**Document Number: DSP0237**

**Date: 2017-05-21**

**Version: 1.1.0**

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

**Supersedes: 1.0.0**

**Document Class: Normative**

**Document Status: Published**

**Document Language: en-US**

# CONTENTS



# Figures

# Tables

# Foreword

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

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

## Acknowledgments

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

**Editor:**

- Yuval Itkin – Mellanox Technologies

**Contributors:**

- Alan Berenbaum – SMSC
- Klodnicki Edward - IBM
- Kozlowski Koe – Dell Inc
- Patrick Caporale – Lenovo
- Patrick Schoeller – Hewlett Packard Enterprise
- Phil Chidester – Dell Inc
- John Leung - Intel Corporation
- Eliel Louzoun – Intel Corporation
- Hemal Shah - Broadcom Limited
- Tom Slaight – Intel Corporation
- Yi Zeng – Intel Corporation

# Introduction

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

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

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

## 1 Scope

This document provides the specifications for the Management Component Transport Protocol (MCTP) transport binding for SMBus/I$^2$C.

## 2 Normative references

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

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

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

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

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

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

NXP Semiconductors, *$I^2$C-bus specification and user manual*, 4 April 2014 http://www.nxp.com/documents/user_manual/UM10204.pdf

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

## 3 Terms and definitions

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

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

The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 5.

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

The terms defined in DSP0004, DSP0223, and DSP1001 apply to this document. The following additional terms are used in this document.

**3.1**

**Address Resolution Protocol**

**ARP**

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

# 4 Symbols and abbreviated terms

The following symbols and abbreviations are used in this document.

**4.1**

**ACK**

acknowledge

**4.2**

**ARP**

Address Resolution Protocol

**4.3**

**ASF**

Alert Standard Format

**4.4**

**BMC**

baseboard management controller

**4.5**

**EEPROM**

Electrically Erasable Programmable Read-Only Memory

**4.6**

**EID**

endpoint identifier

**4.7**

**I$^2$C**

Inter-Integrated Circuit

**4.8**

**I/O**

input/output

**4.9**

**IPMB**

Intelligent Platform Management Bus

**4.10**
**IPMI**
Intelligent Platform Management Interface

**4.11**
**kHz**
kilohertz

**4.12**
**LSb**
least significant bit

**4.13**
**LSB**
least significant byte

**4.14**
**max**
maximum

**4.15**
**MCTP**
Management Component Transport Protocol

**4.16**
**min**
minimum

**4.17**
**ms**
millisecond

**4.18**
**MSB**
most significant byte

**4.19**
**MTU**
Maximum Transmission Unit

**4.20**
**NACK**
not acknowledge

**4.21**
**PCI**
peripheral component interconnect

**4.22**
**PCIe®**
PCI Express™

**4.23**
**PEC**
packet error code

**4.24**
**PMCI**
Platform Management Component Intercommunications

**4.25**
**PSA**
persistent slave address

**4.26**
**rsvd**
reserved (not case sensitive)

**4.27**
**SCL**
serial clock

**4.28**
**SDA**
serial data

**4.29**
**sec**
second

**4.30**
**SEEPROM**
serial EEPROM

**4.31**
**SMBus**
System Management Bus

**4.32**
**UDID**
unique device identifier

# 5 Conventions

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

## 5.1 Reserved and unassigned values

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

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

## 5.2 Byte ordering

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

# 6 MCTP over SMBus/I$^2$C transport

The MCTP over SMBus/I$^2$C transport binding defines how MCTP packets are delivered over a physical SMBus or I$^2$C medium using SMBus transactions. This includes how physical addresses are used, how fixed addresses are accommodated, how physical address assignment is accomplished for hot-plug or other devices that require dynamic physical address assignment, and how MCTP support is discovered. Timing specifications for bus and MCTP control operations are also given, and a "fairness" protocol is defined for the purpose of avoiding deadlock and starvation/lockout situations among MCTP endpoints.

The binding has been designed to be able to share the same bus as devices communicating using earlier SMBus/I$^2$C management protocols such as Alert Standard Format (ASF) and IPMI, and with vendor-specific devices using SMBus/I$^2$C protocols. The specifications can also allow a given device to incorporate non-MCTP SMBus functions alongside MCTP. This is described in more detail in 6.21.

