

JEDEC STANDARD

Universal Flash Storage (UFS) Unified Memory Extension

Version 1.1

JESD220-1A

(Revision of JESD220-1, September 2013)

MARCH 2016

JEDEC SOLID STATE TECHNOLOGY ASSOCIATION



NOTICE

JEDEC standards and publications contain material that has been prepared, reviewed, and approved through the JEDEC Board of Directors level and subsequently reviewed and approved by the JEDEC legal counsel.

JEDEC standards and publications are designed to serve the public interest through eliminating misunderstandings between manufacturers and purchasers, facilitating interchangeability and improvement of products, and assisting the purchaser in selecting and obtaining with minimum delay the proper product for use by those other than JEDEC members, whether the standard is to be used either domestically or internationally.

JEDEC standards and publications are adopted without regard to whether or not their adoption may involve patents or articles, materials, or processes. By such action JEDEC does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the JEDEC standards or publications.

The information included in JEDEC standards and publications represents a sound approach to product specification and application, principally from the solid state device manufacturer viewpoint. Within the JEDEC organization there are procedures whereby a JEDEC standard or publication may be further processed and ultimately become an ANSI standard.

No claims to be in conformance with this standard may be made unless all requirements stated in the standard are met.

Inquiries, comments, and suggestions relative to the content of this JEDEC standard or publication should be addressed to JEDEC at the address below, or refer to www.jedec.org under Standards and Documents for alternative contact information.

Published by
©JEDEC Solid State Technology Association 2016
3103 North 10th Street
Suite 240 South
Arlington, VA 22201-2107

This document may be downloaded free of charge; however JEDEC retains the copyright on this material. By downloading this file the individual agrees not to charge for or resell the resulting material.

PRICE: Contact JEDEC

Printed in the U.S.A.
All rights reserved

PLEASE!

DON'T VIOLATE
THE
LAW!

This document is copyrighted by JEDEC and may not be
reproduced without permission.

For information, contact:

JEDEC Solid State Technology Association
3103 North 10th Street
Suite 240 South
Arlington, VA 22201-2107

or refer to www.jedec.org under Standards-Documents/Copyright Information.

UNIVERSAL FLASH STORAGE (UFS) UNIFIED MEMORY EXTENSION, VERSION 1.1**Contents**

	Page
1 Scope	1
2 Normative Reference.....	1
3 Terms and Definitions	2
4 Introduction	6
4.1 Overview	6
4.2 Purpose of Unified Memory Support	7
4.3 Framework of Unified Memory	8
4.3.1 Host Memory Map	8
4.3.2 Configuration of Unified Memory	9
4.4 Speed Estimation.....	9
5 Modified UFS Architecture Overview	11
5.1 Modified UFS Top level Architecture.....	11
5.2 UFS Bus Master Transport Protocol (UBMTP) Layer.....	12
6 UFS UIC Layer: MIPI M-PHY and MIPI UniPro.....	13
6.1 Overview	13
6.2 HIBERNATE	13
7 UFS Bus Master Transport Protocol (UBMTP) Layer.....	15
7.1 Overview	15
7.2 Unified Memory Protocol Information Unit	15
7.2.1 UMPIU Transaction Codes	15
7.3 General Unified Memory Protocol Information Unit Format	17
7.3.1 Overview	17
7.3.2 Basic Header Format	18
7.3.3 COPY DATA UMPIU	19
7.3.4 UM COPY DATA UMPIU	21
7.3.5 ACKNOWLEDGE COPY UMPIU	22
7.3.6 ACCESS UM BUFFER UMPIU	23
7.3.7 WRITE UM BUFFER UMPIU	25
7.3.8 ACKNOWLEDGE UM BUFFER UMPIU.....	26
7.3.9 UM DATA OUT UMPIU	27
7.3.10 UM DATA IN UMPIU	29
7.3.11 Unified Memory Basic Operations.....	31
7.3.12 SCSI Read / Write and Unified Memory Operations	34
8 UFS Functional Descriptions Related to Unified Memory	41
8.1 UFS Boot with Unified Memory Support	41
8.1.1 Introduction	41
8.1.2 Initialization of Unified Memory	41
8.2 UFS Cache in Unified Memory.....	42
8.2.1 Example of How to Use Write Buffer Cache in Unified Memory	43
8.3 Host Device Interaction.....	43
8.3.1 Background Operation Mode with Unified Memory	43
9 UFS Descriptors, Flags and Attributes.....	44
9.1 Device Descriptor.....	44
9.2 Flags	45
9.3 Attributes.....	46
Annex A Differences between JESD220-1A and JESD220-1.....	47

Contents (cont'd)

	Page
Figures	
Figure 4.1 – Conventional Memory Structure	6
Figure 4.2 – UMA Memory Structure	6
Figure 4.3 – Inter-Memory Communication of Device-Integrated RAM Buffering	7
Figure 4.4 – Inter-Memory Communication of UM Buffering	7
Figure 4.5 – Host Memory Map (Physical)	8
Figure 4.6 – Definition of Latency	9
Figure 4.7 – Definition of Multiple Outstanding	10
Figure 5.1 – Modified UFS Top Level Architecture	11
Figure 6.1 – Flag Manipulation before HIBERNATE	14
Figure 6.2 – Flag Manipulation after HIBERNATE	14
Figure 7.1 – Unified Memory Read	31
Figure 7.2 – Unified Memory Write	32
Figure 7.3 – Copy between System Memory and Unified Memory	33
Figure 7.4 – Generic Copy in the Unified Memory	33
Figure 7.5 – SCSI Read (Non-Cached) and Subsequent Unified Memory Operations	35
Figure 7.6 – SCSI Write (Non-Cached) and Subsequent Unified Memory Operations	35
Figure 7.7 – SCSI Write (Cached) and Subsequent Unified Memory Operations	36
Figure 7.8 – SCSI Read (Cached) and subsequent Unified Memory Operations (Simple Single Read)	37
Figure 7.9 – SCSI Read (Cached) and subsequent Unified Memory Operations (Large Data Read with Filling Missing Write Buffer)	38
Figure 7.10 – SCSI Read (Cached) and subsequent Unified Memory Operations (Large Data Read with Flushing Dirty Write Buffer)	39
Figure 7.11 – L2P Table Operation	40
Figure 8.1 – Unified Memory Initialization Sequence Diagram	42
Tables	
Table 7.1 – UMPIU Transaction Codes	15
Table 7.2 – UMPIU Transaction Code Definitions	16
Table 7.3 – General Format of the Unified Memory Protocol Information Unit	17
Table 7.4 – Basic Header Format	18
Table 7.5 – Transaction Type Format	18
Table 7.6 – UMPIU Flags	18
Table 7.7 – COPY DATA UMPIU	19
Table 7.8 – Flags Definition of ACKNOWLEDGE COPY UMPIU	20
Table 7.9 – UM COPY DATA UMPIU	21
Table 7.10 – ACKNOWLEDGE COPY UMPIU	22
Table 7.11 – Flags Definition of ACKNOWLEDGE COPY UMPIU	22
Table 7.12 – ACCESS UM BUFFER UMPIU	23
Table 7.13 – Flags Definition for ACCESS UM BUFFER UMPIU	24
Table 7.14 – WRITE UM BUFFER UMPIU	25
Table 7.15 – ACKNOWLEDGE UM BUFFER UMPIU	26
Table 7.16 – Flags Definition for ACKNOWLEDGE UM BUFFER UMPIU	26
Table 7.17 – UM DATA OUT UMPIU	27
Table 7.18 – Flags Definition for UM DATA OUT UMPIU	28
Table 7.19 – UM DATA IN UMPIU	29
Table 9.1 – Device Descriptor	44
Table 9.2 – Flags	45
Table 9.3 – Attributes	46

Foreword

This Universal Flash Storage (UFS) Unified Memory Extension standard is an extension to JESD220, Universal Flash Storage (UFS) Standard.

Introduction

The UFS standard defines a managed storage device. UFS devices are designed to offer high performance with low power consumption. The UFS device contains features that support both high throughput for large data transfers and performance for small random data accesses.

The UFS Unified Memory Extension standard describes the requirements to implement unified memory functionality in a UFS device.

UNIVERSAL FLASH STORAGE (UFS) UNIFIED MEMORY EXTENSION, Version 1.1

(From JEDEC Board Ballot, JCB-16-13, formulated under the cognizance of the JC-64.1 Subcommittee on Electrical Specifications and Command Protocols (Item 133.11).)

1 SCOPE

This document provides a comprehensive definition of the unified memory support for UFS. It also provides some details in how to utilize the unified memory for realizing high performance in UFS devices.

2 NORMATIVE REFERENCE

The following normative documents contain provisions that, through reference in this text, constitute provisions of this standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

[UFS], JEDEC JESD220C, Universal Flash Storage (UFS), Version 2.1.

[UFSHCI], JEDEC JESD223C, Universal Flash Storage Host Controller Interface (UFSHCI), Version 2.1.

[UFSHCI-UME], JEDEC JESD223-1A, Universal Flash Storage Host Controller Interface (UFSHCI) Unified Memory Extension Version 1.1.

3 TERMS AND DEFINITIONS

For the purposes of this standard, the terms and definitions given in the document included in section 2, Normative Reference, and the following apply.

3.1 Acronyms

CPORT	A CPort is a Service Access Point on the UniPro Transport Layer (L4) within a Device that is used for Connection-oriented data transmission
DMA	Direct Memory Access
GB	Gigabyte
HCI	Host Controller Interface
KB	Kilobyte
LUN	Logical Unit Number
MB	Megabyte
MIPI	Mobile Industry Processor Interface
TB	Terabyte
TC	Traffic Class
UFS	Universal Flash Storage
UM	Unified Memory
UMA	Unified Memory Architecture
UME	Unified Memory Extension
UMPIU	Unified Memory Protocol Information Unit
UniPro	Unified Protocol.
UPIU	UFS Protocol Information Unit
UTP	UFS Transport Protocol

3.2 Terms and Definitions

Application Client: An entity that is the source of SCSI commands and task management function requests in the host.

Byte: An 8-bit data value with the most significant bit labeled as bit 7 and the least significant bit as bit 0.

Device Server: An entity in the device that processes SCSI commands and task management functions.

Doubleword: A 32-bit data value with the most significant bit labeled as bit 31 and the least significant bit as bit 0.

Dword: A 32-bit data value, a Doubleword.

Gigabyte: 1,073,741,824 or 2^{30} bytes.

Host: An entity or a device with the characteristics of a primary computing device that includes one or more SCSI initiator devices.

Initiator Device: Within a transaction, the originator of a SCSI command request message to a target device.

Kilobyte: 1024 or 2^{10} bytes.

Logical Unit: A logical unit is an internal entity of a bus device that performs a certain function or addresses a particular space or configuration within a bus device.

Logical Unit Number: A numeric value that identifies a logical unit within a device.

Megabyte: 1,048,576 or 2^{20} bytes.

Quadword: A 64-bit data value with the most significant bit labeled as bit 63 and the least significant bit as 0.

Segment: A specified number of sequentially addressed bytes representing a data structure or section of a data structure.

Target Device: Within a transaction, the recipient of a SCSI command request message from an initiator device.

Task: A task is a SCSI command which includes all transactions to complete all data transfers and a status response that will satisfy the requirements of the requested services of the command.

Terabyte: 1.099.511.627.776 or 2^{40} bytes.

Transaction: A UFS bus primitive action which results in transmission of serial data packets between a target and initiator.

UFS Protocol Information Unit: Information transfer (communication) between a UFS host and device is done through messages which are called UFS Protocol Information Units. These messages are UFS defined data structures that contain a number of sequentially addressed bytes arranged as various information fields.

Word: A 16-bit data value with the most significant bit labeled as bit 15 and the least significant bit as bit 0.

3.3 Keywords

Several keywords are used to differentiate levels of requirements and options, as follow:

Can - A keyword used for statements of possibility and capability, whether material, physical, or causal (*can equals is able to*).

Expected - A keyword used to describe the behavior of the hardware or software in the design models assumed by this standard. Other hardware and software design models may also be implemented.

Ignored - A keyword that describes bits, bytes, quadlets, or fields whose values are not checked by the recipient.

Mandatory - A keyword that indicates items required to be implemented as defined by this standard.

May - A keyword that indicates a course of action permissible within the limits of the standard (*may equals is permitted*).

Must - The use of the word *must* is deprecated and shall not be used when stating mandatory requirements; *must* is used only to describe unavoidable situations.

Optional - A keyword that describes features which are not required to be implemented by this standard. However, if any optional feature defined by the standard is implemented, it shall be implemented as defined by the standard.

Reserved - A keyword used to describe objects—bits, bytes, and fields—or the code values assigned to these objects in cases where either the object or the code value is set aside for future standardization. Usage and interpretation may be specified by future extensions to this or other standards. A reserved object shall be zeroed or, upon development of a future standard, set to a value specified by such a standard. The recipient of a reserved object shall not check its value. The recipient of a defined object shall check its value and reject reserved code values.

Shall - A keyword that indicates a mandatory requirement strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall equals is required to*). Designers are required to implement all such mandatory requirements to assure interoperability with other products conforming to this standard.

Should - A keyword used to indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain course of action is deprecated but not prohibited (*should equals is recommended that*).

Will - The use of the word *will* is deprecated and shall not be used when stating mandatory requirements; *will* is only used in statements of fact.

3.4 Abbreviations

etc. - And so forth (Latin: et cetera)

e.g. - For example (Latin: exempli gratia)

i.e. - That is (Latin: id est)

3.5 Conventions

UFS specification follows some conventions used in SCSI documents since it adopts several SCSI standards.

A binary number is represented in this standard by any sequence of digits consisting of only the Western-Arabian numerals 0 and 1 immediately followed by a lower-case b (e.g., 0101b). Spaces may be included in binary number representations to increase readability or delineate field boundaries (e.g., 0 0101 1010b).

A hexadecimal number is represented in this standard by any sequence of digits consisting of only the Western-Arabian numerals 0 through 9 and/or the upper-case English letters A through F immediately followed by a lower-case h (e.g., FA23h). Spaces may be included in hexadecimal number representations to increase readability or delineate field boundaries (e.g., B FD8C FA23h).

A decimal number is represented in this standard by any sequence of digits consisting of only the Western-Arabian numerals 0 through 9 not immediately followed by a lower-case b or lower-case h (e.g., 25).

A range of numeric values is represented in this standard in the form "a to z", where a is the first value included in the range, all values between a and z are included in the range, and z is the last value included in the range (e.g., the representation "0h to 3h" includes the values 0h, 1h, 2h, and 3h).

When the value of the bit or field is not relevant, x or xx appears in place of a specific value.

The first letter of the name of a Flag is a lower-case f (e.g., fMyFlag).

