The host is only allowed to read this structure
342 (never write). Any kind of hardware-based enforcement of the read-only behavior is out-of-scope of this
343 specification. This structure is initialized by the (B)MC to the default values and later updated by (B)MC
344 during normal communication—it is read-writeable for the (B)MC.

345 **Table 4 – MMBI Host Read-Only Structure (Host_ROS)**

| Byte(s) | Description |
|---|---|
| 0:3 | **[31:2] B2H Write Pointer (B2H_WP)**<br><br>Bits [31:2] of the offset where the (B)MC can write the next data in the B2H circular buffer, counted from the beginning of the B2H buffer.<br><br>Bits [1:0] of the offset are assumed to always be zero (for 4-byte alignment).<br><br>The host uses this pointer to determine how many bytes of valid data are present in the Circular Buffer (by comparing it with B2H_RP offset)<br><br>The (B)MC shall advance the pointer once data is written to the Buffer to mark the next available offset.<br><br>Note: (B)MC shall not overwrite the data not read by host, as indicated by the B2H_RP. |
| | **[1] (B)MC Interface Up (B_UP)**<br><br>1 indicates that the (B)MC side of the interface is up and running which means that the data structures are initialized and can be used |
| | **[0] (B)MC Reset Request (B_RST)**<br><br>Setting this flag to 1 will initiate a reset sequence to get the circular buffers into a known good state (see section 8.1 for more information). |
| 4:7 | **[31:2] H2B Read Pointer (H2B_RP)**<br><br>Bits [31:2] of the offset where the host reads data from the H2B circular buffer, counted from the beginning of the H2B buffer.<br><br>Bits [1:0] of the offset are assumed to always be zero (for 4-byte alignment).<br><br>The host uses this pointer to determine how much of data is read by the (B)MC. Comparing this with the H2B write pointer will provide how much space is left to write.<br><br>(B)MC shall only advance the pointer once the data available in H2B is read by the (B)MC. |
| | **[1] Reserved** |
| | **[0] (B)MC Ready (B_RDY)**<br><br>0 indicates that the (B)MC is performing some tasks that keep it busy and so it may be unresponsive – host however can use the data structures and, for example, put data into the buffers as long as B_UP = 1<br><br>1 indicates that the (B)MC is ready to exchange data (see section 8.1 for more information). |

346 MMBI uses two circular buffers: H2B and B2H. Each buffer is a memory range defined in the descriptor
347 with the following access:

348 • H2B (Host-to-BMC buffer) is RW for the host and RO for the (B)MC.
349 • B2H (BMC-to-Host buffer) is RO for the host and RW for the (B)MC.

The Read Pointer and Write Pointer are used to indicate the read and write location in the buffer. For each read or write the pointer shall be advanced. It means pointer increment with a rollover at the buffer size.

These pointers, along with the Buffer Length fields (B2H_L or H2B_L), are used to calculate the number of filled bytes to read or the number of empty bytes available for write.

The circular buffers will be used to send packets of arbitrary size. A packet may require multiple memory reads and/or write transfers.

# 8 Runtime Flows

## 8.1 MMBI Interface Initialization and Reset

This section describes the steps to allow the (B)MC to complete the initialization of the data structures and indicating when both sides of communication are ready to exchange data.

The goal of the reset, on the other hand, is to reinitialize the data structures when at least one side wants a clean start, which may be due to unexpected device events, malfunction, error, etc. It may also be used to reinitialize the data structures after, for example, a (B)MC firmware update in which the data structure needs some new values (e.g., when the circular buffer size changes after the firmware update). A graceful reset follows the state diagram presented in Figure 5, and it guarantees that MMBI protocol layer does not drop any packets (note that other protocol layers may still be unable to guarantee delivery).

The reset sequence is also automatically initiated when hardware errors lead to all-ones or all-zeros memory reads, as is typical with some media. This is thanks to the fact that when all the flags are zeros or are all ones, it indicates an initialization or transition to initialization states. Such unexpected resets do not follow the handshake protocol, and so are ungraceful and may lead to packet losses.

These flags are used to indicate the (B)MC's status as related to initialization and reset:

- (B)MC Interface Up (B_UP)
- (B)MC Reset Request (B_RST)

Similar flags are used to indicate the host's status:

- Host Interface Up (H_UP)
- Host Reset Request (H_RST)

All these flags are used in combination to achieve the proper handshake mechanism between the host and the (B)MC during initialization or reset.

### 8.1.1 Initialization of Descriptor Structures after Power Up

The (B)MC must initialize the expected content of the MMBI data structures (see section 7) during power up and make the shared memory available to the host. Initialization is expected to complete before the host software accesses these structures so that the host can find the *MMBI Capability Descriptor* (*MMBI_Desc*) using the MMBI signature bytes. MMBI structures and buffers must always remain available in the shared memory when the host is using the MMBI interface.

During the initial accesses after the host's power up or reset, the host's software is expected to verify if the content of the MMBI version and MMBI signature are as expected. If the above requirements are met, the host is expected to check the interface state.

If the host's software does not find the proper *MMBI Capability Descriptor (MMBI_Desc)* content at the expected location, the host should consider the MMBI as not present or, optionally, it may implement a

wait option with a timeout. Such a timeout mechanism is system-dependent and is out of scope of this specification.

