



1
2
3
4

Document Number: DSP0237

Date: 2017-05-21

Version: 1.1.0

5
6
7

Management Component Transport Protocol (MCTP) SMBus/I2C Transport Binding Specification

8
9
10
11

Supersedes: 1.0.0

Document Class: Normative

Document Status: Published

Document Language: en-US

12

13 Copyright Notice

14 Copyright © 2017 Distributed Management Task Force, Inc. (DMTF). All rights reserved.

15 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems
16 management and interoperability. Members and non-members may reproduce DMTF specifications and
17 documents, provided that correct attribution is given. As DMTF specifications may be revised from time to
18 time, the particular version and release date should always be noted.

19 Implementation of certain elements of this standard or proposed standard may be subject to third party
20 patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations
21 to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose,
22 or identify any or all such third party patent right, owners or claimants, nor for any incomplete or
23 inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to
24 any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize,
25 disclose, or identify any such third party patent rights, or for such party's reliance on the standard or
26 incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any
27 party implementing such standard, whether such implementation is foreseeable or not, nor to any patent
28 owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is
29 withdrawn or modified after publication, and shall be indemnified and held harmless by any party
30 implementing the standard from any and all claims of infringement by a patent owner for such
31 implementations.

32 For information about patents held by third-parties which have notified the DMTF that, in their opinion,
33 such patent may relate to or impact implementations of DMTF standards, visit
34 <http://www.dmtf.org/about/policies/disclosures.php>.

35 PCI-SIG, PCIe, and the PCI HOT PLUG design mark are registered trademarks or service marks of PCI-
36 SIG.

37 All other marks and brands are the property of their respective owners.

38

CONTENTS

40	Foreword	5
41	Introduction.....	6
42	1 Scope	7
43	2 Normative references	7
44	3 Terms and definitions	7
45	4 Symbols and abbreviated terms.....	8
46	5 Conventions	10
47	5.1 Reserved and unassigned values.....	10
48	5.2 Byte ordering.....	11
49	6 MCTP over SMBus/I ² C transport	11
50	6.1 Terminology	11
51	6.2 Transport binding use with I ² C.....	12
52	6.3 MCTP packet encapsulation	12
53	6.4 Bridges and packet formatting	13
54	6.5 MCTP support discovery.....	13
55	6.6 Support for fixed-address devices	14
56	6.7 Supported media.....	14
57	6.8 Physical address format for MCTP control messages.....	15
58	6.9 Get endpoint ID Medium-Specific Information	15
59	6.10 Bus owner address	15
60	6.11 Bus address assignment	15
61	6.12 SMBus/I ² C considerations for MCTP messages	19
62	6.13 Fairness arbitration	20
63	6.14 NACK window	21
64	6.15 Fairness arbitration requirements for MCTP bridges.....	22
65	6.16 Fairness arbitration requirements for non-bridge endpoints.....	23
66	6.17 Fairness arbitration timing	24
67	6.18 MCTP packet timing requirements	25
68	6.19 MCTP control message timing requirements.....	27
69	6.20 "Stuck 0" condition handling	28
70	6.21 MCTP over SMBus/I ² C protocol anti-aliasing	29
71	6.22 Well-known and reserved slave addresses	30
72	6.23 Fixed address allocation	31
73	6.24 Recommended address range allocation for computer systems	32
74	ANNEX A (informative) Notation.....	35
75	ANNEX B (informative) Change log.....	36
76		

77 Figures

78	Figure 1 – MCTP over SMBus/I ² C packet format	12
79	Figure 2 – Address assignment flow.....	18
80	Figure 3 – Allowed byte range for first NACK'd byte.....	21
81	Figure 4 – Fairness arbitration timing measurement for SMBus and I ² C	24
82	Figure 5 – Example system configuration.....	32
83		

84 **Tables**

85	Table 1 – Packet header field descriptions	12
86	Table 2 – Supported media.....	15
87	Table 3 – Physical address format.....	15
88	Table 4 – Medium-Specific Information	15
89	Table 5 – Fairness arbitration timing values for 100 kHz SMBus/I ² C	24
90	Table 6 – Fairness arbitration timing values for 400 kHz I ² C.....	25
91	Table 7 – Fairness arbitration timing values for 1MHz I ² C.....	25
92	Table 8 – Timing specifications for MCTP packets on SMBus/I ² C	26
93	Table 9 – Timing Specifications for MCTP control messages on SMBus	27
94	Table 10 – Well-known and reserved slave addresses	30
95	Table 11 – Slave address allocation for computer systems	33
96		

97

Foreword

98 The *Management Component Transport Protocol (MCTP) SMBus/I2C Transport Binding Specification*
99 (DSP0237) was prepared by the PMCI Subgroup of the Pre-OS Working Group.

100 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems
101 management and interoperability.

102 Acknowledgments

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

104 Editor:

- 105 • Yuval Itkin – Mellanox Technologies

106 Contributors:

- 107 • Alan Berenbaum – SMSC
- 108 • Klodnicki Edward - IBM
- 109 • Kozlowski Koe – Dell Inc
- 110 • Patrick Caporale – Lenovo
- 111 • Patrick Schoeller – Hewlett Packard Enterprise
- 112 • Phil Chidester – Dell Inc
- 113 • John Leung - Intel Corporation
- 114 • Eliel Louzoun – Intel Corporation
- 115 • Hemal Shah - Broadcom Limited
- 116 • Tom Slaight – Intel Corporation
- 117 • Yi Zeng – Intel Corporation

118

119

Introduction

120 The Management Component Transport Protocol (MCTP) over SMBus/I2C transport binding defines a
121 transport binding for facilitating communication between platform management subsystem components
122 (e.g., management controllers, managed devices) over SMBus/I2C.

123 The *MCTP Base Specification* ([MCTP](#)) describes the protocol and commands used for communication
124 within and initialization of an MCTP network. The MCTP over SMBus/I2C transport binding definition in
125 this specification includes a packet format, physical address format, message routing, and discovery
126 mechanisms for MCTP over SMBus/I2C communications.

127

128
129

Management Component Transport Protocol (MCTP) SMBus/I2C Transport Binding Specification

130 1 Scope

131 This document provides the specifications for the Management Component Transport Protocol (MCTP)
132 transport binding for SMBus/I²C.

133 2 Normative references

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

138 DMTF DSP0136, *Alert Standard Format Specification 2.0*,
139 <http://www.dmtf.org/standards/documents/ASF/DSP0136.pdf>

140 DMTF, DSP0236, *Management Component Transport Protocol (MCTP) Base Specification 1.3*, MCTP,
141 http://www.dmtf.org/sites/default/files/standards/documents/DSP0236_1.3.0.pdf

142 DMTF, DSP0239, *Management Component Transport Protocol (MCTP) IDs and Codes 1.4*, MCTP_ID,
143 http://www.dmtf.org/sites/default/files/standards/documents/DSP0239_1.4.0.pdf

144 IPMI Consortium, *Intelligent Platform Management Interface Specification, Second Generation 2.0*, 2006,
145 ftp://download.intel.com/design/servers/ipmi/IPMIv2_0rev1_0.pdf

146 ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*,
147 <http://isotc.iso.org/livelink/livelink?func=ll&objId=4230456&objAction=browse&sort=subtype>

148 NXP Semiconductors, *I²C-bus specification and user manual*, 4 April 2014
149 http://www.nxp.com/documents/user_manual/UM10204.pdf

150 SBS Implementers Forum, *System Management Bus (SMBus) Specification v2.0*, SMBus, August 2000,
151 <http://www.smbus.org/specs/smbus20.pdf>

152 3 Terms and definitions

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

155 The terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"),
156 "may", "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described
157 in [ISO/IEC Directives, Part 2](#), Annex H. The terms in parentheses are alternatives for the preceding term,
158 for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that
159 [ISO/IEC Directives, Part 2](#), Annex H specifies additional alternatives. Occurrences of such additional
160 alternatives shall be interpreted in their normal English meaning.

161 The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as
162 described in [ISO/IEC Directives, Part 2](#), Clause 5.

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

166 The terms defined in [DSP0004](#), [DSP0223](#), and [DSP1001](#) apply to this document. The following additional
167 terms are used in this document.

168 3.1

169 Address Resolution Protocol

170 ARP

171 refers to the procedure used to dynamically determine the addresses of devices on a shared
172 communication medium

173 4 Symbols and abbreviated terms

174 The following symbols and abbreviations are used in this document.

175 4.1

176 ACK

177 acknowledge

178 4.2

179 ARP

180 Address Resolution Protocol

181 4.3

182 ASF

183 Alert Standard Format

184 4.4

185 BMC

186 baseboard management controller

187 4.5

188 EEPROM

189 Electrically Erasable Programmable Read-Only Memory

190 4.6

191 EID

192 endpoint identifier

193 4.7

194 I²C

195 Inter-Integrated Circuit

196 4.8

197 I/O

198 input/output

199 4.9

200 IPMB

201 Intelligent Platform Management Bus

202	4.10
203	IPMI
204	Intelligent Platform Management Interface
205	4.11
206	kHz
207	kilohertz
208	4.12
209	LSb
210	least significant bit
211	4.13
212	LSB
213	least significant byte
214	4.14
215	max
216	maximum
217	4.15
218	MCTP
219	Management Component Transport Protocol
220	4.16
221	min
222	minimum
223	4.17
224	ms
225	millisecond
226	4.18
227	MSB
228	most significant byte
229	4.19
230	MTU
231	Maximum Transmission Unit
232	4.20
233	NACK
234	not acknowledge
235	4.21
236	PCI
237	peripheral component interconnect
238	4.22
239	PCIe®
240	PCI Express™

241 **4.23**
242 **PEC**
243 packet error code

244 **4.24**
245 **PMCI**
246 Platform Management Component Intercommunications

247 **4.25**
248 **PSA**
249 persistent slave address

250 **4.26**
251 **rsvd**
252 reserved (not case sensitive)

253 **4.27**
254 **SCL**
255 serial clock

256 **4.28**
257 **SDA**
258 serial data

259 **4.29**
260 **sec**
261 second

262 **4.30**
263 **SEEPROM**
264 serial EEPROM

265 **4.31**
266 **SMBus**
267 System Management Bus

268 **4.32**
269 **UDID**
270 unique device identifier

271 **5 Conventions**

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

273 **5.1 Reserved and unassigned values**

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

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

278 5.2 Byte ordering

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

281 6 MCTP over SMBus/I²C transport

282 The MCTP over SMBus/I²C transport binding defines how MCTP packets are delivered over a physical
283 SMBus or I²C medium using SMBus transactions. This includes how physical addresses are used, how
284 fixed addresses are accommodated, how physical address assignment is accomplished for hot-plug or
285 other devices that require dynamic physical address assignment, and how MCTP support is discovered.
286 Timing specifications for bus and MCTP control operations are also given, and a "fairness" protocol is
287 defined for the purpose of avoiding deadlock and starvation/lockout situations among MCTP endpoints.