The first letter of the name of a parameter included in a Descriptor or the first letter of the name of an Attribute is:

- a lower-case b if the parameter or the Attribute size is one byte (e.g., bMyParameter),
- a lower-case w if the parameter or the Attribute size is two bytes (e.g., wMyParameter),
- a lower-case d if the parameter or the Attribute size is four bytes (e.g., dMyParameter),
- a lower-case q if the parameter or the Attribute size is eight bytes (e.g., qMyParameter).

4 INTRODUCTION

4.1 Overview

Unified Memory (UM) architecture is a very versatile memory architecture which enables sharing the Host memory with Device. A typical example is today’s graphics memory used in popular CPUs, graphics workstations, etc.

Unified Memory Architecture (UMA) enables using Host memory as Device working memory, e.g., as L2P table cache, write buffer, etc. The UM area is physically located on the Host side, but logically belongs to the Device as if it would be physical RAM in the Device. Thus the UM area is logically a property of the Device, and it is a replacement of the Device-integrated RAM.

Figure 4.1 shows conventional memory structure of Host and Device, and Figure 4.2 shows that of UMA.

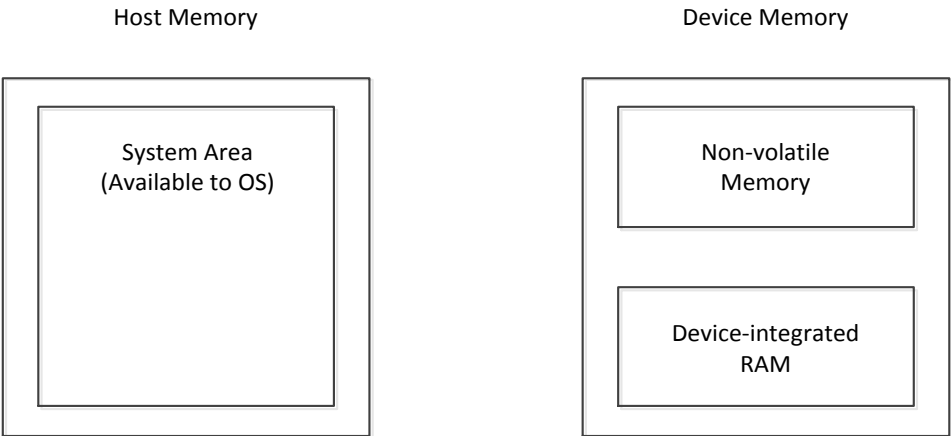


Figure 4.1 — Conventional Memory Structure

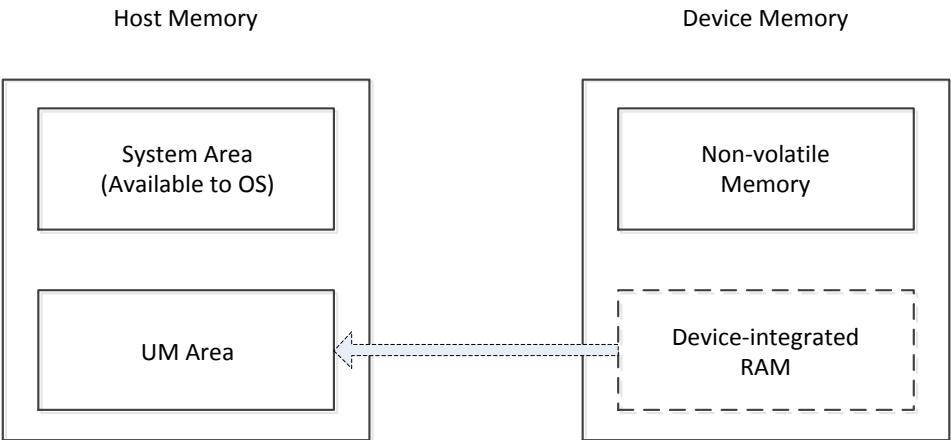


Figure 4.2 — UMA Memory Structure

4.2 Purpose of Unified Memory Support

The purpose of UM is to realize high performance at a reasonable cost. UM provides large working memory (e.g., several Mbytes), which is larger than that of Device-integrated RAM, to the Device.

L2P table access speed is important for high performance operation of non-volatile memory. Device-integrated RAM or single-level cell (SLC) area of non-volatile memory is usually used to speedup L2P table access. Buffer RAM is effective to realize high performance for small random data accesses. Device-integrated RAM is usually used for high performance storage like Solid State Drive (SSD). However such a Device-embedded RAM is costly, and it is sometimes difficult to use for embedded storage. UMA gives an appropriate solution to this problem.

The basic concept of UMA is supported by high-speed communication of UFS. The speed is slightly slower than that of internal bus, but still much faster than that of non-volatile memory access. Thus UM can work as working memory of Device, and it is a best use of hi-speed feature of UFS.

Figure 4.3 shows Inter-Memory Communication of Device-Integrated RAM Buffering in SCSI WRITE, and Figure 4.4 shows Inter-Memory Communication of UM Buffering. The difference in SCSI data phase is that the UFS link is used for the former and the Host internal bus is used for the latter. In write buffer flush, to the contrary, the Device internal bus and the UFS link are used, respectively.

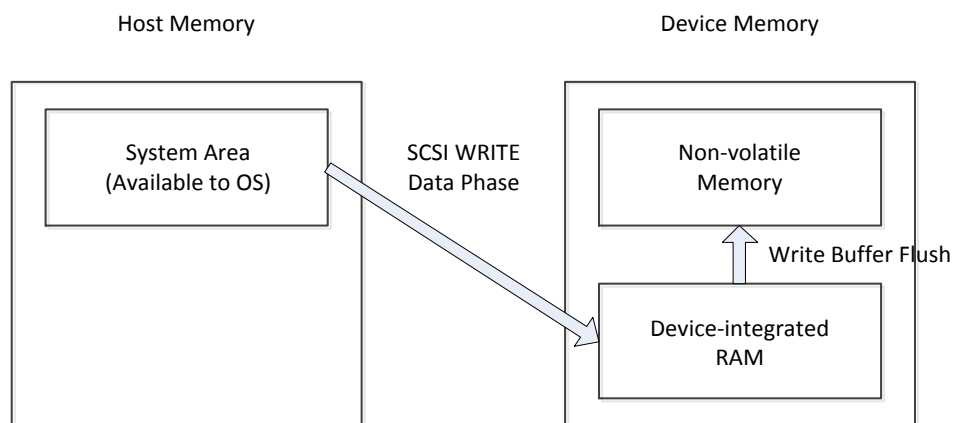


Figure 4.3 — Inter-Memory Communication of Device-Integrated RAM Buffering

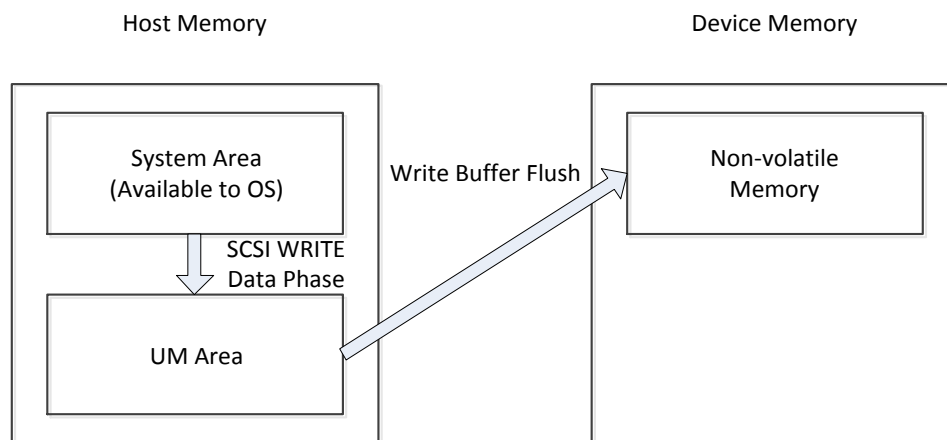


Figure 4.4 — Inter-Memory Communication of UM Buffering

4.3 Framework of Unified Memory

Today's computer usually has the virtual memory architecture, where CPU accesses the system memory using virtual address. However devices connected to the internal bus access the memory using either virtual address or physical address, depending on the position of the memory management unit. Using physical address is more common, and it is assumed in this section.

4.3.1 Host Memory Map

UM is a part of Host memory, more precisely a part of Host physical memory. The Host memory has at least the system memory that is managed by the operating system. Some parts of the Host memory may be used for other dedicated purposes, e.g., buffering data of devices, and UM is included as one of them. The UM Base Address Register points to the start address of UM (UM Base Address) as shown in Figure 4.5.

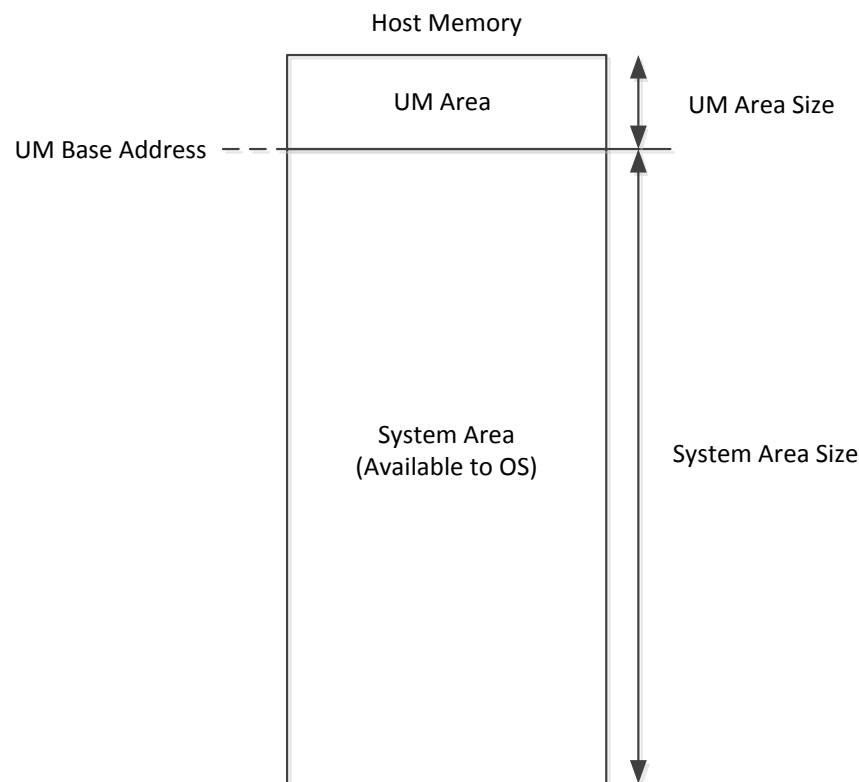


Figure 4.5 — Host Memory Map (Physical)

The UM Area Size is decided by the Host, after referring to `dMinUMAreaSize` in the Device Descriptor (See Table 9.1 for detail.), and is told to the Device in `dUMAreaSize` in the Attributes (See Table 9.3 for detail.).

4.3.2 Configuration of Unified Memory

UM is used as a working memory of Device. The UM may contain the following items:

- Logical-to-Physical address translation table (L2P Table) cache,
- Write Buffer (WB) cache,
- L2P Table cache tag,
- WB cache tag.

Some of the above may exist in the Device, or more items may be contained in the UM; it is an implementation matter, and the standard is UM content agnostic. The UM accesses are performed only using physical addresses. Only offsets from the UM Base Address are specified in the command from the Device to the Host, and the Device is responsible for all of the UM operations and all contents of the UM.

4.4 Speed Estimation

It is difficult to estimate operation speed correctly because it varies according to running applications. Thus benchmark tests are adopted to estimate the speed.

The following three parameters give hints of speed estimation:

- Bandwidth,
- Latency,
- Number of outstanding commands.

Bandwidth depends on the Gear of UFS, e.g., 2.9 Gbps with Gear 2 (single lane) and 5.8 Gbps with Gear 3 (single lane).

The definition of latency is shown in Figure 4.6.

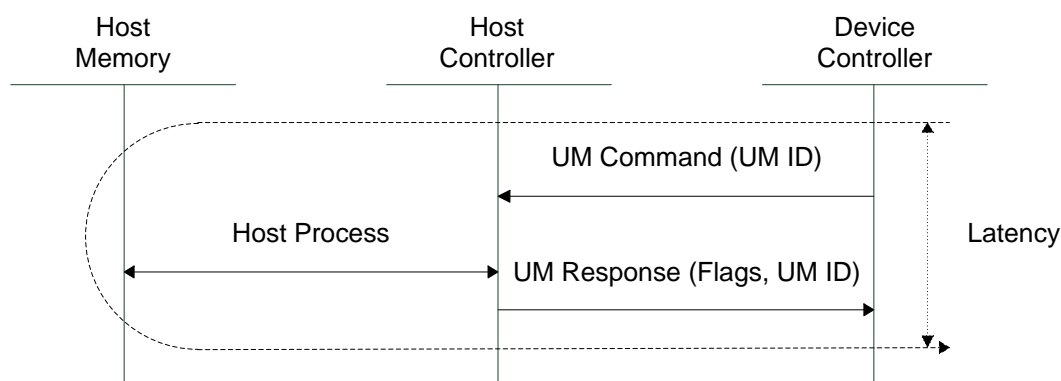


Figure 4.6 — Definition of Latency

Latency can be partly hidden by issuing UM commands in parallel, i.e., multiple outstanding commands (See Figure 4.7 for the definition of multiple outstanding commands.). Thus the number of outstanding commands is important. In this version of the standard, 32 is the default.

4.4 Speed Estimation (cont'd)

The concrete parameter values of a product should be specified in the data sheet provided by the manufacturer.