If the MMBI signature and MMBI version fields match, but the size and location of the buffers cannot be fulfilled by the host, it shall indicate the initialization mismatch error by transitioning to the *Initialization Mismatch* state as described below. With this indication, the (B)MC may consider the interface as inoperable or attempt to reinitialize the *MMBI_Desc* structure with, for example, a smaller buffer size. Before updating the data structure content, the (B)MC shall first clear the B_UP flag and then clear the H_RST flag to return back to the *Initialization in Progress* state. Such attempts to repair the situation are system-dependent and are out of scope of this specification.

### 8.1.2 Interface States and Graceful Reset

When _RST and _UP are both set on one side of communication, it means the entity is requesting a reset sequence. When B_RST = H_RST = B_UP = H_UP = 1, it means that both entities are ready to perform the reset sequence (in fact, the host is just waiting for the (B)MC to do all the initialization).

All the states are summarized in Table 5. The “Host Write Access” and “(B)MC Write Access” columns define write-access restrictions to the data structures by host and (B)MC, respectively. There are no read restrictions for the (B)MC and host. Note that the host is expected to re-read the data structure contents after initialization is completed.

407 **Table 5 – MMBI Interface States**

| B_UP | B_RST | H_UP | H_RST | State Description | Host Write Access | (B)MC Write Access |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **Initialization in Progress**<br>The (B)MC is initializing the data structures.<br>The host can only monitor the data structures, waiting for B_UP = 1 and B_RST = 0 flags. | Host not allowed to write to any MMBI structures | (B)MC allowed to write to any MMBI structures |
| 1 | 0 | 0 | 0 | **Initialization Completed**<br>The (B)MC has completed initialization of the data structures and is ready to exchange data—waiting for the host to be ready. The host should re-read the *MMBI_Desc* structure and any dependent structures.<br>During this state, the (B)MC is allowed to deposit packets into the circular buffer. | Host allowed to write to MMBI structures as per section 7 | (B)MC allowed to write to MMBI structures as per section 7 |
| 1 | 0 | 1 | 0 | **Normal Runtime**<br>Both the (B)MC and host use the data structures and the circular buffers for data exchanges. | Host allowed to write to MMBI structures as per section 7 | (B)MC allowed to write to MMBI structures as per section 7 |
| 1 | 1 | 1 | 0 | **Reset Request by (B)MC**<br>The (B)MC is requesting reset—waiting for the host to notice the request.<br>When the host notices the request, it should consume the data from the B2H (if any) and shall set H_RST flag as an ACK and wait for the initialization to complete (B_UP = 1 and B_RST = 0 status). | Host allowed to write to MMBI structures as per section 7 | (B)MC allowed to write to MMBI structures as per section 7 |
| 1 | 0 | 1 | 1 | **Reset Request by Host**<br>The host is requesting reset—waiting for the (B)MC to notice the request and reinitialize the interface. When the host sets the H_RST flag, it shall not perform any further updates in the MMBI data structures but shall only wait for the initialization to be completed by (B)MC (B_UP = 1 and B_RST = 0 status).<br>When the (B)MC notices the request, it should consume the data from the B2H (if any) and shall set B_RST flag as an ACK. | Host not allowed to write to any MMBI structures | (B)MC allowed to write to any MMBI structures |

| B_UP | B_RST | H_UP | H_RST | State Description | Host Write Access | (B)MC Write Access |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | **Reset ACKed**<br>The host and (B)MC are ready to perform graceful interface reset. This is a transient state when the host is waiting for the (B)MC to complete the initialization. The host is not allowed to write to MMBI data structures. The (B)MC is expected to clear the B_UP and B_RST flags (in this order) and reinitialize all the data structures. | Host not allowed to write to any MMBI structures | (B)MC allowed to write to any MMBI structures |
| 0 | 1 | 1 | 1 | **Transitioning to Initialization**<br>Transient state after the “Reset ACKed” state. The host is not allowed to write to MMBI data structures. | Host not allowed to write to any MMBI structures | (B)MC allowed to write to any MMBI structures |
| 0 | 1 | 1 | 0 | **Temporary Transition States**<br>These states may be observed during initialization when the (B)MC updates the data structures (reinitialization of all the data structures is not an atomic operation).<br>They are unexpected during normal operation and if they happen it means that MMBI structures have been corrupted. The (B)MC may initialize the interface or stop using MMBI and report a fatal error. | Host not allowed to write to any MMBI structures | (B)MC allowed to write to any MMBI structures |
| 0 | 1 | 0 | 1 | | | |
| 0 | 1 | 0 | 0 | | | |
| 0 | 0 | 0 | 1 | | | |
| 1 | 0 | 0 | 1 | **Initialization Mismatch**<br>The host causes transition into this state from *Initialization Completed* when it is unable to use the interface due to unsupported content in the *MMBI Capability Descriptor* structure. | Host not allowed to write to any MMBI structures | (B)MC allowed to write to any MMBI structures |
| 1 | 1 | 0 | 1 | **Unexpected States**<br>These states shall never happen:<br>• If the (B)MC reads this state, it indicates that the host does not follow MMBI protocol or some other corruption happened—the (B)MC should initialize the interface or it may stop using MMBI and report a fatal error, depending on system policy.<br>• If the host reads this state, it may wait for the reinitialization to complete or stop using MMBI and report a fatal error, depending on system policy. | | |
| 1 | 1 | 0 | 0 | | | |
| 0 | 0 | 1 | 0 | | | |
| 0 | 0 | 1 | 1 | | | |