288 The binding has been designed to be able to share the same bus as devices communicating using earlier
289 SMBus/I²C management protocols such as Alert Standard Format (ASF) and IPMI, and with vendor-
290 specific devices using SMBus/I²C protocols. The specifications can also allow a given device to
291 incorporate non-MCTP SMBus functions alongside MCTP. This is described in more detail in 6.21.

292 6.1 Terminology

293 According to SMBus, SMBus devices are categorized as follows, where Address Resolution Protocol
294 (ARP) refers to the SMBus Address Resolution Protocol (a dynamic slave address assignment protocol)
295 and UDID refers to a "unique device identifier", a 128-bit value that a device uses during the ARP process
296 to uniquely identify itself. Because these protocols are implemented with command transactions that are
297 run on top of the SMBus physical specification, it is possible to use these protocols on devices that
298 support an I²C physical interface.

299 • **ARP-capable**

300 [SMBus](#) term indicating a device that supports all [SMBus](#) ARP commands with the exception of
301 the optional Host Notify command. The slave address is assignable. The device supports both
302 Reset commands.

303 • **Fixed and Discoverable**

304 [SMBus](#) term indicating a device supports the Prepare to ARP, directed Get UDID, general Get
305 UDID, and Assign Address commands. The slave address is fixed; the device will accept the
306 Assign Address command but will not allow address reassignment. The device supports both
307 Reset commands.

308 • **Fixed - Not Discoverable**

309 [SMBus](#) term indicating a device supports the directed Get UDID command. The slave address
310 is fixed.

311 • **Non-ARP-capable**

312 [SMBus](#) term indicating a device does not support any ARP commands. The slave address is
313 fixed.

314 • **Fixed Address**

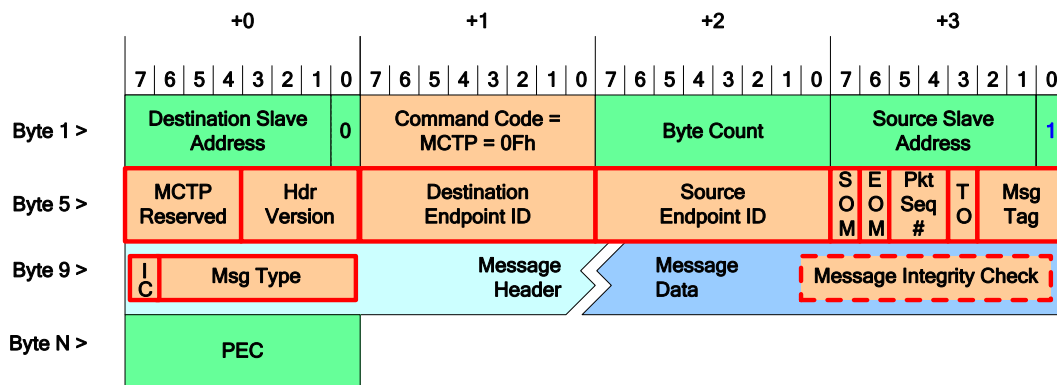
315 For this specification, this term is be used to refer to any device that uses a fixed slave address,
316 without distinguishing whether it is "Fixed and Discoverable", "Fixed, not Discoverable", or
317 "Non-ARP-capable".

318 **6.2 Transport binding use with I²C**

319 The transport binding defined in this specification has also been designed to be able to work with
 320 standard-mode fast-mode (400 kHz) and Fast-mode Plus (1MHz) I²C buses that use 7-bit addressing; 10-
 321 bit addressing is not supported. This binding has not been specified for use with high-speed I²C
 322 specifications.

323 **6.3 MCTP packet encapsulation**

324 All MCTP transactions are based on the SMBus Block Write bus protocol. The first 8 bytes make up the
 325 packet header. The first three fields—Destination Slave Address, Command Code, and Length—map
 326 directly to SMBus functional fields. The remaining header and payload fields map to SMBus Block Write
 327 "Data Byte" fields, as indicated in Figure 1. Hence, the inclusion of the Source Slave Address in the
 328 header is specified by [MCTP](#) rather than [SMBus](#). This is done to facilitate addressing required for
 329 establishing communications back to the message originator.



330

331 **Figure 1 – MCTP over SMBus/I²C packet format**

332 **Table 1 – Packet header field descriptions**

Byte	Block Write Field(s)	Description
1	Slave Address Wr	[7:1] SMBus Destination Slave Address: The slave address of the target device for the local SMBus link [0]: SMBus R/W# bit: Shall be set to 0b as all MCTP messages use SMBus write transactions.
2	Command Code	Command Code: SMBus Command Code All MCTP over SMBus messages use a command code of 0x0F.
3	Byte Count	Byte Count: Byte count for the SMBus Block Write protocol transaction that is carrying the MCTP packet content. This value is the count of bytes that follow the Byte Count field up to, but not including, the PEC byte. For example, if the MCTP packet payload length (starting with byte 9) is 64 bytes, the value in the Byte Count field would be 69. (The count of 69 accounts for 64 bytes of MCTP packet payload plus the five bytes [bytes 4 through 8, inclusive] that comprise the bytes of the SMBus-specific header and MCTP header that follow the Byte Count field.)

Byte	Block Write Field(s)	Description
4	Data Byte 1	SMBus Source Slave address [7:1] : For the local SMBus link, the slave address of the source device. [0]: This bit shall be set to 1b. The value enables MCTP to be differentiated from IPMI over SMBus and IPMB (IPMI over I ² C) protocols.
5	Data Byte 2	[7:4] MCTP reserved: This nibble is reserved for definition by the MCTP Base Specification . [3:0] MCTP header version: Set to 0001b for MCTP devices that are conformant to the MCTP Base Specification 1.0 and this version of the SMBus transport binding. All other values = Reserved.
6	Data Byte 3	Destination endpoint ID (*)
7	Data Byte 4	Source endpoint ID (*)
8	Data Byte 5	[7] SOM: Start Of Message flag (*) [6] EOM: End Of Message flag (*) [5:4] Packet sequence number (*) [3] Tag Owner (TO) bit (*) [2:0] Message tag (*)
9	Data Byte 6	[7] IC: Integrity Check bit (*) [6:0] Message type (*)
10:N-1	Data Bytes 7:M	Message header and data (*)
N	PEC	Packet error code (PEC): The PEC as defined in the SMBus 2.0 Specification . All MCTP transactions shall include a PEC byte. The PEC byte shall be transmitted by the source and checked by the destination.
(*) Indicates a field that is defined by the MCTP Base Specification .		

333 **6.4 Bridges and packet formatting**

334 As an MCTP packet travels through a bridge from one SMBus/I²C port to another, the bridge leaves all
 335 packet header and message header and data fields alone with the exception of the source and
 336 destination slave address, which shall be modified to route across the intended bus/link. When an MCTP
 337 bridge forwards a message from an input port to an output port, it replaces the destination slave address
 338 with the targeted slave on the destination bus, and replaces the source slave address with the bridge's
 339 slave address.

340 The MCTP SMBus/I²C bridge shall re-calculate the PEC byte to account for changes in the source and
 341 destination slave address fields.

342 A similar process is used when bridging between different media. The physical addressing and header
 343 information gets changed by the bridge to match the requirements of the target bus, and any packet-level
 344 integrity check information is also updated.

345 **6.5 MCTP support discovery**

346 All SMBus devices that support an MCTP endpoint and the SMBus Get UDID command for a particular
 347 SMBus/I²C interface (that is, devices with ARP-capable, fixed and discoverable, or fixed-not discoverable
 348 interfaces) are required to have their MCTP support discoverable through the Get UDID command. To do

349 this, endpoints shall return a value of 1b in bit 5 (the ASF bit) in the Interface field in the Get UDID
350 command.

351 Once support for ASF has been indicated, an MCTP control message (for example, Get MCTP Version
352 Support) can be issued to the device to determine whether it supports MCTP. The SMBus command byte
353 for MCTP packets uses a value that has been allocated by the DMTF for MCTP use and does not overlap
354 values used for ASF. This enables older devices that indicate ASF support to be queried for MCTP
355 support without conflict. This is described in more detail in 6.6. Devices that do not support the Get UDID
356 command will need to have their support for MCTP configured into the bus owner as described in 6.6.

