Frequently Asked Questions (FAQ) for MIPI I3C® v1.1 and MIPI I3C Basic^{SM} v1.0

FAQ Version 1.0
25 March 2020

MIPI Board Approved 25 March 2020

Public Release Edition

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

1 Introduction

This FAQ has been developed to introduce the MIPI I3C ***[MIPI01][MIPI12]*** and I3C Basic ***[MIPI10]*** Specifications to developers and users. The I3C WG has compiled these frequently asked questions (FAQs) to assist Member implementation activity. Some areas also include clarification when an area of the Specification was ambiguous, and this FAQ will show the intended resolution of the ambiguity. Many of these topics reflect areas of planned improvements for a future update to I3C v1.0 and I3C Basic v1.0, showing how a v1.0 implementation can be better prepared and/or compliant in advance.

For I3C v1.1 ***[MIPI12]*** new FAQs have been added, and the existing FAQs have been updated as needed.

Throughout this FAQ document, unless otherwise noted, the terms 'MIPI I3C' and 'I3C' refer to both MIPI I3C ***[MIPI01][MIPI12]*** and MIPI I3C Basic ***[MIPI10]***.

Note:

None of the answers in this FAQ are intended to overwrite or overrule the information in the I3C Specification itself ***[MIPI01][MIPI10][MIPI12]***.

The FAQ questions are organized into Sections by topic, based on the reader's level of familiarity with the I3C Specification and implementation:

Section	Title	Focus
2.1	**Introduction to MIPI I3C**	I've heard about I3C. Where can I read a bit more about it?
2.2	**MIPI I3C Specification**	I've started to read the I3C Specification. Tell me a bit more about the frequently asked questions, so I can better understand some of the details in the Specification.
2.3	**Implementation: Ecosystem**	Questions related to design kits, IP, test, and other parts of the enablement ecosystem.
2.4	**Implementation: As a System Designer**	Questions asked by early system designers.
2.5	**Implementation: As a Software Developer**	Questions asked by early software developers.
2.6	**Interoperability Workshops**	Questions asked by early Interoperability Workshop participants.
2.7	**Up and Coming**	Questions related to the next revision (and/or Errata) of the I3C Specification.
2.8	**Clarification, Disambiguation, and Allowances**	Questions related to ambiguous text in the I3C v1.0, I3C v1.1, and I3C Basic Specifications.
2.9	**Conformance Testing**	Questions related to testing device conformance to the I3C Specification.
2.10	**Legal & Intellectual Property Related Questions**	Questions related to legal and IPR aspects of the I3C and I3C Basic Specifications and implementations.

2 Frequently Asked Questions

This FAQ is organized into several topics:

- ***Section 2.1: Introduction to MIPI I3C***
- ***Section 2.2: MIPI I3C Specification***
- ***Section 2.3: Implementation: Ecosystem***
- ***Section 2.4: Implementation: As a System Designer***
- ***Section 2.5: Implementation: As a Software Developer***
- ***Section 2.6: Interoperability Workshops***
- ***Section 2.7: Up and Coming***
- ***Section 2.8: Clarification, Disambiguation, and Allowances***
- ***Section 2.9: Conformance Testing***
- ***Section 2.10:Legal & Intellectual Property Related Questions***

Within each topic, FAQs addressing v1.0 appear first, followed by FAQs addressing I3C v1.1 specifically.

2.1 Introduction to MIPI I3C

I3C v1.0 and I3C Basic v1.0 FAQs

Q1.1 What is MIPI I3C and I3C Basic?

MIPI I3C is a serial communication interface specification that improves upon the features, performance, and power use of I^{2}C, while maintaining backward compatibility for most devices.

MIPI I3C Basic is technically identical to MIPI I3C, except with a reduced feature set and RAND-Z licensing (see ***Section 2.10***).

Q1.2 What does the I3C acronym mean?

The official name is *MIPI Alliance Improved Inter Integrated Circuit*.

Q1.3 Why is MIPI I3C being introduced?

The main purpose of MIPI I3C is threefold:

1. To standardize sensor communication,
2. To reduce the number of physical pins used in sensor system integration, and
3. To support low power, high speed, and other critical features that are currently covered by I^{2}C and SPI.