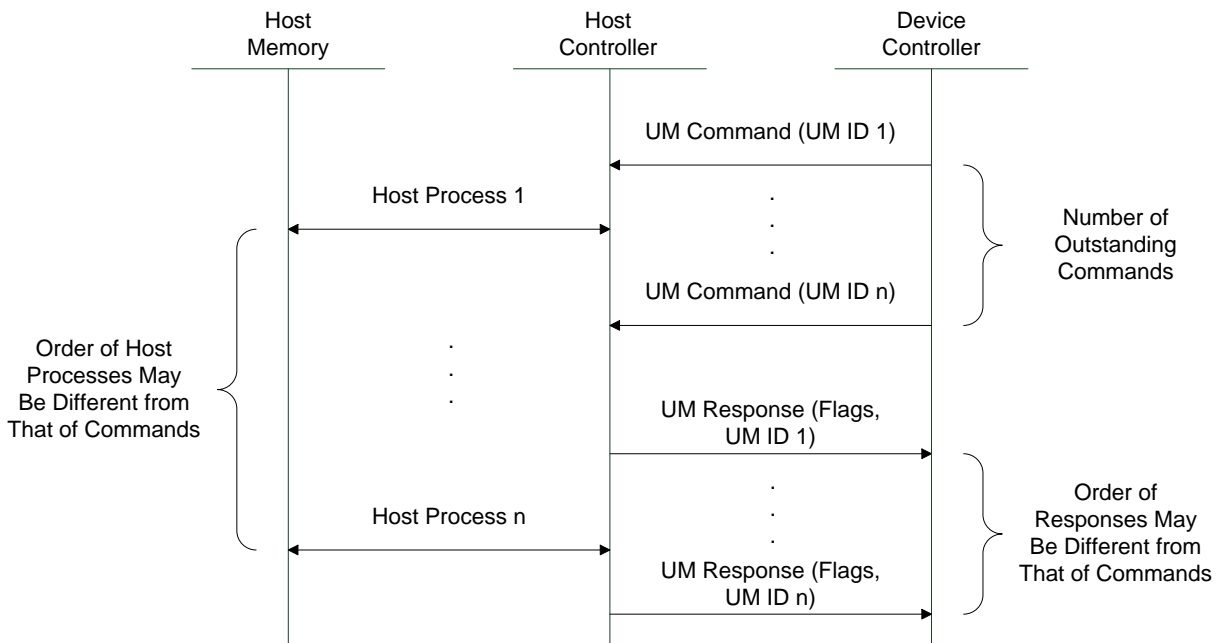


Figure 4.7 — Definition of Multiple Outstanding Commands

5 MODIFIED UFS ARCHITECTURE OVERVIEW

5.1 Modified UFS Top level Architecture

Figure 5.1 shows the top level architecture of Modified Universal Flash Storage (UFS) with Unified Memory Extension.

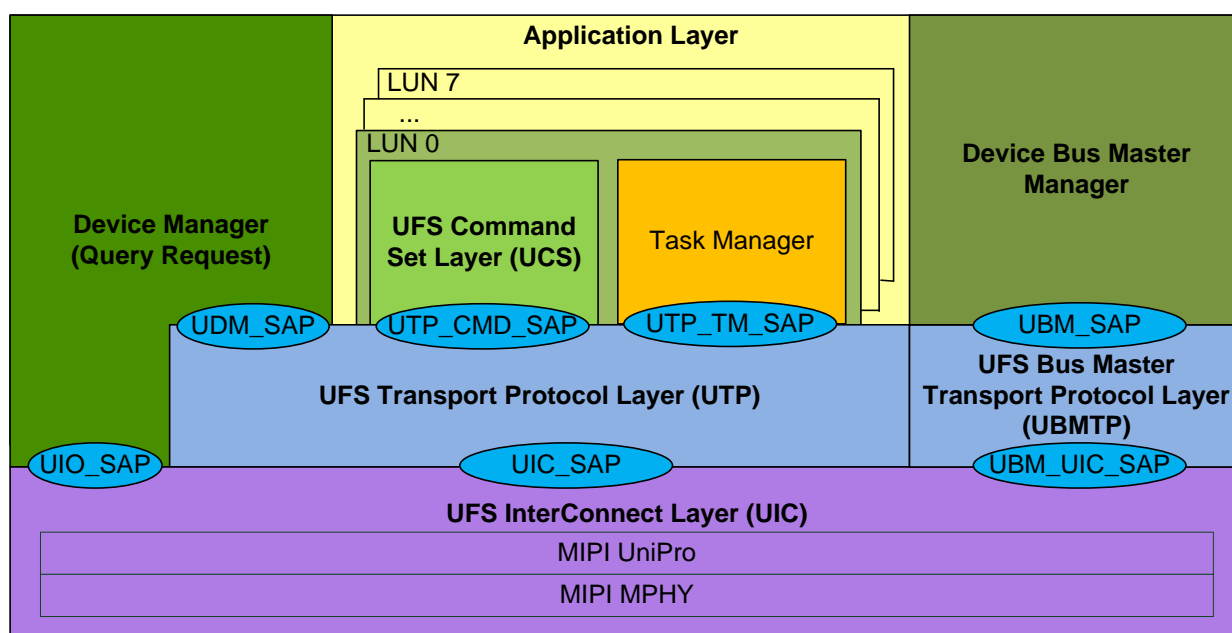


Figure 5.1 — Top Level Architecture of Modified UFS with Unified Memory Extension

UFS Command Set (UCS) is based on SCSI architectural model, but note that Device Bus Master Manager is outside it.

5.1.1 Application Layer

The application layer consists of the Device Manager, the UFS Command Set (UCS) layer, the Task Manager and the Device Bus Master Manager. The UCS layer will handle the normal commands like READ, WRITE, etc. UFS may support multiple command sets, and the UM command set is one of them. The Task Manager handles commands meant for command queue control. The Device Manager will provide device level control like Query Request and lower level link-layer control. The Device Bus Master Manager handles the UM operation commands.

5.1.2 Device Bus Master Manager

All UM commands are initiated by the Device, and the Device Bus Master Manager is primarily responsible for UM operation. The Host Device Bus Master Manager supports up to “32 x (bMaxUMPIURequests + 1)” outstanding requests from the Device. This Attribute is defined in Table 9.3.

The UM operations are classified into the following 3 categories:

- UM Read,
- UM Write,
- Memory Copy.

5.1.3 Service Access Points

As seen from the diagram the Device Bus Master Manager interacts with lower layers using the service access point,

- UBM_SAP.

UBM_SAP is the service access point exposed by the UBMTTP for the Device Bus Master Manager to allow handling of UM operations.

5.1.4 UFS Bus Master Transport Protocol Layer

The UFS Bus Master Transport Protocol (UBMTTP) layer provides services to the higher layer. UMPIU is “UM Protocol Information Unit” which is exchanged between UBMTTP layers of Device and Host. For example, if the Device side UBMTTP receives the request from application layer or Device Bus Master Manager, the UBMTTP generates a UMPIU for that request and transports the generated UMPIU to the peer UBMTTP in Host side. The UBMTTP layer provides the UFS Bus Master Service Access Point (UBM_SAP).

5.2 UFS Bus Master Transport Protocol (UBMTTP) Layer

As mentioned previously, the Transport Layer is responsible for encapsulating the protocol into the appropriate frame structure for the interconnect layer. UFS is protocol agnostic and thus any protocol will need the appropriate translation layer. For this version of UFS standard, there are UTP (UFS Transport Protocol) layer and UBMTTP (UFS Bus Master Transport Protocol) layer.

In this version of the standard, all accesses through UTP layer are only SCSI-based, but UM operations through UBMTTP are not SCSI-based.

6 UFS UIC LAYER: MIPI M-PHY AND MIPI UNIPRO

6.1 Overview

As described in Section 5 (Figure 5.1), the UFS Interconnect Layer is commonly used in both normal UFS and UME, which are subject to all functions and properties of the layer.

6.2 HIBERNATE Negotiation

HIBERNATE is a power management scheme that the UniPro stack inside the UIC layer provides. It is used to reduce power consumption when there is no communication traffic. It is defined in the UniPro specification. The specification defines the requirement of negotiation in the application layer before putting the link into the HIBERNATE mode, but the negotiation itself was not defined in UFS. On one hand, if the communications through the link are only SCSI commands, the Host knows everything about the communications and no negotiation is necessary. On the other hand, if some of the communications are Device initiated, such as UMPIU, the negotiation is essential.

The Flags “fSuspendUM” and “fUMSuspended” are used for the negotiation purpose, which is defined in the UniPro specifications. The Flags operation is expected to be faster than flushing dirty data in UM. UM suspension means that there is no UMPIU in operation and that there will be no UMPIU. Data in UM and/or outstanding UM operations may exist while suspended.

The basic flag operation principle is as follows:

The Host sets “fSuspendUM” to request the Device to suspend UM operation, or resets “fSuspendUM” to tell the Device that UM may operate. The Device shall set “fUMSuspended” to show UM has been suspended. The Host checks “fUMSuspended” before putting the link into HIBERNATE mode.

Flag manipulation sequence examples before and after HIBERNATE are shown in Figure 6.1 and Figure 6.2, respectively. When the Host will put the link line into the HIBERNATE mode, it sets “fSuspendUM” and checks “fUMSuspended”. If there is no on-going UMPIU in the device, the Device shall immediately set “fUMSuspended”. Otherwise, “fUMSuspended” remains reset. If “fUMSuspended” is set, the Host may put the link line into HIBERNATE. Otherwise, the Host waits for it to be set.

After the Host puts the link line out of the HIBERNATE mode, it resets “fSuspendUM”. The Device shall reset “fUMSuspended” before it sends QUERY RESPONSE in return to QUERY REQUEST to reset “fSuspendUM”.

6.2 HIBERNATE Negotiation (cont'd)

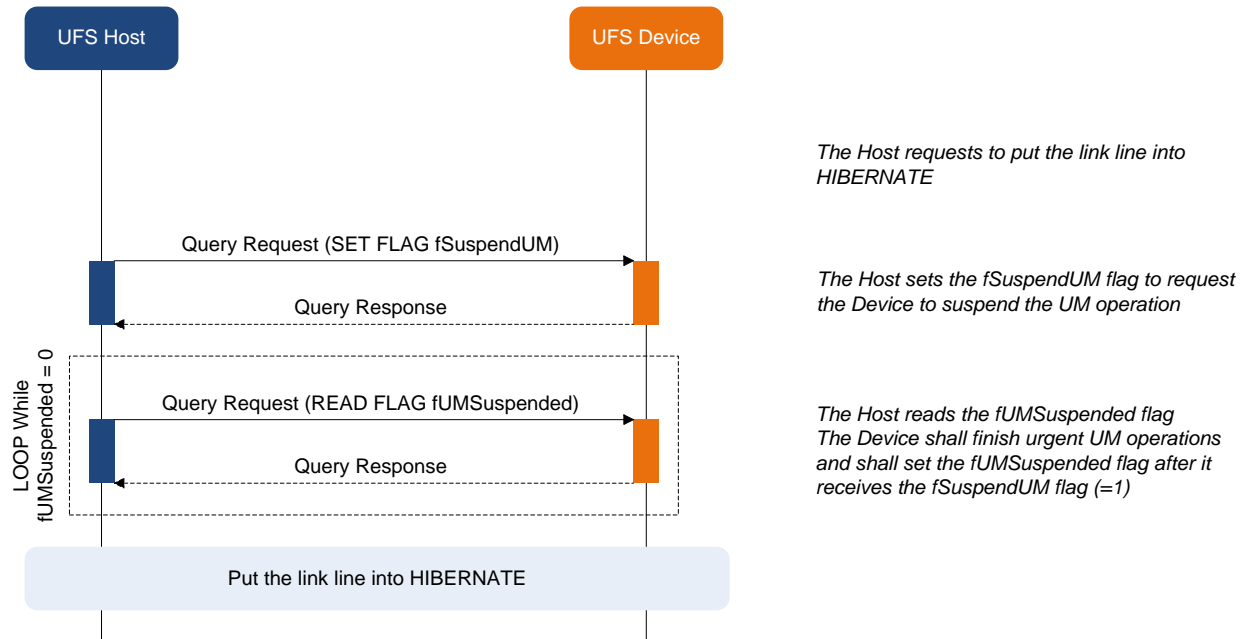


Figure 6.1 — Flag Manipulation before HIBERNATE

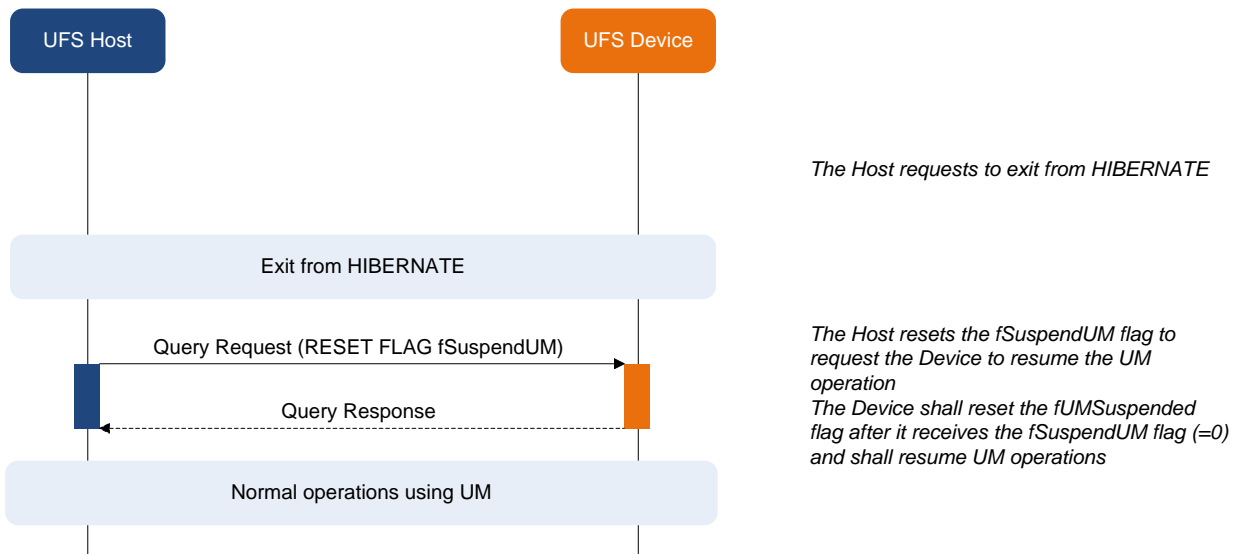


Figure 6.2 — Flag Manipulation after HIBERNATE

7 UFS BUS MASTER TRANSPORT PROTOCOL (UBMTP) LAYER

7.1 Overview

In this version of the standard, Unified Memory operations use CPort 1 / TC1 and CPort 2 / TC0. All UM commands use CPort 1 / TC1, and all data packets use either CPort 2 / TC0 or CPort 1 / TC1. All UM commands are transmitted via Traffic Class (TC) 1. Depending on the priority of a data transfer, the ACCESS UM BUFFER UMPIU offers the ability to select the Traffic Class in which the subsequent data shall be transmitted. This is done via the P Flag (e.g., “P = 1” means CPort 1 / TC1. Details of the P Flag are described in 7.3.6). To avoid complex dynamic Traffic Class switching, every Traffic Class used by UM operations is bound to a fixed CPort. Therefore the P Flag also selects the CPort which is utilized for the data transfer. However bCPortConfEn (See Table 9.1.) exists in the Device Descriptor for more flexible CPort assignment in a future version of the standard.

UM operations are always initiated by the Device, and the Host responds to these requests.