The expected state transitions are presented in Figure 5:

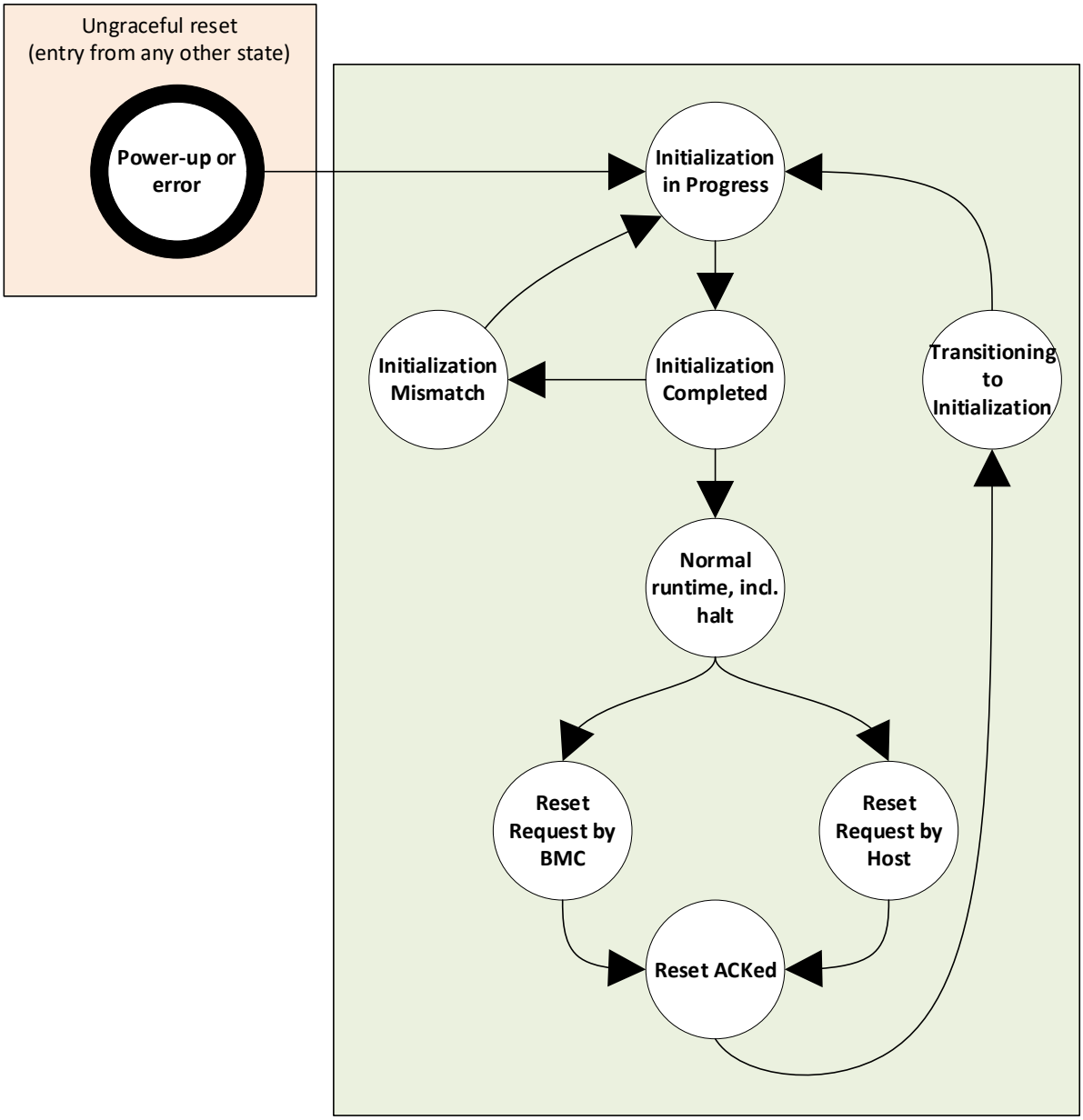



**Figure 5 – MMBI Interface States**

The host shall check the MMBI Interface state before writing any new data to the H2B buffer (as described in Table 5, the host is only allowed to transfer new data in the Normal Runtime state, i.e., B_UP=1 & B_RST=0 & H_UP=1 & H_RST=0). Similarly, the (B)MC shall check the status before writing

any new data to the B2H buffer. These status flags are conveniently located in the B2H_WP or H2B_WP bytes which the host or (B)MC, respectively, read anyway during any use of the circular buffers.