MIPI I3C's purpose is now widening to cover many types of devices currently using I^{2}C/SMbus, SPI, and UART.

Q1.4 What are the main features of MIPI I3C?

MIPI I3C carries the advantages of I^{2}C 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 then adds higher throughput for a given frequency, in-band interrupts (from Slave to Master), dynamic addressing, advanced power management, and hot-join.

Q1.5 How many signal lines does I3C have?

I3C has two signal lines: Data (SDA) and Clock (SCL).

Q1.6 Does I3C require pull-up resistors on the bus like I^{2}C?

No, I3C Masters control an active pull-up resistance on SDA, which they can enable and disable. This may be a board-level resistor controlled by a pin, or it may be internal to the Master.

Q1.7 Who is I3C intended for?

I3C was initially intended for mobile applications as a single interface that can be used for all digitally-interfaced sensors. However it is now intended for all mid-speed embedded and deeply-embedded applications across sensors, actuators, power regulators, MCUs, FPGAs, etc. The interface is also useful for other applications, as it offers high-speed data transfer at very low power levels while allowing multi-drop, which is highly desirable for any embedded system.

Q1.8 Why replace I^{2}C with I3C?

While I^{2}C has seen wide adoption over the years, it lacks some critical features – especially as mobile and mobile-influenced systems continue to integrate more and more sensors and other components. I^{2}C limitations worth mentioning include: 7-bit fixed address (no virtual addressing), no in-band interrupt (requires additional wires/pins), limited data rate, and the ability of Slaves to stretch the clock (thus potentially hanging up the system, etc.). I3C aims both to fix these limitations, and to add other enhancements.

Q1.9 Does I3C use less power than I^{2}C?

The power consumption per bit-transfer in all I3C modes is more efficient than I^{2}C, due to the use of push-pull (vs. open-drain) and strong pull-up signaling.

Further, I3C can save considerable device power through higher data rates (because the device can be put back to sleep sooner), built-in configuration and control (without intruding on the main communication protocols), in-band interrupt (IBI) as a low-cost wake mechanism, and the ability for Slaves to shut down all internal clocks while still operating correctly on the I3C Bus.

Q1.10 Is anyone currently using I3C?

Yes, a number of companies have released products which feature integrated I3C Master and I3C Slave support. Other companies offer IP blocks and associated verification software for adding I3C Bus support into various integrated circuit designs. Some companies also offer protocol analyzers and verification hardware to help analyze I3C Bus traffic for testing and development.

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

Q1.11 How is I3C different from I^{2}C?

I3C offers dynamic address assignment, Slave-initiated communication, and significantly higher communication speeds than I^{2}C.

Q1.12 Is I3C backward compatible with I^{2}C?

Yes, most I^{2}C Slave devices can be operated on an I3C Bus, as long as they have a 50 ns glitch filter and do not attempt to stall the clock. Such use will not degrade the speed of communications to I3C Slaves; it will require decreased speed only when communicating with the I^{2}C Slaves.

I3C Slave Devices with a Static Address can operate as I^{2}C Slaves on an I^{2}C bus; optionally, they can also have a 50 ns spike filter.

Q1.13 What is the maximum number of I3C Devices that can be connected on the same Bus?

The I3C Bus is limited to around a dozen devices.

Q1.14 Can there be more than one I3C Slave inside a chip?

Yes, multi-Slave I3C chips are possible.

Q1.15 Can I3C and I^{2}C co-exist on the same bus?

Yes, both I3C and I^{2}C can share the same bus. For details, see ***Question Q1.12*** regarding I3C backward compatibility.

Q1.16 What is the bit rate for I3C?

I3C has several Modes, each with associated bit rate(s). The base raw bitrate is 12.5 Mbps, with 11 Mbps real data rate at 12.5 MHz clock frequency (this is the only Mode supported in I3C v1.0 and I3C Basic v1.0). The maximum raw bitrate is 33.3 Mbps at 12.5 Mhz, with real data rate of 30 Mbps; this is achieved via HDR Modes that are currently only available in I3C v1.0.

Most traffic will use the 10 to 11 Mbps rate, while large messages can use one of the higher data rate Modes.

Q1.17 Can I3C Slaves initiate communication (i.e., interrupt the Master)?

Yes, I3C Slaves can initiate communication. Communication conflicts are solved by Slave Address Arbitration.

