Frequently Asked Questions (FAQ) for MIPI I3C HCI^{SM} v1.2

FAQ Version 1.0
11 August 2023

MIPI Board Approved 20 September 2023
Public Release Edition

This is an informative document, not a MIPI Specification.

Various rights and obligations that apply solely to MIPI Specifications (as defined in the MIPI Membership Agreement and MIPI Bylaws) including, but not limited to, patent license rights and obligations, do not apply to this document.

Further technical changes to this document are expected as work continues in the MIPI Software Working Group.

Copyright © 2023 MIPI Alliance, Inc.

NOTICE OF DISCLAIMER

The material contained herein is provided on an "AS IS" basis. To the maximum extent permitted by applicable law, this material is provided AS IS AND WITH ALL FAULTS, and the authors and developers of this material and MIPI Alliance Inc. ("MIPI") hereby disclaim all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness of responses, of results, of workmanlike effort, of lack of viruses, and of lack of negligence. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THIS MATERIAL.

IN NO EVENT WILL ANY AUTHOR OR DEVELOPER OF THIS MATERIAL OR MIPI BE LIABLE TO ANY OTHER PARTY FOR THE COST OF PROCURING SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA, OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, ARISING IN ANY WAY OUT OF THIS OR ANY OTHER AGREEMENT RELATING TO THIS MATERIAL, WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

The material contained herein is not a license, either expressly or impliedly, to any IPR owned or controlled by any of the authors or developers of this material or MIPI. Any license to use this material is granted separately from this document. This material is protected by copyright laws, and may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of MIPI Alliance. MIPI, MIPI Alliance and the dotted rainbow arch and all related trademarks, service marks, tradenames, and other intellectual property are the exclusive property of MIPI Alliance Inc. and cannot be used without its express prior written permission. The use or implementation of this material may involve or require the use of intellectual property rights ("IPR") including (but not limited to) patents, patent applications, or copyrights owned by one or more parties, whether or not members of MIPI. MIPI does not make any search or investigation for IPR, nor does MIPI require or request the disclosure of any IPR or claims of IPR as respects the contents of this material or otherwise.

Without limiting the generality of the disclaimers stated above, users of this material are further notified that MIPI: (a) does not evaluate, test or verify the accuracy, soundness or credibility of the contents of this material; (b) does not monitor or enforce compliance with the contents of this material; and (c) does not certify, test, or in any manner investigate products or services or any claims of compliance with MIPI specifications or related material.

Questions pertaining to this material, or the terms or conditions of its provision, should be addressed to:

MIPI Alliance, Inc.
c/o IEEE-ISTO
445 Hoes Lane, Piscataway New Jersey 08854, United States
Attn: Executive Director

Special Note Concerning MIPI I3C and MIPI I3C Basic

As described in the I3C Basic specification, certain parties have agreed to grant additional rights to I3C Basic implementers, beyond those rights granted under the MIPI Membership Agreement or MIPI Bylaws. Contribution to or other participation in the development of this FAQ document does not create any implication that a party has agreed to grant any additional rights in connection with I3C Basic. Consistent with the statements above, nothing in or about this FAQ document alters any party's rights or obligations associated with I3C or I3C Basic.

Contents

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

Release History

Date	Version	Description
20-Sep-2023	FAQ v1.0	Initial Board Approved release.

This page intentionally left blank.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

1 Introduction

This FAQ was developed to introduce the MIPI I3C HCI Specification to developers and users. The MIPI Software WG compiled these Frequently Asked Questions (FAQs) to assist Member implementation activity and to clarify and resolve some ambiguous Sections of the Specification.

Since the I3C HCI Specification describes the behaviors and requirements of an I3C Controller on an I3C Bus, the reader should also be aware of the current versions of the I3C Specifications:

Note:

> ***The most current version of the full I3C Specification is I3C v1.1.1.*** *FAQ entries reflect all technical and editorial updates (i.e., the changes from I3C v1.0 or v1.1 to v1.1.1).*
>
> ***The most current version of the I3C Basic Specification is I3C Basic v1.1.1.*** *FAQ entries reflect all technical and editorial updates (i.e., the changes from I3C Basic v1.0 to v1.1.1). Note that* ***there is no I3C Basic v1.0****; this version number was skipped.*

Throughout this FAQ, unless otherwise noted, the terms “MIPI I3C” and ‘I3C’ refer to both I3C ***[MIPI01][MIPI08][MIPI10]*** and I3C Basic ***[MIPI07][MIPI11]***, unless otherwise specified.

Note:

> *Unless otherwise noted, a reference to a numbered section of the I3C Specification applies to both the I3C Specification and the I3C Basic Specification (i.e., section numbers are aligned across the two Specifications).*
>
> *None of the answers in this FAQ are intended to overwrite or overrule the information in the I3C Specification* ***[MIPI01][MIPI08][MIPI10]*** *or the I3C Basic Specification* ***[MIPI07][MIPI11]****.*

The questions are organized into Sections by topic and grouped into two higher-level categories: general questions about I3C HCI and the ecosystem and detailed technical questions about the material in the I3C HCI Specification.

Section	Title	Focus
2.1	**Introduction to MIPI I3C HCI**	I've heard about I3C HCI. Where can I read more about it?
2.2	**I3C HCI Versions and Releases**	What versions of I3C HCI have been released, and what has changed?
2.3	**Up and Coming**	Questions related to the next revision (and/or Errata) of the I3C HCI Specification.
2.4	**Naming and Terminology**	Questions related to recent changes to I3C terminology and how these affect the I3C HCI Specification.
2.5	**Ecosystem**	Questions related to design kits, IP, test, and other parts of the enablement ecosystem for I3C HCI.
2.6	**Interacting with Targets**	Questions about Host Controller usage with I3C and I^{2}C Target Devices
2.7	**Interoperability Workshops**	Questions asked by early Interoperability Workshop participants.
2.8	**Conformance Testing**	Questions related to testing device conformance to the I3C Specification.
2.9	**Support for Optional I3C HCI Features**	Questions about minimum requirements and which optional features should be supported by implementers.
2.10	**Implementation: As a Host Controller Implementer**	Questions relating to Host Controller implementation and design choices.
2.11	**Implementation: As a Software Developer**	Questions asked by software developers.
2.12	**Operation in PIO Mode**	Specific questions about changes to PIO Mode operational details.
2.13	**Backwards Compatibility with I^{2}C**	How can I3C Host Controllers interoperate with Legacy I^{2}C Target Devices?
2.14	**Dynamic Address Assignment and Group Address Assignment**	Questions about Address Assignment and how the software uses the Host Controller to manage I3C Target Device addresses.
2.15	**In-Band Interrupt and Hot-Join**	Questions about how the Host Controller responds to I3C Target-initiated requests on the I3C Bus.
2.16	**Common Command Codes (CCCs)**	Questions about how the Host Controller processes Transfer Commands to send CCCs to I3C Targets.
2.17	**Resets and Error Handling**	How do the Host Controller and Driver handle error scenarios and Target reset operations?

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

2 Frequently Asked Questions

This FAQ is organized by general topic area and I3C HCI features and capabilities:

This page intentionally left blank.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

General Questions

2.1 Introduction to MIPI I3C HCI

Q1.1 What is MIPI I3C HCI?

MIPI I3C HCI is a Host Controller Interface Specification that defines the features and behaviors of an I3C Bus Controller using a register-based interface.

Q1.2 What is MIPI I3C TCRI?

MIPI I3C TCRI defines the Transfer Command Response Interface that is useful for various applications where an I3C Controller processes Transfer Commands and generates Transfer Responses. I3C HCI defines the behavior of a Host Controller that relies on the standard Transfer Commands and Transfer Responses defined in the I3C TCRI Specification.

Q1.3 What are MIPI I3C and I3C Basic?

MIPI I3C is a serial communication interface Specification that improves upon the features, performance, and power use of I²C while maintaining backwards compatibility for most Devices.

MIPI I3C Basic is technically identical to I3C except with a reduced feature set and different licensing.

I3C HCI provides a Host Controller interface for I3C Bus Controller logic that implements support for I3C or I3C Basic, although certain features and capabilities will not be available with I3C Basic.

Q1.4 Why was I3C HCI introduced?

I3C HCI's main purposes are to:

1. Standardize the host interface to I3C Bus Controller logic.
2. Define the behavioral expectations for an I3C Bus Controller that accepts Transfer Commands and generates Transfer Responses.
3. Define a framework for implementers to implement optional extended capabilities that provide additional features specific to various applications.

Q1.5 What are the main features of I3C HCI?

MIPI I3C carries the advantages of I²C in simplicity, low pin count, easy board design, and multi-drop (vs. point-to-point) but provides the higher data rates, simpler pads, and lower power of SPI. I3C adds higher throughput for a given frequency, In-Band Interrupts (from Target to Controller), Dynamic Addressing, advanced power management, and Hot-Join.

Q1.6 For which applications or use cases is I3C HCI intended?

I3C HCI is intended for a wide range of applications, with implementations that can be in micro-controllers (MCs), application processing units (APUs), or PC peripherals attached to a CPU (either integrated or as expansion Devices). It can scale up to more advanced applications that require additional performance and autonomy from the Host APU/CPU if the Host system memory can be accessed directly by the Host Controller.

Q1.7 How can the I3C HCI Specification and related MIPI Specifications be obtained?

- **MIPI I3C HCI Specification**: MIPI Alliance members have access and rights to the ***I3C HCI Specification*** through their MIPI membership and the member website. The latest adopted version is I3C HCI v1.2 ***[MIPI12]***.
  - **Non-members may download** a public version of the I3C HCI Specification by visiting the I3C HCI page on the MIPI Alliance website: https://www.mipi.org/specifications/i3c-hci.
- **MIPI I3C TCRI Specification:** MIPI Alliance members have access and rights to the ***I3C TCRI Specification*** through their MIPI membership and the member website. The latest adopted version is I3C TCRI v1.0 ***[MIPI06]***.

    - **Non-members may download** a public version of the I3C TCRI Specification by visiting the I3C TCRI page on the MIPI Alliance website: https://www.mipi.org/specifications/i3c-tcri.
- **MIPI I3C Specification:** MIPI Alliance members have access and rights to the ***I3C Specification*** through their MIPI membership and the member website. The latest adopted version is I3C v1.1.1 ***[MIPI10]***.
- **MIPI I3C Basic Specification:** MIPI Alliance made the ***I3C Basic v1.0 Specification [MIPI07]*** publicly available for download in December 2018. The latest adopted version is I3C Basic v1.1.1 ***[MIPI11]***. MIPI Alliance members have access and rights to the I3C Basic Specification through their MIPI membership and the member website.
    - **Non-members may download** a copyright-only version of the I3C Basic Specification by visiting the I3C Basic page on the MIPI Alliance website: https://www.mipi.org/specifications/i3c-sensor-specification.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

2.2 I3C HCI Versions and Releases

Q2.1 What is new in I3C HCI v1.1?

I3C HCI v1.1 is an advancement of the I3C HCI Specification that includes clarifications and corrections from the Errata of I3C HCI v1.0 and new features and capabilities provided by versions 1.1 and later of the I3C Specification and the I3C Basic Specification. Some of these new features and capabilities are required.

The new features in I3C HCI v1.1 include:

- Transfer Command support for Broadcast CCCs and Directed CCCs with Defining Bytes
- Command flows for the I3C Target Reset Pattern, including the **RSTACT** CCC for configurable Target Reset actions
- Support for additional Dynamic Address Assignment methods and Group Addresses
- Support for various I3C Bus recovery procedures
- Numerous improvements to the standard Host Controller interface for configuration, command, and control

I3C HCI v1.1 also includes improved definitions of normative content, including an updated Theory of Operation section that more fully describes PIO Mode and how the Host interacts with the Host Controller in PIO Mode and improved definitions of error handling for I3C transfers. This version of I3C HCI underwent significant improvements to clarify the Host Controller's implementation requirements and behavioral expectations.

Note:

I3C HCI v1.1 has been superseded by I3C HCI v1.2. MIPI Alliance recommends that implementers use I3C HCI v1.2 as the basis for new implementations.

Q2.2 What is new in I3C HCI v1.2?

I3C HCI v1.2 is an advancement of the I3C HCI Specification that includes clarifications to facilitate a richer interface and new features that make I3C HCI even more attractive to a broader set of use cases and applications. Many of these new features are optional-normative and can be selectively supported by the implementer as determined by the specific application, the use case for the I3C Bus, and the intended size and complexity of the Host Controller logic.