357 I²C devices can also support the SMBus protocols and commands for being an ARP-able device that is
358 also discoverable as an MCTP device. This is required for hot-plug I²C devices using MCTP.

359 6.6 Support for fixed-address devices

360 MCTP bus owners shall include non-volatile options to record the addresses used by fixed-address
361 devices on SMBus/I²C buses that they own, and which of those devices support MCTP.

362 For non-MCTP devices, the MCTP bus owner needs this information to know which fixed addresses to
363 avoid when performing SMBus ARP for the bus. (Alternatively, the bus owner could be configured with a
364 range of SMBus slave addresses that the bus owner is allowed to allocate from.)

365 For MCTP devices, the bus owner needs this information to perform EID assignment and, if the bus
366 owner is also an MCTP bridge, routing table initialization and operation.

367 For fixed-address MCTP devices that do not support the Get UDID command (that is, non-ARP-capable
368 devices), the bus owner needs to also be configured with information that identifies the device as
369 supporting MCTP.

370 For fixed-address devices that support the [SMBus](#) Get UDID command (that is, devices with ARP-
371 capable, Fixed and Discoverable, or Fixed-Not Discoverable SMBus interfaces) the bus owner can either
372 discover whether the device supports MCTP by using the discovery approach described in 6.5, or it could
373 have this information configured at the same time that the slave address information for the fixed-address
374 device is provided.

375 It is recommended that general-purpose devices that act as MCTP bus owners allow being configured to
376 support at least 16 different fixed-address devices for each SMBus/I²C bus they own. This number would
377 include both MCTP and non-MCTP devices.

378 6.7 Supported media

379 This physical transport binding has been designed to work with the media specified in [DSP0239](#). Table 2
380 quotes relevant physical media identifiers from [DSP0239](#). In case of any contradiction [DSP0239](#) shall be
381 used as the normative definition. Use of this binding with other types of physical media is not covered by
382 this specification. At least one of the physical media identifiers listed in Table 2 shall be supported to
383 comply with this specification.

384

Table 2 – Supported media

Physical Media Identifier	Description
0x01	SMBus 100 kHz compatible
0x02	SMBus + I²C 100 kHz compatible
0x03	I²C 100 kHz compatible
0x04	I²C 400 kHz compatible
0x05	SMBus + I²C 1 MHz compatible

385 **6.8 Physical address format for MCTP control messages**

386 The address format shown in Table 3 shall be used for MCTP control commands that require a physical
 387 address parameter to be returned for a bus that uses this transport binding with one of the supported
 388 media types listed in 6.7. This includes commands such as the Resolve Endpoint ID, Routing Information
 389 Update, and Get Routing Table Entries commands.

390

Table 3 – Physical address format

Format Size	Layout and Description
1 byte	[7:1] slave address bits [0] 0b

391 **6.9 Get endpoint ID Medium-Specific Information**

392 The Medium-Specific Information as shown in Table 4 shall be used for the medium-specific Information
 393 field returned in the response to the Get Endpoint ID MCTP control message.

394

Table 4 – Medium-Specific Information

Description	
[7:1]	reserved
[0]	fairness arbitration support (see 6.13) 0b = not supported 1b = supported

395 **6.10 Bus owner address**

396 In order to be the target of the SMBus Notify ARP Master protocol transaction the MCTP bus owner shall
 397 be configurable to be accessed at the SMBus host slave address. This configuration does not need to be
 398 used if the bus implementation does not include any MCTP devices that require dynamic address
 399 assignment of their slave address. For more information, see 6.11.4.

400 The bus owner may use a different, second slave address for all other MCTP communication functions.

401 **6.11 Bus address assignment**

402 This clause describes the configuration, setup, and operation of communication between MCTP
 403 endpoints using SMBus/I²C as the communication medium.

404 6.11.1 Slave addresses

405 Each device on SMBus/I²C shall have a slave address to be the target of transactions by bus masters.
406 The MCTP transport protocol solely utilizes Master Write transactions to transfer MCTP packets between
407 MCTP endpoints. For endpoint "A" to send an MCTP packet to endpoint "B", endpoint A shall master the
408 bus and issue a Block-Write transaction to the slave address of endpoint B. Similarly, for endpoint B to
409 send an MCTP packet to endpoint A, it shall master the bus and issue a Block-Write transaction to the
410 slave address of endpoint A. Thus, bi-directional transfer of MCTP packets requires that both sides of the
411 communication have slave addresses.

412 Device support for slave addresses can be of two general types: fixed or assignable. Devices with
413 assignable addresses (also referred to as "ARP-capable" or "ARP-able") can use the [SMBus](#) ARP. The
414 entity that assigns slave addresses to ARP-able devices is referred to as the "ARP master".

415 A bus can include a mix of fixed-address and ARP-able devices. Most fixed-address devices do not
416 include a discovery mechanism, and neither [SMBus](#) nor [I²C](#) require one. Therefore, for a generic bus
417 implementation that support ARP-able devices (such as SMBus to PCI/PCIe connectors) the ARP master
418 needs to know what ranges of addresses are being used for fixed-address devices so that it doesn't give
419 an ARP-able device an address that conflicts with a fixed-address device.

420 This transport binding allows for non-MCTP devices (both fixed address and ARP-able) to reside on the
421 same bus segment used for MCTP devices. The use and assignment of slave addresses shall therefore
422 be compatible with pre-existing devices. To accomplish this, the following approach is used for managing
423 devices on a bus that supports MCTP.

424 6.11.2 Well known and reserved slave addresses

425 The [SMBus](#) and [I²C](#) specifications define certain slave addresses that should either be avoided by
426 devices or are reserved (not to be used as a general device slave address) because those addresses are
427 related to functions that are used by MCTP. These addresses are listed in Table 10.

428 6.11.3 Fixed-address recommendations for device manufacturers

429 MCTP may be used within a typical computer system application where the motherboard/baseboard may
430 come from one supplier, the chassis from another supplier, and possibly add-in modules from yet
431 another.

432 Referring to Table 11, it is thus recommended that devices that use fixed addresses and are targeted for
433 uses that can include baseboard (B), chassis/system (C), and add-in (A) applications are configurable to
434 cover for at least three different "B" addresses, at least three different "C" addresses, and at least two
435 different "A" addresses to help avoid address conflicts in those applications.

436 6.11.4 Dynamic address assignment (SMBus ARP) support

437 MCTP buses that support connections to standard PCI/PCIe add-in cards are required by the PCI
438 specifications to support SMBus ARP (be ARP-capable) to allow the devices to be dynamically assigned
439 addresses to avoid address conflicts and eliminate the need for manual configuration of addresses.

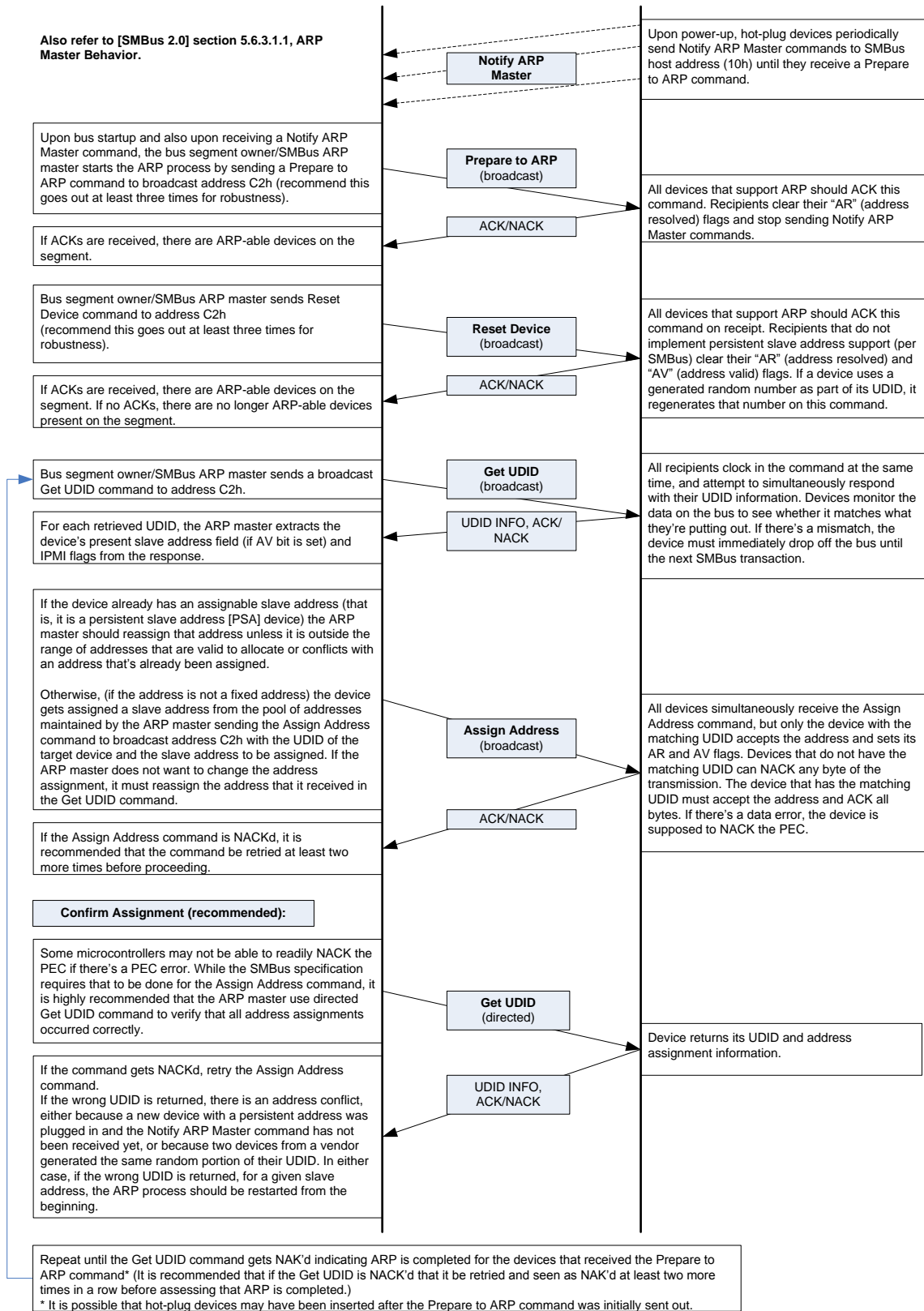
440 Figure 2 presents an overview of the address assignment process.