## 6.1 Terminology

According to SMBus, SMBus devices are categorized as follows, where Address Resolution Protocol (ARP) refers to the SMBus Address Resolution Protocol (a dynamic slave address assignment protocol) and UDID refers to a "unique device identifier", a 128-bit value that a device uses during the ARP process to uniquely identify itself. Because these protocols are implemented with command transactions that are run on top of the SMBus physical specification, it is possible to use these protocols on devices that support an I$^2$C physical interface.

- **ARP-capable**

  SMBus term indicating a device that supports all SMBus ARP commands with the exception of the optional Host Notify command. The slave address is assignable. The device supports both Reset commands.

- **Fixed and Discoverable**

  SMBus term indicating a device supports the Prepare to ARP, directed Get UDID, general Get UDID, and Assign Address commands. The slave address is fixed; the device will accept the Assign Address command but will not allow address reassignment. The device supports both Reset commands.

- **Fixed - Not Discoverable**

  SMBus term indicating a device supports the directed Get UDID command. The slave address is fixed.

- **Non-ARP-capable**

  SMBus term indicating a device does not support any ARP commands. The slave address is fixed.

- **Fixed Address**

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

## 6.2 Transport binding use with I²C

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

## 6.3 MCTP packet encapsulation

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

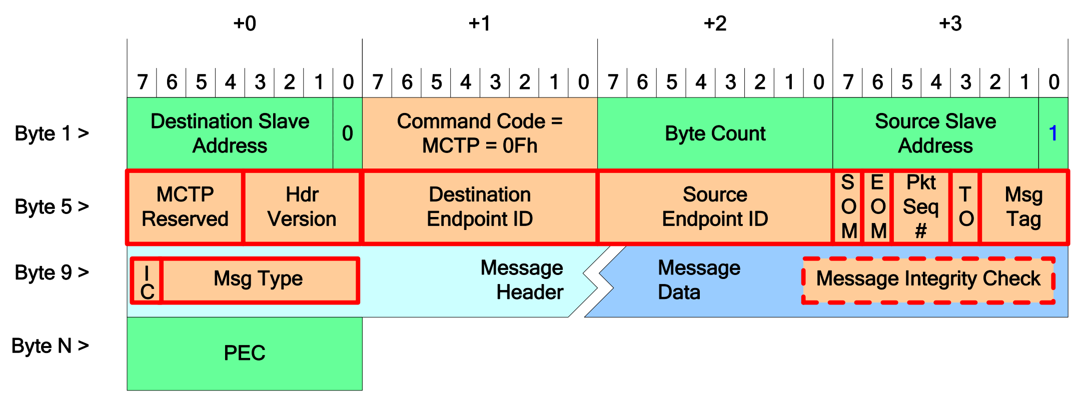


**Figure 1 – MCTP over SMBus/I²C packet format**

**Table 1 – Packet header field descriptions**

| Byte | Block Write Field(s) | Description |
|---|---|---|
| 1 | Slave Address<br>Wr | [7:1] SMBus Destination Slave Address: The slave address of the target device for the local SMBus link<br>[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<br>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.<br>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<br>[7:1] : For the local SMBus link, the slave address of the source device.<br>[0]: This bit shall be set to `1b`. The value enables MCTP to be differentiated from IPMI over SMBus and IPMB (IPMI over $I^2C$) protocols. |
| 5 | Data Byte 2 | [7:4] MCTP reserved: This nibble is reserved for definition by the *MCTP Base Specification*.<br>[3:0] MCTP header version:<br>Set to 0001b for MCTP devices that are conformant to the *MCTP Base Specification* *1.0* and this version of the SMBus transport binding.<br>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 (*)<br>[6] EOM: End Of Message flag (*)<br>[5:4] Packet sequence number (*)<br>[3] Tag Owner (TO) bit (*)<br>[2:0] Message tag (*) |
| 9 | Data Byte 6 | [7] IC: Integrity Check bit (*)<br>[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*. | | |

## 6.4 Bridges and packet formatting

As an MCTP packet travels through a bridge from one SMBus/$I^2C$ port to another, the bridge leaves all packet header and message header and data fields alone with the exception of the source and destination slave address, which shall be modified to route across the intended bus/link. When an MCTP bridge forwards a message from an input port to an output port, it replaces the destination slave address with the targeted slave on the destination bus, and replaces the source slave address with the bridge's slave address.