Q1.18 Is it possible to have multiple Masters on the same I3C Bus?

Yes, I3C allows for multiple Masters on the same Bus. I3C has one Main Master that initially configures the Bus and acts as the initial Current Master. Optionally, the Bus can have one or more Secondary Master Devices that initially act as Slaves. Any Secondary Master Device can request to become the Current Master. Once the Current Master agrees to the request, and transfers Current Master control ('Mastership') to a given Secondary Master Device, then that Device becomes the Current Master.

Q1.19 How can Masters and Slaves communicate on the I3C Bus?

The basic byte-based messaging schemes of I²C and SPI map easily onto I3C. Additionally, a set of common command codes (CCCs) has been defined for standard operations like enabling and disabling events, managing I3C-specific features (e.g., Dynamic Addressing, Timing Control, etc.), and other functions. CCCs can be either Broadcasted (i.e., sent to all Devices on the I3C Bus), or else Directed at a particular Device on the I3C Bus.

CCCs do not interfere with, and do not consume any of the message space of, normal Master-to-Slave communications. That is, I3C provides a separate namespace for CCCs.

Q1.20 What are CCCs (Common Command Codes) and why are they used?

The CCCs are the commands that an I3C Master uses to communicate to some or all of the Slaves on the I3C Bus. The CCCs are sent to the I3C Broadcast address (which is 7'h7E) so as not to interfere with normal messages sent to a Slave. In other words, CCCs are separated from the standard "content protocol" used by normal messages, such as Private Write and Read transfers (in SDR Mode).

The CCCs are used for standard operations like enabling/disabling events, managing I3C-specific features, and other Bus operations. CCCs can be either Broadcasted (i.e., sent to all Devices on the I3C Bus), or else Directed at specific Devices on the I3C Bus. All CCCs (i.e., the command numbers themselves) are allocated by MIPI Alliance, and some are reserved for specific purposes including MIPI Alliance enhancements and other extensions (see ***Q1.31***).

Q1.21 Why replace SPI (Serial Peripheral Interface) with I3C?

SPI requires four wires and has many different implementations because there is no clearly defined standard. In addition SPI requires one additional chip select (or enable) wire for each additional device on the bus, which quickly becomes cost-prohibitive in terms of number of pins and wires, and power. I3C aims to fix that, as it uses only two wires and is well defined.

I3C covers most of the speed range of SPI, but is not intended for the highest speed grades that really only work well with a point-to-point interface, such as for SPI Flash.

I3C v1.1 FAQs

Q1.22 What is MIPI I3C v1.1?

MIPI I3C v1.1 is an advancement of the MIPI I3C Specification that includes not only clarifying edits to make for a more easily-implemented interface, but also new optional features that make I3C even more attractive to a broader set of use cases and industries.

The new features include:

- HDR-BT (Multi-Lane Bulk Transport)
- Device to Device(s) Tunneling
- Grouped Addressing
- Multi-Lane for Speed
- Slave Reset

Q1.23 What are the "required" features in I3C v1.1 vs. I3C 1.0?

Almost all new features of I3C v1.1 are optional.

However, it should be noted that:

- The formerly optional CCC GETHDRCAPS has been changed to GETCAPS, and its support is required for I3C v1.1 Devices, so as to indicate it is a v1.1 (or later) Device.
- Additionally, it is necessary to support the new RSTACT CCC; as a result, support for a minimal Slave Reset is required.

MIPI recommends that Slaves consider support of improvement features (such as Flow control for HDR, and Group Addressing), as well as features they likely could use for their application.

Q1.24 Are there any I3C v1.0 features that are not supported in I3C v1.1?

The Directed form of the RSTDAA CCC is not supported in I3C v1.1.

Q1.25 What is new in I3C v1.1?

The new features include:

- HDR-BT (Multi-Lane Bulk Transport)
- Device to Device(s) Tunneling
- Grouped Addressing
- Multi-Lane for Speed
- Slave Reset

Additionally, many clarifying edits have been made to the Specification, and the GETHDRCAPS CCC has been replaced with the GETCAPS CCC (which is required for I3C v1.1 Devices).

Q1.26 How does an I3C Slave behave with an I^{2}C Master vs. with an I3C Master?