7.2 Unified Memory Protocol Information Unit

The Unified Memory transactions are grouped into data structures called Unified Memory Protocol Information Unit (UMPIU).

7.2.1 UMPIU Transaction Codes

Every UMPIU data structure contains a Transaction Code. This code defines the content and implied function or use of the UMPIU data structure. The following table lists currently defined transaction codes. UPIU and UMPIU transaction codes currently do not overlap, but may overlap since UPIU transaction codes are valid only on CPort 0 and UMPIU transaction codes are valid only on CPorts 1 and 2.

Table 7.1 — UMPIU Transaction Codes

Device to Host	Transaction Code	Host to Device	Transaction Code
COPY DATA	00 1000b	ACKNOWLEDGE COPY	10 1000b
ACCESS UM BUFFER	00 1001b	Reserved	10 1001b
UM DATA IN	00 1010b	UM DATA OUT	10 1010b
WRITE UM BUFFER	00 1011b	ACKNOWLEDGE UM BUFFER	10 1011b
UM COPY DATA	00 1100b	Reserved	10 1100b
Reserved	Others	Reserved	Others

NOTE 1 Bit 5 of the Transaction Code indicates the direction of flow and the originator of the UMPIU: when equal “0” the originator is the Device , when equal “1” the originator is the Host.

7.2.1 UMPIU Transaction Codes (cont'd)**Table 7.2 — UMPIU Transaction Code Definitions**

UMPIU Data Structure	Description
COPY DATA	The COPY DATA transaction originates in the Device and is sent to the Host. A COPY DATA command is used to copy data between a UFS data buffer inside the system memory and the UM area. In contrast to the UM COPY DATA command, the buffer address in the host memory is automatically extracted by the Host from the original transfer request.
UM COPY DATA	The UM COPY DATA transaction originates in the Device and is sent to the Host. A UM COPY DATA command is used to copy data from a source location inside the UM area into a destination location within the UM area. In contrast with the COPY DATA command, the Device directly provides the source address. This is a generic copy command.
ACKNOWLEDGE COPY	The ACKNOWLEDGE COPY transaction originates in the Host and is sent back to the Device. This is used to acknowledge COPY DATA and UM COPY DATA commands.
ACCESS UM BUFFER	The ACCESS UM BUFFER originates in the Device and is sent to the Host. An ACCESS UM BUFFER is used to initiate data transfer between the Device and UM. Actual data transfer is done through UM DATA IN / OUT UMPIUs.
WRITE UM BUFFER	The WRITE UM BUFFER originates in the Device and is sent to the Host. A WRITE UM BUFFER command is used to write a small amount of data (e.g., up to 512 bytes). The data is directly incorporated into the WRITE UM BUFFER command, which eliminates the need for an additional UM DATA IN UMPIU. Only write accesses are supported.
ACKNOWLEDGE UM BUFFER	The ACKNOWLEDGE UM BUFFER transaction originates in the Host and is sent back to the Device. This is used to acknowledge ACCESS UM BUFFER (in write case; UM DATA OUT is used for read) / WRITE UM BUFFER command.
UM DATA OUT	The UM DATA OUT transaction originates in the Host and is sent back to the Device. This is the actual data transfer from Host to Device.
UM DATA IN	The UM DATA IN transaction originates in the Device and is sent to the Host. This UMPIU is used to carry the actual data transfer that has been requested by an ACCESS UM BUFFER command. The direction is from Device to Host.

7.3 General Unified Memory Protocol Information Unit Format

Table 7.3 represents the general structure of a UMPIU. All UMPIUs contain a fixed size and location basic header and additional fields as required to support the transaction type.

Table 7.3 — General Format of the Unified Memory Protocol Information Unit

General UMPIU Format			
0 Transaction Type	1 Flags	2 Reserved	3 UM ID
4 (MSB) Source UM Area Offset / LUN	5 Source UM Area Offset / Task Tag	6 Source UM Area Offset / Data Segment Length	7 (LSB) Source UM Area Offset / Data Segment Length
8 (MSB)	9	10	11 (LSB)
Source/Destination UM Area Offset			
12 (MSB)	13	14	15 (LSB)
Data Length			
16			19
Data			
...			
Data Segment Length + 12	Data Segment Length + 13	Data Segment Length + 14	Data Segment Length + 15
Data			

7.3.1 Overview

UMPIU total size will vary depending upon the UMPIU transaction type, but all UMPIU sizes will be an integer multiple of 32-bits, meaning they will be addressed on a 4 byte (DWORD) boundary. If the aggregation of data and header segments does not end on a 32-bit boundary then additional padding will be added to round up the UMPIU to the next 32-bit, 4 byte address boundary.

The UMPIU size can be fixed or variable depending upon the Transaction Type field and extension flags. Some Transaction Types will have different lengths for the same code whereas others will always be a fixed size.

7.3.2 Basic Header Format

This is the format of the basic header contained within every UMPIU structure. This data packet will be sent between Devices and Hosts and will be part of a larger function specific UMPIU. There is enough information in this header to allow the Device or the Host to track the destination and the source, the function request, if additional data and parameters are required and whether they are included in this UMPIU or will follow in subsequent UMPIUs.

The smallest sized UMPIU is currently defined to have 16 bytes. The 16 bytes area contains the basic header plus additional fields. This means that the smallest datum sent over the Service Delivery Subsystem will be 16 bytes.

Table 7.4 — Basic Header Format

Basic UMPIU Header Format			
Transaction Type	Flags	Reserved	UM ID

a) Transaction Type

The Transaction Type indicates the type of request or response contained within the data structure. The Transaction Type contains an opcode as defined in 7.2.1, UMPIU Transaction Codes.

Table 7.5 — Transaction Type Format

Transaction Type Bits							
7	6	5	4	3	2	1	0
Reserved		Transaction Code					

b) Transaction Code

The Transaction Code indicates the operation that is represented within the data fields of the UMPIU and the number and location of the defined fields within the UMPIU.

c) Flags

The content of the Flags field vary with the Transaction Type opcode.

Table 7.6 — UMPIU Flags

UMPIU Type	Operational Flags				Reserved			
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
COPY DATA	-	R	-	-	-	-	-	-
UM COPY DATA	-	-	-	-	-	-	-	-
ACKNOWLEDGE COPY	-	I	-	E	-	-	-	-
ACCESS UM BUFFER	-	R	W	P	-	-	-	-
WRITE UM BUFFER	-	-	-	-	-	-	-	-
ACKNOWLEDGE UM BUFFER	-	I	-	E	-	-	-	-
UM DATA OUT	-	I	-	E	-	-	-	-
UM DATA IN	-	-	-	-	-	-	-	-

7.3.2 Basic Header Format (cont'd)

d) Reserved

All fields marked as reserved shall contain a value of zero.

e) UM ID

UM ID is generated by the Device. This field is used for identifying the request of the Device. The Device shall not generate overlapping UM IDs for UM commands (ACCESS UM BUFFER, WRITE UM BUFFER, COPY DATA, UM COPY DATA). Response and data packets shall use the same UM ID as the associated UM commands.

7.3.3 COPY DATA UMPIU

7.3.3.1 Overview

The COPY DATA UMPIU is a request by the Device to the Host to copy data between the system memory and the Unified Memory. The COPY DATA UMPIU (with Flags.R = 0) is a request to copy data from the source location inside the system memory to a destination location inside the Unified Memory. This UMPIU is mainly used in manipulating random write, where data are buffered in the Unified Memory. Then COPY DATA UMPIU (with Flags.R = 0) is the data phase of the corresponding SCSI WRITE command. The COPY DATA UMPIU (with Flags.R = 1) is a reverse operation, a request to copy data from the source location in the Unified Memory to a destination location in the system memory. Therefore it is the data phase of the corresponding SCSI READ command. Only a single COPY DATA UMPIU is allowed for a SCSI READ / WRITE.

Table 7.7 — COPY DATA UMPIU

COPY DATA UMPIU			
0 0000 1000b	1 Flags	2 Reserved	3 UM ID
4 LUN	5 Task Tag	6 Reserved	7 Reserved
8 (MSB)	9	10	11 (LSB)
Source/Destination UM Area Offset			
12 (MSB)	13	14	15 (LSB)
Data Length			

7.3.3.2 Basic Header

The first 4 bytes of the COPY DATA UMPIU contain the Basic Header as described in 7.3.2, Basic Header Format. Specific details are as follows:

a) Transaction Type

A type code value of 0000 1000b indicates a COPY DATA UMPIU.

b) LUN

LUN is copied from the original SCSI command.

c) Task Tag

The Task Tag is copied from the original SCSI command.

d) Source/Destination UM Area Offset

Source/Destination UM Area Offset specifies the source/destination address offset of UM in byte. The Host has a Base Address Register to show the starting address of the UM area, and the actual source/destination address is the result of adding the content of the register and the offset.

e) Data Length

Data Length is the length of copied Data in byte.

f) Flags

Table 7.8 — Flags Definition of ACKNOWLEDGE COPY UMPIU

Flag	Description
Flags.R	A value of “0” in the .R flag indicates that a data copy operation from the system area to the UM area is required. This case corresponds to a SCSI WRITE command. A value of “1” in the .R flag indicates that a data copy operation from the UM area to the system area is required. This case corresponds to a SCSI READ command.

7.3.4 UM COPY DATA UMPIU

7.3.4.1 Overview

The UM COPY DATA UMPIU is a request by the Device to the Host to perform generic memory copy in the Unified Memory.

Table 7.9 — UM COPY DATA UMPIU

UM COPY DATA UMPIU			
0 0000 1100b	1 Flags	2 Reserved	3 UM ID
4 (MSB)	5	6	7 (LSB)
Source UM Area Offset			
8 (MSB)	9	10	11 (LSB)
Destination UM Area Offset			
12 (MSB)	13	14	15 (LSB)
Data Length			

7.3.4.2 Basic Header

The first 4 bytes of the UM COPY DATA UMPIU contain the Basic Header as described in 7.3.2, Basic Header Format. Specific details are as follows:

a) Transaction Type

A type code value of 0000 1100b indicates a UM COPY DATA UMPIU.

b) Source UM Area Offset

Source UM Area Offset specifies the source address offset of UM in byte. The Host has a Base Address Register to show the starting address of the UM area, and the actual source address is the result of adding the content of the register and the offset.

c) Destination UM Area Offset

Destination UM Area Offset specifies the destination address offset of UM in byte. The Host has a Base Address Register to show the starting address of the UM area, and the actual destination address is the result of adding the content of the register and the offset.

d) Data Length

Data Length is the length of copied Data in byte.

7.3.5 ACKNOWLEDGE COPY UMPIU

7.3.5.1 Overview

The ACKNOWLEDGE COPY UMPIU contains the basic UMPIU header plus additional information to indicate the status of the associated COPY DATA or UM COPY DATA UMPIU. The ACKNOWLEDGE COPY UMPIU is sent by the Host to the Device in response to a COPY DATA or a UM COPY DATA UMPIU to indicate that the request has been executed either successfully or unsuccessfully. Error cases are flagged inside the Flags field.

Table 7.10 — ACKNOWLEDGE COPY UMPIU

ACKNOWLEDGE COPY UMPIU			
0 0010 1000b	1 Flags	2 Reserved	3 UM ID
4 Reserved	5 Reserved	6 Reserved	7 Reserved
8 Reserved	9 Reserved	10 Reserved	11 Reserved
12 Reserved	13 Reserved	14 Reserved	15 Reserved

7.3.5.2 Basic Header

The first 4 bytes of the ACKNOWLEDGE COPY UMPIU contain the Basic Header as described in 7.3.2, Basic Header Format. Specific details are as follows:

a) Transaction Type

A type code value of 0010 1000b indicates an ACKNOWLEDGE COPY UMPIU.

b) Flags

Table 7.11 — Flags Definition of ACKNOWLEDGE COPY UMPIU

Flag	Description
Flags.I	The .I flag will be set to “1” to indicate that an illegal address access error occurred during the task execution of a COPY DATA or a UM COPY DATA command: the COPY DATA or the UM COPY DATA command requested to access the offset beyond the maximum offset. Setting the .I flag to “1” additionally sets the .E flag to “1”.
Flags.E	The .E flag will be set to “1” to indicate that an error occurred during the task execution of a COPY DATA or a UM COPY DATA command. It shows any error detected in the Host. The .E Flag shall be set to “1” if .I Flag is set.

7.3.6 ACCESS UM BUFFER UMPIU

7.3.6.1 Overview

The ACCESS UM BUFFER UMPIU is a request by the Device to the Host to access (read / write) data from / to the Unified Memory. The write command shall be acknowledged by an ACKNOWLEDGE UM BUFFER UMPIU after the UM DATA IN UMPIU. In case of the read command, the Host shall not send an ACKNOWLEDGE UM BUFFER UMPIU. Information about success or failure of the read command shall be provided inside the “Flag” of the UM DATA OUT UMPIU. UM DATA IN / OUT UMPIUs are sent via either CPort 2 (TC0) or CPort 1 (TC1) according to the P flag. The P flag shall be set to “1” if there is no mechanism in the Host to guarantee the processing order of two UMPIUs sent in different Traffic Classes.

Table 7.12 — ACCESS UM BUFFER UMPIU

ACCESS UM BUFFER UMPIU			
0 0000 1001b	1 Flags	2 Reserved	3 UM ID
4 Reserved	5 Reserved	6 Reserved	7 Reserved
8 (MSB)	9	10	11 (LSB)
Source/Destination UM Area Offset			
12 (MSB)	13	14	15 (LSB)
Data Length			

7.3.6.2 Basic Header

The first 4 bytes of the ACCESS UM BUFFER UMPIU contain the Basic Header as described in 7.3.2, Basic Header Format. Specific details are as follows:

a) Transaction Type

A type code value of 0000 1001b indicates an ACCESS UM BUFFER UMPIU.

b) Source/Destination UM Area Offset

Source/Destination UM Area Offset specifies the source/destination address offset of UM in byte. The Host has a Base Address Register to show the starting address of the UM area, and the actual source/destination address is the result of adding the content of the register and the offset.

c) Data Length

Data Length is the length of copied Data in byte. The upper 15 bits shall be “0” (maximum length is 64 Kbytes). The minimum value of Data Length is 1.