#### 8.1.2.1 Host Initiating Graceful Reset Sequence

Assuming *Normal Runtime* state, the host shall use the following sequence to request MMBI interface reset:

1) The host sets H_RST = 1 to initiate the reset flow. If (B)MC interrupts are enabled, the host notifies the (B)MC.
    a. In response, the (B)MC is expected to set B_RST = 1, which indicates the transition to the *Reset ACKed* state. If host interrupts are enabled, the host is expected to be notified about the update (or else it uses polling). At this point, the (B)MC reinitializes all the data structures.
2) The host waits for B_UP = 1 and B_RST = 0 (and H_UP = H_RST = 0), which indicates the transition to the *Initialization Completed* state. Host interrupts are not used at this stage until H_UP is set by host software.
3) The host transitions to the *Normal Runtime* state by setting H_UP = 1. The host is also expected to set the B_RDY flag, indicating that it can receive and handle new packets—see section 8.3. If (B)MC interrupts are enabled, the host notifies the (B)MC after the flags are updated.

Figure 6 presents a sample flow:

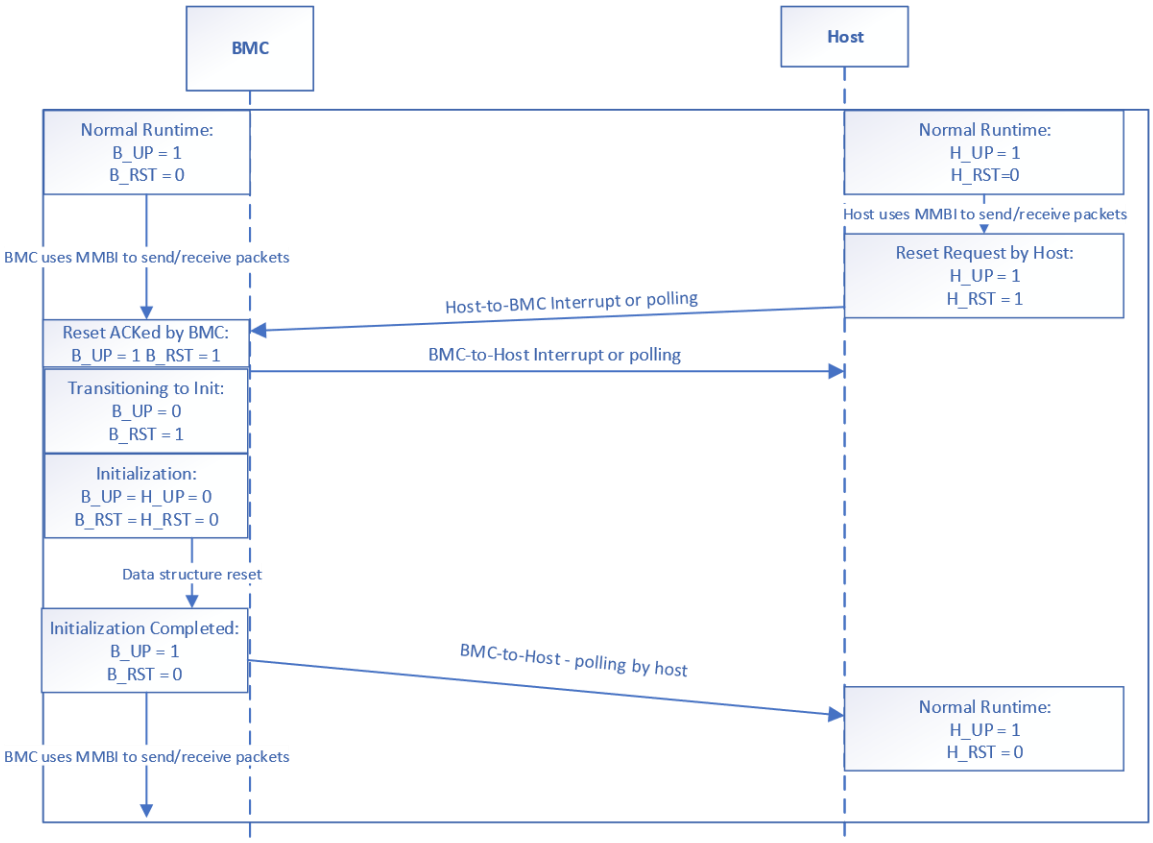



**Figure 6 – Sample MMBI Reset by Host**

#### 8.1.2.2 (B)MC Initiating Graceful Reset Sequence

Assuming *Normal Runtime* state, the (B)MC shall use the following sequence to request MMBI interface reset:

1) The (B)MC sets B_RST = 1 to initiate the reset flow. If host interrupts are enabled, the (B)MC notifies the host.

2) The (B)MC waits for H_UP = 1 and H_RST = 1, which indicates the transition to the *Reset ACKed* state. If (B)MC interrupts are enabled, the (B)MC is expected to be notified about the update (or else (B)MC uses polling).

3) The (B)MC clears the B_UP flag (B_RST still set). Host interrupts are no longer enabled.

4) The (B)MC clears the H_UP and H_RST flags (this may cause transient states to be observed by the host).

5) The (B)MC clears the B_RST flag.

6) The (B)MC reinitializes all the data structures.

7) The (B)MC sets B_UP = 1. Host interrupts are not used at this stage until H_UP is set by host software.