All I3C Slaves come up in I^{2}C mode, as they do not yet possess a Dynamic Address. If they have an I^{2}C static address, then they may operate on a legacy I^{2}C bus, using that static address. I3C Slaves may also support an I^{2}C 50 ns Spike filter for Fm and Fm+ modes, and they may support other I^{2}C features that are not supported by I3C (such as Device ID). All I3C Masters will emit the first START,7'h7E slowly enough to be seen through an I^{2}C Spike filter, allowing the I3C Slaves to disable the spike filter once they see the first START,7'h7E.

If the I3C Slave does not have an I^{2}C-type static address, then it will simply wait for the START,7'h7E from an I3C Master. Such a Slave would be of no value on a legacy I^{2}C bus, as I^{2}C relies on each slave having a static address.

Note that I3C Slaves should not use I^{2}C Clock stretching unless they know with certainty that they are on a legacy I^{2}C Bus. I3C Masters are never required to support I^{2}C Clock stretching, so the I3C Slave must not use that facility unless it has reason to know it is safe to do so (i.e., that it is on a legacy I^{2}C bus). (Note also that an I3C Master may choose to support I^{2}C Clock stretching when communicating with a legacy I^{2}C Slave, though that is outside of the scope of the I3C Specification.)

Q1.27 Must all I3C Slaves support Dynamic Address Assignment (DAA)?

No. If an I3C Slave will be used only on I3C Buses that rely on the SETAASA CCC (which auto-sets the Slave's Dynamic Address from its I^{2}C Static Address) and/or the SETDASA CCC (where the Master sets the Slave's Dynamic Address using a Direct CCC that references its I^{2}C static address), then that Slave would never be asked to use DAA and could participate on the I3C Bus without implementing DAA. However, MIPI Alliance strongly recommends supporting the DAA (ENTDAA) approach as well, otherwise the Device will only ever be usable on that narrow subset of I3C Buses.

Q1.28 How can an I3C Slave lose its I3C Dynamic Address, and how does it become an I^{2}C Slave again?

Once assigned an I3C Dynamic Address, an I3C Slave normally retains it until the Slave is de-powered. However, an I3C Slave will lose its I3C Dynamic Address through the RSTDAA Broadcast CCC, which resets all I3C Slaves back to their initial I^{2}C state. The RSTDAA Broadcast CCC is not normally used; it would only be used to assign a new Dynamic Address.

An I3C Slave might also lose its Dynamic Address if the Device is reset: for example, pin-reset, full Slave Reset (starting with I3C v1.1), deepest-sleep (power down), etc. In the case of an I3C Device losing its Dynamic Address in non-standard ways, the Hot-Join mechanism is provided to allow the Slave to notify the Master of the event and receive a new Dynamic Address. In cases where the Master has deliberately caused the Slave to lose its Dynamic Address (e.g., by sending the RSTDAA CCC, or by causing a Slave reset), the I3C Master will start a new Dynamic Address Assignment process using the ENTDAA, SETDASA, or SETAASA CCC.

See also Offline Mode for I3C Slaves, where the Devices retain their Dynamic Addresses through a power-down or deepest-sleep cycle.

Q1.29 What are the differences between I3C v1.0 and I3C v1.1 in how CCCs are defined?

CCCs in I3C v1.1 are defined and marked differently than in I3C v1.0, in two important ways:

1. **Conditionally Required CCCs:** In I3C v1.1, ***Table 16*** in ***Section 5.1.9.3*** (i.e., the main table that defines the I3C Common Command Codes) now marks each CCC as either Required ('R'), Conditional ('C'), or Optional ('O'). Required and Optional retain the same meanings they had in I3C v1.0, but in I3C v1.1 some CCCs have been changed to Conditional.

   For Conditional CCCs the Description column includes a note indicating the condition(s) under which the CCC is required ("Required If:"). Typical conditions would be the use of a particular feature, or support for another particular CCC. If the conditions for a given Conditional CCC are not met, then there is no requirement to support that CCC, and support for it can be regarded as Optional.

   Example: The ENTAS0 CCC is marked as Conditional and only "Required If" any of the other ENTASx CCCs (i.e., ENTAS1, ENTAS2, and/or ENTAS3, all of which are Optional) are supported. Of course an implementer may choose to support the ENTAS0 CCC even if none of these other Optional ENTASx CCCs are supported. But if at least one of the other ENTASx CCCs are supported, then support for the ENTAS0 CCC is required for that configuration.