7.3.6.2 Basic Header (cont'd)**d) Flags****Table 7.13 — Flags Definition for ACCESS UM BUFFER UMPIU**

Flag	Description
Flags.R	A value of “1” in the .R flag indicates that the command requires a data READ operation (incoming data) from the Host to the Device. If .R is set to “1”, then .W must be set to “0”. If .R and .W are both set to “0”, then no data transfer is required for this command, and the Data Length field is ignored.
Flags.W	A value “1” in the .W flag indicates that the command requires a data WRITE operation (outgoing data) from the Device to the Host. If .W is set to “1”, then .R must be set to “0”. If .W and .R are set to “0”, then no data transfer is required for this command, and the Data Length field is ignored.
Flags.P	The .P flag shall be set to “1” to request subsequent UM DATA IN / OUT UMPIU in TC 1 via CPort 1. It shall be set to “1” if there is no mechanism in the Host to guarantee the processing order of two UMPIUs sent in different Traffic Classes. If the .P flag is set to “0”, the UM DATA IN / OUT UMPIU is in TC0 via CPort 2.

7.3.7 WRITE UM BUFFER UMPIU

7.3.7.1 Overview

The WRITE UM BUFFER UMPIU is a request by the Device to the Host to write data in the Unified Memory with the actual data inside the WRITE UM BUFFER UMPIU. The command shall be acknowledged by an ACKNOWLEDGE UM BUFFER UMPIU.

Table 7.14 — WRITE UM BUFFER UMPIU

WRITE UM BUFFER UMPIU			
0 0000 1011b	1 Flags	2 Reserved	3 UM ID
4 Reserved	5 Reserved	6 (MSB) Data Segment Length	7 (LSB) Data Segment Length
8 (MSB)	9	10	11 (LSB)
Destination UM Area Offset			
12 Reserved	13 Reserved	14 Reserved	15 Reserved
16			19
Data			
...			
Data Segment Length + 12	Data Segment Length + 13	Data Segment Length + 14	Data Segment Length + 15
Data			

7.3.7.2 Basic Header

The first 4 bytes of the WRITE UM BUFFER UMPIU contain the Basic Header as described in 7.3.2, Basic Header Format. Specific details are as follows:

a) Transaction Type

A type code value of 0000 1011b indicates a WRITE UM BUFFER UMPIU.

b) Data Segment Length

Data Segment Length is the length of data in byte. The value “0” means 65536, and the minimum length is “1”.

c) Destination UM Area Offset

Destination UM Area Offset specifies the destination address offset of UM in byte. The Host has a Base Address Register to show the starting address of the UM area, and the actual destination address is the result of adding the content of the register and the offset.

d) Data

This is the Data Segment area shall be padded out to the next nearest 4 byte boundary.

7.3.8 ACKNOWLEDGE UM BUFFER UMPIU

7.3.8.1 Overview

The ACKNOWLEDGE UM BUFFER UMPIU contains the basic UMPIU header plus additional information to indicate the completion status of the associated WRITE UM BUFFER or ACCESS UM BUFFER command.

Table 7.15 — ACKNOWLEDGE UM BUFFER UMPIU

ACKNOWLEDGE UM BUFFER UMPIU			
0 0010 1011b	1 Flags	2 Reserved	3 UM ID
4 Reserved	5 Reserved	6 Reserved	7 Reserved
8 Reserved	9 Reserved	10 Reserved	11 Reserved
12 Reserved	13 Reserved	14 Reserved	15 Reserved

7.3.8.2 Basic Header

The first 4 bytes of the ACKNOWLEDGE UM BUFFER UMPIU contain the Basic Header as described in 7.3.2, Basic Header Format. Specific details are as follows:

a) Transaction Type

A type code value of 0010 1011b indicates an ACKNOWLEDGE UM BUFFER UMPIU.

b) Flags

Table 7.16 — Flags Definition for ACKNOWLEDGE UM BUFFER UMPIU

Flag	Description
Flags.I	The .I flag will be set to “1” to indicate that an illegal address access error occurred during the task execution of an ACCESS UM BUFFER or a WRITE UM BUFFER command: the ACCESS UM BUFFER or the WRITE UM BUFFER command requested to access the offset beyond the maximum offset. Setting the .I flag to “1” additionally sets the .E flag to “1”.
Flags.E	The .E Flag will be set to “1” to indicate that an error occurred during the task execution of an ACCESS UM BUFFER or a WRITE UM BUFFER command. It shows any error detected in the Host. The .E Flag shall be set to “1” if .I Flag is set.

7.3.9 UM DATA OUT UMPIU

7.3.9.1 Overview

The UM DATA OUT UMPIU contains the basic UMPIU header plus additional information needed to manage the data transfer. The data transfer flows from the Host to the Device (READ). The UM DATA OUT UMPIU always contains a data segment. Note that it is not allowed to have a null data segment in the UM DATAOUT UMPIU: The value “0” in Data Segment Length means the size of the data is 64 Kbytes.

The UM DATA OUT UMPIU is sent in response to a Device generated ACCESS UM Buffer UMPIU with setting Flags.R to “1”.

Table 7.17 — UM DATA OUT UMPIU

UM DATA OUT UMPIU			
0 0010 1010b	1 Flags	2 Reserved	3 UM ID
4 Reserved	5 Reserved	6 (MSB) Data Segment Length	7 (LSB) Data Segment Length
8 Reserved	8 Reserved	10 Reserved	11 Reserved
12 Reserved	13 Reserved	14 Reserved	15 Reserved
16	Data		19
...			
Data Segment Length + 12	Data Segment Length + 13	Data Segment Length + 14	Data Segment Length + 15
Data			

7.3.9.2 Basic Header

The first 4 bytes of the ACCESS UM BUFFER UMPIU contain the Basic Header as described in 7.3.2, Basic Header Format. Specific details are as follows:

a) Transaction Type

A type code value of 0010 1010b indicates an UM DATA OUT UMPIU.

b) Data Segment Length

Data Segment Length is the length of data in byte. The value “0” means 65536, and the minimum length is “1”.

c) Data

This is the Data Segment area that contains the data payload. The maximum data payload size is 65536 bytes as shown in the above Data Segment Length. The Data Segment area always starts on a 4 byte boundary. The Data Segment area shall be entirely filled with data payload to a 4 byte boundary. If necessary, the Data Segment shall be padded out to the next nearest 4 byte boundary.

d) Flags

Table 7.18 — Flags Definition for UM DATA OUT UMPIU

Flag	Description
Flags.I	The .I flag will be set to “1” to indicate that an illegal address access error occurred during the task execution of an ACCESS UM BUFFER: the ACCESS UM BUFFER command requested to access the offset beyond the maximum offset. Setting the .I flag to “1” additionally sets the .E flag to “1”.
Flags.E	The .E Flag will be set to “1” to indicate that an error occurred during the task execution of an ACCESS UM BUFFER or a WRITE UM BUFFER command. It shows any error detected in the Host. The .E Flag shall be set to “1” if .I Flag is set.

7.3.10 UM DATA IN UMPIU

7.3.10.1 Overview

The UM DATA IN UMPIU contains the basic UMPIU header plus additional information needed to manage the data transfer. The data transfer flows from the Device to the Host (WRITE). The UM DATA IN UMPIU always contains a data segment. Note that it is not allowed to have a null data segment in the UM DATA IN UMPIU: The value “0” in the Data Segment Length means the size of the data is 64 Kbytes.

The UM DATA IN UMPIU is sent after a Device generated ACCESS UM BUFFER UMPIU with setting Flags.W to “1”. The command shall be acknowledged by an ACKNOWLEDGE UM BUFFER UMPIU.

Table 7.19 — UM DATA IN UMPIU

UM DATA IN UMPIU			
0 0000 1010b	1 Flags	2 Reserved	3 UM ID
4 Reserved	5 Reserved	6 (MSB) Data Segment Length	7 (LSB) Data Segment Length
8 (MSB)	9	10	11 (LSB)
Destination UM Area Offset			
12 Reserved	13 Reserved	14 Reserved	15 Reserved
16			19
Data			
...			
Data Segment Length + 12	Data Segment Length + 13	Data Segment Length + 14	Data Segment Length + 15
Data			

7.3.10.2 Basic Header

The first 4 bytes of the UM DATA IN UMPIU contain the Basic Header as described in 7.3.2, Basic Header Format. Specific details are as follows:

a) Transaction Type

A type code value of 0000 1010b indicates an UM DATA IN UMPIU.

b) Data Segment Length

Data Segment Length is the length of data in byte. The value “0” means 65536, and the minimum length is “1”.

c) Destination UM Area Offset

Destination UM Area Offset specifies the destination address offset of UM in byte. The Host has a Base Address Register to show the starting address of the UM area, and the actual destination address is the result of adding the content of the register and the offset.

d) Data

This is the Data Segment area that contains the data payload. The maximum data payload size is 65536 bytes as shown in the above Data Segment Length. The Data Segment area always starts on a 4 byte boundary. The Data Segment area shall be entirely filled with data payload to a 4 byte boundary. If necessary, the Data Segment area shall be padded out to the next nearest 4 byte boundary.

7.3.11 Unified Memory Basic Operations

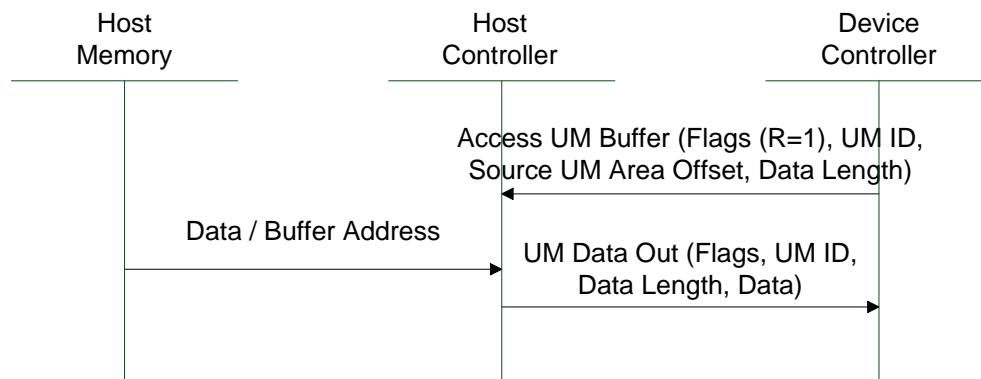
7.3.11.1 Overview

The UM operations are classified into the following 3 categories:

- Unified Memory Read: Transfer data from UM to Device,
- Unified Memory Write: Transfer data from Device to UM, and
- Memory Copy: Copy data either between System Memory and UM or inside UM.

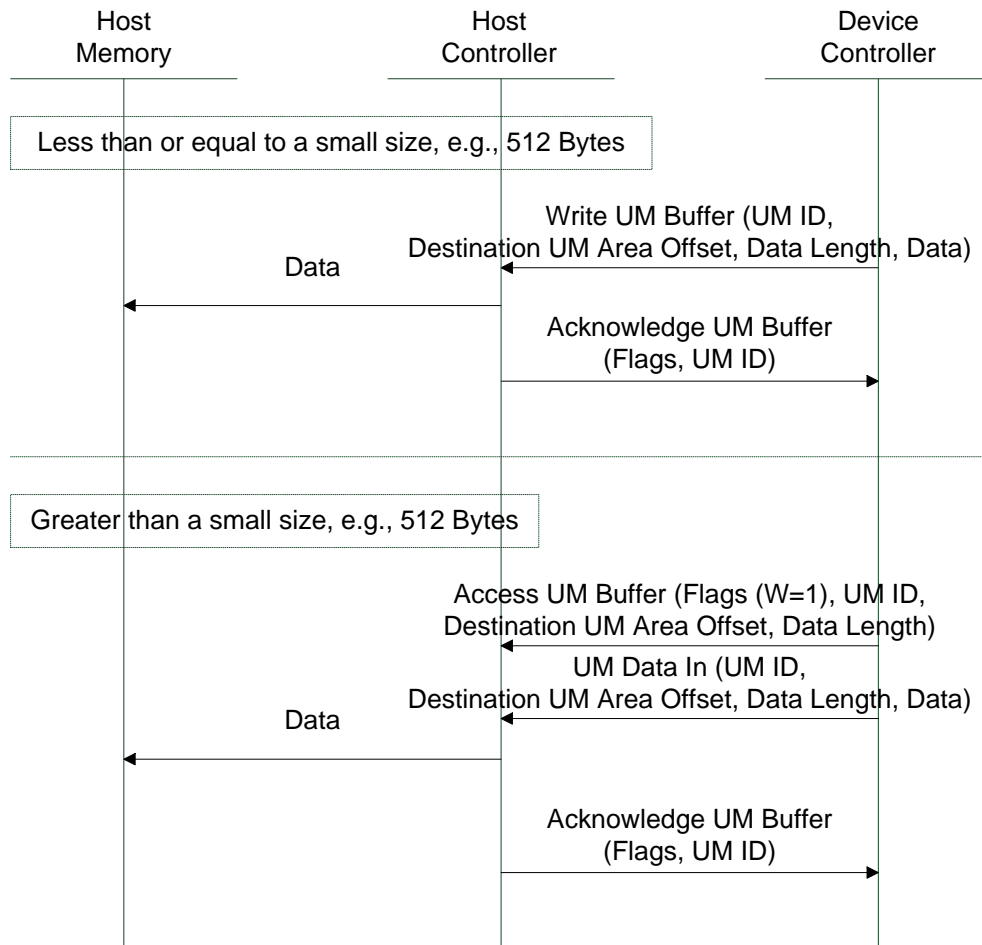
Actual data transfer between UM and Device uses two UMPIUs according to the direction of the data transfer. Status report uses two UMPIUs according to the commands.

Figure 7.1 shows Unified Memory Read, Figure 7.2 shows Unified Memory Write, Figure 7.3 shows System Memory to Unified Memory Copy, and Figure 7.4 shows Generic Unified Memory Copy.



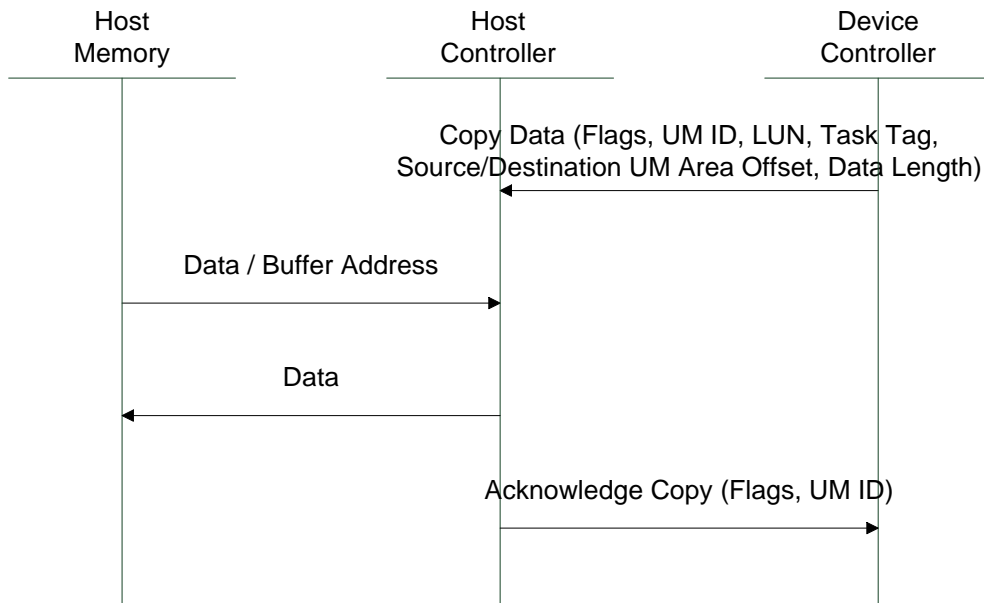
UM ID shall be the same.