441 **6.11.5 Devices supporting multiple interfaces**

442 Devices that support multiple, separate [SMBus](#) or [I²C](#) interfaces where the interfaces are intended to be
443 connected to the same bus shall meet the following requirements:

- 444 • The interfaces shall be either be ARP-capable or be fixed-address interfaces that are configured
445 to use a different slave address for each interface.
- 446 • If the interfaces support SMBus ARP, (as either ARP-able or ARP-enumerable devices) a
447 different SMBus UDID shall be used for each SMBus ARP-able interface.

448 NOTE Devices that have internal hardware interfaces that may be implemented as separate blocks but are designed
449 to share a slave address are not considered to have separate interfaces in this context.



450

451

Figure 2 – Address assignment flow

452 6.11.6 MCTP requirements on SMBus ARP master support

453 If the bus supports ARP-able devices, MCTP requires that each bus shall have a controller that operates
454 as the ARP master and assigns slave addresses to all ARP-able devices on the segment. Because the
455 MCTP bus owner shall know the physical addresses of ARP-able devices that support MCTP, the ARP
456 master role will typically be handled by the same device that serves as the MCTP bus owner.

457 If a different physical device than the device holding the bus owner functions as the ARP master, there
458 shall be a mechanism to communicate the address assignment information to the bus owner function.
459 The mechanism for this is not specified by MCTP.

460 Only one controller is allowed to function as the ARP master for the segment at a given time. The ARP
461 master function is allowed to *fail-over* or be transferred to another controller. The mechanism for this
462 capability, if provided, is not specified by MCTP.

463 6.11.7 Recommendations on ARP master allocation of slave addresses

464 For PCI and PCI Express™ (PCIe) bus implementations, it is recommended that, by default, the ARP
465 master only assigns addresses to ARP-able devices from the "B" range. This is because the PCI
466 slots/connectors themselves are most commonly implemented as part of the board set.

467 Device manufacturers of controllers that function as ARP masters should provide a mechanism to enable
468 system integrators to either configure which fixed addresses that ARP should avoid, or a pool of non-
469 conflicting addresses from which ARP can draw.

470 For PCI and PCIe SMBus implementations, the ARP master should be able to assign at least two
471 addresses for each PCI connector on the segment.

472 6.11.8 MCTP requirements on hot-pluggable bridges using SMBus

473 Hot-pluggable MCTP devices that include bridging functionality are required to have static, pre-assigned,
474 SMBus UDIDs. This is because it is considered a more robust and reliable mechanism than randomly
475 generated UDIDs, and because it simplifies tracking and managing MCTP device hot-add and hot-
476 removal.

477 If devices regenerate their UDIDs on hot-plug, the MCTP bus owner/ARP master cannot rely on the UDID
478 to determine whether a device was newly added to the system. When a hot-plug device includes MCTP
479 bridging functionality, the bus owner shall be able to allocate the device a range of EIDs from a fixed pool
480 of IDs. Thus, it is important for the bus owner to be able to determine which devices have been removed
481 so that any EIDs it had given out can be returned to the pool.

482 It is straightforward for the ARP master to re-enumerate the UDIDs on the bus and determine which
483 UDIDs (if any) are no longer present (re-enumeration is a natural fallout of the ARP process). If there are
484 MCTP devices without fixed UDIDs in the mix, however, the bus owner would need to take additional
485 steps to check to see which devices had already been allocated EIDs to determine by elimination which
486 ranges, if any, had become freed. With fixed UDID, the bus owner can track which EIDs have been
487 allocated to which UDIDs and thereby determine which have been freed by a hot swap by just re-
488 enumerating the UDIDs.

489 6.12 SMBus/I²C considerations for MCTP messages

490 The following applies to MCTP messages on SMBus regardless of their message type. Note that MCTP
491 messages require Block Write byte count sizes that exceed limits specified by [SMBus](#). Additional
492 restrictions on MCTP packets over what the [SMBus](#) and [I²C](#) allow are given in 6.3 and 6.18.

493 6.12.1 Slave address ACKs/NACKs

494 Per [SMBus](#) and [I²C](#), the NACK of a slave address indicates the physical absence of the device interface.

- 495 • Devices are therefore required to always ACK their slave addresses. This includes ACK'ing
496 slave addresses used for ARP if the device is ARP-able or ARP-enumerable.
- 497 • An MCTP device *shall* ACK its slave address(es) when the R/W bit on the slave address is 0.

498 6.12.2 Clock stretching for non-addressed devices

499 MCTP devices that are monitoring the bus as slaves and do not have a slave address that matches the
500 transaction shall not clock stretch past the ACK bit for the slave address byte. This requirement only
501 applies to MCTP packet transactions. It does not apply to non-MCTP-defined messages or transactions,
502 such as those used for SMBus ARP.

503 6.13 Fairness arbitration

504 6.13.1 General

505 The following clauses describe an extension to the SMBus/I²C arbitration mechanism for device ports that
506 are used with MCTP. The extensions define a 'fairness' mechanism that helps ensure that ports that are
507 arbitrating for access to the bus will eventually get access and will not be locked out of access by other
508 MCTP ports that are using the bus.

509 NOTE Fairness arbitration only applies for messages using the MCTP base protocol. SMBus messages such as
510 Host Notify are not required to use fairness arbitration.

511 This mechanism works as follows:

- 512 • An MCTP port that wins bus arbitration (per [SMBus](#) or [I²C](#)) for a given transaction shall wait
513 until it detects a particular bus idle interval before the device can again attempt to arbitrate for
514 the bus. This is referred to as the device waiting to detect the "FAIR_IDLE" condition.
- 515 • Once the port has succeeded in detecting the FAIR_IDLE condition, it can attempt to get on the
516 bus and no longer needs to wait to detect the FAIR_IDLE condition. The port can continue to
517 attempt to access the bus without waiting for FAIR_IDLE until the next time the port wins
518 arbitration. After winning arbitration, the port shall again wait to detect the FAIR_IDLE condition
519 before it can attempt to get on the bus.

520 With this approach, all ports that lose arbitration will eventually get a turn at accessing the bus, because
521 any ports that win arbitration will need to wait until a bus idle interval is detected, while those that have
522 lost arbitration will not need to wait.

523 For this to work, endpoints shall be able to do two things:

- 524 1) Be able to recognize the FAIR_IDLE condition. Ports that are waiting to detect a FAIR_IDLE
525 condition shall recognize that no other port has made the bus become busy within a particular
526 window of time (T_{IDLE_WINDOW}) after the bus becomes free.
- 527 2) Ports that have not won arbitration shall be able to issue a START condition soon enough after
528 the bus becomes free so that a bus busy condition is seen by ports that are waiting to detect a
529 FAIR_IDLE condition. To ensure this condition is met, START shall be issued by the port within
530 a particular window of time (T_{START_WINDOW}) after the bus becomes free.

531 NOTE There is actually no explicit indication in [SMBus](#) or [I²C](#) that arbitration has been won. Instead, what the
532 master detects is that it was able to access the bus and did not have a collision (lose arbitration) with another master.
533 For this specification, this is referred to as *winning arbitration*. Because of the way arbitration works, an MCTP
534 endpoint that is transmitting as a master onto the bus will know that it has won arbitration if it is able to transmit from
535 the destination slave address byte through the end of the source slave address byte (byte 4) without receiving a
536 collision or NACK.

537 **6.13.2 Deadlock avoidance with fairness arbitration**

538 A device that wins arbitration but is subsequently NACK'd for its write transaction shall return to waiting
 539 for the FAIR_IDLE period before it can attempt the transaction again.

540 **6.13.3 Fairness arbitration support**

541 Bridges and endpoints should support fairness arbitration. An endpoint's support for fairness arbitration
 542 shall be reported through the medium-specific Information field in the response to the Get Endpoint ID
 543 MCTP control message.

544 **6.13.4 Bus busy sampling requirements for fairness arbitration**

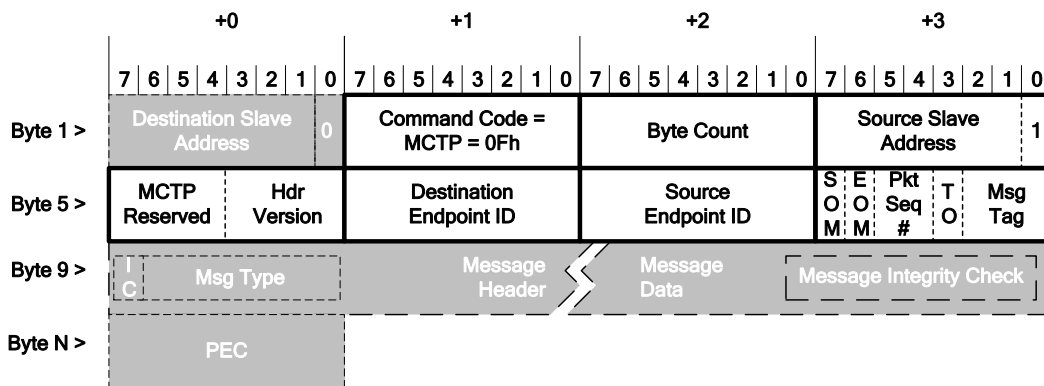
545 It is atypical and unlikely that the bus will go busy and then free again within T_{IDLE_WINDOW} . This is because
 546 T_{IDLE_WINDOW} is shorter than the time required to send one byte on the bus. Thus, this condition would only
 547 occur on an error or under a usage of the bus that is not legal within the specifications. Therefore, an
 548 implementation is not required to continuously check the bus busy status during the entire duration of
 549 T_{IDLE_WINDOW} (though this is recommended). An implementation is allowed to check the bus busy status
 550 only at the conclusion of the T_{IDLE_WINDOW} interval that is measured by the device.