8) The (B)MC waits for the host to set H_UP = 1. If (B)MC interrupts are enabled, the (B)MC is expected to be notified about the update.

Note that the (B)MC is also expected to set the B_RDY flag, typically in step 7, indicating that it can receive and handle new packets—see section 8.3.

Figure 7 presents a sample flow.

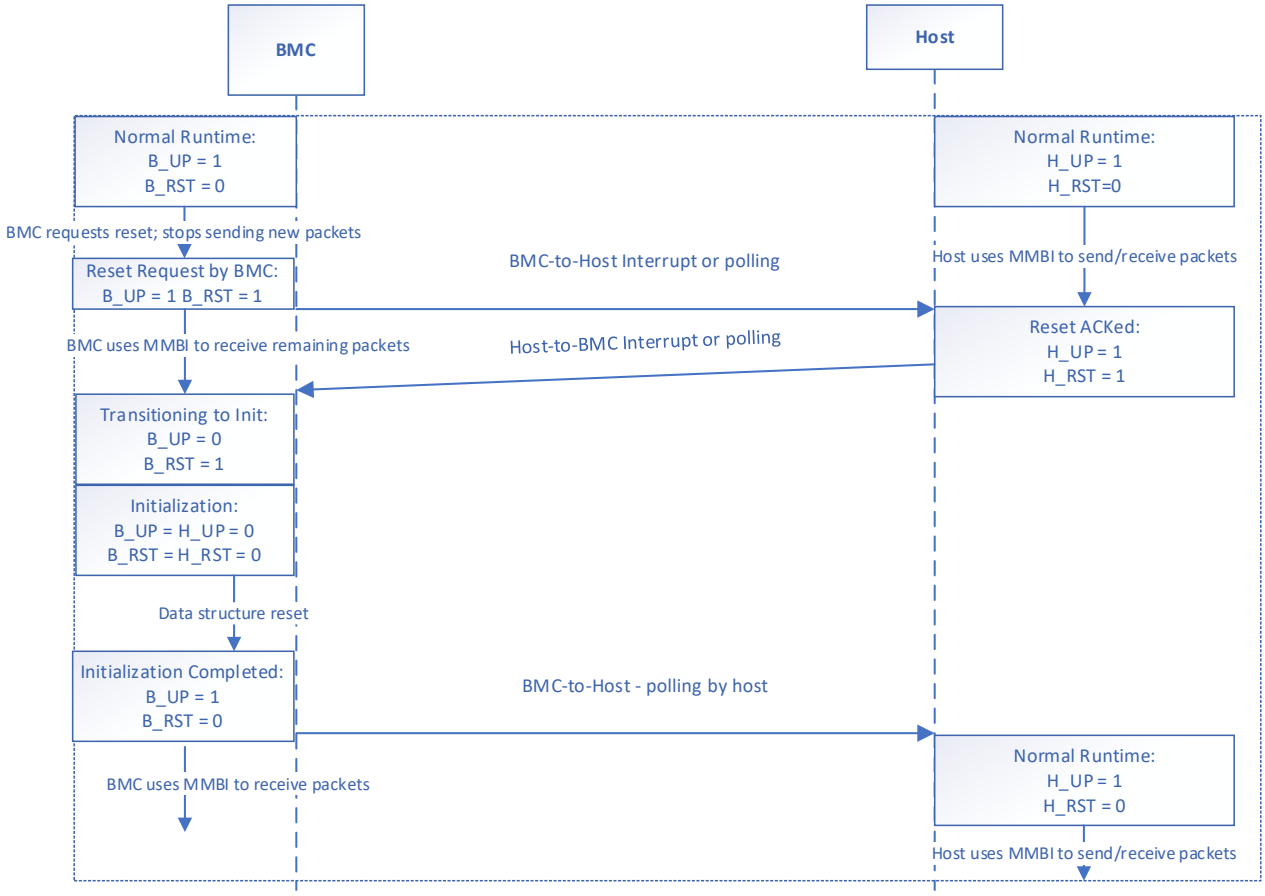



**Figure 7 – Sample MMBI Reset by (B)MC**

### 8.1.3 Ungraceful Reset Considerations

If an ungraceful reset/crash happens, MMBI does not guarantee delivery. However, provisions are present in the MMBI design to handle the following scenarios:

1. In the case of a (B)MC FW-only reset (HW continues to work, memory content, including buffers stay intact in shared memory and accesses are still handled by HW): the host will still see the MMBI in the normal state and write to MMBI Circular buffers to deposit or read data as long as there is any space available in the buffers. In this situation, host may timeout waiting for a response but this is handled by higher layers above MMBI.
2. (B)MC HW reset (buffers are wiped and MMIO mechanisms are broken): the host will see errors on reads/writes and must handle them as per host-specific mechanisms. Additionally, MMBI encoding of status in B_UP, B_RST, H_UP, & H_RST is such that all-zeros or all-ones are recognized as transient states (see Table 5). So, even if there would be no other mechanisms in the system, the host would still recognize this as an error and would have to wait for reinitialization by the (B)MC (the host is not allowed to write to the buffers in the transient state, i.e., until the data structures are reinitialized by (B)MC FW).
3. Unexpected host reset (SW or HW reset is the same outcome): the host's unexpected reset will leave the data structures intact in (B)MC memory, so the (B)MC can still read the data from the buffers. Assuming the (B)MC understands the host's status via other mechanisms, the (B)MC can take informed decisions about how to respond to such situations.