The new features in I3C HCI v1.2 include:

- Scheduled Command Processing
- Standby Controller mode with I3C Secondary Controller logic
- Dead Bus Recovery mechanism
- Improvements to In-Band Interrupt (IBI) status reporting for different I3C Bus events
- I3C Target credit counting mechanism for managing IBI Requests based on Host readiness
- Support for Host-initiated abort of IBI data payloads
- Improvements to the PIO Mode capabilities for Host Controller implementations that choose to support PIO Mode
- Support for HDR Mode Data Transfer and Early Termination capabilities for optional HDR Modes

I3C HCI v1.2 also includes fixes, corrections, and clarifications for all issues addressed by the Errata 01 for I3C HCI v1.1.

Starting with I3C HCI v1.2, the definitions of standard Transfer Commands and Transfer Responses were moved to the I3C TCRI Specification to promote reuse of these definitions for other applications of I3C. I3C HCI v1.2 includes references to the I3C TCRI Specification for the I3C Bus Controller logic behaviors and Transfer Command and Transfer Response processing. I3C HCI v1.2 also focuses on the Host Controller's specific requirements and features that pertain to its interface with the Host and other optional advanced capabilities, including those listed above, that are independent of Transfer Command/Response processing.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

Q2.3 What are the required features in I3C HCI v1.2 vs. I3C HCI v1.1 / v1.0?

Almost all of the new features of I3C HCI v1.2 are optional. While I3C HCI v1.1 defines fewer new features, most of them are required.

Specifically, it should be noted that:

- The Transfer Command and Transfer Response formats are now defined in I3C TCRI v1.0.
  - Additionally, the Transfer Command and Transfer Response formats in I3C HCI v1.1 were updated and extended (compared to I3C HCI v1.0) with improved support for Broadcast CCCs and Direct CCCs, including Defining Bytes. Support for CCCs with Defining Bytes is now mandatory for I3C HCI v1.1 and later.
- If the Host Controller supports both PIO Mode and Direct Memory Access (DMA) Mode, then changing the operating mode (i.e., switching active operation between PIO Mode and DMA Mode) is now explicitly controlled by writing to an HCI register field and is no longer controlled by the number of currently enabled Ring Bundles (as defined in I3C HCI v1.0).
- I3C HCI v1.1 and later define new requirements for stalled Transfer Command sequences (i.e., successive Transfer Commands that are separated by **TOC**=0) for situations where the Host fails to enqueue all such Transfer Commands before the Host Controller processes the last enqueued Transfer Command that ends with **TOC**=0 (i.e., when the Host does not intend to drive a STOP condition on the I3C Bus).
- I3C HCI v1.1 and later also define new Internal Control Command subtypes, including a special command that initiates an I3C Target Reset pattern. This can be used on its own or in a special sequence that uses **RSTACT** CCCs (either Broadcast or Direct) to address one or more I3C Targets and configure a specific Target Reset action.
- I3C HCI v1.2 uses the I3C v1.1.1 and I3C Basic v1.1.1 Specifications as normative references for behavior for an I3C Bus Controller. A Host Controller that complies with I3C HCI v1.2 must also comply with all normative requirements for an I3C Bus Controller as defined in these specifications.

MIPI recommends that implementers study the updated I3C HCI v1.2 Specification and other related MIPI I3C Specifications carefully to understand the requirements and any optional features they might use for their applications.

Q2.4 Are there any I3C v1.1/v1.1.1 features that are not supported in I3C HCI v1.2?

Yes. I3C HCI v1.2 does not yet support the following features:

- HDR-BT (Bulk Transfer) Mode
- Multi-Lane transfers (in any I3C Mode)
- CCCs in HDR Modes
- Specific types of 'combo' transfers that are necessary for more advanced use cases, such as Device-to-Device Tunneling

Q2.5 Does a Host Controller that complies with I3C HCI v1.2 still interoperate with I3C Target Devices that comply with I3C v1.0 or I3C Basic v1.0?

Yes. A Host Controller can still address I3C Target Devices that comply with the earlier (and superseded) versions of the I3C and I3C Basic Specifications, although such I3C Target Devices will not support any of the newer features defined in the current I3C and I3C Basic Specifications.

When driving Transfer Commands to such Target Devices, the Host should not attempt to send such I3C Target Devices any CCCs that are not defined in such I3C and I3C Basic Specifications, as such Target Devices will not support them.

Q2.6 Can a Host Controller that supports the full I3C Specification interoperate with I3C Target Devices that only support the I3C Basic Specification?

Yes. The I3C Basic Specification is, in general, a subset of the full I3C Specification, so the Host Controller can still drive Transfer Commands in SDR Mode and any optional HDR Modes that such an I3C Target Device might optionally support (for I3C Basic v1.1.1 only). The Host Controller can use the **GETCAPS** CCC to determine which version of the I3C Specification or I3C Basic Specification and which optional capabilities are supported.

The Host Controller should not try to drive Transfer Commands in HDR Modes to I3C Target Devices that do not support these optional HDR Modes (or, for I3C Basic v1.0, any HDR Modes). However, this does not prevent the Host Controller from driving Transfer Commands in HDR Modes to other I3C Target Devices that support the full I3C Specification and one or more optional HDR Modes.

Note:

*I3C Target Devices that only comply with I3C Basic v1.0 will not support the **GETCAPS** CCC. This CCC (formerly defined as the **GETHDRCAP** CCC in I3C v1.0) was not defined in I3C Basic v1.0. In this case, such an I3C Target Device will NACK the **GETCAPS** CCC. The Host should interpret this NACK as an indication that no HDR Modes are supported by this I3C Target Device, and the I3C Target Device only supports I3C Basic v1.0.*

Q2.7 Can a Host Controller that only supports the I3C Basic Specification interoperate with I3C Target Devices that support the full I3C Specification?

Yes, but such a Host Controller cannot use any optional transfer modes or capabilities not included in the I3C Basic Specification.

2.3 Up and Coming

Q3.1 Are there any impending fixes or Errata for I3C HCI v1.2 that should be applied now?

MIPI currently has no impending fixes or Errata for I3C HCI v1.2.

Note that all fixes or Errata for previous versions of I3C HCI have been applied to I3C HCI v1.2.

Based on learning from earlier implementations, I3C Interoperability Workshops, queries from adopters, and reviews by the Software WG and the I3C WG, this FAQ includes clarifications, improvements that can be implemented by the I3C Host Controller, and other guidance for implementers.

Q3.2 Are any revisions to I3C HCI v1.2 expected?

Not currently. However, the MIPI Software WG meets regularly and is considering proposals to revise and extend the Host Controller interface and its capabilities. As part of I3C HCI Specification maintenance, the MIPI Software WG seeks to improve the I3C HCI Specification with clarifications and additional explanation. Please direct any comments or suggestions to MIPI Alliance.

Q3.3 What new features, if any, are coming to I3C HCI?

There are no new approved features. However, the MIPI Software WG is considering the following:

- Support for HDR-BT (Bulk Transfer) Mode
- Support for Multi-Lane transfers
- Support for Device-to-Device Transfers
- Support for CCCs in HDR Modes
- Improvements and optimizations to both PIO Mode and DMA Mode
- Additional support for Secondary Controller capabilities

Implementers who are interested in these new features, and others interested in a particular use case should direct any comments or suggestions to MIPI Alliance.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

2.4 Naming and Terminology

Q4.1 What is an I3C Controller Device, and why was the I3C Master Device renamed?

As part of a terminology replacement effort across MIPI Alliance, starting with I3C v1.1.1, I3C Basic v1.1.1, and I3C HCI v1.1, the terms Master and Slave have been deprecated. An I3C v1.0/v1.1 Master Device is now called a Controller. There is no change to the technical definition of such an I3C Device or its role on an I3C Bus. The term Controller is a better, more accurate description of the Device's role on an I3C Bus.

Due to this change, the names of various CCCs and other related terms have also changed, starting with v1.1.1, including:

Deprecated Prior Term *I3C and I3C Basic before v1.1.1*	**Replacement Term** *I3C and I3C Basic v1.1.1 and Later*
Master	Controller or Bus Controller
Current Master	Active Controller
Secondary Master	Secondary Controller
Main Master	Primary Controller
New Master (relating to Handoff)	New Active Controller
Master-Capable Device	Controller-Capable Device
Mastership, Mastering the Bus, etc.	Controller Role, Control of the Bus, etc.
Mastership Request	Controller Role Request
GETACCMST CCC	**GETACCCR** CCC
Error Types M0 through M3	Error Types CE0 through CE3
Deprecated Prior Term *I3C HCI before v1.1*	**Replacement Term** *I3C HCI v1.1 and Later*
Register **MASTER_DEVICE_ADDR**	Register **CONTROLLER_DEVICE_ADDR**
Field **NON_CURRENT_MASTER_CAP** in register **HC_CAPABILITIES**	Field **STANDBY_CR_CAP**
Field **M2_ERROR_COUNT** in register **MX_ERROR_COUNTERS**	Field **CE2_ERROR_COUNT** (the register name was not changed)

Note:

*In I3C HCI v1.1 and later, the general term Controller is used to mean "Bus Controller" for sections of the implementation where it would directly interface with the I3C Bus, and "Host Controller" for sections of the implementation where it would be accessed by the Host. See also **Q4.3**. This distinction does not apply to the I3C and I3C Basic Specifications: For the I3C HCI Specification, the term "Bus Controller" generally applies to the behaviors defined in the I3C and I3C Basic Specifications.*

See also ***Q4.2***.

Q4.2 What is an I3C Target Device, and why was the I3C Slave Device renamed?

As part of a terminology replacement effort across MIPI Alliance, starting with I3C v1.1.1, I3C Basic v1.1.1, and I3C HCI v1.1, the terms Master and Slave have been deprecated. An I3C v1.0/v1.1 Slave Device is now called a Target. There is no change to the technical definition of such an I3C Device or its role on an I3C Bus.

The term Target is a better, more accurate description of the Device's role on an I3C Bus. In particular, the previous term did not describe I3C transfers, which are typically sent by the I3C Controller to individual I3C Devices or all I3C Devices. Target better describes how individual transfers are addressed to specific I3C Devices.

Due to this change, the names of various CCCs and other related terms have also changed, starting with v1.1.1, including:

Deprecated Prior Term *I3C and I3C Basic before v1.1.1*	**Replacement Term** *I3C and I3C Basic v1.1.1 and Later*
Slave	Target
Slave Reset Pattern	Target Reset Pattern
DEFSLVS CCC	**DEFTGTS** CCC
Error Types S0 through S6	Error Types TE0 through TE6

See also ***Q4.1***.

Q4.3 What is the difference between a Bus Controller and a Host Controller?

The I3C HCI Specification defines an I3C Bus Controller as a Device that complies with the I3C Specification or the I3C Basic Specification and holds the role of Controller (i.e., it is currently the Active Controller of the I3C Bus). In the I3C HCI Specification, Bus Controller logic refers to the part of an I3C HCI implementation that interfaces directly with the I3C Bus and directly drives Transfer Commands to one or more I3C Targets while it is the Active Controller. In most respects, Bus Controller logic is normatively defined within the I3C TCRI Specification, specifically how this logic processes and executes Transfer Commands and generates Transfer Responses. If the Host Controller is in Standby Controller mode because it is not currently the Active Controller (i.e., it is currently acting as a Secondary Controller on the I3C Bus), the Bus Controller logic is typically idle and cannot drive Transfer Commands.

In contrast, Host Controller describes the overall entity that contains one or more instances of Bus Controller logic (i.e., one for each connected I3C Bus) as well as the part of the implementation facing the Host. This term can refer to the specific interface elements and entities that are accessed by the Host (i.e., the Driver or other higher-level software) and used to convey Transfer Commands, Transfer Responses, and associated data bytes for such activities on the I3C Bus. This includes the configuration registers, the specific details of the operating mode (PIO Mode and/or DMA Mode), and any higher-level features and capabilities that are not directly related to lower-level I3C transfer activity.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

2.5 Ecosystem

Q5.1 Who is defining the I3C HCI Specification and other related MIPI Specifications?

The I3C HCI Specification and the I3C TCRI Specification are defined by the MIPI Alliance Software Working Group, which was formed in 2014.

The I3C Specification is defined by the MIPI Alliance I3C Working Group (originally named the Sensor Working Group), which was formed in 2013. The I3C Basic Specification is defined by the MIPI Alliance I3C Basic Ad-Hoc Working Group, which was formed in 2018.

Q5.2 Is anyone currently using I3C HCI?

Yes. Several companies have released products or IP blocks that feature integrated I3C Controller support that complies with the I3C HCI Specification. Other companies offer associated verification software for testing the behavior of I3C Host Controllers in various integrated circuit designs. Some companies also offer protocol analyzers and verification hardware to analyze I3C Bus traffic for testing and development.

Since this document cannot provide a comprehensive list of such products, those interested in learning more about products that support or enable I3C or the I3C HCI should contact MIPI Alliance.