The MCTP SMBus/$I^2C$ bridge shall re-calculate the PEC byte to account for changes in the source and destination slave address fields.

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

## 6.5 MCTP support discovery

All SMBus devices that support an MCTP endpoint and the SMBus Get UDID command for a particular SMBus/$I^2C$ interface (that is, devices with ARP-capable, fixed and discoverable, or fixed-not discoverable interfaces) are required to have their MCTP support discoverable through the Get UDID command. To do this, endpoints shall return a value of `1b` in bit 5 (the ASF bit) in the Interface field in the Get UDID command.

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

$I^2C$ devices can also support the SMBus protocols and commands for being an ARP-able device that is also discoverable as an MCTP device. This is required for hot-plug $I^2C$ devices using MCTP.

## 6.6 Support for fixed-address devices

MCTP bus owners shall include non-volatile options to record the addresses used by fixed-address devices on SMBus/$I^2C$ buses that they own, and which of those devices support MCTP.

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

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

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

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

It is recommended that general-purpose devices that act as MCTP bus owners allow being configured to support at least 16 different fixed-address devices for each SMBus/$I^2C$ bus they own. This number would include both MCTP and non-MCTP devices.

## 6.7 Supported media

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

**Table 2 – Supported media**

| Physical Media Identifier | Description |
|---|---|
| `0x01` | SMBus 100 kHz compatible |
| `0x02` | SMBus + $I^2C$ 100 kHz compatible |
| `0x03` | $I^2C$ 100 kHz compatible |
| `0x04` | $I^2C$ 400 kHz compatible |
| `0x05` | SMBus + $I^2C$ 1 MHz compatible |

## 6.8 Physical address format for MCTP control messages

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

**Table 3 – Physical address format**

| Format Size | Layout and Description |
|---|---|
| 1 byte | [7:1] slave address bits<br>[0] `0b` |

## 6.9 Get endpoint ID Medium-Specific Information

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

**Table 4 – Medium-Specific Information**

| Description |
|---|
| [7:1] reserved<br>[0] fairness arbitration support (see 6.13)<br>`0b` = not supported<br>`1b` = supported |

## 6.10 Bus owner address

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

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

## 6.11 Bus address assignment

This clause describes the configuration, setup, and operation of communication between MCTP endpoints using SMBus/$I^2C$ as the communication medium.

### 6.11.1 Slave addresses

Each device on SMBus/I$^2$C shall have a slave address to be the target of transactions by bus masters. The MCTP transport protocol solely utilizes Master Write transactions to transfer MCTP packets between MCTP endpoints. For endpoint "A" to send an MCTP packet to endpoint "B", endpoint A shall master the bus and issue a Block-Write transaction to the slave address of endpoint B. Similarly, for endpoint B to send an MCTP packet to endpoint A, it shall master the bus and issue a Block-Write transaction to the slave address of endpoint A. Thus, bi-directional transfer of MCTP packets requires that both sides of the communication have slave addresses.

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

A bus can include a mix of fixed-address and ARP-able devices. Most fixed-address devices do not include a discovery mechanism, and neither SMBus nor I$^2$C require one. Therefore, for a generic bus implementation that support ARP-able devices (such as SMBus to PCI/PCIe connectors) the ARP master needs to know what ranges of addresses are being used for fixed-address devices so that it doesn’t give an ARP-able device an address that conflicts with a fixed-address device.

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

### 6.11.2 Well known and reserved slave addresses

The SMBus and I$^2$C specifications define certain slave addresses that should either be avoided by devices or are reserved (not to be used as a general device slave address) because those addresses are related to functions that are used by MCTP. These addresses are listed in Table 10.

### 6.11.3 Fixed-address recommendations for device manufacturers

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

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

### 6.11.4 Dynamic address assignment (SMBus ARP) support

MCTP buses that support connections to standard PCI/PCIe add-in cards are required by the PCI specifications to support SMBus ARP (be ARP-capable) to allow the devices to be dynamically assigned addresses to avoid address conflicts and eliminate the need for manual configuration of addresses. Figure 2 presents an overview of the address assignment process.