551 **6.14 NACK window**

552 An endpoint/bridge is required to NACK an incoming packet if the device does not have input buffer
 553 space available for the packet. For the NACK to be recognized by the transmitter as the NACK for a
 554 packet retry, the first NACK bit shall be issued no earlier than byte two (that is, the Command Code byte)
 555 and no later than byte 8 (the MCTP flags byte). These bytes are represented by the bold outlined bytes in
 556 the figure below **Error! Reference source not found.** After the first NACK has been issued any
 557 subsequent bytes that are received for the packet shall also be NACK'd until a START, STOP, or bus free
 558 condition is detected.

559 An endpoint/bridge that NACKs a packet shall continue to NACK any remaining bytes for the transaction
 560 until it recognizes the next START or STOP condition on the bus.

561



562

563 **Figure 3 – Allowed byte range for first NACK'd byte**

564

6.15 Fairness arbitration requirements for MCTP bridges

MCTP bridges that support fairness arbitration shall meet the following requirements:

- 567 • The bridge shall support FAIR_IDLE detection and implement the corresponding fairness policy
568 separately for each port on the bridge.
- 569 • Upon device power up or initialization, a port does not need to detect a FAIR_IDLE condition
570 before first attempting to access the bus.
- 571 • A bridge that loses arbitration when attempting to transmit shall continue to retry the transaction
572 when the bus becomes free for up to PN2 retries (see Table 8). If the retry limit is reached, the
573 bridge shall drop the packet data.
- 574 • A bridge that receives a NACK when attempting to transmit to a given physical address shall
575 continue to retry the transaction when the bus becomes free for up to PN2 retries. The bridge
576 will return to attempting to arbitrate for the bus as described in the preceding requirement,
577 restarting its number of arbitration retries. If the retry limit is reached, the bridge shall drop the
578 packet data.
- 579 • An MCTP bridge shall provide dedicated input buffer space per port. The minimum input buffer
580 size is large enough to store one full baseline MTU-sized MCTP packet. It is recommended, but
581 not required, that a bridge also implement a dedicated output buffer per port, sized to store at
582 least one full baseline MTU-sized MCTP packet.
- 583 • If the MCTP bridge is the target of an MCTP packet and it does not have enough buffer space in
584 its input buffer to store the full packet, it shall NACK the packet. If the bridge has an output
585 packet to transmit on that same port, it shall be able to issue a START within T_{START_WINDOW} after
586 issuing the retry NACK.
- 587 • A bridge is required to drop a received packet if it finds that the packet error code (PEC) byte for
588 the transaction is incorrect.
- 589 • An MCTP bridge is not allowed to perform "connected" transactions where the decision to ACK
590 or NACK an incoming packet is dependent on the bridge's ability to acquire the destination bus
591 prior to accepting the packet.
- 592 • MCTP bridges are required to implement "store and forward" packet processing. That is, once a
593 bridge has accepted a packet for routing, it shall retain that packet until it can successfully
594 transmit it onto the target bus (except when running out of retries when trying to access the
595 target bus, or upon receiving a packet for a bus that is unavailable or an endpoint that is not
596 present.)
- 597 • A bridge cannot make the acceptance of a receive packet on its upstream port (port that
598 connects to a bus that is not owned by the bridge itself) conditional on its ability to transmit a
599 packet on its upstream port. This requirement does not apply to a downstream port on a bridge
600 (that is, a downstream port may elect to NACK an incoming packet to allow the bridge to
601 transmit from that port). This requirement is to help avoid deadlock situations if a bridge is
602 required to route a packet back onto the bus from which the packet came.
- 603 • A bridge that receives a NACK while it is performing a Master Write operation is not required to
604 immediately conclude the Master Write operation and drop off the bus. The bridge may continue
605 the write operation through its conclusion. In either case, the master shall always conclude its
606 transaction with a STOP condition, unless some other device on the bus first produces a
607 START or STOP condition. The latter situation is an erroneous condition on the bus, but bridges
608 shall be able to handle it. Devices shall always recognize START and STOP conditions
609 regardless of the transaction or bit position on which they occur.

610 6.16 Fairness arbitration requirements for non-bridge endpoints

611 Non-bridge/bus owner endpoints ("simple endpoints") are required to implement the MCTP fairness
612 arbitration extensions (when enabled) as follows:

613 • The endpoint's port shall support FAIR_IDLE detection and implement the corresponding
614 fairness policy.

615 • Upon device power up or initialization, the endpoint does not need to detect a FAIR_IDLE
616 condition before first attempting to access the bus.

617 • The endpoint cannot make the acceptance of a receive packet conditional on its ability to
618 transmit a packet (that is, a simple endpoint shall not NACK incoming packets because it is
619 trying to send an outgoing packet).

620 Meeting this requirement may require the endpoint to have separate transmit and receive
621 buffers. This is the recommended implementation.

622 If a device is severely limited in buffer space and cannot allocate separate space for both
623 transmit and received data, options are for the endpoint to allow its buffer to be over-written by
624 the receive packet, or in some cases the endpoint may elect to do a dummy receive of the
625 incoming packet (that is, ACK the incoming bytes, but internally drop them as they are coming
626 in.)

627 • Higher layer protocols shall be used to handle the case when the endpoint is targeted by more
628 messages than it can process. The buffering requirement for the MCTP Control Protocol
629 messages is defined in the MCTP Base Specification. Buffering requirements for other message
630 types are defined in the respective specifications for the message type.

631 • An endpoint is allowed to NACK a packet if it is temporarily unable to accept it (for example,
632 because of an *input* buffer-full condition). This should typically only occur if the endpoint is the
633 target of packets from more than one source endpoint.

634 • There is no direct limit of how long a non-bridge endpoint is allowed to successively NACK
635 incoming packets. However, there are limits on how many packet-level retries a transmitter will
636 attempt before it drops the transmitted packet, as well as message type-specific limits on how
637 long and how many times a given message will be retried.

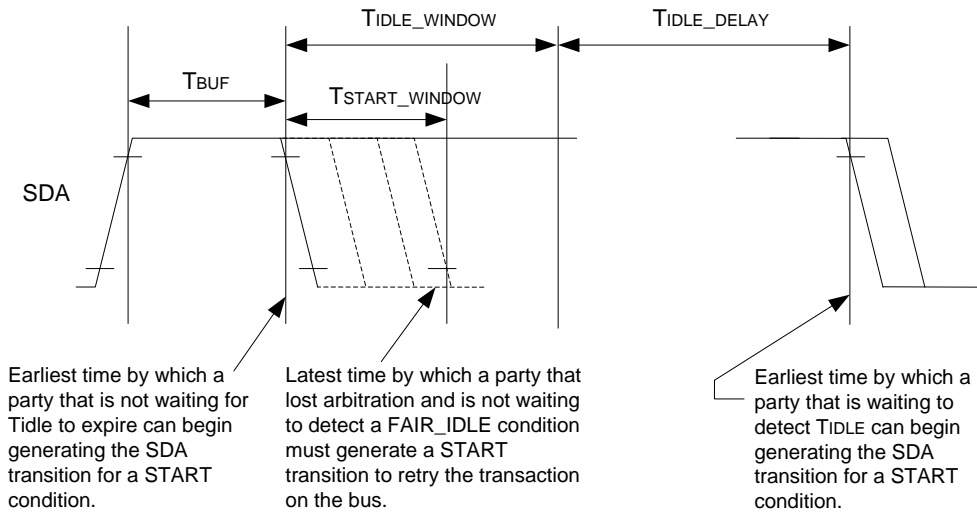
638 • If an endpoint has an output transmit packet and it NACKs an input receive packet from lack of
639 input buffer space, it shall be able to issue a START condition to transmit the output packet
640 within T_{START_WINDOW} after the bus becomes free, unless the endpoint is waiting to detect T_{IDLE} .

641 • An endpoint that receives a NACK while it is performing a Master Write operation is not required
642 to immediately conclude the Master Write operation and drop off the bus. The endpoint may
643 continue the write operation through its conclusion. In either case, the master shall always
644 conclude its transaction with a STOP condition, unless some other device on the bus first
645 produces a START or STOP condition. The latter situation is an erroneous condition on the bus,
646 but bridges shall be able to handle it. Devices shall always recognize START and STOP
647 conditions regardless of the transaction or bit position on which they occur.

648 • Endpoints that are NACK'd or lose arbitration shall retry transaction for PN1 retries (see
649 Table 8).

650 **6.17 Fairness arbitration timing**

651 Figure 4, Table 5, and Table 6 present the specifications for the timing intervals for fairness arbitration on
 652 SMBus and I²C relative to the data (SDA) signal. Refer to [SMBus](#) and [I²C](#) for the additional specifications
 653 on the relationship between SCL and SDA for STOP, bus idle, and START conditions.