Q5.3 What is the I3C HCI IP core availability in the market?

Some vendors now offer I3C HCI-compliant Controller IP cores for integration into ASIC Devices and FPGAs.

2.6 Interacting with Targets

Q6.1 Will all I3C Targets be compatible with all CCCs?

No. Some CCCs are mandatory, whereas others are optional or conditionally supported. A specific I3C Target will either support each optional or conditionally supported CCC or not, depending upon that Target's capabilities. Refer to the I3C FAQ for more information.

Q6.2 Can the Host send any CCCs using a Transfer Command?

While most CCCs can be sent in a Transfer Command, there are some exceptions. The Host Controller does not block the use of any CCCs in Transfer Commands except for the following:

- The **ENTDAA** and **SETDASA** CCCs, which cannot be sent directly in a Transfer Command.
  - The Host should use the Address Assignment Command instead of a Transfer Command, because these CCCs have special requirements and require additional behaviors that a Transfer Command does not support.
- The **ENTHDR0**-**ENTHDR7** CCCs, which cannot be sent directly in a Transfer Command.
  - Since HDR Modes use fundamentally different signaling protocols on the I3C Bus, these CCCs must not be sent directly, as they imply that the I3C Bus will use a different signaling protocol per the indicated HDR Mode for subsequent activity.
  - To drive transfers in HDR Modes, the Host must specify the I3C Mode in the **MODE** field of the Transfer Command, and the Bus Controller logic will automatically drive the appropriate **ENTHDRx** CCC to enter the HDR Mode, if it is supported (see ***Q9.2***).
- The **GETACCCR** CCC, which cannot be sent directly in a Transfer Command.
  - Since this CCC usually initiates the Controller Role handoff procedure from the Bus Controller logic to another Controller-capable device on the I3C Bus, this CCC must not be sent directly, as ACK from the addressed Target device (i.e., a Secondary Controller) will usually imply that the handoff procedure has been initiated.
  - If the Host Controller supports Standby Controller mode with Secondary Controller logic, then the Host may use the special Internal Control Command format that automatically sends the **GETACCCR** CCC to an indicated Secondary Controller and then conditionally start the handoff procedure based on the result.

If the Host Controller receives a Transfer Command with any of these blocked CCCs, then it will generate a Transfer Response with error code 0xA (**NOT_SUPPORTED**) in field **ERR_STATUS** and halt execution of any subsequent Transfer Commands. The Host will need to handle this error and clear any error condition as a result.

All CCCs not listed above can be sent in Transfer Commands, including those that might be used for Vendor/Standards Extensions. It is the Host's responsibility (i.e., the Driver or software) to determine which CCCs are supported or not supported by each I3C Target. In most cases, the Host can attempt to drive Transfer Commands that are CCCs to any I3C Target (either Broadcast or Direct), and the Target will ignore any Broadcast CCCs that it does not support and NACK any Direct CCCs that it does not support, as well as any non-supported Defining Bytes with Direct CCCs.

Q6.3 Will Legacy I^{2}C Targets respond to I3C commands?

In general, no. The Host Controller should begin all I3C transactions with the Broadcast Address (7'h7E), although this can be controlled by field **IBA_INCLUDE** in register **HC_CONTROL** (see ***[MIPI12] Section 7.4.2***) and the special Internal Control Command type that enables or disables automatic transmission of the Broadcast Address after a START (see ***[MIPI12] Section 8.4.2.2***).

If the automatic Broadcast Address transmission is enabled, as noted above, then the Host Controller will send START followed by the Broadcast Address for all transfers, including Private Writes and Private Reads to I3C Targets. (See ***Q10.2***.)

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

Per the I3C Application Note ***[MIPI05]*** and the I^{2}C specification, Legacy I^{2}C Targets will never respond to the Broadcast Address. However, a Legacy I^{2}C Target will respond to a transaction that begins with a START followed by its Static Address. The Host determines whether an addressed Target is an I3C Target or a Legacy I^{2}C Target by setting field **DEVICE** in the Target's DAT entry (see ***[MIPI12] Section 8.1***). This determines whether automatic Broadcast Address transmission is used after a START (if the setting was previously enabled) when initiating a transaction with that Target.

Q6.4 How does a Host Controller handle a NACK of a Transfer Command?

This depends on the value of field **DEV_NACK_RETRY_CNT** in the DAT entry for the addressed I3C Target and whether the Transfer Command is a Direct Read CCC.

- For Transfer Commands that are Direct Read CCCs, the Host Controller will always do a single retry, per the mandatory single-retry model (see ***Q16.3***).
- For all other Transfer Commands:
  - If field **DEV_NACK_RETRY_CNT** has a value of 0x0, then the Host Controller will not retry the Transfer Command if it receives a NACK.
  - If field **DEV_NACK_RETRY_CNT** has a non-zero value, then the Host Controller will retry the Transfer Command if it receives a NACK on the first attempt. The Host Controller will retry up to ***N*** attempts (i.e., where ***N*** is the value in field **DEV_NACK_RETRY_CNT**).
    - If the Host Controller receives an ACK on a subsequent attempt, then the Host Controller will deem the Transfer Command to be successful and report the status in the Response Descriptor. No additional retries will be performed if an ACK is received. However, on a subsequent attempt, the Transfer Command might fail for reasons other than a NACK.
    - If the Host Controller receives a NACK on the last attempt, then the Host Controller will deem the Transfer Command to have failed, and it will return a Response Descriptor with a value of 0x5 (**NACK**) in field **ERR_STS**.

2.7 Interoperability Workshops

Q7.1 What is a MIPI I3C Interoperability Workshop?

A MIPI I3C Interoperability Workshop is a MIPI Alliance-sponsored event where different vendors bring their I3C implementations and check their interoperation with other vendors' implementations.

Q7.2 What is the output from a MIPI I3C Interoperability Workshop?

There are three major outputs from a MIPI I3C Interoperability Workshop:

- Participating vendors can get detailed information about how well their I3C implementations interoperate with other vendors' implementations. Vendors can also compare their results.
- MIPI Alliance can generate an overall picture of the industry state-of-the-I3C implementation. For example, they can determine how many vendors have implemented I3C and how many implementations pass or fail when tested with one another.
- The MIPI I3C Working Group gains a better understanding of any major issues with the I3C Specification. The WG can then leverage that learning by adding to this FAQ, other supporting documents (such as Application Notes, per ***Q11.2***), and possible future revisions of MIPI I3C Specifications.

Q7.3 Are MIPI I3C Interoperability Workshops an ongoing activity?

MIPI Alliance arranges I3C Interoperability Workshop events in response to requests from its membership. They have typically been co-located with regularly scheduled MIPI Member Meetings.

Q7.4 Who can attend or participate in a MIPI I3C Interoperability Workshop?

In general, any MIPI Alliance members who have I3C hardware ready to interoperate can participate.

Q7.5 What HW/SW is typically needed to participate in a MIPI I3C Interoperability Workshop?

The minimum requirements to date are the availability of a board with an I3C Device that can connect to other Devices via the three wires SDA, SCL, and GND. However, this could change in the future. It is also useful to have software (e.g., running on a laptop connected to the board and an I3C Device) to interactively view transmitted and received Bus communications, but this might not be required for Targets.

Currently, there are solutions working at 3.3V and 1.8V.

Q7.6 Are there any I3C Interoperability Workshops planned for I3C v1.1.1 or I3C Basic v1.1.1?

MIPI Alliance has been hosting I3C Interoperability Workshops in conjunction with MIPI Member Meetings, typically two to three times per year. This FAQ was last updated in August 2023.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

2.8 Conformance Testing

Q8.1 What is a MIPI Conformance Test Suite (CTS)?

A MIPI WG develops a CTS document to improve the interoperability of products that implement a given MIPI interface Specification. The CTS defines a set of conformance or interoperability tests, whereby a product can be tested against other implementations of the same Specification.

Q8.2 Is there a MIPI CTS for I3C HCI?

MIPI Alliance has not yet released a CTS for I3C HCI. Interested parties who are MIPI members should reach out to the MIPI Software WG.

This page intentionally left blank.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

Detailed Technical Questions

2.9 Support for Optional I3C HCI Features

Q9.1 Which optional Host Controller features should an implementer support?

While the choice of which optional features to support will depend on the specific use case, the MIPI Software WG recommends that implementers should choose to implement support for the following optional Host Controller features:

- **Combo Transfer Commands** efficiently perform more complicated Transfer Commands (e.g., combined Write-then-Read) with a single Command Descriptor (see the I3C TCRI Specification ***[MIPI06]***, ***Section 7.1.2.3***). Many common I3C Targets have a register access mechanism that uses Write-then-Read transactions to select a register offset in one phase and then read the register data in the next phase with a Repeated START between the phases. Combo Transfer Commands can be useful in such applications.
  - If the Host Controller supports Combo Transfer Commands, then field **COMBO_COMMAND** in register **HC_CAPABILITIES** will have a value of 1'b1.
- **Auto-Command** (see ***[MIPI12] Section 6.11***) allows the Host Controller to automatically respond to IBI Requests with a given Mandatory Data Byte (MDB) value and initiate a read transfer from the same I3C Target that sent the IBI Request. I3C Targets can use this mechanism (i.e., a Pending Read Notification) to signal the I3C Controller and inform them that it has data bytes to be read. The Auto-Command capability makes this more efficient and offloads this responsibility from the Driver, so the Host Controller can automatically fetch the read data from such I3C Targets.
  - If the Host Controller supports Auto-Commands, then field **AUTO_COMMAND** in register **HC_CAPABILITIES** will have a value of 1'b1.
- **Debug-Specific registers** (see ***[MIPI12] Section 7.7.7***) provide detailed information about the internal operating status of the Host Controller, which can be especially helpful for implementations that support PIO Mode. If present, these registers can be used by the Driver to check the status of ongoing transfers and monitor the current levels of the PIO Queues.
  - If the Host Controller supports these Debug-Specific registers, then the Host Controller will have an Extended Capability structure of type Debug-Specific (ID = 0x0C).

Q9.2 Are Host Controllers required to support optional I3C features such as HDR Modes or Timing Control modes?

No, these features are optional. A Host Controller implementer may choose not to implement support for any HDR Modes or Timing Control modes. Additionally, a Host Controller implementer is limited to the I3C features that are included in the I3C Specification used (i.e., either the full I3C Specification or the I3C Basic Specification), which defines I3C Bus Controller functionality.

If the implementer does not wish to support any HDR Modes or chooses only to implement support for some of the available HDR Modes, then the Host Controller implementation will still be fully functional in SDR Mode. In this case, the Host Controller will reject (i.e., refuse to process) any Transfer Commands that indicate a transfer in any non-supported HDR Modes (see the I3C TCRI Specification ***[MIPI06]***, ***Section 7.1.2***) and return a Response Descriptor with value 0xA (**NOT_SUPPORTED**) in field **ERR_STATUS**. Register **HC_CAPABILITIES** (see ***[MIPI12] Section 7.4.4***) will then indicate which, if any, HDR Modes are supported. If no HDR Modes are supported, then fields **HDR_DDR_EN** and **HDR_TS_EN** will both have a value of 1'b0.

If the implementer does not wish to support any Timing Control modes, then the Host Controller implementation will still support IBIs (In-Band Interrupts) from I3C Targets. In this case, writing to field **TS** in the DAT entry will have no effect (i.e., the Host Controller will never timestamp any IBIs).

Q9.3 Are Host Controllers required to support Standby Controller Mode as a Secondary Controller?

No, this capability is not always required. However, the use case for a particular Host Controller implementation will typically determine whether a Host Controller needs to act as a Secondary Controller on the I3C Bus.

If the Bus is relatively simple and the system does not need to have any other Secondary Controllers on the Bus, then the Host Controller can omit the Secondary Controller logic and not support Standby Controller Mode. In such applications, the Host Controller initializes the Bus (i.e., acts as Primary Controller) and does not pass the Controller Role to any other I3C Devices. If the Host Controller enters a standby mode or low-power mode, then it is expected to respond to Bus events, such as IBI Requests and Hot-Join Requests, by waking its system even though it does not relinquish the Controller Role at any time.

However, for more complicated applications, the system designer might require one or more Secondary Controller devices on the Bus. These devices might keep the Bus operational in a limited capacity while the Host Controller enters a deep sleep or low-power mode where it cannot respond to Bus events as mentioned previously. Alternately, other applications might rely on special-purpose Secondary Controllers to consume data from other I3C Targets that are acting as sensors and temporarily act as the Active Controller during that time. Both examples would require the Host Controller to support Standby Controller Mode and act as a Secondary Controller while another I3C Device is the Active Controller of the Bus.