#### 6.11.5 Devices supporting multiple interfaces

Devices that support multiple, separate SMBus or I$^2$C interfaces where the interfaces are intended to be connected to the same bus shall meet the following requirements:

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

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

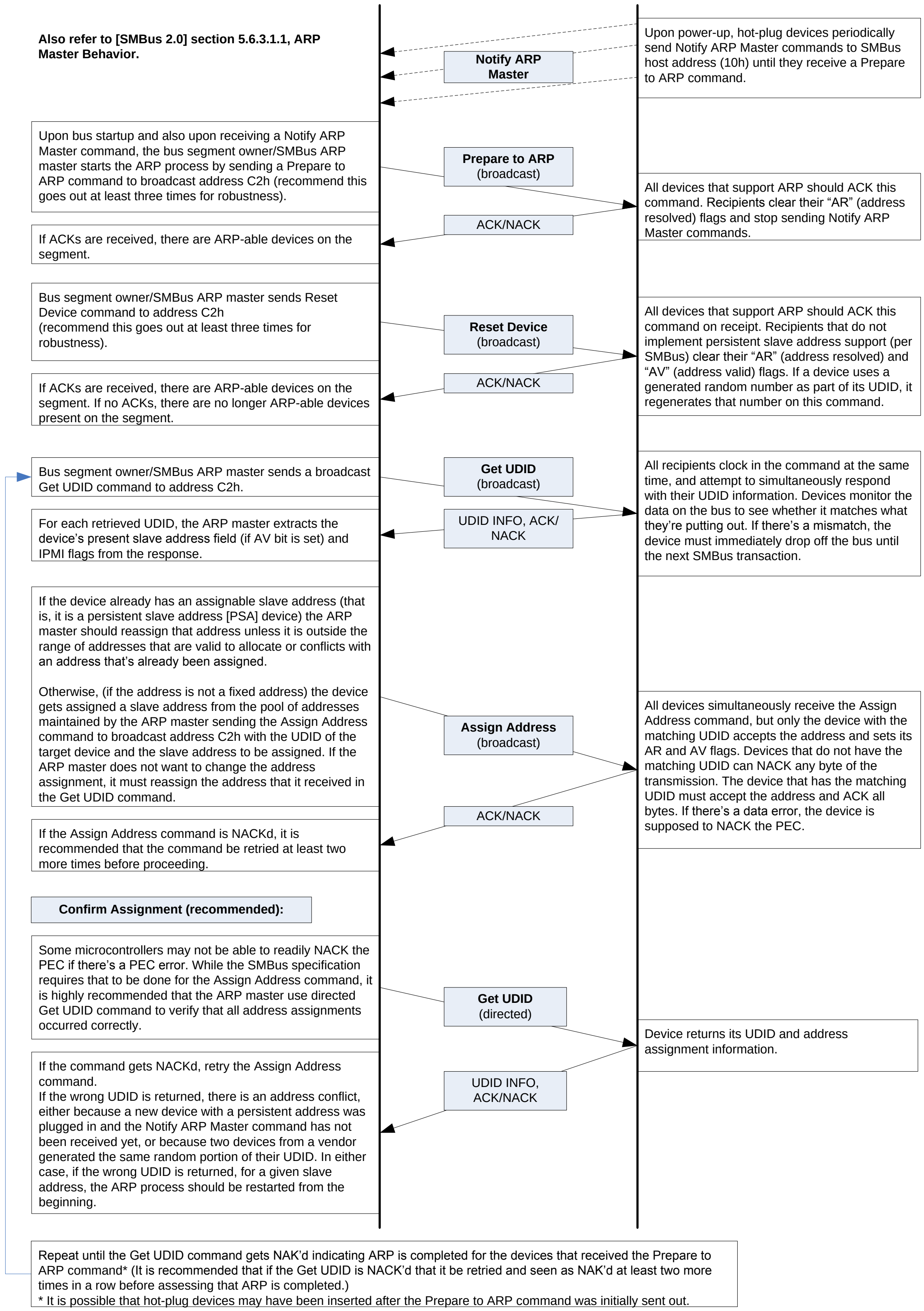


**Figure 2 – Address assignment flow**

### 6.11.6 MCTP requirements on SMBus ARP master support

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

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

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

### 6.11.7 Recommendations on ARP master allocation of slave addresses

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

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

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

### 6.11.8 MCTP requirements on hot-pluggable bridges using SMBus

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

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

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

## 6.12 SMBus/$I^2C$ considerations for MCTP messages

The following applies to MCTP messages on SMBus regardless of their message type. Note that MCTP messages require Block Write byte count sizes that exceed limits specified by SMBus. Additional restrictions on MCTP packets over what the SMBus and $I^2C$ allow are given in 6.3 and 6.18.