2. **CCCs in HDR Modes:** At the discretion of the implementer, CCCs may also be supported in any supported HDR Modes. ***Section 5.2.1.2*** defines the general concepts of CCC framing while in HDR Modes, and specifies key requirements and details regarding their use within an HDR Mode. In general, the use of CCCs within any HDR Mode provides the option to send certain CCCs efficiently while remaining in HDR Mode (i.e., without the extra overhead of exiting HDR Mode and then returning to SDR Mode). The specific CCC framing details for each supported HDR Mode are listed in ***Table 53***, and defined in the sections specifying each HDR Mode.

Table 51 defines which CCCs may be supported and permitted for use in any HDR Mode, both for Broadcast CCC and Direct CCC flows. Additionally, ***Table 52*** defines which CCCs are prohibited in any HDR Mode, and explains why. Since support for CCCs in HDR Modes is optional, and since an implementer might choose to support a subset of CCCs from ***Table 16*** (i.e., those which are also permitted for use, as per ***Table 51*** and ***Table 52***), it is important for the I3C Master to know which CCCs are supported in HDR Modes. This might include CCCs set aside for Vendor / Standard Extension use, or ones reserved for another MIPI Alliance WG. It is also required that any CCC that is supported in any HDR Mode is supported in SDR Mode too.

Note:

This last point means that if an I3C Device does not acknowledge a given CCC in a given HDR Mode, then the I3C Master can determine whether that CCC is only unsupported in that HDR Mode (vs. is not supported at all) by retrying that same CCC in SDR Mode.

Q1.30 What has changed in CCC use or coding in I3C v1.1?

The coding and framing details for CCCs have been changed or extended, in three important areas:

1. **Direct Write/Read Commands:** In I3C v1.0, Direct CCCs were either Direct Read Commands (Direct GET CCC) or Direct Write Commands (Direct SET CCC). I3C v1.1 adds an additional Direct Write/Read Command (Direct SET/GET CCC) form: a Direct Write immediately followed by a Direct Read with the same Command Code; this form is used for alternatively writing to, and then reading data from, one or more specific I3C Slave Devices. The Direct Write/Read Command uses the Direct CCC framing model per ***Section 5.1.9.2.2***.

   Example: An I3C Master might use the MLANE CCC (***Section 5.1.9.3.30***) to configure an I3C Slave Device to use a specific Multi-Lane configuration using a Direct SET CCC, followed by a Direct GET CCC to read back the active Multi-Lane configuration and confirm that it was selected for operation. Using both forms of Direct SET CCC and Direct GET CCC within the same SDR frame is considered a Direct SET/GET CCC.

2. **Defining Byte for Directed CCCs:** In I3C v1.1, some Directed CCCs add support for a Defining Byte.

   A. In CCC framing for SDR Mode, the coding of the initial CCC is:

      S, 7’h7E, CCC_byte, Defining_Byte, Sr, Slave_Addr, ...

   For more framing details for SDR Mode, see ***Table 15*** in ***Section 5.1.9.1***.

   B. In CCC framing for HDR Modes, the Defining Byte is sent in the HDR-x CCC Header Block of type Indicator. This framing element is defined differently for each HDR Mode; for more framing details, see ***Table 53*** in ***Section 5.2.1.2.1***.

   Depending on the particular CCC, this Defining Byte can be used in several possible ways:

   - It might define a register/code to set, either with or without additional per-Slave info
   - It might select a particular register/code to read, from among several available for that CCC
   - It might indicate one of several completely different sub-commands, each of which might be either required or optional, as needed for the particular CCC definition and use case.

   Example: When used with the MLANE CCC, a Defining Byte selects a sub-command. One sub-command might be used to read the available Multi-Lane capabilities for an I3C Slave Device, as a Direct GET CCC; another sub-command might be used to either configure an I3C Slave Device to use a specific Multi-Lane configuration, or read back the same active Multi-Lane configuration, as either a Direct GET CCC or a Direct SET CCC. For this use case, each Defining Byte has a different purpose and a different, specific data format.