The various options for Standby Controller Mode (see ***[MIPI12] Section 6.17***) provide a high degree of flexibility for the implementer to choose which capabilities a Host Controller will support, including the following:

- Whether the Host Controller is the Primary Controller or can join the Bus later (i.e., by sending a Hot-Join Request in a Secondary Controller Role and respond to a subsequent Dynamic Address Assignment)
- Whether the Host Controller can detect the state of the Bus, attempt to determine whether there is currently an Active Controller, and then try to claim the Controller Role if it sees no Active Controller (i.e., use the Dead Bus Recovery mechanism, per ***[MIPI12] Section 6.18***; see also the Error Type DBR method, defined in the I3C Specification ***Section 5.1.10.1.8***)
- Whether the Host Controller can initiate a Controller Role Request while it is a Secondary Controller
- Whether the Host Controller waits in a ‘primed’ state while it is a Secondary Controller to automatically accept the Controller Role from the current Active Controller, or whether it will refuse to accept the Controller Role (i.e., remain as a Secondary Controller)
- Whether the Host Controller optionally responds to other transaction types or CCCs while it is a Secondary Controller, using implementer-defined extensions to Secondary Controller Logic
- Whether the Host Controller monitors the Bus for specific CCCs that might be sent by the Active Controller to provide information on other I3C Devices, such as the **DEFTGTS** and **DEFGRPA** CCCs

When deciding which of these optional capabilities to support, the implementer should consider I3C’s specific application and the specific capabilities and behaviors expected of any other Secondary Controller Devices on the Bus. Implementers should be aware that ***[MIPI12] Section 6.17*** defines the minimum requirements for capabilities that the Host Controller must support, if the implementer chooses to support Standby Controller Mode for any use case.

Implementers seeking additional guidance on Standby Controller Mode and optional Secondary Controller capabilities should contact the MIPI Software WG.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

2.10 Implementation: As a Host Controller Implementer

Q10.1 How many Target Devices can a Host Controller support?

In theory, an I3C Bus is only limited by the maximum per-Device capacitance on SDA and SCL, although capacitance alone is not sufficient to determine the maximum number of Targets or the maximum frequency that can be used to drive transfers on the I3C Bus. The *I3C Application Note: General Topics* ***[MIPI05]*** lists several considerations for the maximum number of Targets.

A Host Controller can support a maximum of 32 Targets (both I3C and I^{2}C), limited only by the size of the DAT. This includes all Targets and Virtual Targets, assigned Group Addresses, or any other addressable entity or Device on the I3C Bus. However, implementers may reduce this maximum if an implementation has a DAT with fewer entries. In this case, such a Host Controller would only support a maximum number of addressable entities/Devices equal to the number of DAT entries.

Q10.2 How does the Host Controller resolve communication conflicts on the I3C Bus?

Per the I3C Specification, I3C Targets are only allowed to drive the Bus under certain situations. For example, I3C Targets may drive SDA to emit their Dynamic Address into the arbitrable Address Header after a START (but never after a Repeated START). After a START, the I3C Bus reverts to Open-Drain Pull-Up resistor mode for the arbitrable Address Header; thus, the I3C Target that drives a low value (i.e., logic 0) would win. This forms the beginning of an IBI Request, Hot-Join Request, or Controller-Role Request.

Note:

I3C Targets may also drive SDA when ACKing the Broadcast Address or their own address during the Address Header or while providing data during a read transfer. However, these situations do not cause potential communication conflicts on the I3C Bus.

When a Host Controller drives a START to initiate a transaction with an I3C Target, it also conditionally drives the Broadcast Address (i.e., 7'h7E) to give I3C Targets the opportunity to drive their own Dynamic Address or another special address into the arbitrable Address Header. (See also entry ***Q6.3***.) This behavior is configurable (as detailed in the note below). If no I3C Targets use this opportunity to do so, then the Host Controller continues with the transaction as planned:

- If the transaction is a Private Read or Private Write transfer in SDR Mode intended for a single Target, then the Host Controller sends a Repeated START followed by the Dynamic Address of the intended Target and waits for the Target to provide ACK. If the intended Target does not provide ACK, this will be interpreted as a NACK.
- If the transaction is a Private Write transfer in SDR Mode intended for a Group, then the Host Controller sends a Repeated START followed by the Group Address and waits for one or more Targets in the Group to provide ACK. If no Targets in the Group provide ACK, this will be interpreted as a NACK.
- If the transaction is a Generic Read or Generic Write transfer in an HDR Mode, then the Host Controller sends the appropriate **ENTHDRx** CCC to enter the selected HDR Mode, then initiates the HDR transfer according to the protocol of that HDR Mode.
- If the transaction is a CCC, then the Host Controller sends the indicated Command Code and optional Defining Byte and continues with the CCC framing.

Note:

For CCCs, the Host Controller always drives the Broadcast Address after START, per the CCC framing defined in the I3C Specification (see ***Section 5.1.9.1****). For SDR Private Read or Private Write transfers, the Host Controller determines whether to drive the Broadcast Address after START based on the current value of field* ***IBA_INCLUDE*** *in register* ***HC_CONTROL*** *and on whether the Driver used the Internal Control Command with sub-command 0x2 (Broadcast Address Enable/Disable). The former acts as a global setting for all transactions, while the latter is specific to the operating context. In DMA Mode, the sub-command can be used to specify the configuration for a particular Ring Bundle if multiple Ring Bundles are supported.*

If an I3C Target does try to drive its Dynamic Address or another special address into the arbitrable Address Header as part of a request, then the Host Controller will conditionally provide either ACK or NACK based on the following factors:

- The values of fields **CRR_REJECT** and **IBI_REJECT** in that Target's DAT entry
- The value of field **CREDIT_COUNT** in that Target's DAT entry, if IBI credit counting is supported and enabled
- The value of field **HOT_JOIN_CTRL** in register **HC_CONTROL**, if the special Hot-Join Address (7'h02) is received in the Address Header
- In PIO Mode, whether the IBI Queue is too full to accept a new IBI Status Descriptor that would report the status of the incoming request
- In DMA Mode, whether the appropriate IBI Status Ring is too full to accept a new IBI Status Descriptor that would report the status of the incoming request
- Whether the Host Controller has been halted for any other reason

Q10.3 How and when should implementers support Device Context in system memory?

For some implementations, the Host Controller will use Host system memory for the DAT entries, the DCT entries, or both. This requires the Host Controller to support the DMA capability into a region of the Host system memory, and it also requires the Host's system Bus to support such DMA requests to read from or write into this memory region. It is the Driver's responsibility to calculate the total amount of memory needed for all such tables the Host Controller expects to store in the Device Context memory and allocate a region of the Host system memory that is sufficiently large to hold all entries.

As part of Host Controller initialization, the Driver programs the base address of this Device Context memory into registers **DEV_CTX_BASE_LO** and **DEV_CTX_BASE_HI** (see ***[MIPI12] Section 7.4.19*** and ***Section 7.4.20***). If Scatter-Gather mode is supported, then the Driver configures this memory region to use Scatter-Gather mode by programming register **DEV_CTX_SG** (see ***[MIPI12] Section 7.4.21***) if the Device Context memory region is not a contiguous block (per ***[MIPI12] Section 6.2.1***).

If a Host Controller does not support the DAT and/or the DCT in regular registers, then it is required to support Device Context in Host system memory. This option is selected by the implementer (i.e., it is not a run-time configuration choice made by the Driver).

- If the DAT is not supported in regular registers, then each DAT entry takes 2 DWORDs of Host system memory.
  - The entire DAT will consume several DWORDs equal to 2 times the number of DAT entries (per field **TABLE_SIZE** in register **DAT_SECTION_OFFSET**). The DAT entries will be contiguous and arranged in ascending order.
  - Within each such DAT entry, the DWORDs are stored with Bits[31:0] in the first DWORD and Bits[63:32] in the second DWORD.
- If the DCT is not supported in regular registers, then each DCT entry takes 4 DWORDs of Host system memory.
  - The entire DCT will consume several DWORDs equal to 4 times the number of DCT entries (per field **TABLE_SIZE** in register **DCT_SECTION_OFFSET**). The DCT entries will be contiguous and arranged in ascending order.
  - Within each such DCT entry, the DWORDs are stored with Bits[31:0] in the first DWORD, Bits[63:32] in the second DWORD, Bits[95:64] in the third DWORD, and Bits[127:96] in the fourth DWORD.
  - Field **TABLE_INDEX** in register **DCT_SECTION_OFFSET** will effectively point to a specific DCT entry in Host system memory.
- If both of the above are true (i.e., if both DAT and DCT are not supported in regular registers), then the Host Controller will expect both such tables to be stored in Host system memory.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

If the Host Controller supports the DAT in Device Context memory, then the implementer should also consider which DAT fields might need to be cached as internal state, as it is likely infeasible for the Host Controller to rely entirely on Host system memory access to read the DAT entries when needed. The DAT is crucial for Target-related transactions, which is pertinent if the Host Controller wishes to reduce latency when performing DAT lookups in response to an incoming IBI Request or Controller Role Request. A non-cached implementation would require the Host Controller to slow or stall the I3C Bus clock while it performed a fresh read from Host system memory, examined the DAT entries received from Device Context memory, compared the incoming Dynamic Address against all DAT entries, and then decided to take the appropriate action. Similarly, a non-cached implementation would require the Host Controller to perform a fresh read from Host system memory when initiating any Transfer Command, as it would need to fetch the Target's Dynamic Address from the DAT entries (which did not reside locally in registers) to drive the Transfer Command on the I3C Bus.

To mitigate this situation, the implementer must determine how the Host Controller caches some internal state derived from the DAT fields that were previously read from Device Context memory. While the I3C HCI Specification does not specify the caching method and the frequency of updates to the cached state, there is a special Internal Control Command type that allows the Driver to tell the Host Controller when it should update its cached state based on changes that the Driver might have made to DAT entries in system memory. Refer to ***[MIPI12] Section 8.4.2.3*** for more details on the Device Context Update sub-command.

The caching concerns do not directly affect the DCT if it resides in Device Context memory. Since the Host Controller only writes to the DCT (and never reads from it) during Dynamic Address Assignment with **ENTDAA**, no internal state needs to be cached. The Host Controller can simply write to the DCT appropriately as and when it reports information on the I3C Targets that successfully complete Dynamic Address Assignment.

Q10.4 If the Host Controller supports both the DAT and DCT in Device Context memory, how is this memory partitioned?

Refer to the previous entry, ***Q10.3***, for more context. If the Host Controller supports both the DAT and the DCT in Device Context memory, then the DWORDs for all DAT entries are stored first (i.e., starting at byte offset 0x0) and are stored consecutively in the Device Context memory region. This means that the first DAT entry (i.e., entry #0) starts at byte offset 0x0 and takes the first 2 DWORDs; the second DAT entry (i.e., entry #1) starts at byte offset 0x8 and takes the next 2 DWORDs, and so on. After the last DAT entry (i.e., as determined by the number of DAT entries, reported by field **TABLE_SIZE** in register **DAT_SECTION_OFFSET**), the DWORDs for all DCT entries are stored contiguously, starting at byte offset **DAT_SECTION_OFFSET.TABLE_SIZE** × 8, with DCT entry taking 4 DWORDs.

For example, if the Host Controller supported a maximum of 32 DAT entries, the first DAT entry would start at byte offset 0x0, the second DAT entry would start at byte offset 0x8, and so on; the last DAT entry would start at byte offset 0xF8. After the last DAT entry, the first DCT entry would start at byte offset 0x100, the second DCT entry would start at byte offset 0x110, and so on.

If the Host Controller only supports one such table in Device Context memory (i.e., either the DAT or the DCT, but not both), then the first entry starts at byte offset 0x0.

Note:

These requirements were not clearly stated in I3C HCI v1.1 and earlier.

Q10.5 What is the recommended number of entries for the DAT and the DCT?

The minimum number of entries for each such table is 1, although this is not a practical or efficient configuration. For general use cases, a Host Controller should have at least 16 DAT entries and at least 4 DCT entries, although more is usually better.

Some considerations:

- As noted in entry ***Q10.1***, the number of DAT entries directly affects how many I3C and Legacy I^{2}C Target Devices can be addressed with Transfer Commands, as well as how many I3C Targets can send IBI Requests or Controller Role Requests that the Host Controller is expected to acknowledge. While it is theoretically possible for a Host Controller to support more Targets than the DAT entries can accommodate, this would require the Driver to swap DAT entries in and out of system memory based on which Targets it expects to use for Transfer Commands. This would not be an efficient use of Host resources and might even block the reception of IBI Requests and Controller Role Requests. The MIPI Software WG recommends that the Host Controller support at least as many DAT entries as Targets (i.e., addressable entities or Devices) on the Bus, with a few extras to be used for Dynamic Address Assignment after a Hot-Join Request.