Figure 7.1 — Unified Memory Read

7.3.11.1 Overview (cont'd)

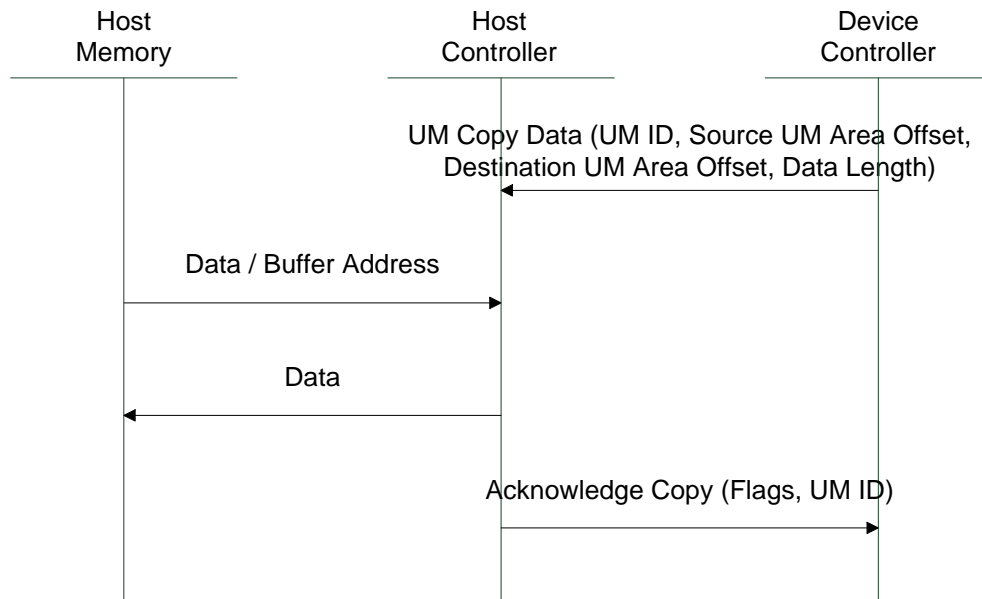
UM ID shall be the same.

Figure 7.2 — Unified Memory Write

7.3.11.1 Overview (cont'd)

UM ID shall be the same.

Figure 7.3 — Copy between System Memory and Unified Memory



UM ID shall be the same.

Figure 7.4 — Generic Copy in the Unified Memory

7.3.12 SCSI Read / Write and Unified Memory Operations

7.3.12.1 Overview

This informational section shows potential use cases of UM, which may or may not be implemented by the Device vendor.

The UM operation is transparent to the software and UFS driver. Due to this no additional load is added to the System Host.

In principle, Unified Memory may be used for any kinds of buffers that may exist in the Device, e.g., Logical to Physical (L2P) Table cache and Write Buffer. However, on one hand, implementers shall keep in mind that the latency for UM access might be higher and/or more volatile compared to the Device integrated RAM. Therefore the UM concept is especially useful for use cases that are less vulnerable to latency variations. In most cases access speed of non-volatile memory is much slower than that of UM, and L2P Table cache and Write Buffer are good examples. Note that caching large data is less effective because it tends to be flushed into the non-volatile memory sooner than small data.

On the other hand, UM may provide more amount of buffers than those made with the Device integrated RAM.

7.3.12.2 Use Case Examples

Figure 7.5 through Figure 7.10 show use cases corresponding to SCSI READ and WRITE, and Figure 7.11 shows L2P cache operation. The last 3 steps of Figure 7.7 shows Write Buffer cache flush. Note that this happens in only required cases, Write Buffer cache collision and before power down. It may be performed while the Device has free time. If the corresponding file has been deleted, the Host may issue a SCSI UNMAP command for invalidating the Write Buffer. It will reduce the cache flush time and will realize fast execution.

The SCSI data phase is actually performed by a COPY DATA UMPIU command as shown in Figure 7.7, Figure 7.8, and Figure 7.9.