   Example: A Defining Byte for the GETCAPS CCC (see ***Section 5.1.9.3.19***) selects among several different capabilities areas, to read from an I3C Slave Device as a Direct GET CCC. For this use case, the Defining Byte may also indicate different formats for the data message returned by the I3C Slave Device.

For some new Direct CCCs defined in I3C v1.1, a Defining Byte is required.

3. **New Optional Defining Bytes:** In I3C v1.1, some CCCs that had no Defining Byte in I3C v1.0 now have optional Defining Bytes to extend their functionality and support other new behaviors. If the I3C Master sends the CCC without the Defining Byte then the original functionality or behavior occurs, but if the CCC is sent with the Defining Byte then the optional extended functionality or new behavior is accessed (see ***Section 5.1.9.2.2***).

   Many Direct GET CCCs that allow optional Defining Bytes also have a method for indicating whether any additional Defining Bytes are supported. This support would be indicated in the data message returned by the Direct version of a particular CCC (which might be the same CCC) when sent without a Defining Byte. This allows an I3C Master to query an I3C Slave Device to determine whether this capability is supported. The particular CCCs that allow optional Defining Bytes are defined in version 1.1 of the I3C Specification (see ***Section 5.1.9.3***). Additionally, for I3C Slave Devices that support such Direct CCCs and also support any optional Defining Bytes, a Defining Byte value of 0x00 generally results in the same behavior as when no Defining Byte is sent.

   Note:

   While Defining Byte support for such Direct GET CCCs is generally optional, certain Defining Byte values are required or strongly recommended for certain use cases, or when used in conjunction with other features or capabilities. Such Direct CCCs list these conditions or recommendations, in addition to any changes in the format of the data message that might be returned when a Defining Byte is used.

   Example: In I3C v1.1, an I3C Master can use the GETSTATUS CCC (see ***Section 5.1.9.3.15***) to determine the operational status of an I3C Slave Device. All I3C Slaves support the use of GETSTATUS without a Defining Byte. But if an I3C Slave also supports the GETSTATUS CCC with any optional Defining Bytes, then the I3C Master may also do either of the following things:

   - Send the Direct GET GETSTATUS CCC with a Defining Byte of 0x00, to return the same data message as if no Defining Byte were used;
   - Send the Direct GET GETSTATUS CCC with any other Defining Byte value listed in ***Table 24***. Any I3C Slave that supports that particular Defining Byte is required to ACK the CCC and return an appropriate data message for that Defining Byte (i.e., not the standard Slave Status). For example, if a Defining Byte value of 0x91 (“SECMST”) is supported, then using this Defining Byte would request the Slave to return alternate status information related to the Slave’s Secondary Master capabilities.

Q1.31 What are “Vendor / Standard Extension” CCCs, who can use them, and how are they differentiated among different uses?

In general, the ranges of command codes that are defined for Vendor or Standards use in ***Table 16*** in ***Section 5.1.9.3*** (i.e., the main table that defines the I3C Common Command Codes) can be used by any implementer or any standards developing organization to define custom CCCs that extend I3C to accommodate new use cases. However, the definition of custom CCCs should be considered carefully, as the practical issues of implementation might lead to situations where interoperability could be affected, across multiple I3C Slave Devices that interpret the same command code differently.

For example, different implementers or standards groups might define a custom CCC using the same command code (i.e., byte value for the CCC) with different interpretations of the message format that their custom Slave Device should return, for a Direct GET CCC; or different expectations about how their custom Slave Device should act, for a Direct SET CCC; or different expectations about which Defining Bytes might be supported for this CCC, for their custom Slave Device. For these conflicting definitions, interoperability issues would certainly arise if the Master used this custom CCC on an I3C Bus that used custom Slave Devices from multiple implementers, or custom Slave Devices that conformed to different standards published by several such standards groups. The Master would not necessarily have knowledge of this situation until it detected protocol issues or encountered other errors.

To help alleviate this situation, I3C v1.1 adds a new SETBUSCON CCC (see ***Section 5.1.9.3.31***) that allows the Master to set the Bus context. This CCC informs Slaves about which version of I3C is used on the Bus, and whether it is a standards-based Bus, and/or a Vendor-private Bus. This information allows the Slave to coordinate its interpretation of extended CCCs, as well as other uses of the Bus, to match the actual current Bus context. The SETBUSCON feature is fully optional, but provides a powerful way to align standards-group-based uses of I3C with coordinated private uses. To foster proper coordination, SETBUSCON Context Byte values are published on the MIPI Alliance public website ***[MIPI11]***, and may be assigned by request.