### 6.12.1 Slave address ACKs/NACKs

Per SMBus and I$^2$C, the NACK of a slave address indicates the physical absence of the device interface.

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

### 6.12.2 Clock stretching for non-addressed devices

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

## 6.13 Fairness arbitration

### 6.13.1 General

The following clauses describe an extension to the SMBus/I$^2$C arbitration mechanism for device ports that are used with MCTP. The extensions define a 'fairness' mechanism that helps ensure that ports that are arbitrating for access to the bus will eventually get access and will not be locked out of access by other MCTP ports that are using the bus.

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

This mechanism works as follows:

- An MCTP port that wins bus arbitration (per SMBus or I$^2$C) for a given transaction shall wait until it detects a particular bus idle interval before the device can again attempt to arbitrate for the bus. This is referred to as the device waiting to detect the "FAIR_IDLE" condition.
- Once the port has succeeded in detecting the FAIR_IDLE condition, it can attempt to get on the bus and no longer needs to wait to detect the FAIR_IDLE condition. The port can continue to attempt to access the bus without waiting for FAIR_IDLE until the next time the port wins arbitration. After winning arbitration, the port shall again wait to detect the FAIR_IDLE condition before it can attempt to get on the bus.

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

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

1) Be able to recognize the FAIR_IDLE condition. Ports that are waiting to detect a FAIR_IDLE condition shall recognize that no other port has made the bus become busy within a particular window of time ($T_{IDLE\_WINDOW}$) after the bus becomes free.

2) Ports that have not won arbitration shall be able to issue a START condition soon enough after the bus becomes free so that a bus busy condition is seen by ports that are waiting to detect a FAIR_IDLE condition. To ensure this condition is met, START shall be issued by the port within a particular window of time ($T_{START\_WINDOW}$) after the bus becomes free.

NOTE There is actually no explicit indication in SMBus or I$^2$C that arbitration has been won. Instead, what the master detects is that it was able to access the bus and did not have a collision (lose arbitration) with another master. For this specification, this is referred to as *winning arbitration*. Because of the way arbitration works, an MCTP endpoint that is transmitting as a master onto the bus will know that it has won arbitration if it is able to transmit from the destination slave address byte through the end of the source slave address byte (byte 4) without receiving a collision or NACK.

### 6.13.2 Deadlock avoidance with fairness arbitration

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

### 6.13.3 Fairness arbitration support

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

### 6.13.4 Bus busy sampling requirements for fairness arbitration

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

## 6.14 NACK window

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

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

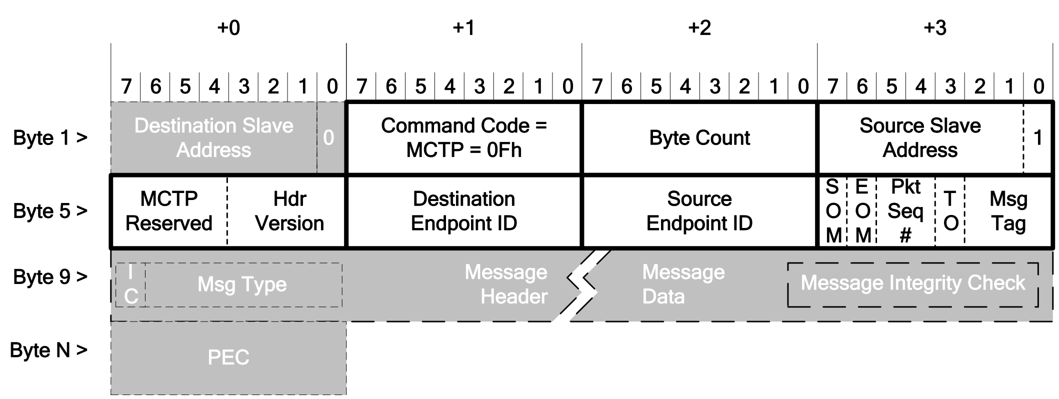


**Figure 3 – Allowed byte range for first NACK'd byte**

## 6.15 Fairness arbitration requirements for MCTP bridges

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

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

## 6.16 Fairness arbitration requirements for non-bridge endpoints

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

- The endpoint's port shall support FAIR_IDLE detection and implement the corresponding fairness policy.
- Upon device power up or initialization, the endpoint does not need to detect a FAIR_IDLE condition before first attempting to access the bus.
- The endpoint cannot make the acceptance of a receive packet conditional on its ability to transmit a packet (that is, a simple endpoint shall not NACK incoming packets because it is trying to send an outgoing packet).