654

655 **Figure 4 – Fairness arbitration timing measurement for SMBus and I²C**

656 **Table 5 – Fairness arbitration timing values for 100 kHz SMBus/I²C**

Symbol	Min	Max	Unit	Notes
T _{BUF}	4.7	–	μs	Per SMBus 100 kHz specification
T _{START_WINDOW}	–	20	μs	Window of time within which a device that is not waiting to detect a FAIR_IDLE condition shall generate START if the device is retrying to gain bus access after losing arbitration.
T _{IDLE_WINDOW}	30	60	μs	Window of time within which a device that is waiting to detect a FAIR_IDLE condition shall not detect a bus busy condition. A FAIR_IDLE condition exists when bus busy is not detected within this interval.
T _{IDLE_DELAY}	31	–	μs	A device that detects FAIR_IDLE condition shall wait this delay before attempting to generate START. This delay accommodates the difference between the T _{IDLE_WINDOW} intervals implemented by different devices on the bus, plus additional time to accommodate bus skews between devices that are generating START and devices that are monitoring for it. This guarantees that one party that has detected T _{IDLE_WINDOW} does not generate START before other devices that are detecting FAIR_IDLE have completed checking for their T _{IDLE} window. Otherwise, the other devices would not see a FAIR_IDLE condition even though one occurred. (Therefore T _{IDLE_DELAY} shall be greater than the difference between the T _{IDLE_WINDOW} maximum and minimum.)

657

Table 6 – Fairness arbitration timing values for 400 kHz I²C

Symbol	Min	Max	Unit	Notes
T _{BUF}	1.3	–	μs	Per I ² C 400 kHz specification
T _{START_WINDOW}	–	4	μs	Window of time within which a device that is not waiting to detect a FAIR_IDLE condition shall generate START if the device is retrying to gain bus access after losing arbitration.
T _{IDLE_WINDOW}	5	20	μs	Window of time within which a device that is waiting to detect a FAIR_IDLE condition shall not detect a bus busy condition. A FAIR_IDLE condition exists when bus busy is not detected within this.
T _{IDLE_DELAY}	16	–	μs	A device that detects FAIR_IDLE condition shall wait for this delay before attempting to generate START. This delay accommodates the difference between the T _{IDLE_WINDOW} intervals implemented by different devices on the bus, plus additional time to accommodate bus skews between devices that are generating START and devices that are monitoring for it. This guarantees that one party that has detected T _{IDLE} does not generate START before other devices that are detecting T _{IDLE} have completed their T _{IDLE} window. Otherwise, the other devices would not see a FAIR_IDLE condition even though one occurred. (Therefore T _{IDLE_DELAY} shall be greater than the difference between the T _{IDLE_WINDOW} maximum and minimum.)

658

Table 7 – Fairness arbitration timing values for 1MHz I²C

Symbol	Min	Max	Unit	Notes
T _{BUF}	0.5	–	μs	Per I ² C 1MHz specification
T _{START_WINDOW}	–	2	μs	Window of time within which a device that is not waiting to detect a FAIR_IDLE condition shall generate START if the device is retrying to gain bus access after losing arbitration.
T _{IDLE_WINDOW}	3	6	μs	Window of time within which a device that is waiting to detect a FAIR_IDLE condition shall not detect a bus busy condition. A FAIR_IDLE condition exists when bus busy is not detected within this interval.
T _{IDLE_DELAY}	3.1	–	μs	A device that detects FAIR_IDLE condition shall wait this delay before attempting to generate START. This delay accommodates the difference between the T _{IDLE_WINDOW} intervals implemented by different devices on the bus, plus additional time to accommodate bus skews between devices that are generating START and devices that are monitoring for it. This guarantees that one party that has detected T _{IDLE_WINDOW} does not generate START before other devices that are detecting FAIR_IDLE have completed checking for their T _{IDLE} window. Otherwise, the other devices would not see a FAIR_IDLE condition even though one occurred. (Therefore T _{IDLE_DELAY} shall be greater than the difference between the T _{IDLE_WINDOW} maximum and minimum.)

659 **6.18 MCTP packet timing requirements**

660 The timing specifications shown in Table 8 are specific to MCTP packet transfers on SMBus. Timing is
 661 specified for a "point-to-point" connection. That is, timing is specified as if there were only two endpoints
 662 in direct communication on the bus. In particular, the timing specifications assume that there is no clock
 663 stretching that occurs due to other parties on the bus.

Table 8 – Timing specifications for MCTP packets on SMBus/I²C

Timing Specification	Symbol	Value	Description
Endpoint packet level retries	PN1	8	Number of times a non-bridge endpoint shall retry sending an MCTP packet upon receiving a NACK during the specified window (see Figure 3). An endpoint that gets successive NACKs shall do one retry for each NACK up to at least this number of retries. This also includes bridges when bridges are transmitting as an endpoint (as opposed to a bridge transmitting from its routing functionality).
Bridge packet level retries	PN2	12	Number of times an MCTP bridge (when transmitting packet for routing) shall retry sending an MCTP packet upon receiving a NACK during the specified window (see 6.14 Error! Reference source not found.). A bridge shall do one retry on each NACK up to this number.
Packet transaction originator duration	PT1a	250 μ s per byte ^[1]	The overall duration shall be less than the specified interval times the number of bytes in the packet, starting from the byte following the slave byte through and including the PEC byte. Individual data byte transmissions may exceed the specification provided the cumulative duration for the packet is met.
Originator slave address byte duration	PT1b	250 μ s ^[1]	The amount of time, including any clock stretching, used to transmit the slave address, Wr, and ACK bits on the bus.
Slave-induced clock stretching	PT1c	250 μ s per byte ^[1]	MCTP devices that are receiving MCTP packets shall not clock stretch the overall packet more than the specified amount. Note that MCTP devices may share the bus with non-MCTP SMBus devices that cause clock stretching that exceeds this specification.
The PT2 parameters are intended to help guide a controller in determining when it is acceptable to initiate a Master Write transaction if the controller powers up or initializes itself on a bus segment that may already be active. It also helps controllers know when it is acceptable to continue under conditions where a STOP condition may have been lost because a controller dropped off the bus due to an error condition. An implementation shall meet at least one of specifications PT2a or PT2b.			
Time-out waiting for bus free without seeing a STOP condition (Bus free determined by not detecting START or STOP)	PT2a	100 ms	For controllers that have hardware that can only detect bus-free/busy-busy status by monitoring for START and STOP conditions, the controller can assume the bus is free if PT2a seconds goes by without detecting a START or STOP condition. If a START condition is detected, the time-out interval restarts. If a STOP condition is detected, the controller can assuming the bus is free following the TBUF interval specified in SMBus . NOTE: This interval effectively places an upper limit on the duration of a single transaction. The byte count in an MCTP packet limits the size of the transaction to 260 bytes. 100 ms is more than sufficient to cover this transfer.

Timing Specification	Symbol	Value	Description
Time-out waiting for bus free without seeing a STOP condition (Bus free determined by data/clock activity)	PT2b	50 μs	The SMBus specification defines a bus-free (idle) condition as TBUF seconds after a STOP condition, or by the data and clock lines being high for PT2b seconds (where the value for PT2b is taken from T _{HIGH, max} as defined in SMBus). If a controller has appropriate hardware support, monitoring PT2b and TBUF can be used to determine the bus-free (idle) condition in lieu of PT2a. This is generally the most efficient and highest performance way to detect bus free on SMBus. SYSTEM IMPLEMENTATION NOTE: If "bit banded" I ² C devices may be used on the same segment, it is important to ensure that those devices do not drive the clock and data high for more than T _{HIGH, max} seconds during transactions.
SDA Low Timeout	PT3	2 sec min, 5 sec max	Time for a bus owner to monitor the SDA low level for a "Stuck 0" before attempting to clear the condition. (See 6.20.)
NOTE 1: Intervals include the ACK bit associated with the byte.			

665 **6.19 MCTP control message timing requirements**

666 The following timing specifications are specific to MCTP control messages on SMBus/I²C. Timing is
 667 specified for a "point-to-point" connection. That is, timing is specified as if there were only two endpoints
 668 in direct communication on the bus. In particular, the timing specifications assume that there is no clock
 669 stretching occurs due to other parties on the bus.

670 Response specifications are given assuming that the requester is able to operate at full speed on the bus.
 671 That is, clock stretching, if any, is solely generated by the requester.

672 Responses are not retried. A "try" or "retry" of a request is defined as a complete transmission of the
 673 MCTP control message.

674 **Table 9 – Timing Specifications for MCTP control messages on SMBus**

Timing Specification	Symbol	Min	Max	Description
Endpoint ID reclaim	Treclaim	5 sec	–	Minimum time that a bus owner shall wait before reclaiming the EID for a non-responsive hot-plug endpoint.
Number of request retries	MN1	2	See descr.	Total of three tries, minimum: the original try plus two retries. The maximum number of retries for a given request is limited by the requirement that all retries shall occur within MT4, max of the initial request.
Request-to-response time	MT1	–	100 ms	This interval is measured at the responder from the end of the reception of the MCTP Control Protocol request to the beginning of the transmission of the response. This requirement is tested under the condition where the responder can successfully transmit the response on the first try.