- The number of DCT entries directly affects how many I3C Target Devices can be detected and assigned a Dynamic Address with a single Address Assignment Command (see ***Section 8.4.1.1***). It is safe for a Host Controller to have fewer DCT entries than DAT entries. However, the Driver must be aware of this limitation and only submit Address Assignment Commands that do not exceed the number of DCT entries. If the I3C Bus has more I3C Targets than DCT entries, the Driver will need to submit multiple Address Assignment Commands, assign Dynamic Addresses in stages, and only initiate the next stage once it consumes the data reported by the Host Controller for any I3C Targets that were detected in the previous stage. Since the DCT stores transient data that the Driver should read immediately, this will impact efficiency but will not affect how many I3C Targets can be used with Dynamic Address Assignment. In theory, a DCT with the minimum size (i.e., only 1 entry) is still functional if the Driver initiates ***N*** Address Assignment Commands (i.e., one for each expected I3C Target with field **DEV_COUNT** set to 1) where ***N*** is equal to the number of I3C Targets. However, this is not as efficient as a DCT with more entries, where the Driver can assign more than 1 Dynamic Address per Address Assignment Command. The MIPI Software WG recommends that the Host Controller support at least 4 DCT entries, although this depends on the use case.
- If the Host Controller uses the DAT or the DCT in Device Context memory (see entries ***Q10.3*** and ***Q10.4*** for context), then the Host Controller should generally not restrict the number of entries in such tables, as the size is only limited by available Host system memory. However, the Host Controller will still need to check the incoming Dynamic Address for an IBI Request or Controller Role Request with lookup logic that searches for this Dynamic Address across all valid DAT entries. In some cases where logic space is at a premium, the implementer may wish to simplify the lookup logic by restricting the number of DAT entries that can be supported (i.e., searched in parallel).

Q10.6 What factors should be considered when deciding whether to implement support for PIO Mode?

In general, PIO Mode works well for smaller integrations, such as micro-controllers, application processors, or FPGAs, where the Driver can dedicate enough cycles to interact with the Host Controller, as the Driver must be more directly involved with the transactions that it sends to the Host Controller. The Driver must be able to do this with appropriately low latency.

PIO Mode is also generally suitable for space-constrained systems: A Host Controller implementation that only supports PIO Mode should require considerably less logic and does not require the Host to support DMA capability on its system Bus. If a Host system does not support DMA capability on its system Bus, then PIO Mode is the only possible operating mode for such a Host Controller implementation.

However, PIO Mode requires a high level of interaction with low latency. The Host Controller will likely send many interrupts that need to be handled by the Driver during transactions. If the Driver does not respond promptly (which could be caused by multitasking within the Host's processing core), then the Driver's delayed interrupt handling could negatively impact the Host Controller's operations and I3C Bus transactions. For example, if the Driver cannot service the PIO Queues promptly or if the PIO Queues are not sized appropriately for the use case and expected latency, then the Host Controller could see data overflow/underflow errors during Transfer Commands or IBI data payloads, as well as possible errors due to Command Sequence stalls or timeouts.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

Q10.7 What factors should be considered when deciding whether to implement support for DMA Mode?

In general, DMA Mode works well for medium-to-large implementations that can afford the additional logic and buffers needed to handle the higher level of automated transaction processing within the Host Controller. For some general computing use cases with an OS, the Driver (typically, software running on the Host) cannot reliably service frequent Host Controller interrupts (as PIO Mode requires) to prevent negative impacts such as overflow/underflow and stalls/timeouts, among others. For such systems, DMA Mode is more efficient from the perspective of the Host, as it allows the Driver to offload more of the transaction processing to Host Controller logic.

DMA Mode is also more suitable for systems that could have multiple Drivers (e.g., firmware agents or other execution contexts) that interact with different Targets on the I3C Bus. If the Host Controller supports more than one Ring Bundle, then each Driver (or agent/context) can interact solely with its own assigned Command/Response Rings and let the Host Controller logic handle the arbitration and serialization.

However, DMA Mode requires more logic and additional buffer memories to handle the transaction processing of the Command/Response Ring Pair and the arbitration and serialization across multiple Ring Pairs if multiple Ring Bundles are supported. In this case, offloading this work to the Host Controller comes with more cost and complexity. DMA Mode also requires DMA capability on the Host's system Bus. As a result, the Driver must spend more effort to manage the allocated memory for the Rings; typically, these are allocated in memory that the Driver can access (i.e., kernel memory), which could be limited in some systems, depending on the OS implementation.

Additionally, if the Host system does not support DMA capability on its system Bus, then DMA Mode cannot be used for such a Host Controller implementation (see ***Q10.6***).

Q10.8 Can a Host Controller implement support for both PIO Mode and DMA Mode?

Yes, if the Host system Bus supports DMA (see ***Q10.7***). However, this requires additional complexity, as the logic for both operating modes must be present even though only one operating mode can be active at a time. This means that the PIO Mode registers will not be used when DMA Mode is selected and vice versa. Additionally, the Host Controller must support mode switching when the Driver writes to the **MODE_SELECTOR** field of the **HC_CONTROL** register.

Q10.9 How much of a Host Controller implementation must be hardware or firmware?

The I3C HCI Specification does not specify which Host Controller behaviors should be handled by dedicated hardware logic or by firmware running on a processing core or engine. However, it is generally expected that lower-level behaviors involving Bus Controller logic should be done in dedicated hardware logic, as it is not typically efficient to implement the Bus Controller logic via bit-bang due to higher latency. Since an I3C Bus Controller is required to react quickly to changing states on the SDA and SCL lines, dedicated hardware logic is preferable for the lower-level operations that interact with the I3C Bus.

With that said, it is possible for the upper-level portion of a Host Controller (e.g., the Host-facing interface) to be implemented either by dedicated hardware logic or by firmware running on a processing core adjacent to the Bus Controller logic. In some implementations, the processing core could utilize programmable logic (e.g., FPGA). If an implementer wishes to use a split hardware/firmware implementation, then the implementer must ensure that the firmware can reliably and quickly react to events generated by the Bus Controller logic that drives operations onto the I3C Bus, and the Host Controller implementation's latency and performance would not be compromised.

Q10.10 How should a Host Controller's registers and queues be implemented?

The I3C HCI Specification does not specify how the registers and queues should be implemented. Host Controller implementers are free to use various implementation methods for the registers and queues if the resulting implementation conforms to the behaviors and expectations defined in the I3C HCI Specification.

In most cases, certain configuration registers would typically be implemented as flip-flop structures, as these would need to be frequently accessed internally. In other cases, some registers and queues could be implemented as other memory-based structures that are accessed via an internal interface (such as register files, SRAMs, or similar).

If PIO Mode is supported, then many of the PIO Queues could be implemented as flip-flops or other memory-based structures. The implementer should determine which structure makes the most sense for the implementation based on the level of complexity, the size of the structure, the latency required to use the queue, and the performance characteristics of the memory structures available on the specific manufacturing process. The implementer is also free to decide whether a queue that uses a memory structure, such as an SRAM, will need to be buffered internally for performance and latency reasons. However, this internal buffering is not defined or required by the I3C HCI Specification, and it would not be visible to the Driver via the register interface. In some cases, an implementer might choose to use a single shared memory structure for multiple queues and use internal buffers for certain elements to mitigate latency issues caused by the use of a shared memory structure.

Q10.11 What I3C data transfer speeds are required, and how should a Host Controller support them?

The I3C HCI Specification and I3C TCRI Specification provide guidelines on the data transfer rates that a Host Controller should support. In SDR Mode, there are 5 standard SDR speeds, from SDR0 to SDR4, which can be specified in the **MODE** field of the Transfer Command.

The following table from the I3C TCRI Specification (in ***[MIPI06] Section 7.1.1.1***) shows the guidance for the maximum data transfer speeds for each such SDR speed.

Maximum Values for I3C SDR Data Transfer Speeds

MODE Field Value	Listed Speed	Maximum Sustainable Data Rate
0x0	I3C SDR0	12.5 MHz, Standard SDR Speed, f_{SCL} Max
0x1	I3C SDR1	8 MHz
0x2	I3C SDR2	6 MHz
0x3	I3C SDR3	4 MHz
0x4	I3C SDR4	2 MHz

Per these guidelines, the SDR0 speed should be the fastest data rate and is typically the maximum sustainable data rate on the I3C Bus (i.e., 12.5 MHz, based on parameter f_{SCL} Max, which is defined in the I3C Specification). However, the implementer may choose to reduce this to a lower data rate (e.g., 10 MHz) for a particular use case, especially if the I3C Bus has a more complex topology or higher overall capacitance and would be unable to use the maximum data rate of 12.5 MHz for a particular system. In special cases, the clock provided within the overall system might not allow the Host Controller to use the maximum sustainable data rate of 12.5 MHz for the SDR0 speed, and a slightly lower data rate could be used. Such a Host Controller will still properly drive I3C transactions on the Bus at the lower data rate, albeit with reduced efficiency. The SDR1 through SDR4 speeds are expected to be slower than the SDR0 speed, with each successively slower than the preceding speed, and the guidelines provide recommended data rates for each of these speeds. However, implementers are not constrained to use these specific data rates for the SDR0 through SDR4 speeds: The actual data rates will depend on the application and on the implementer’s choice of clock logic used by the Host Controller or the clock provided within the overall system (i.e., the product that integrates the Host Controller).

For example, assume that a Host Controller implementation receives a 100 MHz clock signal from its system and internally divides this clock signal by various integers to derive each specific data rate used by its Bus Controller logic for transfers that can use the SDR0 through SDR4 speeds. In this example, the SDR0 speed can operate at the recommended value (also the maximum value) of 12.5 MHz, since this can be achieved with an integer divider of 8. However, for the SDR1 speed, a data rate of 8 MHz would not be possible, as

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

there is no integer divider for 100 MHz that results in 8 MHz. In this case, the implementer could choose to use an integer divider of 12 or lower (which is not recommended, as that would produce a data rate higher than 8 MHz) but instead should choose another integer divider, such as 13 (i.e., to produce a data rate of 7.6923 MHz), 14 (i.e., to produce a data rate of 7.1428 MHz), or higher. Similarly, the other data rates for the SDR2 through SDR4 speeds would depend on the available integer dividers that produce data rates that should approach but not exceed the guidance on maximum values.

Alternately, assume that a Host Controller implementation uses a tick-counter approach to derive data rates. Using the base clock as a reference, it would need to determine the number of clock ticks for the number of High and Low cycles for the data rate of each SDR speed. For example, if the Host Controller receives a 100 MHz clock signal from its system, it can use 4 clock ticks for the High cycle and 4 clock ticks for the Low cycle to achieve an effective data rate of 12.5 MHz with a 50/50 duty cycle for the SDR0 speed. The Host Controller can then use additional ticks for the High and Low cycles of the SDR1 through SDR4 speeds if each produces a valid duty cycle.

Regardless of the provided clock or the method for deriving the data rates, the Host Controller must drive all I3C transactions with a valid duty cycle that is acceptable to I3C Devices and any Legacy I^{2}C Devices that might be present, such as for a Mixed Bus. If this is a Mixed Bus, then the Host Controller should also ensure that the High-cycle time of I3C transactions is short enough to be filtered out by the 50 ns spike filter of such Legacy I^{2}C Devices (i.e., since Legacy I^{2}C Devices should not see any traffic addressed only to I3C Devices). This typically means that the Low-cycle time in the SDR1 through SDR4 speeds will be successively greater as the data rate is decreased.

While the I3C HCI Specification does not define the method for exposing the effective data rates that will be used for I3C transactions, implementers should consider exposing these timing parameters in registers within an implementer-defined Extended Capability structure. This allows the Driver to read the timing parameters and know the effective data rates for the SDR0 through SDR4 speeds. In some cases, it might also be advised to allow the Driver to adjust the duty cycle and effective data rate for each speed by providing writeable register fields. Since the SDR0 through SDR4 speeds only apply to the Push-Pull phase of an I3C transaction in SDR Mode, the timing parameters for HDR transactions and the Open Drain phases of SDR transactions should also be exposed by such implementer-defined registers.

Q10.12 How does a Host Controller advertise which types of Internal Control Commands are supported?

Starting with I3C HCI v1.1, implementers are required to support register **INT_CTRL_CMD_EN** (see ***[MIPI12] Section 7.4.16***). This register has a read-only field to indicate the specific sub-commands of the Internal Control Command Descriptor that the Host Controller supports.

Certain Internal Control Command sub-commands are required to be supported, and other sub-commands are conditionally required based on other Host Controller capabilities. While these requirements were not clearly stated, and the register above was not defined, in I3C HCI v1.0, later versions of the I3C HCI Specification state these requirements clearly.