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

  If a device is severely limited in buffer space and cannot allocate separate space for both transmit and received data, options are for the endpoint to allow its buffer to be over-written by the receive packet, or in some cases the endpoint may elect to do a dummy receive of the incoming packet (that is, ACK the incoming bytes, but internally drop them as they are coming in.)
- Higher layer protocols shall be used to handle the case when the endpoint is targeted by more messages than it can process. The buffering requirement for the MCTP Control Protocol messages is defined in the MCTP Base Specification. Buffering requirements for other message types are defined in the respective specifications for the message type.
- An endpoint is allowed to NACK a packet if it is temporarily unable to accept it (for example, because of an *input* buffer-full condition). This should typically only occur if the endpoint is the target of packets from more than one source endpoint.
- There is no direct limit of how long a non-bridge endpoint is allowed to successively NACK incoming packets. However, there are limits on how many packet-level retries a transmitter will attempt before it drops the transmitted packet, as well as message type-specific limits on how long and how many times a given message will be retried.
- If an endpoint has an output transmit packet and it NACKs an input receive packet from lack of input buffer space, it shall be able to issue a START condition to transmit the output packet within $T_{START\_WINDOW}$ after the bus becomes free, unless the endpoint is waiting to detect $T_{IDLE}$.
- An endpoint that receives a NACK while it is performing a Master Write operation is not required to immediately conclude the Master Write operation and drop off the bus. The endpoint may continue the write operation through its conclusion. In either case, the master shall always conclude its transaction with a STOP condition, unless some other device on the bus first produces a START or STOP condition. The latter situation is an erroneous condition on the bus, but bridges shall be able to handle it. Devices shall always recognize START and STOP conditions regardless of the transaction or bit position on which they occur.
- Endpoints that are NACK'd or lose arbitration shall retry transaction for PN1 retries (see Table 8).

## 6.17 Fairness arbitration timing

Figure 4, Table 5, and Table 6 present the specifications for the timing intervals for fairness arbitration on SMBus and $I^2C$ relative to the data (SDA) signal. Refer to SMBus and $I^2C$ for the additional specifications on the relationship between SCL and SDA for STOP, bus idle, and START conditions.

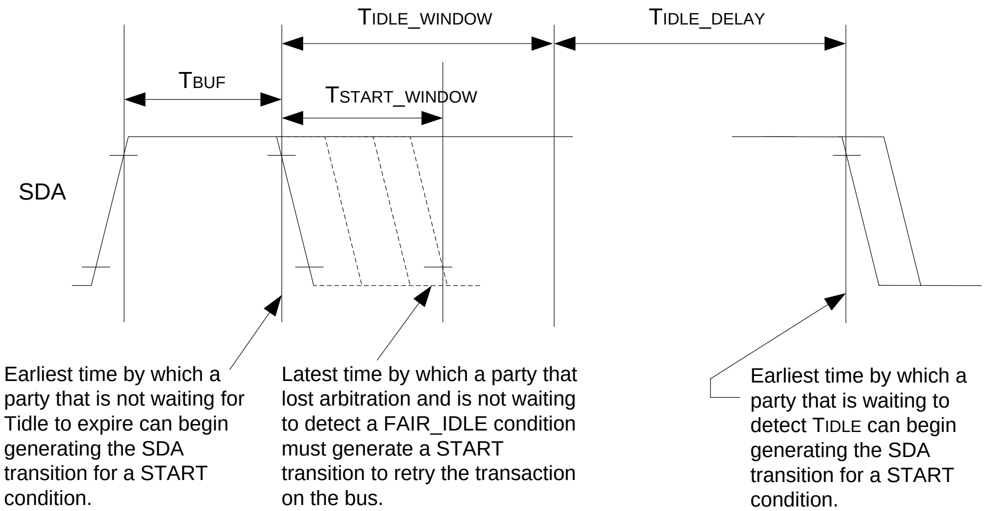


**Figure 4 – Fairness arbitration timing measurement for SMBus and $I^2C$**

**Table 5 – Fairness arbitration timing values for 100 kHz SMBus/$I^2C$**

| 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.) |

**Table 6 – Fairness arbitration timing values for 400 kHz $I^2C$**

| Symbol | Min | Max | Unit | Notes |
|---|---|---|---|---|
| $T_{BUF}$ | 1.3 | – | µs | Per $I^2C$ 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.) |