7.3.12.2 Use Case Examples (cont'd)

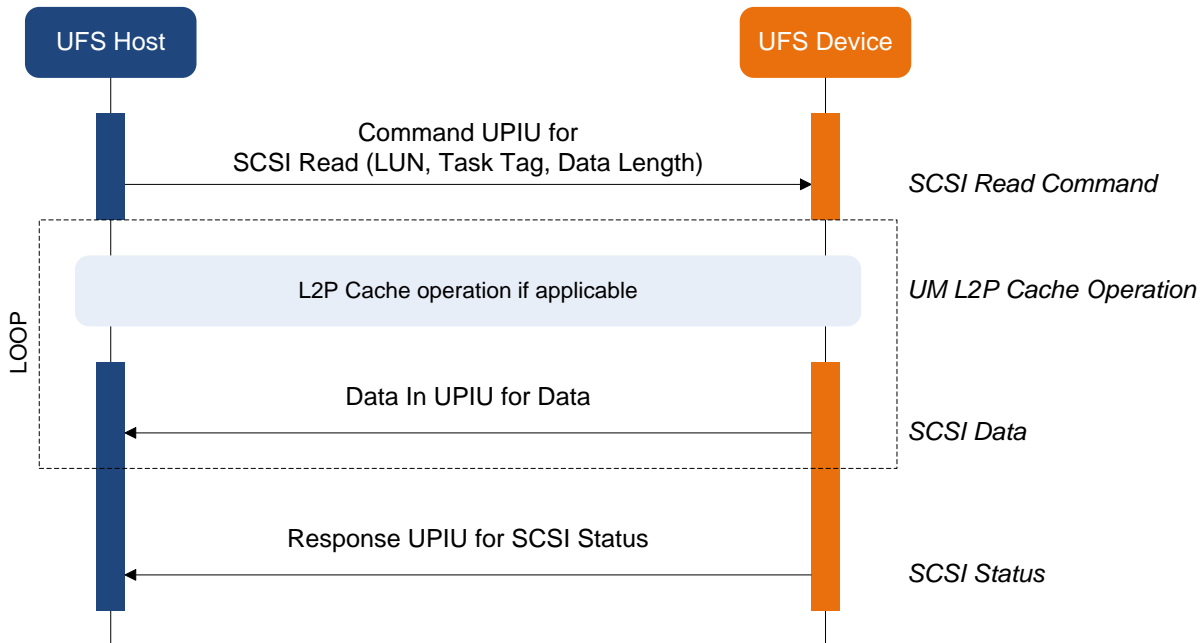


Figure 7.5 — SCSI Read (Non-Cached) and Subsequent Unified Memory Operations

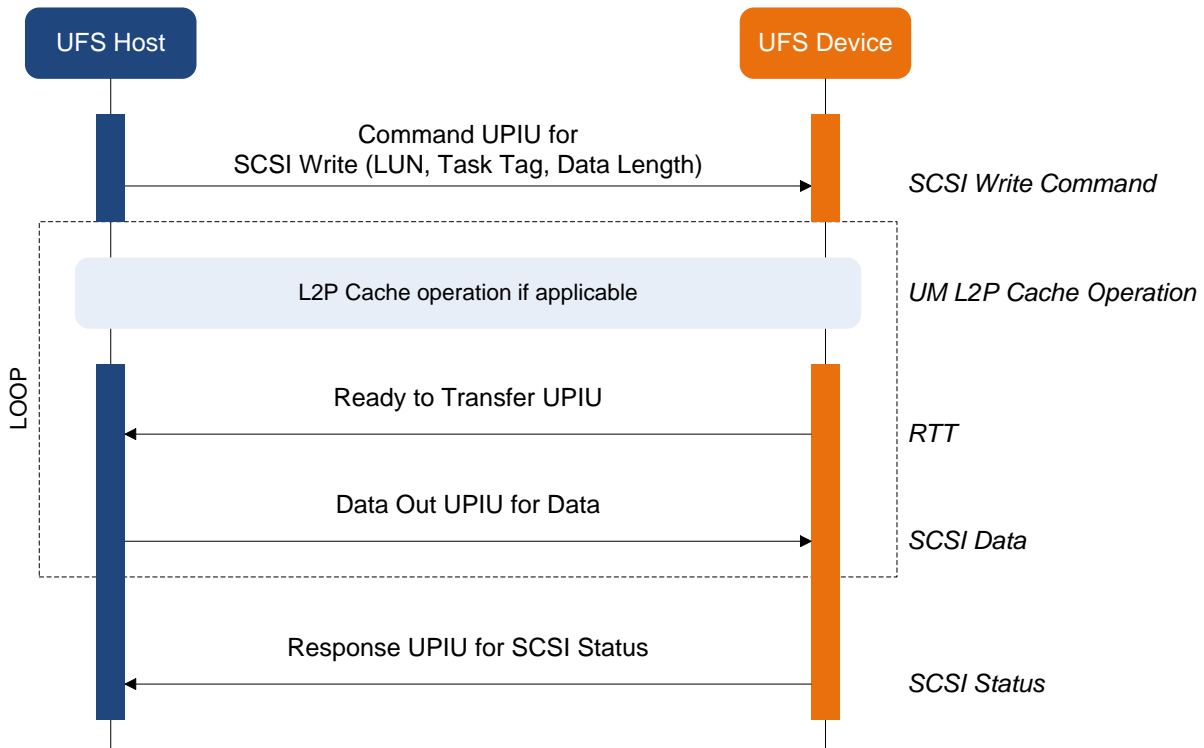


Figure 7.6 — SCSI Write (Non-Cached) and Subsequent Unified Memory Operations

7.3.12.2 Use Case Examples (cont'd)

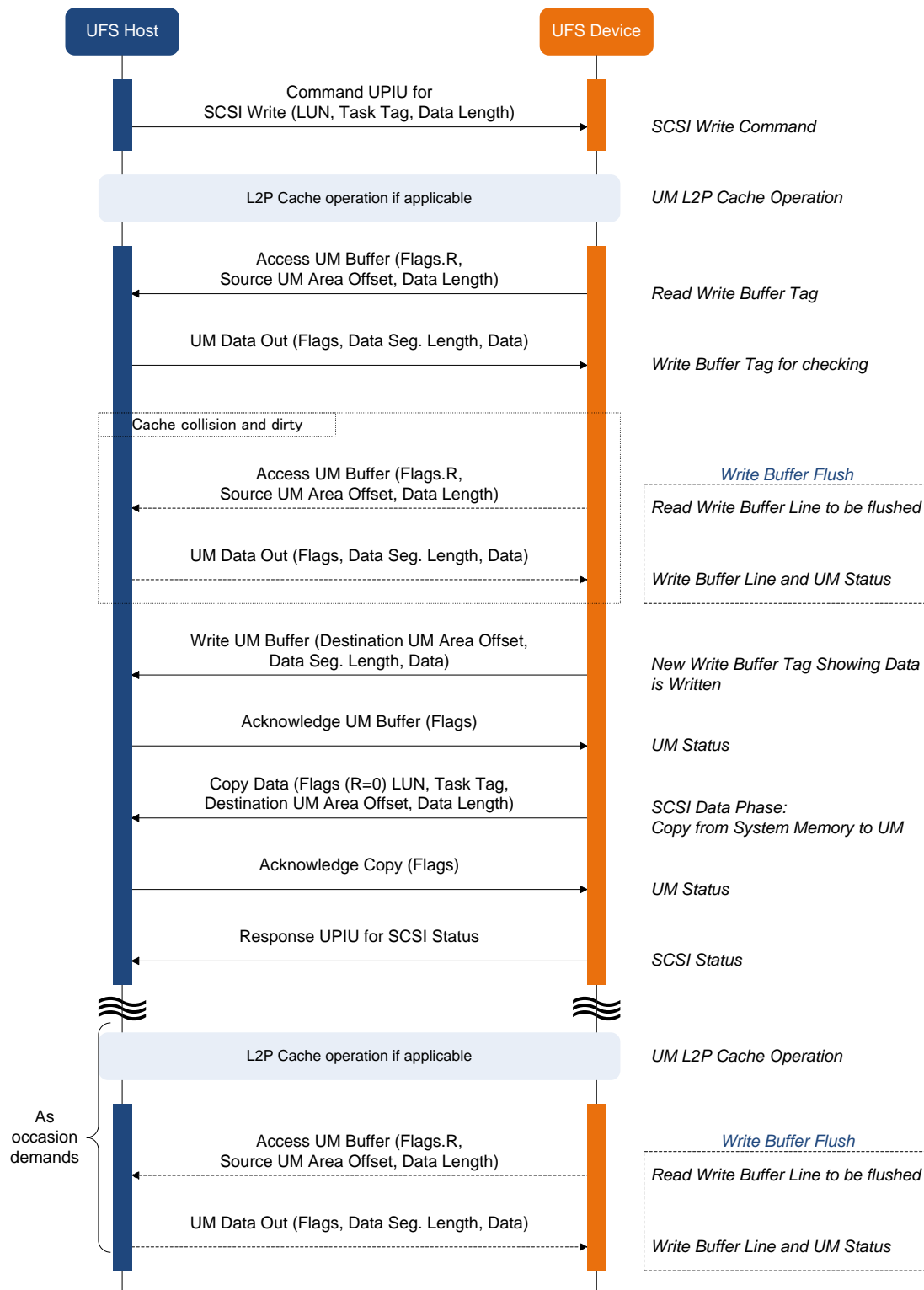


Figure 7.7 — SCSI Write (Cached) and Subsequent Unified Memory Operations

7.3.12.2 Use Case Examples (cont'd)

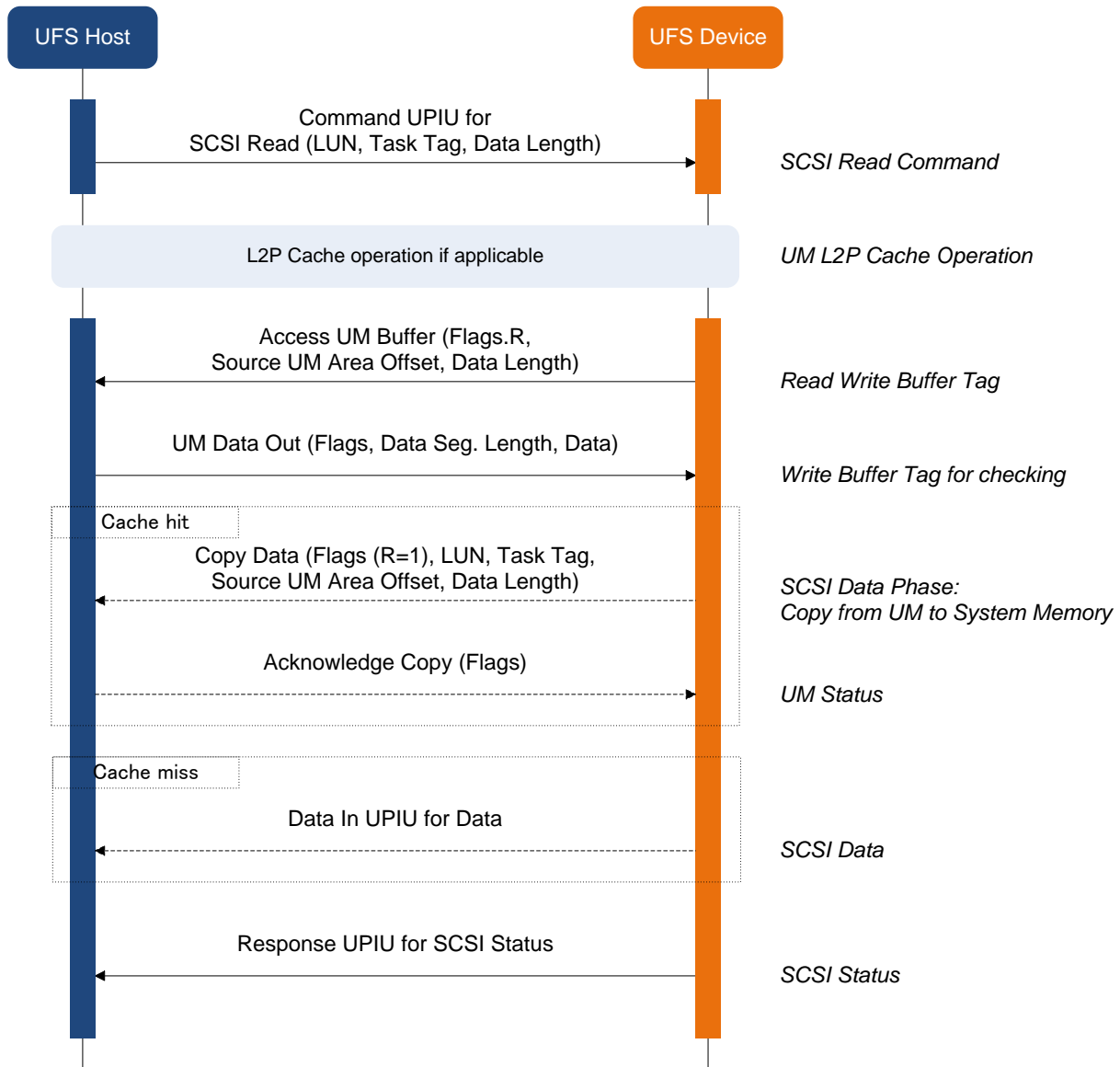


Figure 7.8 — SCSI Read (Cached) and subsequent Unified Memory Operations (Simple Single Read)

7.3.12.2 Use Case Examples (cont'd)

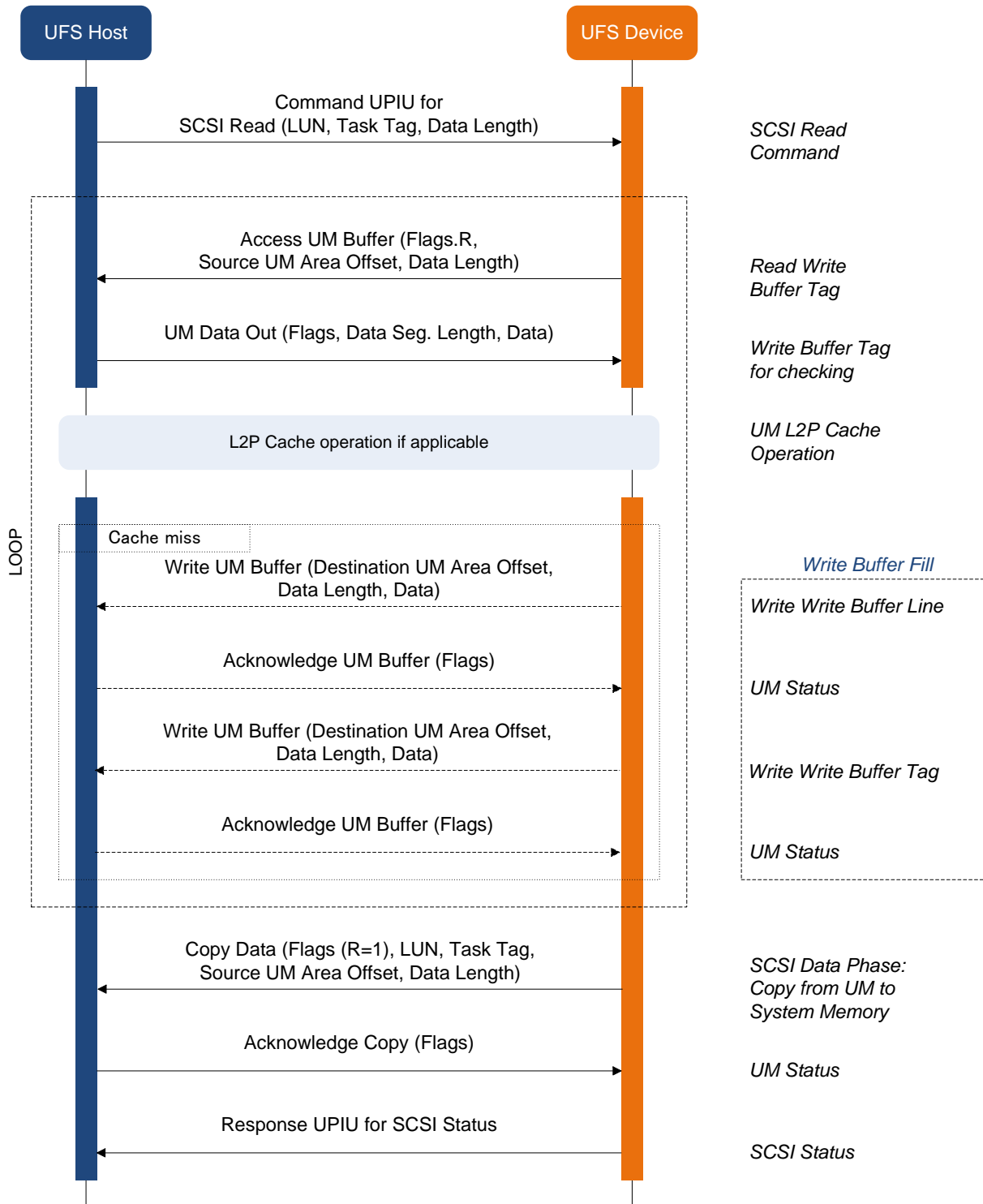


Figure 7.9 — SCSI Read (Cached) and subsequent Unified Memory Operations (Large Data Read with Filling Missing Write Buffer)

7.3.12.2 Use Case Examples (cont'd)

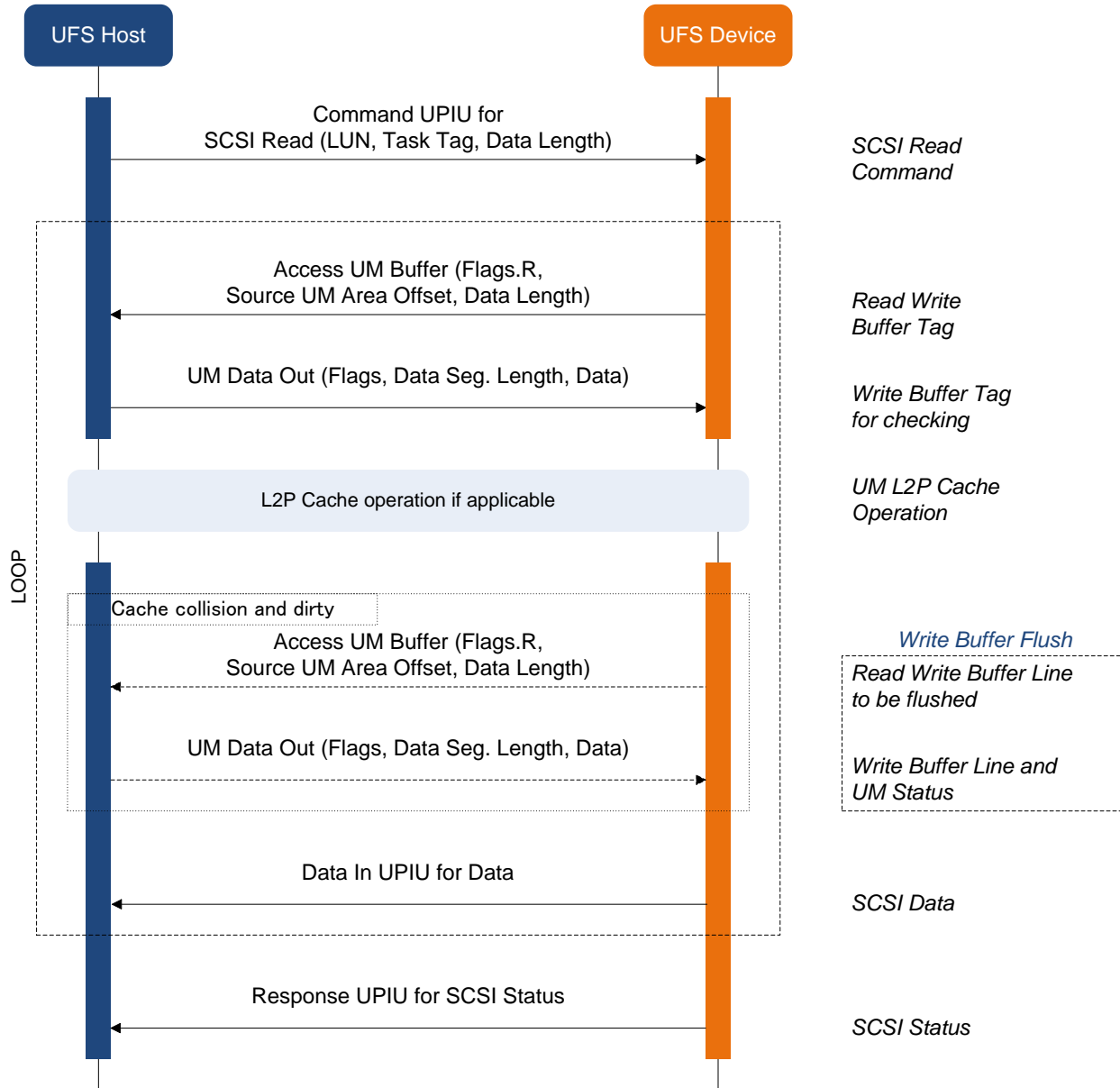


Figure 7.10 — SCSI Read (Cached) and subsequent Unified Memory Operations (Large Data Read with Flushing Dirty Write Buffer)

7.3.12.2 Use Case Examples (cont'd)

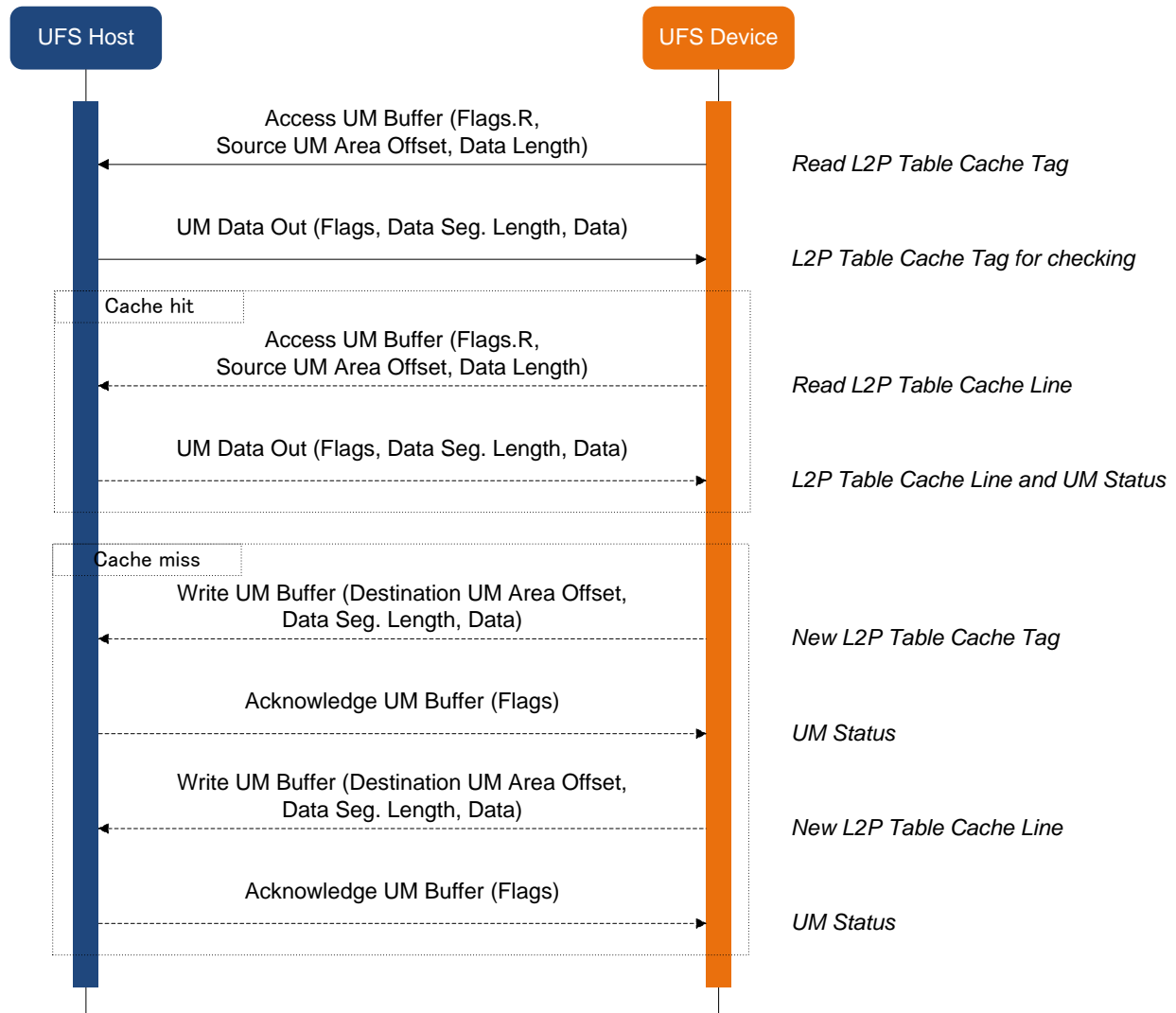


Figure 7.11 — L2P Table Operation

8 UFS FUNCTIONAL DESCRIPTIONS RELATED TO UNIFIED MEMORY

8.1 UFS Boot with Unified Memory Support

8.1.1 Introduction

Several features of the boot functionality can be configured in order to be adapted to different system requirements. If Unified Memory is supported, it shall be initialized after the normal UFS boot procedure.