- Sub-Command 0x1 (Ring Bundle Lock) is conditionally required: If the Host Controller supports DMA Mode with at least two Ring Bundles, then it shall support this sub-command.
- Sub-Command 0x2 (Broadcast Address Enable/Disable) is always required.
- Sub-Command 0x3 (Device Context Update) is conditionally required, if the Host Controller supports Device Context (i.e., if the DAT entries are expected to be read from Host system memory).
- Sub-Command 0x4 (Target Reset Pattern) is always required. This sub-command was added in I3C HCI v1.1.
- Sub-Command 0x5 (Controller SDA Recovery or Bus Reset Procedure) is always required. This sub-command was added in I3C HCI v1.1.
- Sub-Command 0x6 (Enable End Transfer Termination and HDR Mode Configuration) is conditionally required: If the Host Controller supports HDR Modes or Monitoring Devices that

rely on the **ENDXFER** CCC for configuration or activation, then it shall support this sub-command. This sub-command was added in I3C HCI v1.2.

- Sub-Command 0x7 (Controller Role Handoff Procedure with **GETACCCR** CCC) is conditionally required: If the Host Controller supports Standby Controller Mode, then it shall support this sub-command. This sub-command was added in I3C HCI v1.2.
- Sub-Command 0xD (Attempt Dead Bus Recovery) is conditionally required: If the Host Controller supports the Dead Bus Recovery Mechanism, then it shall support this sub-command. This sub-command was added in I3C HCI v1.2.

Implementers should consult ***Section 8.4.2*** and its sub-sections for more details.

2.11 Implementation: As a Software Developer

Q11.1 Where are the definitions for Transfer Command and Transfer Response structures?

In I3C HCI v1.0 and v1.1, the Transfer Commands and Transfer Responses were normatively defined within ***Section 8***. Starting with I3C HCI v1.2, the Transfer Commands and Transfer Responses are normatively defined in the I3C TCRI Specification. The latest adopted version is I3C TCRI v1.0 ***[MIPI06]***. Note that the Transfer Commands and Transfer Responses supported by I3C TCRI v1.0 have the same definitions as in I3C HCI v1.1.

Developers should refer to the I3C HCI and I3C TCRI Specifications to fully understand all Host Controller behaviors with respect to Transfer Commands and Transfer Responses. In general, I3C HCI defines the *how* – how the Host Controller interacts with its Host to receive Transfer Commands and generate Transfer Responses – and the I3C TCRI defines the *what* – what I3C Bus Controller behavior is expected when processing either a single Transfer Command or a sequence of Transfer Commands.

Q11.2 Are there any companion MIPI I3C Specifications that enable software development or system integration?

Yes. The following MIPI Specifications are expected to enable software development and system integration:

- **MIPI Specification for Discovery and Configuration (DisCo), v1.0** *[MIPI03]*

  Describes a standardized device discovery and configuration mechanism for interfaces based on MIPI Specifications, which can simplify component design and system integration. Also oriented to application processors.

- **MIPI DisCo Specification for I3C, v1.1** *[MIPI04]*

  Allows operating system software to use Advanced Configuration and Power Interface (ACPI) structures to discover and configure the I3C Host Controller and attached I3C Devices in ACPI-compliant systems. Also oriented to application processors.

In addition to these MIPI Specifications, the MIPI I3C WG has released several application notes that can help ASIC hardware developers, system designers, and others working in the more deeply embedded I3C Devices.

Q11.3 Are there software libraries available for I3C?

Yes. Core I3C infrastructure has been added to the Linux Kernel as part of the I3C subsystem. The I3C subsystem also includes Drivers for several I3C Controller Devices and IP core implementations, including MIPI I3C HCI-compliant Host Controllers (see ***[MIPI09]***).

The current list of Linux Kernel Patches for the I3C subsystem can be accessed via ***[LINX01]***.

2.12 Operation in PIO Mode

Q12.1 What has changed in I3C HCI v1.1 relating to PIO Mode?

In I3C HCI v1.0, the queue threshold control register (**QUEUE_THLD_CTRL**) had inconsistent definitions for the various threshold fields. For example, some threshold fields were defined to be "zero-based," meaning that a value of 0x0 defined a threshold of 1 or more entries, a value of 0x1 defined a threshold of 2 or more entries, and so on. However, other threshold fields were defined with a value of 0x0 to indicate either zero (i.e., no) or more entries or an empty queue. This confused implementers. In I3C HCI v1.1, the definitions for the queue threshold fields were made consistent. In some cases, this means that a value of 0x0 is no longer a valid value, since it did not make sense in order to trigger an event.

In I3C HCI v1.0, if a Host Controller supported both PIO Mode and DMA Mode, the selected operating mode was implicitly determined based on the number of enabled Ring Bundles. For example, if no Ring Bundles were enabled, then the Host Controller logic would use PIO Mode. However, if the Host enabled at least one Ring Bundle, then the Host Controller would automatically switch into DMA Mode. This confused implementers and software developers. In I3C HCI v1.1, an explicit control field was added to register **HC_CONTROL** to select the operating mode.

Additionally, an extensive Theory of Operation section covering PIO Queue Management was added to explain the fundamental aspects of PIO Mode operation and how the Host should interact with the PIO Queues to drive transfers.

Q12.2 What has changed in I3C HCI v1.2 relating to PIO Mode?

An error in I3C HCI v1.1 (corrected by Errata 01) referred to the wrong field in the definition of register **PIO_INTR_STATUS**. The descriptive text for field **CMD_QUEUE_READY_STAT** incorrectly referred to field **RESP_EMPTY_BUF_THLD** when it should have referred to field **CMD_EMPTY_BUF_THLD**. This error was also corrected in I3C HCI v1.2.

I3C HCI v1.2 also gives implementers more options to flexibly define various PIO Queues to suit the application. These alternate PIO Queue sizes are now indicated in register **ALT_QUEUE_SIZE** (new for this version). However, if implementers choose not to use these options, then this register can be defined with a value of 0x0 (i.e., all fields have zero values), which is backwards compatible with I3C HCI v1.1.

- In I3C HCI v1.1 and earlier, the sizes of the Command Queue and the Response Queue were always linked (i.e., both queues always had the same number of entries). While this worked well for many use cases, other specialized use cases could benefit from allowing these queues to be sized independently. In I3C HCI v1.2, the implementer may choose to define these queues with different sizes. Then, the implementer must indicate the actual defined size of the Response Queue in register **ALT_QUEUE_SIZE** and indicate that the Response Queue has a different size than the Command Queue (where the size is still defined by register **QUEUE_SIZE**).
- In I3C HCI v1.1 and earlier, the size of the IBI Queue is limited to a maximum of 255 DWORDs. While this worked well for many use cases, other specialized use cases could benefit from a larger IBI Queue, particularly systems that frequently use IBIs to communicate I3C Target status and longer data payloads using Auto-Command. For such use cases, a larger IBI Queue allows the Driver more time to consume the data, particularly systems with higher latency in processing interrupts. In I3C HCI v1.2, the implementer may choose to define a larger IBI Queue, up to a maximum of 2040 DWORDs. Then, the implementer must indicate this in register **ALT_QUEUE_SIZE**, which effectively multiplies the value of field **IBI_STATUS_SIZE** in register **QUEUE_SIZE** by 8.

Additionally, I3C HCI v1.2 adds a new **PIO_CONTROL** register that gives the Driver control over the operation of the PIO Queues, providing similar functionality to what the **RING_CONTROL** register provides for each Ring Bundle.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

2.13 Backwards Compatibility with I²C

Q13.1 Is an I3C Host Controller backwards compatible with I²C? Can I3C and I²C Devices coexist on the same Bus?

Yes. Most Legacy I²C Target Devices can be operated on an I3C Bus, provided they have a 50 ns spike (glitch) filter and do not attempt to stall the clock. Such use will not degrade the speed of communications to I3C Targets; it will require decreased speed only when communicating with the I²C Targets.

I3C supports Legacy I²C Target Devices using Fast-mode (Fm, 400 KHz) and FastMode+ (Fm+, 1 MHz) with the 50 ns spike filter. It does not support the other, faster I²C modes, I²C Devices that lack the spike filter, or I²C Devices that stretch the clock.

I3C Host Controllers can communicate with Legacy I²C Targets on the Bus if such devices meet these requirements as defined in the I3C Specification, ***Section 5.1.1.1***. During initialization, the Driver must write 1'b1 to field **I2C_DEV_PRESENT** in register **HC_CONTROL** (see ***[MIPI12] Section 7.4.2***).

If the Bus has one or more such Legacy I²C Targets (i.e., is a Mixed Bus), the Host Controller must initiate transactions and use an appropriate data rate and duty cycle, depending on whether it is addressing an I3C Target or a Legacy I²C Target. For example, when addressing I3C Devices, the Host Controller must observe the timing requirements and ensure that the High cycle of SCL is not longer than 50 ns, as that would allow the transaction to pass through the Legacy I²C Target's 50 ns spike filter. Similarly, when addressing Legacy I²C Targets, the Host Controller must ensure that the data rate and duty cycle are compatible with I²C Targets. To ensure the Host Controller knows the difference, the Driver must write the appropriate value into the **DEVICE** field of the DAT entry for the Target (see ***[MIPI12] Section 8.1***).

Note:

*Certain HDR Modes may not be used on a Mixed Bus. Additionally, some I3C timing parameters need to accommodate the presence of Legacy I²C Devices on a Mixed Bus. Consult the I3C Specification **Section 5.1.2.4** for more details.*

Additionally, the Host Controller must account for the Static Address of each Legacy I²C Target device on the Bus and ensure that it does not assign that same Dynamic Address to any I3C Targets. If the system has a configuration resource (e.g., DisCo for I3C) that contains the list of known Legacy I²C Targets on the Bus, then the Driver should read that data and account for such devices. Otherwise, it must acquire that data on Legacy I²C Targets from another source. The Driver will then go through the initialization procedure with full awareness of all Legacy I²C Targets on the Bus before it sends any Address Assignment Commands as part of Bus initialization.

Note:

I3C Buses do not support Legacy I²C Controller Devices or any Legacy I²C Devices that attempt to act as the Controller and drive the SCL line (i.e., when arbitrating for control to drive transactions). Such Devices cannot share the same Bus with I3C Devices. I3C Buses also do not support any Legacy I²C Devices that use clock stretching, since the SCL line is owned by the Active Controller and driven in Push-Pull mode.

Q13.2 Does the Driver need to take any additional steps if the Bus has I3C Targets that initially act as Legacy I²C Targets?

Possibly. Such I3C Targets might be used on either an I3C Bus or a Legacy I²C Bus, and when powered on, they will have a 50 ns spike filter enabled by default and will not inherently know whether they are on an I3C Bus. As a result, they will not know whether they can pull SDA Low to initiate a Hot-Join Request, since that capability is not supported on Legacy I²C Buses. As a result, the Primary Controller must send a START followed by the Broadcast Address (7'h7E) at a data rate that will be seen by such an I3C Device with its 50 ns spike filter enabled (i.e., the default state). (Refer to the I3C Specification ***Section 5.1.2.1.1*** for more details.)

Host Controllers are not required to do this automatically when initialized by the Driver. If a Host Controller does this automatically, then no additional steps must be taken. However, if this does not happen, then the Driver needs to send a Transfer Command using a DAT index for an imaginary Legacy I²C Device (i.e., a

DAT entry that is temporarily configured with a dummy address and field **DEVICE** set to 1'b1) for any valid Broadcast CCC (such as **ENEC** or **DISEC**). The Transfer Command should use a safe data rate (i.e., field **MODE** set to 0x0, for I²C Fast Mode at 400 KHz) to ensure that the Broadcast CCC is sent slowly enough to pass through the enabled spike filter of such an I3C Target.

If some other I3C Target (i.e., one that knows it is on an I3C Bus by default) raises a Hot-Join Request that wins arbitration over the Broadcast Address (7'h7E), this will take priority, and the other I3C Targets will not see the START, 7'h7E/W pattern that informs them they are definitely on an I3C Bus. As a result, the Driver should resend the same Transfer Command if it receives any Hot-Join Request notifications to ensure that the other I3C Targets can clearly see the START, 7'h7E/W pattern.

Note:

Implementers may choose to add support for this initialization step in hardware to be activated automatically when the Driver enables and initializes the Host Controller before processing any Transfer Commands.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

2.14 Dynamic Address Assignment and Group Address Assignment

Q14.1 Can the Host Controller detect a PID collision during Dynamic Address Assignment with the ENTDAA CCC?