Timing Specification	Symbol	Min	Max	Description
Time-out waiting for a response	MT2	MT1 max+ 2*MT3 max	MT4, min[1]	This interval is measured at the requester from the end of the successful transmission of the MCTP Control Protocol request to the beginning of the reception of the corresponding MCTP Control Protocol response. This interval at the requester sets the minimum amount of time that a requester should wait before retrying an MCTP Control Protocol request. Note: This specification does not preclude an implementation from adjusting the minimum time-out waiting for a response to a smaller number than MT2 based on measured response times from responders. The mechanism for doing so is outside the scope of this specification.
Transmission Delay	MT3	-	100 ms	Time to take into account transmission delay of an MCTP Control Protocol Message. Measured as the time between the end of the transmission of an MCTP Control Protocol message at the transmitter to the beginning of the reception of the MCTP Control Protocol message at the receiver.
Instance ID expiration interval	MT4	5 sec [2]	6 sec	Interval after which the instance ID for a given response will expire and become reusable if a response has not been received for the request. This is also the maximum time that a responder tracks an instance ID for a given request from a given requester.
NOTE 1: Unless otherwise specified, this timing applies to the mandatory and optional MCTP commands.				
NOTE 2: If a requester is reset, it may produce the same sequence number for a request as one that was previously issued. To guard against this, it is recommended that sequence number expiration be implemented. Any request from a given requester that is received more than MT4 seconds after a previous, matching request should be treated as a new request, not a retry.				

675

676 6.20 "Stuck 0" condition handling

677 A possible condition exists in SMBus and I²C where a slave device that is being read or is driving ACK
678 could be left driving a low (0) level onto the data line (SDA) of the bus. The bus uses a "wire OR'd"
679 approach, where the low (0) level takes precedence over the high (1) level. Therefore, if one party drives
680 a low (0) level onto the bus, the bus cannot go to a high (1) level until the low level is released.

681 This means that no other transactions can occur until this condition is cleared (because generating a
682 START or STOP condition on the bus requires being able to drive a high-to-low or low-to-high transition
683 on the data line, respectively).

684 This condition can occur due to the premature termination of a transaction from the master (as could
685 happen on device resets, power cycles, or firmware restarts, for example) or could occur due to the loss
686 of a clock due to electrical noise.

687 Effectively, what happens is that the device that was being accessed does not recognize that the
688 transaction has been terminated or that a clock was missed. The device continues to drive the 0 onto the

689 bus because it is waiting to get more clocks from the master to conclude the transaction, but those clocks
690 will never come unless some bus master takes steps to generate them.

691 The solution to this condition is to have a master clock the bus until the SDA line goes high, at which point
692 the master can issue a START or STOP condition to get the bus back in synchronization.

693 To accomplish this, the master needs to be able to access and clock the bus without paying attention to
694 the present state of the SDA line.

695 Many microcontrollers have the ability to have firmware dynamically reconfigure their SMBus pins as
696 general purpose I/O pins. If this is supported, it is straightforward for firmware to generate the necessary
697 clocks on the SCL line by bypassing the SMBus controller hardware and using programmed I/O to control
698 the pins instead. The firmware would then simply clock the bus until it sees a "1" condition on the SDA
699 line and then a new SMBus transaction can be launched.

700 NOTE It is recommended that MCTP bus owners include a provision to detect and clear Stuck 0 conditions on
701 SMBus buses that they own. The controller should do this if it can detect that a constant 0 condition has existed on
702 the SDA line for more than PT3 seconds.

703 **6.21 MCTP over SMBus/I²C protocol anti-aliasing**

704 MCTP over SMBus has been designed to allow one endpoint to support multiple protocols, such as ASF,
705 IPMI, or legacy device-specific protocols with a single slave address. The following clauses describe
706 provisions that can help support implement MCTP over SMBus in devices that also need to support other
707 SMBus or I²C protocols.

708 **6.21.1 IPMI**

709 The IPMI protocols for SMBus (IPMI over SMBus) and I²C (Intelligent Platform Management Bus, IPMB)
710 use the fourth byte of the transaction as a Source Slave Address byte, as does MCTP over SMBus.
711 However, the IPMI protocols require the least significant bit of that byte to be 0b, whereas MCTP over
712 SMBus requires the bit to be 1b. Thus, a device that needs to differentiate between MCTP over SMBus
713 and the IPMI SMBus/I²C protocols can do so using that bit.

714 **6.21.2 ASF**

715 MCTP over SMBus uses the ASF specification reserved value of 0x0F for the command byte. Thus, the
716 ASF-defined commands that use SMBus block-write protocol can be differentiated from MCTP over
717 SMBus block-write using the command byte value. If necessary, other ASF SMBus write transactions,
718 such as those for legacy sensor and control access can be differentiated from MCTP packets based on
719 the length of the transaction. The ASF transactions are all shorter.

720 **6.21.3 Integrating MCTP with legacy SMBus functions**

721 This clause describes some possible options if MCTP is being added to a device that shall also support
722 functions using a non-MCTP SMBus interface.

723 In general, there should be no problems having those functions co-exist with MCTP provided that the
724 legacy SMBus operations do not require generating or accepting write transactions that use the MCTP
725 value of 0x0F.

726 If the SMBus device currently uses the 0x0F MCTP command value for a device-specific purpose and it
727 wants to use the same slave address, the following can be done:

- 728 • The device-specific command can be moved to a different command value. This is generally the
729 most straightforward approach if it can be supported.
- 730 • Depending on the device-specific command definition, it may be possible to differentiate
731 between the command and MCTP packets based on other differences, such as the overall
732 length of the command or differences between the values in the fourth or fifth bytes of the
733 command. (MCTP always uses 1b as the least significant bit of the fourth byte, and the fifth
734 byte holds a fixed 4-bit value for the Header Version.)
- 735 • The device can implement MCTP over SMBus on a separate slave address from the legacy
736 functions.

737 6.22 Well-known and reserved slave addresses

738 For bus segments that support ARP-able devices, Table 10 summarizes addresses that are generally
739 reserved by SMBus or I²C and should either be avoided by devices. In addition, some are reserved (not
740 to be used as a general device slave address) because those addresses are related to functions that are
741 used by MCTP.

742 **Table 10 – Well-known and reserved slave addresses**

Slave Address bits [7:1]	R/W# bit [0]	Hex ^[7]	Comment	Disposition
0000 000	0	0x00	I ² C general call address, IPMI broadcast	avoid ^[1]
0000 000	1	0x01	START byte	avoid ^[2]
0000 001	X	0x02, 0x03	CBUS address	avoid ^[3]
0000 010	X	0x04, 0x05	Address reserved for different bus format	avoid
0000 011	X	0x06, 0x07	Reserved for future use by I ² C specifications	avoid
0000 1XX	X	0x08–0x0F	I ² C specification, high-speed mode master code	avoid ^[4]
0001 000	X	0x10	SMBus host	rsvd
0001 100	X	0x18, 0x19	SMBus Alert Response address	rsvd
0010 000	X	0x20, 0x21	IPMI BMC address	avoid ^[5]
0101 000	X	0x50, 0x51	Reserved for ACCESS.bus host	avoid (ACCESS.bus defunct)
0110 111	X	0x6E, 0x6F	Reserved for ACCESS.bus default address	avoid
1111 0XX	X	0xF0–0xF7	I ² C 10-bit slave addressing ^[1]	avoid ^[6]
1111 1XX	X	0xF8–0xFF	Reserved for future use by I ² C specifications	avoid

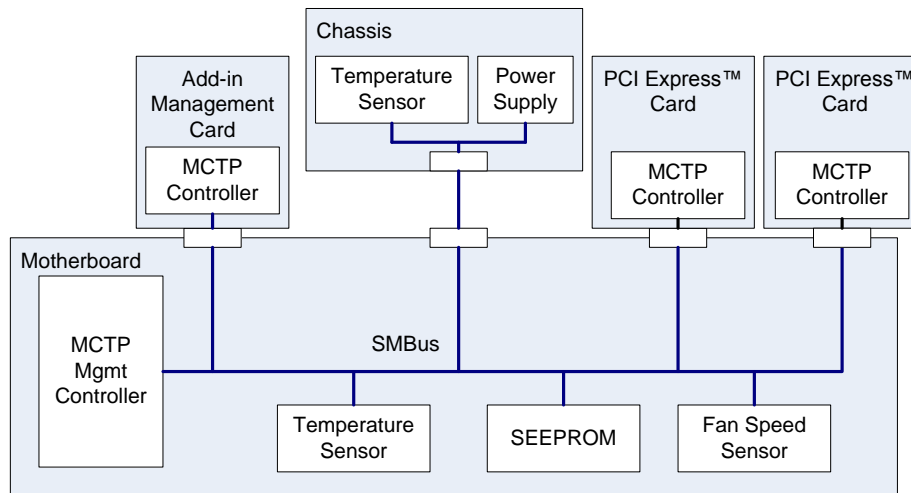
Slave Address bits [7:1]	R/W# bit [0]	Hex ^[7]	Comment	Disposition
1100 001	x	0xC2, 0xC3	SMBus Device Default address	rsvd. Used for SMBus ARP with MCTP
<p>NOTE 1. This address is used as a broadcast address in IPMI and I²C. It should be avoided if IPMI management controllers may be used on the same bus segment. In I²C, it is reserved for two purposes: to broadcast a portion of an address that is used for devices that have a portion of their address that is configurable, and as an optional mechanism for a device to master and broadcast its slave address onto the bus. MCTP does not support the use of this address for the I²C address assignment or slave address broadcast purposes.</p> <p>NOTE 2. The I²C START byte is a pre-amble to the slave address that is intended to provide time for firmware driven I²C interfaces to shift into polling of I²C clock and data lines after a START condition has been detected. This is a very rarely used option in I²C. MCTP does not support the use of the START byte with MCTP or non-MCTP devices.</p> <p>NOTE 3. CBUS is an ancestor of I²C, developed by Philips Semiconductor. It uses a data and clock signal similar to I²C, but with a third signal (SEN) used to generate the START and STOP conditions on the bus. This address range was reserved by the I²C specification to enable a degree of backward compatibility with CBUS devices sharing the I²C SCL and SDA lines as the CBUS clock and data lines, respectively. While listed as a reserved address in the I²C specification, few SMBus/I²C implementations using MCTP will have any need to also support CBUS devices.</p> <p>NOTE 4. MCTP is not defined to support I²C high-speed mode operation.</p> <p>NOTE 5. This address is the "well known address" for an IPMI BMC. This address should be avoided if an IPMI BMC may be used on the same MCTP segment.</p> <p>NOTE 6. Used in conjunction with the R/W# bit position to deliver the most-significant three address bits for I²C 10-bit addressing. MCTP protocols and data structures do not support 10-bit addressing on SMBus or I²C segments. MCTP only supports 7-bit addresses for MCTP and non-MCTP devices on a bus segment.</p> <p>NOTE 7. By convention, when the 7-bit slave address field is represented as a two-digit hexadecimal number, it is treated as an 8-bit value where the 7-bit address occupies the upper 7 bits and the least significant bit is 0b or 1b according to the value of the SMBus/I2C Read-Write bit associated with the slave address.</p>				