8.1.2 Initialization of Unified Memory

The UM initialization is made up of the following phases: Device Descriptor Reading, Attribute Setting and Initialization Completion.

a) Device Descriptor Reading

The UFS host controller can get relevant device info for the initialization process by accessing the Device Descriptor (i.e., Device Subclass, bCPortConfEn, dMinUMAreaSize, etc.). The Host controller can know whether Unified Memory is supported by the Device checking Device Subclass (Bit2 shows UM support, and it is set to “1” if UM is supported. See Table 9.1 for detail.). If UM is supported, the Host controller sets the registers (See UFSHCI spec [UFSHCI-UME] for detail).

b) Attribute Setting

The Host then sets the Attributes (i.e., dUMAreaSize and bMaxUMPIURequests). The value of dUMAreaSize shall be greater than or equal to dMinUMAreaSize. If bMaxUMPIURequests is not set, the default value “0” is applied, which means the maximum number of UMPIU requests is “32”.

c) Initialization Completion of Unified Memory

The Host sets the fUM flag to “1” for initializing UM. Then the Host polls the fUM flag to be reset by the Device.

8.1.2 Initialization of Unified Memory (cont'd)

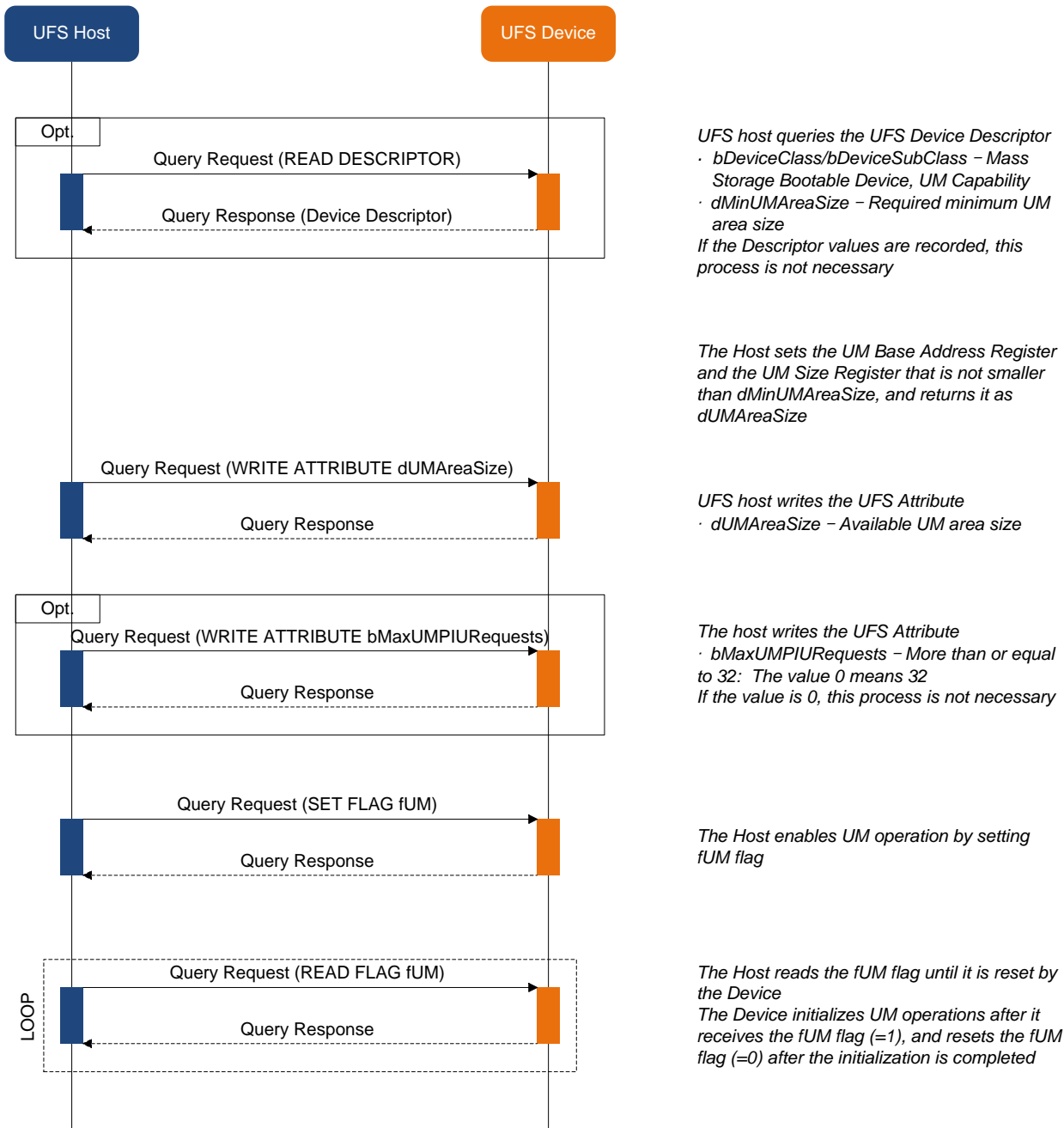


Figure 8.1 — Unified Memory Initialization Sequence Diagram

8.2 UFS Cache in Unified Memory

If Write Buffer Cache is allocated in UM, the caching process is subject to the UFS Cache function since the cache in UM is a replacement of the cache in the Device (See [UFS] 13.5.) . It is recommended to have Write Buffer Cache in UM for taking advantage of UMA.

8.2.1 Example of How to Use Write Buffer Cache in Unified Memory

Application program uses the `write` function to write data as shown in the following example:

```
write (file descriptor, buffer address, data size);
```

The device driver in the Host is responsible to manage the SCSI WRITE command. For the above example, `file descriptor` is translated into LUN, Task Tag, and Logical address in SCSI WRITE in Command UPIU; `buffer address` is set to DMA controller in HCI; `data size` is translated into data length in Command UPIU, transfer length in SCSI WRITE, and transfer data count in Data Out UPIU. Thus the device driver knows the data size.

As described in section 7, caching small size data is more effective than doing large size one, and thus small size data should be cached whereas large size one should be directly written to non-volatile memory. A threshold is necessary to decide the data should be cached or not. Let 8 Kbytes be the threshold for example, then the data whose size is not greater than 8 Kbytes is cached, and the data whose size is greater than 8 Kbytes is not cached.

The device driver creates a Command UPIU in which FUA = 1 for the large data, and do with FUA = 0 for the small data. It also creates a DATA OUT UPIU for the large data according to an RTT UPIU from the Device, but does not do for the small data. Instead the Device generates a COPY DATA UMPIU for the small data according to the given parameters in the Command UPIU.

The data cached in UM will be flushed to non-volatile memory in the Device as the occasion arises. Write Buffer cache collision is an example of the occasion, and transition to the Pre-PowerDown Power Mode is another example.

8.3 Host Device Interaction

8.3.1 Background Operation Mode with Unified Memory

8.3.1.1 Introduction

The Background Operation with Unified Memory is subject to that defined in [UFS] 13.4.4. The use of the function is recommended for more safety in UM.

8.3.1.2 Power Failure

It is the Device's responsibility to ensure that the data in the Device is not corrupted if a power failure occurs during a background operation. Note that some buffers may exist in UM, which is physically in the Host. The buffers in UM are basically the same as those in the Device, but may be more fragile. Power failure in the Host, where the Device is still alive, may affect more than the case with the Device-integrated buffers.

8.3.1.3 Implementation

Refer to Table 14.18 "Flags" and Table 14.20 "Attributes" of UFS Standard [UFS].

9 UFS DESCRIPTORS, FLAGS AND ATTRIBUTES

9.1 Device Descriptor

UM support is specified by Bit2 of bDeviceSubClass, and the minimum size of UM is specified by dMinUMAreaSize. For future extension in CPort assignment, bCPortConfEn is reserved.

Table 9.1 — Device Descriptor

DEVICE DESCRIPTOR					
Offset	Size	Name	MDV ¹	User Conf.	Description
00h	1	bLength	40h	No	Size of this descriptor
04h	1	bDeviceSubClass	Device specific	No	UFS Mass Storage Subclass Bits (0/1) specify as follows: Bit0: Bootable / Non-Bootable Bit1: Embedded / Removable Bit2: No UM Support / UM Support Others: Reserved Examples: 00h: Embedded Bootable 01h: Embedded Non-Bootable 02h: Removable Bootable 03h: Removable Non-Bootable 04h: Embedded Bootable with UM Support
1Fh	17	Reserved			
30h	1	bCPortConfEn	Device specific	No	CPort configuration enable 00h: Use of CPort s are defined as follows CPort 0 for normal UFS, CPorts 1 and 2 for UM operations Others: Reserved for future CPort configurability
31h	4	dMinUMAreaSize	Device specific	No	Minimum UM area size requested by Device. Lower 10 bits shall be 0 (1 Kbyte boundary).
NOTE 1 The column "MDV" (Manufacturer Default Value) specifies parameter values after device manufacturing. Some parameters may be configured by the user writing the Configuration Descriptor.					
NOTE 2 "User Conf." column specifies which fields can be configured by the user writing the Configuration Descriptor: "Yes" means that the field can be configured, "No" means that the field is a capability of the device and cannot be changed by the user. The desired value shall be set in the equivalent parameter of the Configuration Descriptor.					

9.2 Flags

There is a flag, fUM, to be set to initialize UM. The flags, fSuspendUM and fUMSuspended, are used for the communication between the Host and the Device to suspend UM operation for some cases, e.g., before/after HIBERNATE.

Table 9.2 — Flags

FLAGS					
IDN	Name	Type	Type ¹ # Ind. ² # Sel. ³	Default	Description
0Ah	fUM	Read / Set only	D	0	UM Initialization Host set fUM flag to initiate UM initialization after device initialization process is completed. Device resets flag when UM initialization is completed.. 0b: UM initialization completed or not started yet. 1b: UM initialization in progress.
0Ch	fSuspendUM	Write only / Volatile	D	0	Host sets fSuspendUM to request Device to suspend UM operation ⁴ . Host resets fSuspendUM to tell Device that UM may operate. 0b: Tell Device that UM may operate. 1b: Request Device to suspend UM.
0Dh	fUMSuspended	Read / Read only	D	0	Device sets fUMSuspended to show UM has been suspended ⁴ . Host checks fUMSuspended before putting the link into HIBERNATE mode. 0b: UM has not been suspended. 1b: UM has been suspended.
<p>NOTE 1 The type “D” identifies a device level flag, while the type “A” identifies an array of flags.</p> <p>NOTE 2 For array of flags, “# Ind.” specifies the amount of valid values for the INDEX field in QUERY REQUEST/RESPONSE UPIU.</p> <p>NOTE 3 For array of flags, “# Sel.” specifies the amount of valid values for the SELECTOR field in QUERY REQUEST/RESPONSE UPIU.</p> <p>NOTE 4 UM suspension means that there is no UMPIU in operation and that there will be no UMPIU. Data in UM and/or outstanding UM operations may exist while suspended. When Host will put the link into HIBERNATE mode, it first sets fSuspendUM and checks fUMSuspended.</p>					

There are attributes, `dUMAreaSize` and `bMaxUMPIURequests`, to specify the UM area size allocated by the Host and to define the maximum number of UMPIU requests supported by the Host, respectively.

ATTRIBUTES

ATTRIBUTES						
IDN	Name	Access Property	Size	Type ¹ # Ind. ² # Sel. ³	MDV ⁴	Description
12h	dUMAreaSize	Write only / Power on reset	4 bytes	D	0000 0000h	Allocated UM area size. Lower 10 bits shall be 0 (1 Kbyte boundary).
13h	bMaxUMPIURequests	Read / Persistent	1 byte	D	00h	The Host sets this value to tell the number of UMPIU requests that the Host Device Bus Master Manager supports to the Device. The length is 32 x (bMaxUMPIURequests + 1). The value of this attribute shall be between 0 and 7.

NOTE 1 The type “D” identifies a device level attribute, while the type “A” identifies an array of attributes.

NOTE 2 For array of attributes, “# Ind.” specifies the amount of valid values for the INDEX field in QUERY REQUEST/RESPONSE UPIU .

NOTE 3 For array of attributes, “# Sel.” specifies the amount of valid values for the SELECTOR field in QUERY REQUEST/RESPONSE UPIU.

NOTE 4 The column “MDV” (Manufacturer Default Value) specifies attribute values after device manufacturing.

Annex A (informative) Differences between JESD220-1A and JESD220-1

This table briefly describes most of the changes made to entries that appear in this standard, JESD220-1A, compared to its predecessor, JESD220-1 (September 2013). If the change to a concept involves any words added or deleted (excluding deletion of accidentally repeated words), it is included. Some punctuation changes are not included.

Clause	Description of change
2	Addition normative references added.
3.1	Added UME
3.2	Added Application Client and Device Server
4.1	Removed 1 st paragraph
6	All New, subclauses and figures
7, 8, 9	in previous document these clauses were 6, 7, 8, all figures and tables renumbered accordingly.



Standard Improvement Form

JEDEC _____

The purpose of this form is to provide the Technical Committees of JEDEC with input from the industry regarding usage of the subject standard. Individuals or companies are invited to submit comments to JEDEC. All comments will be collected and dispersed to the appropriate committee(s).

If you can provide input, please complete this form and return to:

JEDEC
Attn: Publications Department
3103 North 10th Street
Suite 240 South
Arlington, VA 22201-2107

Fax: 703.907.7583

1. I recommend changes to the following:

☐ Requirement, clause number _____

☐ Test method number _____ Clause number _____

The referenced clause number has proven to be:

☐ Unclear ☐ Too Rigid ☐ In Error

☐ Other _____

2. Recommendations for correction:

3. Other suggestions for document improvement:

Submitted by

Name: _____

Phone: _____

Company: _____

E-mail: _____

Address: _____

City/State/Zip: _____

Date: _____