In all the above cases, MMBI data structures can be reinitialized after the reset to allow a clean restart.

## 8.2 Calculation of Filled Space and Empty Space in Circular Buffer

The procedure for calculating the number of filled bytes in a circular buffer is analogous for both the H2B and B2H buffers: the difference between the write pointer and read pointer indicates the amount of valid data, accounting for the rollover at the end of the buffer. The write pointer cannot advance beyond the read pointer, accounting for the rollover at the end of the buffer.

The following steps allow calculation of the number of filled slots in a circular buffer:

1. The write and read pointers must start with zero after initialization. Since read pointer = write pointer, there is no valid data/packets in the buffer on initialization.
2. Once data is written to the buffer, the source (the host or (B)MC) will advance the write buffer pointer.
3. Read pointer is advanced once data is read/consumed by the receiver (the host or (B)MC).
4. Rollover: when the pointers reach the maximum offset within the buffer during writing/reading, data must be written/read starting back at zero offset, and the pointers roll over accordingly.

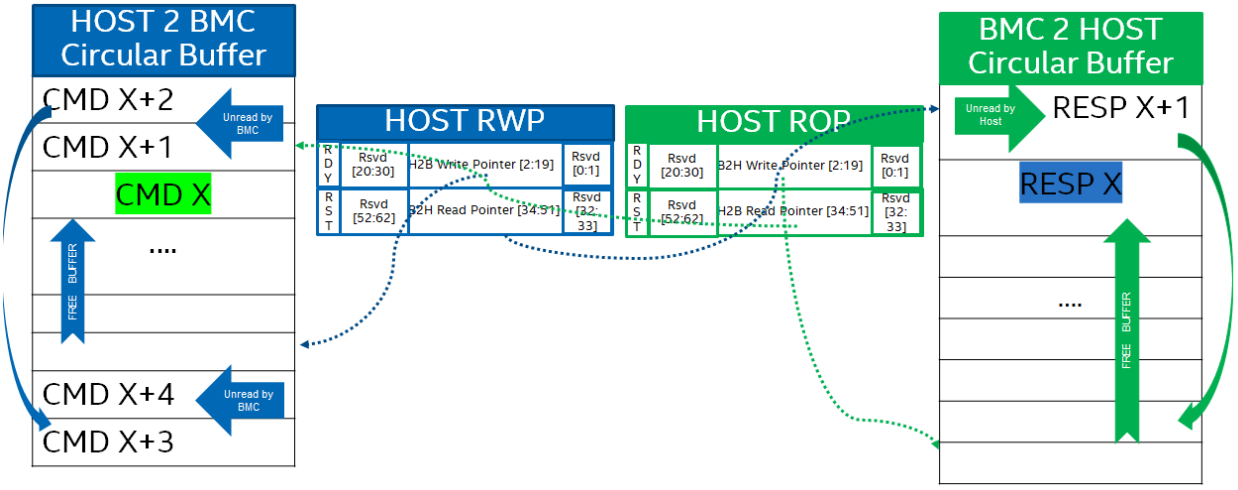



**Figure 8 – Filled and Empty Space in Circular Buffers**

## 8.3 Device Readiness and Communication Pause

In addition to the reinitialization or reset states, the MMBI interface also uses the H_RDY and B_RDY flags to indicate the device's readiness to consume incoming packets and handle them. When the host or (B)MC are ready to receive and handle packets, they set the B_RDY or H_RDY flags, respectively. If a B_RDY or H_RDY flag is clear but the B_UP and H_UP flags are set, it means that the MMBI interface is up but the target device is not ready to consume and handle new packets. When the interface is up, it means that the data structures are ready to accept new packets so the sender can:

- wait for the receiver to become ready before writing new packets to the buffer—this is important if the sender expects an action to be taken by the receiver, such as providing a response

- deposit new packets to the buffer in order for the receiver to consume them later—this capability may be used if the sender does not expect a response from the receiver; for example, when the sender needs to deposit some logs in the shared memory

An example flow when (B)MC firmware / host software undergoes a reset and indicates its non-readiness during a reboot is presented in Figure 9. Note that in this example it is assumed that the MMBI data structures are still intact in shared memory during the reset.