**Table 7 – Fairness arbitration timing values for 1MHz $I^2C$**

| Symbol | Min | Max | Unit | Notes |
|---|---|---|---|---|
| $T_{BUF}$ | 0.5 | – | µs | Per $I^2C$ 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.) |

## 6.18 MCTP packet timing requirements

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

**Table 8 – Timing specifications for MCTP packets on SMBus/I$^2$C**

| 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 µs 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 µs[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 µs per byte[1] | MCTP devices that are receiving MCTP packets shall not clock stretch the overall packet more than the specified amount.<br>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.<br>If a START condition is detected, the time-out interval restarts.<br>If a STOP condition is detected, the controller can assuming the bus is free following the TBUF interval specified in SMBus.<br>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 banged" $I^2C$ 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. | | | |

## 6.19 MCTP control message timing requirements

The following timing specifications are specific to MCTP control messages on SMBus/$I^2C$. Timing is specified for a "point-to-point" connection. That is, timing is specified as if there were only two endpoints in direct communication on the bus. In particular, the timing specifications assume that there is no clock stretching occurs due to other parties on the bus.

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

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

**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.<br><br>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.<br><br>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. | | | | |

## 6.20 "Stuck 0" condition handling

A possible condition exists in SMBus and $I^2C$ where a slave device that is being read or is driving ACK could be left driving a low (0) level onto the data line (SDA) of the bus. The bus uses a "wire OR'd" approach, where the low (0) level takes precedence over the high (1) level. Therefore, if one party drives a low (0) level onto the bus, the bus cannot go to a high (1) level until the low level is released.

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

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

Effectively, what happens is that the device that was being accessed does not recognize that the transaction has been terminated or that a clock was missed. The device continues to drive the 0 onto the bus because it is waiting to get more clocks from the master to conclude the transaction, but those clocks will never come unless some bus master takes steps to generate them.

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

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

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

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

## 6.21 MCTP over SMBus/I$^2$C protocol anti-aliasing

MCTP over SMBus has been designed to allow one endpoint to support multiple protocols, such as ASF, IPMI, or legacy device-specific protocols with a single slave address. The following clauses describe provisions that can help support implement MCTP over SMBus in devices that also need to support other SMBus or I$^2$C protocols.

### 6.21.1 IPMI

The IPMI protocols for SMBus (IPMI over SMBus) and I$^2$C (Intelligent Platform Management Bus, IPMB) use the fourth byte of the transaction as a Source Slave Address byte, as does MCTP over SMBus. However, the IPMI protocols require the least significant bit of that byte to be `0b`, whereas MCTP over SMBus requires the bit to be `1b`. Thus, a device that needs to differentiate between MCTP over SMBus and the IPMI SMBus/I$^2$C protocols can do so using that bit.

### 6.21.2 ASF

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

### 6.21.3 Integrating MCTP with legacy SMBus functions

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

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

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

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

## 6.22 Well-known and reserved slave addresses

For bus segments that support ARP-able devices, Table 10 summarizes addresses that are generally reserved by SMBus or I$^2$C and should either be avoided by devices. In addition, some are reserved (not to be used as a general device slave address) because those addresses are related to functions that are used by MCTP.

**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** |

**NOTE 1.** This address is used as a broadcast address in IPMI and I$^2$C. It should be avoided if IPMI management controllers may be used on the same bus segment. In I$^2$C, 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 I$^2$C address assignment or slave address broadcast purposes.

**NOTE 2.** The I$^2$C START byte is a pre-amble to the slave address that is intended to provide time for firmware driven I$^2$C interfaces to shift into polling of I$^2$C clock and data lines after a START condition has been detected. This is a very rarely used option in I$^2$C. MCTP does not support the use of the START byte with MCTP or non-MCTP devices.