Collisions are not possible when each I3C Target Device has its own Manufacturer ID and unique part number. If more than one instance of the same Target is used on a given I3C Bus, then each such instance must have a separate instance ID; otherwise, there would be a collision. Alternately, if the Driver previously sent a Transfer Command with the **ENTTM** Broadcast CCC to enter Vendor Test Mode (per ***Section 5.1.9.3.8*** of the I3C Specification ***[MIPI10][MIPI11]***), and if two or more I3C Target Devices chose the same random 32-bit value, then a collision would be possible (although unlikely).

If the Host Controller knows the number of I3C Targets on the I3C Bus (either from prior knowledge or by reading the system's configuration, such as from a DisCo resource), then it can detect this condition: The number of Dynamic Addresses assigned would be less than the expected number of Targets. If the Driver detects this, then it can take steps to resolve such collisions, for example by resetting all Dynamic Addresses with the **RSTDAA** CCC and restarting the process by re-sending the **ENTTM** Broadcast CCC to exit the test mode (if applicable) or by eventually declaring a system error after a set maximum number (e.g., 3) of such attempts fail.

However, if the Host Controller does not know the number of I3C Targets in advance, then it cannot detect PID collisions, meaning that multiple I3C Targets that return the same PID + BCR + DCR values during the Dynamic Address Assignment process (with the **ENTDAA** CCC) will effectively be assigned the same Dynamic Address. This will cause communication errors on the I3C Bus.

Q14.2 Does the Host Controller require any special support for Group Addressing?

No, Group Addresses are implicitly supported. Each assigned Group Address effectively needs to be treated as its own Target, meaning that the Host must allocate one DAT entry per Group Address in addition to other DAT entries for each Target's Dynamic Address. Once a DAT entry is set up for a Group, and once a Group Address is assigned to one or more Targets with the **SETGRPA** CCC (which the Driver must enqueue via Transfer Commands), the Driver can subsequently send Transfer Commands using the DAT index for that assigned Group Address.

Note:

Per the I3C Specification, all transactions addressed to a Group Address must be Write transfers, as Read transfers cannot be addressed to Group Addresses. Additionally, each I3C Target must first have an assigned Dynamic Address before it can be assigned any Group Addresses.

Q14.3 How does the use of any Address Assignment CCCs affect existing DAT entries?

The Host Controller does not update the contents of DAT entries when it sends any such Address Assignment CCCs on the I3C Bus. This includes the Address Assignment Commands (i.e., for **ENTDAA** or **SETDASA** CCCs) and Transfer Commands that send any other such CCCs (e.g., for **SETNEWDA**, **SETAASA**, **RSTDAA**, **SETGRPA**, or **RSTGRPA** CCCs). It is the Host's responsibility to manage/update the DAT entries before sending new Transfer Commands to I3C Targets that are affected by Address Assignment CCCs.

Q14.4 How can an I3C Target's assigned Dynamic Address be changed?

If the I3C Target supports the **SETNEWDA** CCC, then the Driver should first enqueue a Transfer Command with the **SETNEWDA** CCC. If this succeeds, then the Driver should update the DAT entry by writing the new Dynamic Address into the **DYNAMIC_ADDRESS** field.

Note:

*Per the I3C Specification, I3C Targets that support the **ENTDAA** CCC must also support the **SETNEWDA** CCC. For special I3C Targets that do not support both the **ENTDAA** and **SETNEWDA** CCCs, the only way to change the assigned Dynamic Address is to reset by sending either the **RSTACT** CCC followed by the Target Reset Pattern or by sending the **RSTDAA** CCC, which affects all I3C Targets (see also **Q14.3** and **Q14.6**). Once this is done, the Driver can assign a different Dynamic Address to that I3C Target using the **SETDASA** CCC.*

If the DAT is stored in Device Context memory, then the Driver should subsequently use the Device Context Update sub-command of the Internal Control Command to inform the Host Controller that the DAT entry has been updated (see also ***Q10.3***). Refer to ***[MIPI12] Section 8.4.2.3*** for more details on the Device Context Update sub-command.

Q14.5 Should the Driver clear an existing DAT entry if an assigned Dynamic Address or Group Address is no longer valid?

Yes. The Driver should write the value 7'h00 into the **DYNAMIC_ADDRESS** field in that DAT entry to mark it as disabled.

If the DAT is stored in Device Context memory, then the Driver should subsequently use the Device Context Update sub-command of the Internal Control Command to inform the Host Controller that the DAT entry has been removed (see also ***Q10.3***). Refer to ***[MIPI12] Section 8.4.2.3*** for more details on the Device Context Update sub-command.

Q14.6 Does the RSTDAA CCC also clear any assigned Group Addresses?

Yes. If the Driver enqueues a Transfer Command that is the **RSTDAA** CCC, this will reset all assigned Dynamic Addresses and all Group Addresses for all I3C Targets, per ***Section 5.1.4.4*** of the I3C Specification. However, this does not clear or otherwise disable any DAT entries that contain previously valid Dynamic Addresses and/or Group Addresses. Consequently, all such DAT entries for I3C Targets will point to unassigned Addresses until the Host updates or disables the entries and then subsequently assigns new Dynamic Addresses and optional Group Addresses.

Q14.7 How does the RSTGRPA CCC affect existing DAT entries with Group Addresses?

If the Driver enqueues a Transfer Command that is the **RSTGRPA** CCC, then this will reset the assigned Group Addresses for affected I3C Targets depending on the format of the **RSTGRPA** CCC that is used. However, this does not clear or otherwise disable any DAT entries that contain previously valid Group Addresses. Consequently, the Host will need to determine which DAT entries for affected Group Addresses must be updated or disabled. If the Host subsequently chooses to assign new Group Addresses, a new DAT entry will need to be allocated for each such Group Address (see ***Q14.2***).

2.15 In-Band Interrupt and Hot-Join

Q15.1 How can an I3C Controller support Pending Read Notifications?

I3C Host Controllers can easily support Pending Read Notifications by implementing the Auto-Command feature that conforms to the Pending Read Notification contract. This allows the Host Controller to automatically initiate a Read transfer after receiving an IBI that is a Pending Read Notification. Once a suitable IBI is seen, the Host Controller will initiate either a Private Read (in SDR Mode) or an HDR Generic Read (in supported HDR Modes) of the same Target, without software intervention, based on matching MDB values. The Driver must configure the Auto-Command fields in the DAT entries for I3C Targets to enable the Auto-Command behavior.

If the implementer chooses not to support the Auto-Command feature, then it is the Driver's responsibility to manage the Pending Read behavior. In some situations, the Driver must pause or cancel any previously enqueued Read transfers if the Host Controller receives such an IBI with a matching MDB value to signal a Pending Read Notification, as this would oblige the Driver to initiate a Read transfer (i.e., SDR Private Read or HDR Generic Read) that is expected for this IBI. If another Read transfer was enqueued and the Driver did not (or could not) stop it in time, then there is a chance that this Read transfer would consume the Pending Read data. Then, it would be the Driver's responsibility to correctly associate the consumed Pending Read data with the IBI notification.

Q15.2 Can the Driver tell the Host Controller to deny any Hot-Join Requests?

Yes, although this is not recommended for most use cases. Hot-Join Requests are fundamental to discovering I3C Targets that join the Bus and need a Dynamic Address to be assigned. If Hot-Join Requests must be temporarily disabled, then the Driver should take the following steps:

- Enqueue a Transfer Command that sends the Broadcast **DISEC** CCC with the **DISHJ** bit set (see the I3C Specification ***Section 5.1.9.3.1***.
- Write 1'b1 to field **HOT_JOIN_CTRL** in register **HC_CONTROL** (see ***[MIPI12] Section 7.4.2***) to tell the Bus Controller logic to respond automatically with NACK followed by the Broadcast **DISEC** CCC with the **DISHJ** bit set.

When the Driver decides to re-enable Hot-Join Requests on the I3C Bus, the Driver should take the following steps:

- Write 1'b0 to field **HOT_JOIN_CTRL** in register **HC_CONTROL** to tell the Bus Controller logic to respond automatically with ACK.
- Enqueue a Transfer Command that sends the Broadcast **ENEC** CCC with the **ENHJ** bit set (see the I3C Specification ***Section 5.1.9.3.1***.

Note:

If Hot-Join Requests are enabled, then the Driver will still need to enqueue new Address Assignment Commands to assign a Dynamic Address, as and when I3C Targets successfully send Hot-Join Requests.

Q15.3 How should the Driver prepare for situations where the Host Controller frequently processes Hot-Join Requests on the I3C Bus?

In many situations where Hot-Join Requests are received frequently, or when they arrive later than I3C Bus initialization (i.e., after the initial set of I3C Targets are discovered and configured), unexpected Hot-Join Requests can create a challenge for Driver implementers. This is especially true if I3C Targets are power-cycled frequently or go through deep sleep/wake cycles where they do not keep their Dynamic Addresses. Here are some recommendations on how to handle such situations:

- Maintain a list of known I3C Targets in Host system memory, along with PID + BCR + DCR values (i.e., the values read from the DCT when each I3C Target is initially discovered).
- As an I3C Target goes offline (i.e., fails to respond to Transfer Commands), mark its entry in Host system memory to show the I3C Target is offline but may rejoin later.

- Reserve one DAT entry with a temporary Dynamic Address (e.g., the last DAT entry) that will be used for subsequent Hot-Join Requests after initial I3C Bus configuration:
  A. As and when each Hot-Join Request is received, enqueue an Address Assignment Command for **ENTDAA** with **DEV_COUNT** = 1 and **DEV_INDEX** pointing to that reserved entry.
  B. Capture the PID + BCR + DCR values from the DCT entry and compare them with the list of previously known I3C Targets. If the values match, then use the **SETNEWDA** CCC to re-assign the Target's previous Dynamic Address. At this point, the previous DAT entry is valid again. If the values do not match (i.e., this Target was not previously known), then set up a new DAT entry to hold a new Dynamic Address, then use the **SETNEWDA** CCC to change the Target's Dynamic Address.
  C. The reserved DAT entry will be free for use again, with the same temporary Dynamic Address.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

2.16 Common Command Codes (CCCs)

Q16.1 What are the differences in CCC behaviors for I3C HCI v1.1 and later?

I3C HCI v1.1 added support for CCCs with Defining Bytes, a new feature that was added with I3C v1.1. This required the Transfer Command types to be extended to indicate whether a Defining Byte would be sent and which Defining Byte value to send. The Managed CCC Transfer Framing model now supports CCCs with and without Defining Bytes and automatically re-sends the Broadcast Address (7'h7E) along with new, optimized Command Codes and Defining Bytes, when needed.

I3C HCI v1.1 and later also clarified details on which CCCs were not allowed to be sent in Transfer Commands, as the Host Controller could only send these CCCs automatically.

Q16.2 Does I3C HCI support sending CCCs in HDR Modes?

Host Controllers do not yet support this capability.

Q16.3 Does the mandated "single-retry model" apply to all Direct Read CCCs?

The I3C v1.1 Specification ***Section 5.1.9.2.3*** states: "I3C mandates a single-retry model for Direct GET CCC Commands." Based on this statement, the Host Controller is required to retry once for the Directed GET CCCs if the Target NACKs the first try. This may not be necessary for other 'read' Directed CCCs (such as **RSTACT**, **MLANE**, vendor read, or others) that are not defined for dedicated Read CCCs (i.e., CCCs that have names in the form **GETxxx**). However, the Host Controller will automatically retry all Direct Read CCCs, regardless of the Command Code, if it receives a NACK on the first attempt. This will happen regardless of the value of field **DEV_NACK_RETRY_CNT** in the DAT entry for the addressed I3C Target.

Q16.4 Does the Host Controller check to see if any CCCs that require Defining Bytes are always indicated with a Defining Byte in the Transfer Command?

No. It is the Driver's responsibility to ensure that Transfer Commands are constructed correctly for all CCCs (both Direct and Broadcast). In some cases, a particular CCC is always required to be sent with a Defining Byte; this is the case for some new CCCs that were added in I3C v1.1 and later and might be the case for some vendor-defined CCCs. If so, then the Driver must indicate the Defining Byte and set the appropriate fields in the Transfer Command.

Q16.5 Does a Host Controller support the use of Vendor/Standard Extension CCCs in Transfer Commands?

Yes. The Command Code field in a Transfer Command can use any valid value, including those that are available for Vendor/Standard Extension CCCs or those that have been assigned to different MIPI WGs.

Q16.6 Can the Driver mix Transfer Commands that are CCCs with Private Read/Write Transfer Commands?

Yes. The Driver can enqueue any Transfer Commands that form a sequence of regular Read/Write transfers and Broadcast/Direct CCCs in any order that is valid for the Target's use case. If all such Transfer Commands (except the last one) use value 0 in field **TOC**, then the Host Controller will execute these Transfer Commands in a continuous sequence (i.e., using Repeated START between transfers) and drive the CCC framing appropriately. As part of the Managed CCC Transfer Framing Model, the Host Controller will automatically enter the CCC framing when it executes a Transfer Command that is a CCC and then exit the CCC framing when it executes a Transfer Command that is a Private Write or Private Read transfer.