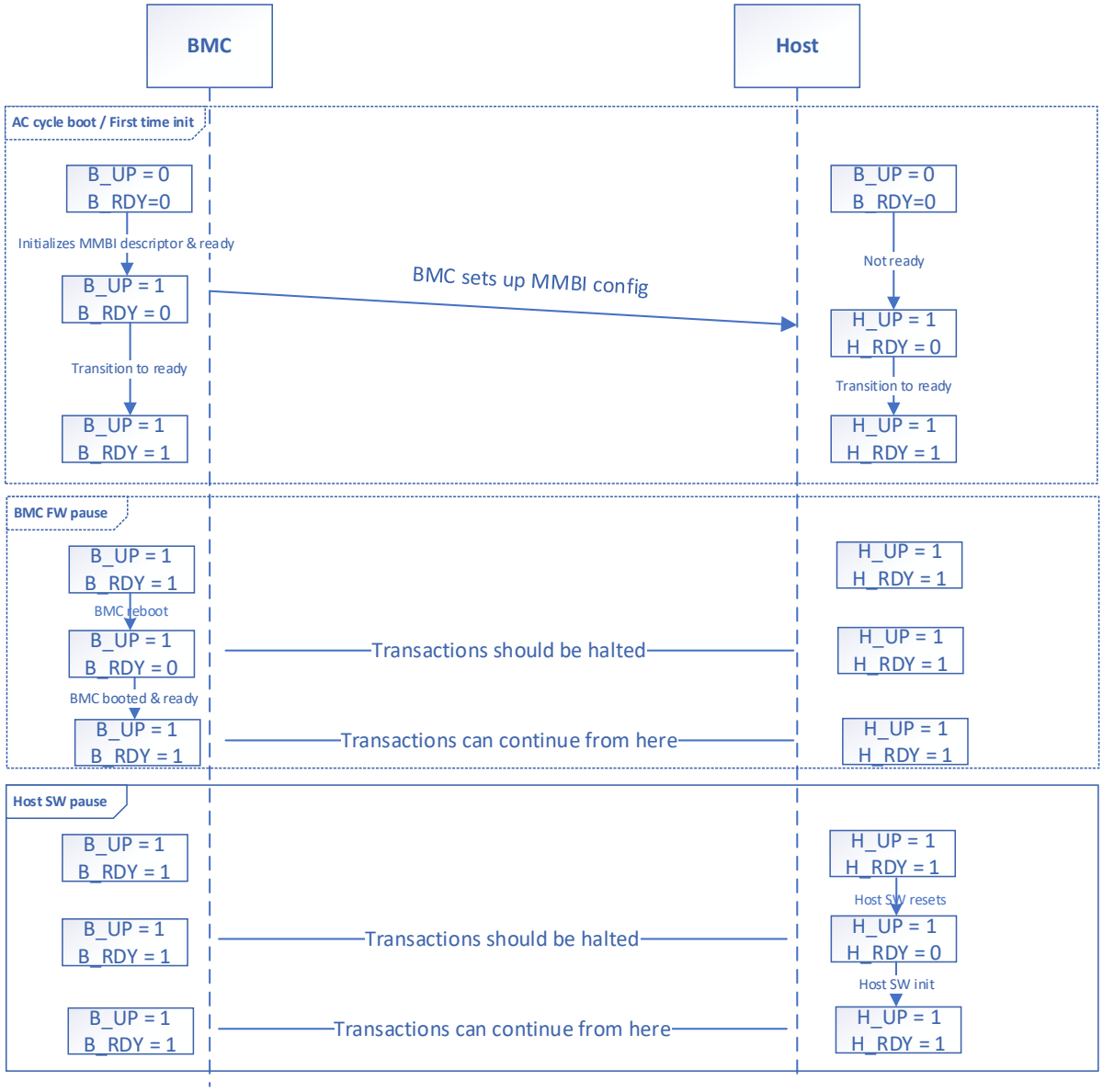



**Figure 9 – Sample MMBI Device Pause Sequences**

## 8.4 Packet Transfer

This flow describes the host-to-(B)MC flow that shall be followed to send a packet. An analogous flow shall be followed to send packets in the opposite direction (swap (B)MC and host in the description and use B2H buffer instead):

1) Host software reads the read and write pointers (H2B_RP and H2B_WP) to determine the number of empty spaces available in its circular buffer.
    a. If there is not enough empty space available in the host's circular buffer, the host waits until there is room in the host's circular buffer. This is done either by polling or waiting for an interrupt.
    b. Host software shall also verify that B_UP = 1 & B_RST = 0 before any packet transfers. If this is not the case, it shall follow the reset process as defined in section 8.1. Note that the host may decide to delay packet transfer depending on B__RDY state and its policy.
2) Once there is enough empty space available in the circular buffer, the host writes the packet into the host's circular buffer. To accomplish this, the host sequentially writes data at the write pointer location but not exceeding the length of the buffer (H2B_L). When it reaches the maximum address of the buffer, it shall continue writing the packet from the buffer base address (H2B_BA). This process shall never overflow the buffer by advancing beyond the H2B_RP.
3) Once the packet write is complete, the host updates the write pointer value in H2B_WP.
4) In Interrupt-enabled mode, the (B)MC firmware is interrupted:
    a. If BMC_Int_T = 1, the host uses the (B)MC Interrupt Info (BMC_Int_L) and (B)MC Interrupt Value (BMC_Int_V) to interrupt the (B)MC FW.
    b. Even if BMC_Int_T = 0, the (B)MC HW may also monitor H2B_WP and generate an interrupt automatically.
    c. Alternatively, a platform-specific method can be used to trigger the interrupt to (B)MC.
5) In polling mode, the (B)MC FW can continuously read the write pointer to see when it changes. In interrupt mode, it is woken up by the (B)MC HW.
6) The (B)MC firmware reads the read/write pointers and determines the number of filled spaces in the circular buffer available for reading.
    a. If the circular buffer is empty, the host has not sent a packet. This interrupt is for another reason, or it indicates that the host has completed reading the packet(s) last transmitted by the (B)MC.
7) The (B)MC FW reads the buffer data written by the host.
8) The (B)MC FW updates the read pointer in B2H_RWS. This indicates to the host how much data has been read by the (B)MC, and the host can use the portion of the buffer that has been read already.
9) If host notifications are enabled, (B)MC FW shall generate an interrupt to the host.
10) When the host software gets interrupted or due to polling of H2B_RP, it can determine that the (B)MC has consumed the data. The host can also poll instead of relying on interrupts.