**NOTE 3.** CBUS is an ancestor of I$^2$C, developed by Philips Semiconductor. It uses a data and clock signal similar to I$^2$C, but with a third signal (SEN) used to generate the START and STOP conditions on the bus. This address range was reserved by the I$^2$C specification to enable a degree of backward compatibility with CBUS devices sharing the I$^2$C SCL and SDA lines as the CBUS clock and data lines, respectively. While listed as a reserved address in the I$^2$C specification, few SMBus/I$^2$C implementations using MCTP will have any need to also support CBUS devices.

**NOTE 4.** MCTP is not defined to support I$^2$C high-speed mode operation.

**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.

**NOTE 6.** Used in conjunction with the R/W# bit position to deliver the most-significant three address bits for I$^2$C 10-bit addressing. MCTP protocols and data structures do not support 10-bit addressing on SMBus or I$^2$C segments. MCTP only supports 7-bit addresses for MCTP and non-MCTP devices on a bus segment.

**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.

## 6.23 Fixed address allocation

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

From a practical point-of-view, SMBus and I$^2$C do not have an effective central registry or other mechanism for avoiding conflicts in the assignment and use of slave addresses among device vendors. While there are potential registries of device slave address usage for SMBus (under the System Management Interface Forum) and I$^2$C (from Philips Semiconductor), these have not generally been used by device vendors and there is no group or standard that works to enforce conformance to those registries.

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

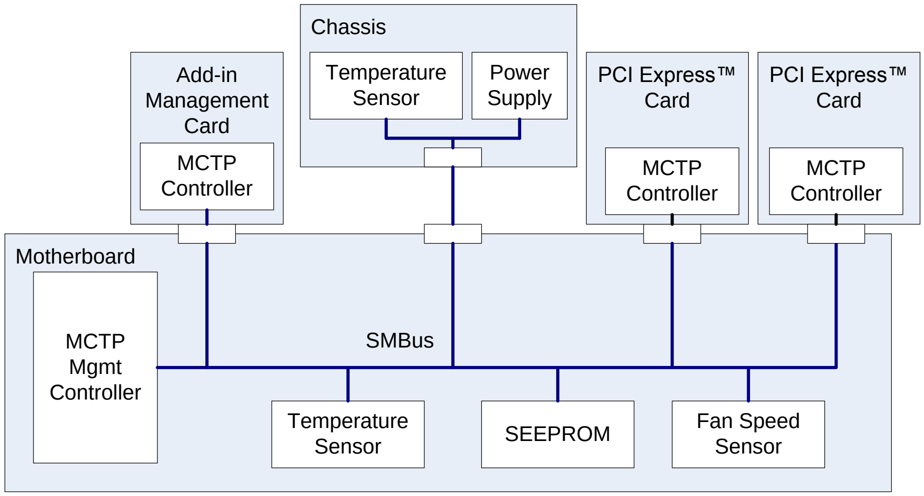


**Figure 5 – Example system configuration**

## 6.24 Recommended address range allocation for computer systems

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

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

- **B** **Board**: An area reserved for board set manufacturer use (where *board set* would be the motherboard and other boards that accompany that motherboard from the same vendor).
- **C** **Chassis**: An area reserved for use by vendors that make chassis in which a third-party board set would be used.
- **A** **Add-in**: For third-party add-in devices (for example, modules or add-in cards that used fixed addresses and would be used in combination with a motherboard or chassis where there is a connection to a SMBus segment implementing MCTP).

  NOTE PCI/PCIe add-in cards that use standard PCI connectors are required to support SMBus ARP and fixed addresses are not used.
- **R** **Reserved** for IPMI, $I^2C$, SMBus, or MCTP uses. Includes the *avoid* addresses from Table 10.

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

789 **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 | SEEPROM | | | |
| | 0xA6 | SEEPROM | | | |
| | 0xA8 | SEEPROM | | | |
| | 0xAA | SEEPROM | | | |
| | 0xC0 | | | | |
| | 0xC2 | SMBus Device Default address | | | |
| | 0xC4-0xCE | | | | |
| | 0xE0-0xE6 | I2C Bus Switch | | | |

# ANNEX A
# (informative)

# Notation

## Notations

Examples of notations used in this document are as follows:

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

# ANNEX B (informative)

# Change log

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