Q16.7 What is the new Command Code value 0x1F for CCCs, and how should it be used?

In I3C v1.1.1 and I3C Basic v1.1.1, a new dummy command code value 0x1F is defined for special use only in CCC flows for HDR Modes that require special structured protocol elements (i.e., Words or Blocks) to conform to that HDR Mode's coding. This 0x1F dummy Command Code has no meaning as a standard CCC, as it is only used in special flows. Since this version of the I3C HCI Specification does not support CCCs in HDR Modes, the Driver should never use Command Code value 0x1F for any Transfer Commands that are CCCs.

Note:

In general, other Host Controller implementations that support CCCs in HDR Modes should drive this dummy Command Code automatically as needed to handle the CCC framing in a manner equivalent to the Managed CCC Transfer Framing Model defined in the I3C HCI Specification. If such a capability is added to a future version of the I3C HCI Specification, then the MIPI Software WG will aim to handle the CCC framing automatically for HDR Modes, as it currently does for SDR Mode.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

2.17 Resets and Error Handling

Q17.1 How does the Host Controller handle the defined I3C Error Types for I3C Controller Devices?

For Error Type CE0, the Host Controller will report the transfer status in the Response Descriptor. It is the Driver's responsibility to determine whether a Target's response to a CCC is an error (e.g., an insufficient number of data bytes returned during a Direct Read CCC).

If the Host Controller supports the optional Error Type CE1, then the Host Controller should detect when the transmitted data is different from what it intended to transmit (e.g., another I3C Device was driving SDA at the wrong time). If this occurs, then the Host Controller should abort the transfer and indicate this in the Response Descriptor with a value of 0x9 (**BUS_ABORTED**) in field **ERR_STATUS**. The Host Controller should also report the error via the appropriate abort register.

For Error Type CE2, the Host Controller will automatically end the transfer with a STOP, then drive the HDR Exit Pattern to recover the Bus from a situation where there is no ACK of the Broadcast Address.

If the Host Controller supports optional Standby Controller mode and detects Error Type CE3 when passing the Controller Role to the indicated Secondary Controller, then the Host Controller will automatically regain the Controller Role and report the error in field **ACR_HANDOFF_ERR_M3_STAT** of register **STBY_CR_INTR_STATUS** (see ***[MIPI12] Section 7.7.11.7***).

Q17.2 How can the Driver initiate a recovery procedure for a stuck SDA lane?

The Driver should send the special Recovery Reset Command Descriptor, which is a subtype of the Internal Control Command, per ***[MIPI12] Section 6.15.2.1***.

Q17.3 How can the Driver initiate a recovery procedure that requires either a STOP condition or an HDR Exit Pattern?

The Host Controller will automatically drive the STOP condition and/or HDR Exit Pattern at appropriate times, including when it detects Error Type CE2. The Driver may also drive this manually using the special Recovery Reset Command Descriptor, which is a subtype of the Internal Control Command, per ***[MIPI12] Section 6.15.2.2*** and ***Section 6.15.2.3***.

Q17.4 Does the Host Controller automatically use the GETSTATUS CCC in cases where a Target does not respond to a transfer or CCC?

No. It is the Driver's responsibility to send a Transfer Command with the **GETSTATUS** CCC to such a Target. The Driver should monitor incoming Response Descriptors for a NACK or other error and drive the **GETSTATUS** CCC when appropriate.

Q17.5 How can the Driver initiate a Target Reset action?

The Driver should send the Target Reset Command Descriptor, which is a subtype of the Internal Control Command, per ***[MIPI12] Section 6.15.1***.

If the Driver needs to send only the Target Reset Pattern (i.e., without any preceding **RSTACT** CCCs), then the Driver will need to enqueue one or more Target Reset Command Descriptors (see ***[MIPI12] Section 6.15.1.1***).

If the Driver needs to configure a specific reset action (i.e., when using the **RSTACT** CCC before the Target Reset Pattern), then the Driver will need to construct an atomic reset sequence starting with the Target Reset Command Descriptor to enter the critical section (see ***[MIPI12] Section 6.15.1.2***). This is followed by one or more Transfer Commands that are **RSTACT** CCCs (either Broadcast or Direct, in the appropriate sequence) and another Target Reset Command Descriptor to leave the critical section. The Host Controller will automatically send the Target Reset Pattern after the last such Transfer Command (i.e., with field **TOC** set to 1'b1).

Q17.6 What is a Command Sequence stall or timeout, and how does the Host Controller handle this situation?

The I3C TCRI Specification (***[MIPI06] Section 6.4.2***) defines a Command Sequence as a situation where the Driver has enqueued several Command Descriptors that are Transfer Commands, all of which have field **TOC** set to 1'b0 except the last Transfer Command. Typically, such Transfer Commands would indicate that the I3C transactions are to be run in a continuous sequence with either a Repeated START or an HDR Restart Pattern between each one, and terminate after the last Transfer Command with field **TOC** set to 1'b1 at the end of the Command Sequence. If the Driver could enqueue the last such Transfer Command promptly, the Host Controller would know the end of the Command Sequence as it processed the Transfer Commands in the sequence and understand that it was supposed to end the transfer with either a STOP (in SDR Mode) or an HDR Exit Pattern (in HDR Modes).

However, if the Driver could not enqueue the last such Transfer Command in time, and the last enqueued Transfer Command had field **TOC** set to 1'b0, then the Host Controller's Command Queue/Ring would effectively run out of Transfer Commands to process but not know which action to take after the last Transfer Command. If this occurs, the I3C Bus Controller logic would be forced to either (a) stall processing, if possible, until the Driver enqueues the last such Transfer Command; or (b) time-out and forcibly end processing by terminating the transaction to prevent an invalid condition on the I3C Bus. Similarly, if the Response Queue/Ring becomes full, then the Host Controller could not enqueue new Response Descriptors, and it would either stall or time-out.

Under typical usage, the Host Controller should never encounter a Command Sequence stall or timeout, as the Driver and its Host could manage the Command Queue/Ring and Response Queue/Ring with appropriate latency and would not allow either situation to occur. However, this situation could occur if the system had high Host latency or the Driver did not optimally configure the Host Controller interrupts.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

3 Terminology

See also ***Section 2*** in the MIPI I3C Specification ***[MIPI01][MIPI07][MIPI08]***.

3.1 Definitions

Controller: The I3C Bus Device controlling the Bus. (I3C and I3C Basic versions prior to v1.1.1 used the deprecated term Master.)

Hot-Join: Targets that join the I3C Bus after it is already started, either because they were not powered previously or because they were physically inserted into the Bus. The Hot-Join mechanism allows the Target to notify the Controller that it is ready to get a Dynamic Address.

In-Band Interrupt (IBI): A method whereby a Target Device emits its Address into the arbitrated Address header on the I3C Bus to notify the Controller of an interrupt.

Master: Deprecated term used in I3C HCI versions prior to 1.1 and in I3C and I3C Basic versions prior to v1.1.1. See Controller.

Primary Controller: Controller-capable Device that has initial control of the I3C Bus. Formerly called "Main Master".

Slave: Deprecated term used in I3C HCI versions prior to 1.1 and in I3C and I3C Basic versions prior to v1.1.1. See Target.

Target: An I3C Target Device can only respond to Common or individual commands from a Controller. (I3C and I3C Basic versions prior to v1.1.1 used the deprecated term Slave.)

3.2 Abbreviations

ACK	Short for "acknowledge" (an I3C Bus operation)
DisCo	Discovery and Configuration (family of MIPI Alliance interface specifications)
e.g.	For example (Latin: exempli gratia)
i.e.	That is (Latin: id est)
NACK	Short for "not acknowledge" (an I3C Bus operation)

3.3 Style Conventions

The following document styles are used for various constants or entities that are defined in the referenced MIPI Specifications:

- **CCCs** that are defined in the I3C Specification
- **BITFIELDS** for CCCs that are defined in the I3C Specification
- **REGISTERS** that are defined in the I3C HCI Specification
- Register **FIELDS** that are defined in the I3C HCI Specification

3.4 Acronyms

See also the acronyms defined in the MIPI I3C HCI Specification and the MIPI I3C Specification.

ACPI	Advanced Configuration and Power Interface
APU	Application Processing Unit
ASIC	Application Specific Integrated Circuit
BCR	Bus Characteristics Register
CCC	Common Command Code (an I3C common command or its unique code number)
CPU	Central Processing Unit
CTS	Conformance Test Suite
DAT	Device Address Table
DCR	Device Characteristics Register
DCT	Device Characteristic Table
DMA	Direct Memory Access
FAQ	Frequently Asked Questions
FPGA	Field Programmable Gate Array
HCI	Host Controller Interface (a MIPI Alliance interface specification ***[MIPI02][MIPI09][MIPI12]***)
HDR	High Data Rate (a set of I3C Bus Modes)
HDR-BT	HDR Bulk Transfer (an I3C Bus Mode)
HDR-DDR	HDR Double Data Rate (an I3C Bus Mode)
I3C	Improved Inter Integrated Circuit (a MIPI Alliance interface specification ***[MIPI01][MIPI07][MIPI08][MIPI10][MIPI11]***)
IBI	In-Band Interrupt (an I3C Bus feature)
MC	Micro-controllers
MDB	Mandatory Data Byte
PID	Provisioned ID (a 48-bit unique ID for each I3C Target on the Bus)
PIO	Programmable I/O
SCL	Serial Clock (an I3C Bus line)
SDA	Serial Data (an I3C Bus line)
SDR	Single Data Rate (an I3C Bus Mode)
SPI	Serial Peripheral Interface (an interface specification)
SRAM	Static Random Access Memory
TCRI	Transfer Command/Response Interface (a MIPI Alliance interface specification ***[MIPI06]***)

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition

4 References

[MIPI01] *MIPI Alliance Specification for I3C® (Improved Inter Integrated Circuit)*, version 1.0, MIPI Alliance, Inc., 23 December 2016 (Adopted 31 December 2016).

[MIPI02] *MIPI Alliance Specification for I3C Host Controller Interface (I3C HCI^{SM})*, version 1.0, MIPI Alliance, Inc., 29 September 2017 (Adopted 4 April 2018).

[MIPI03] *MIPI Alliance Specification for Discovery and Configuration (DisCo^{SM})*, version 1.0, MIPI Alliance, Inc., 1 July 2016 (Adopted 28 December 2016).

[MIPI04] *MIPI Alliance DisCo^{SM} Specification for I3C^{SM}*, version 1.1, MIPI Alliance, Inc., 26 September 2022 (Adopted 28 February 2023).

[MIPI05] *MIPI Alliance I3C Application Note: General Topics*, App Note version 1.1, MIPI Alliance, Inc., 27 April 2022 (approved 27 July 2022).

[MIPI06] *MIPI Alliance Specification for I3C Transfer Command Response Interface (I3C TCRI^{SM})*, version 1.0, MIPI Alliance, Inc., 24 May 2022 (Adopted 7 September 2022).

[MIPI07] *MIPI Alliance Specification for I3C Basic^{SM} (Improved Inter Integrated Circuit)*, version 1.0, MIPI Alliance, Inc., 19 July 2018 (Adopted 8 October 2018).

[MIPI08] *MIPI Alliance Specification for I3C® (Improved Inter Integrated Circuit)*, version 1.1, MIPI Alliance, Inc., 27 November 2019 (Adopted 11 December 2019).

[MIPI09] *MIPI Alliance Specification for I3C Host Controller Interface (I3C HCI^{SM})*, version 1.1, MIPI Alliance, Inc., 20 May 2021 (Adopted 20 May 2021).

[MIPI10] *MIPI Alliance Specification for I3C® (Improved Inter Integrated Circuit)*, version 1.1.1, MIPI Alliance, Inc., 11 June 2021 (Adopted 8 June 2021).

[MIPI11] *MIPI Alliance Specification for I3C Basic^{SM} (Improved Inter Integrated Circuit)*, version 1.1.1, MIPI Alliance, Inc., 9 June 2021 (Adopted 21 July 2021).

Note:

Version number v1.1 was not used for I3C Basic.

[MIPI12] *MIPI Alliance Specification for I3C Host Controller Interface (I3C HCI^{SM})*, version 1.2, MIPI Alliance, Inc., 15 February 2023 (Adopted 12 April 2023).

[LINX01] Linux Kernel Patches for I3C subsystem, <https://patchwork.kernel.org/project/linux-i3c/list/>, last accessed 11 August 2023.

This page intentionally left blank.

Copyright © 2023 MIPI Alliance, Inc.
Public Release Edition