## 8.5 Interrupts (Optional)

Interrupts, if enabled by the discovery/control mechanisms of MMBI, shall be triggered for the following reasons (both for host software and (B)MC firmware):

- A packet has just been written to the circular buffer.
- A packet has just been read from the circular buffer.
- The host or (B)MC has initiated an interface reset sequence.
- The host or (B)MC has completed its portion of the interface reset sequence and normal operation can begin.

An interrupt handler shall:

- check the status flags in the *MMBI Capability Descriptor* (*MMBI_Desc*)—if a reset is initiated, the flow defined in section 8.1 shall be followed
- check if there is a packet in the circular buffer—this can be calculated as per section 8.2—and, if there is data present in the buffer, the interrupt handler should initiate the packet receive flow, as defined in section 8.4.

If there are multiple instances of the MMBI interface sharing the same interrupt, the interrupt handler shall check all the instances for the reasons listed above. The order of such a check and interrupt affinity are implementation-specific and out of scope of this specification.

# 9 Multi-Protocol Packet Format

If BUFT=`0001b` (VPSCB) and Packet Protocol Type = `0001b` (Multi-protocol Type), the multi-protocol MMBI packets will have the following defined header fields, as shown in Table 6. There is a 4-byte alignment expectation, meaning that padding must be added if necessary for the packet length to be a multiple of 4 bytes.

570 **Table 6 – Multi-Protocol Packet Format**

| Byte(s) | Description |
|---|---|
| 0:2 | **[23:2] Packet Length (PKT_LEN)**<br>The size of the packet, calculated as PKT_LEN+1 multiplied by 4 bytes (can represent up to 16MB packet).<br>Values `0x3FFFFF` and zero are reserved. |
| | **[1:0] Packet padding (PKT_PAD)**<br>Number of padding bytes |
| 3 | [7:4] – Reserved |
| | **[3:0] – Packet type (PKT_TYPE)**<br>Defines the format of the remaining bytes:<br>`0100b` – MCTP over MMBI (see *Management Component Transport Protocol (MCTP) Memory-Mapped Buffer Interface (MMBI) Transport Binding Specification*)<br>`0101b` – Vendor defined content as defined below<br>Other values are reserved. |
| 4:N-1 | **Protocol type specific fields**<br>This field depends on the PKT_TYPE value:<br>If PKT_TYPE = MCTP = `0100b`: format follows MCTP over MMBI (see *Management Component Transport Protocol (MCTP) Memory-Mapped Buffer Interface (MMBI) Transport Binding Specification*)<br>If PKT_TYPE = Vendor defined = `0101b`: the following vendor-defined format shall be used: |

| Byte(s) | Description |
|---|---|
| 4:7 | Vendor IANA Enterprise Number encoded in little-endian format; for more information about IANA Enterprise Numbers, please see *Internet Assigned Numbers Authority – Private Enterprise Numbers* |
| 8:N-1 | Content defined by the vendor |

| | |
|---|---|
| (N:M) | **Padding (PAD) – optional**<br>Padding bytes as defined in PKT_PAD field.<br>Note: padding is added to ensure packets are 4-byte aligned |

571

# ANNEX A
# (informative)

# Notations

Examples of notations used in this document are as follows:

- 2:N — In field descriptions, this will typically be used to represent a range of byte offsets starting from byte two and continuing to and including byte N. The lowest offset is on the left; the highest is on the right.
- (6) — Parentheses around a single number can be used in packet field descriptions to indicate a byte field that may be present or absent.
- (3:6) — Parentheses around a field consisting of a range of bytes indicates the entire range may be present or absent. The lowest offset is on the left; the highest is on the right.
- PCIe — Underlined blue text is typically used to indicate a reference to a document or specification called out in clause 2, “Normative References” or to items hyperlinked within the document.
- [4] — Square brackets around a number are typically used to indicate a bit offset. Bit offsets are given as zero-based values (that is, the least significant bit offset = 0).
- [7:5] — A range of bit offsets. The most significant bit is on the left, the least significant bit is on the right.
- `1b` — A number consisting of `0`s and `1`s followed by a lowercase “`b`” indicates that the number is in binary format.
- `0x12A` — A leading “`0x`” indicates that the number is in hexadecimal format.

# ANNEX B
# (informative)

## Change log

| Version | Date | Description |
|---|---|---|
| 1.0.0 | 2023-07-14 | Initial release |
| 1.0.1 | 2024-10-09 | Document title change (“Memory-Mapped BMC Interface” to “Memory-Mapped Buffer Interface”) to better reflect potential broader uses of MMBI beyond just BMC |