743 **6.23 Fixed address allocation**

744 One of the problems that an implementer often faces is choosing which slave address to use. For the
 745 PCI™ and PCI Express™ bus specifications, the specifications require that devices on standard
 746 connectors defined by those specifications have their addresses set through SMBus ARP. Therefore,
 747 fixed address allocation is not an option for PCI add-in cards themselves. In fixed bus implementations,
 748 however, there are many situations where it is desired or necessary to utilize fixed-address devices.

749 From a practical point-of-view, SMBus and I²C do not have an effective central registry or other
 750 mechanism for avoiding conflicts in the assignment and use of slave addresses among device vendors.
 751 While there are potential registries of device slave address usage for SMBus (under the System
 752 Management Interface Forum) and I²C (from Philips Semiconductor), these have not generally been used
 753 by device vendors and there is no group or standard that works to enforce conformance to those
 754 registries.

755 Most device vendors provide a configurable range of three or more addresses to enable an implementer
 756 to reconcile address conflicts on a single segment. Because typically only a small number of fixed-
 757 address devices are used on a given segment, it is frequently possible to configure devices so they do
 758 not have overlapping addresses. This approach is problematic, however, in situations where a component
 759 that is attached to that segment in the platform may come from several sources. Clause 6.23 provides
 760 guidelines to allocating fixed addresses that are designed to reduce the number of conflicts that could
 761 occur when multiple suppliers provide different elements of a computer system (see Figure 5 for an
 762 example).



763

764

Figure 5 – Example system configuration

765 6.24 Recommended address range allocation for computer systems

766 This clause provides a recommended allocation of SMBus addresses between board, chassis, and add-in
 767 uses that help avoid address conflicts when fixed addresses are used. It also serves as a general
 768 guideline of what addresses a generic ARP master should use for allocation to PCI/PCIe add-in cards.

769 There might be cases when MCTP is used within a typical computer system application where the
 770 motherboard may come from one supplier, the chassis from another supplier, and possibly add-in
 771 modules from yet another supplier. To facilitate the mix-and-match of these elements and to help avoid
 772 the need for every system manufacturer to set up their own address allocation conventions with suppliers,
 773 MCTP recommends that system manufacturers follow the address allocation approach initially defined by
 774 the IPMI specifications (see Table 11). This approach splits the available fixed addresses (addresses
 775 other than reserved addresses) into four main usage areas:

776 **B Board:** An area reserved for board set manufacturer use (where *board set* would be the
 777 motherboard and other boards that accompany that motherboard from the same vendor).

778 **C Chassis:** An area reserved for use by vendors that make chassis in which a third-party board
 779 set would be used.

780 **A Add-in:** For third-party add-in devices (for example, modules or add-in cards that used fixed
 781 addresses and would be used in combination with a motherboard or chassis where there is a
 782 connection to a SMBus segment implementing MCTP).

783 NOTE PCI/PCIe add-in cards that use standard PCI connectors are required to support SMBus ARP
 784 and fixed addresses are not used.

785 **R Reserved** for IPMI, I²C, SMBus, or MCTP uses. Includes the *avoid* addresses from Table 10.

786 By following this convention, future motherboards can offer connections to chassis elements and third-
 787 party modules where those devices can use fixed addresses, if required. It also provides a convention to
 788 avoid conflicts if legacy non-MCTP devices share the same SMBus segment.

Table 11 – Slave address allocation for computer systems

Use	Address	Typical Device	Use	Address	Typical Device
R	0x00	I ² C, IPMB broadcast	C		
	0x01	I ² C			
	0x02	I ² C		0x48	SMBUS/I2C IO Expander, such as 8574
	0x04-0x0E	I ² C		0x4A	SMBUS/I2C IO Expander, such as 8574
	0x20	IPMB uC (BMC)		0x4C	SMBUS/I2C IO Expander, such as 8574
	0x50	ACCESS.bus		0x4E	SMBUS/I2C IO Expander, such as 8574
	0x6E	ACCESS.bus		0x52-0x6C	58h, 5Ah, 5Ch = Heceta
	0xF0-0xF6	I ² C			
	0xF8-0xFE	I ² C			
A	0x10	SMBus host (B)	0x78	SMBUS/I2C IO Expander, such as 8574A	
	0x12-0x16		0x7A	SMBUS/I2C IO Expander, such as 8574A	
	0x18	SMBus Alert Response address (B)	0x7Ch	SMBUS/I2C IO Expander, such as 8574A	
	0x1A-0x1E		0x7E	SMBUS/I2C IO Expander, such as 8574A	
	0x30-0x3E		0x9A	TEMPERATURE SENSORS, SUCH AS LM75, DS1624, DS1621	
			0x9C	uC (pri. HSC), DS1624, DS1621	
	0xD0-0xDE		0x9E	uC (sec. HSC), DS1624, DS1621	
B	0x22	uC (FPC, ICMB) ^[1]	0xA0-0xA2	FRU (Power Supply FRU or SEEPROM)	
	0x24	uC (PBC) ^[1]	0xAC	SEEPROM	
	0x26		0xAE	SEEPROM	
	0x28	SM Card ^[1]	0xB0-0xB2	Power Supply Device (PMBus)	
	0x2A-0x2E		0xE8-0xEE	I2C Bus Switch	
	0x40	SMBUS/I2C IO Expander, such as 8574A	NOTE 1: Term from IPMI usage. FPC = front panel controller, PBC = Power Backplane Controller, ICMB = Intelligent Chassis Management Bus bridge, SM Card = System Management Card		
	0x42	SMBUS/I2C IO Expander, such as 8574A			
	0x44	SMBUS/I2C IO Expander, such as 8574A			
	0x46	SMBUS/I2C IO Expander, such as 8574A			
	0x70	SMBUS/I2C IO Expander, such as 8574A			
	0x72	SMBUS/I2C IO Expander, such as 8574A			

Use	Address	Typical Device	Use	Address	Typical Device
	0x74	SMBUS/I2C IO Expander, such as 8574A			
	0x76	SMBUS/I2C IO Expander, such as 8574A			
	0x80-0x8E				
	0x90	TEMPERATURE SENSORS, SUCH AS LM75, DS1624, DS1621, 8591			
	0x92	TEMPERATURE SENSORS, SUCH AS LM75, DS1624, DS1621, 8591			
	0x94	TEMPERATURE SENSORS, SUCH AS LM75, DS1624, DS1621, 8591			
	0x96	TEMPERATURE SENSORS, SUCH AS LM75, DS1624, DS1621, 8591			
	0x98	TEMPERATURE SENSORS, SUCH AS LM75, DS1624, DS1621, 8591			
	0xA4	EEPROM			
	0xA6	EEPROM			
	0xA8	EEPROM			
	0xAA	EEPROM			
	0xC0				
	0xC2	SMBus Device Default address			
	0xC4-0xCE				
	0xE0-0xE6	I2C Bus Switch			

ANNEX A (informative)

Notation

790
791
792
793
794

795 Notations

796 Examples of notations used in this document are as follows:

- 797 • 2:N In field descriptions, this will typically be used to represent a range of byte offsets
798 starting from byte two and continuing to and including byte N. The lowest offset is on
799 the left, the highest is on the right.
- 800 • (6) Parentheses around a single number can be used in message field descriptions to
801 indicate a byte field that may be present or absent.
- 802 • (3:6) Parentheses around a field consisting of a range of bytes indicates the entire range
803 may be present or absent. The lowest offset is on the left, the highest is on the right.
- 804 • PCIe Underlined, blue text is typically used to indicate a reference to a document or
805 specification called out in 2, "Normative References" or to items hyperlinked within the
806 document.
- 807 • rsvd Abbreviation for "reserved." Case insensitive.
- 808 • [4] Square brackets around a number are typically used to indicate a bit offset. Bit offsets
809 are given as zero-based values (that is, the least significant bit [LSb] offset = 0).
- 810 • [7:5] A range of bit offsets. The most significant bit is on the left, the least significant bit is
811 on the right.
- 812 • 1b The lower case "b" following a number consisting of 0s and 1s is used to indicate the
813 number is being given in binary format.
- 814 • 0x12A A leading "0x" is used to indicate a number given in hexadecimal format.
- 815

816
817
818
819
820

ANNEX B (informative)

Change log

Version	Date	Description
1.0.0	2009-07-28	
1.1.0	2017-05-21	Added 1MHz speed mode Moved NACK window to a separate section Corrected timing parameters description Updated reserved addresses mapping in Table 11 Updated Table 2 Updated revisions for the normative references

821