MIPI Alliance strongly recommends that standards groups utilize the SETBUSCON CCC to prevent such interoperability issues, by requesting an assigned Context Byte for their particular usage, which might include a specific interpretation of any command codes that are defined for Vendor or Standards use. Additionally, such custom Slave Device implementations should not enable this custom interpretation by default, until they receive the SETBUSCON CCC with an assigned Context Byte from the I3C Master.

MIPI Alliance offers a similar recommendation to implementers seeking to define custom CCCs for private use in a custom Slave Device, by using similar logic in such a Slave implementation to not enable this custom interpretation by default, until receiving the SETBUSCON CCC with a suitable Context Byte from the I3C Master to explicitly enable a private interpretation.

Q1.32 How should custom CCCs be used as part of a content protocol based on I3C?

For most use cases, custom CCCs (including the command codes that are defined for Vendor or Standards use, see ***Q1.31***) should generally be used as configuration or control commands, sent from an I3C Master to one or more I3C Slave Devices. Using custom CCCs for larger data transfers is not recommended.

When considering the practical concerns of implementing support for custom CCCs in an I3C Slave, it is important to note that CCCs in I3C are generally used as shorter messages that are separated from the standard content protocol (i.e., Write and Read transfers in SDR Mode, or any HDR Modes) and are not intended to interfere with normal messages sent to a Slave (see ***Q1.20***).

Additionally, since the Command Code and Defining Byte for CCCs are sent to the I3C Broadcast Address (i.e., 7'h7E) and rely on a special 'modality' that must be observed by all Slaves, handling for custom CCCs might need to be implemented differently within the Slave.

With these considerations in mind, implementers should consider using custom CCCs for special purposes, taking advantage of the CCC flows and their separation from the content protocol, to affect changes to the flow or meaning of transfers that are used in their content protocol. By contrast, transfers within their content protocol (such as Write or Read transfers in SDR Mode, or any HDR Modes) should be used for data-intensive transactions. In most cases, custom CCCs should enable new mechanisms for configuring or controlling a Slave Device, while transfers in their content protocol should be used for data transfers between a Master and a Slave.

The following examples are provided as guidance for custom CCC definitions:

- Configuration commands that switch between various supported formats of index or selection that might be used in a Write command, or the first phase of a two-phase Write+Read command
- Configuration commands that send a directed mode change to particular Slaves, or broadcast mode changes to all Slaves that support a particular capability or feature (if applicable)
  - Note that custom Broadcast CCCs (i.e., for Vendor or Standards use) should be used with caution, as these are received by all Slaves on the I3C Bus, and various Slaves might handle such CCCs differently (i.e., by not using a common interpretation; see ***Q1.31*** for guidance).
- Control commands that switch between multiple endpoints within a Slave, using a multiplex model that chooses how and where a standard Write transfer might be directed and processed, or from where a Slave might source the data message for a Read transfer
  - In this case, the custom CCC acts as a command to the multiplexor logic.
  - However, such a multiplex model means that only one endpoint at a time can be selected for active use, which might present a limitation for the use case, and might restrict the modality for using such a Slave.

- Special messages sent only to the Slave hardware (i.e., the Peripheral logic) whereas the data-intensive transfers in the standard content protocol might be implemented via software or other agents that connect to the Peripheral logic, and would not need to see such special messages
  - In this case, the Peripheral logic would be expected to offer a quick response due to CCC framing, so a shorter CCC message would offer rapid response due to the expectations based on capabilities in Peripheral logic, versus waiting for software or other agents to respond, which might lead to delays.
  - By contrast, an implementation that used Peripheral logic with software to respond to Direct GET CCCs might not be capable of successfully responding to the first such attempt, and would rely on ideal timing conditions to successfully respond to subsequent attempts.
  - Note that this might only apply to implementations that relied on software, versus implementations that handled custom CCCs natively (i.e., entirely within hardware).
- Commands that change modes to initialize new functions or enable a new modality, which might change the interpretation of standard transfers for the content protocol (e.g., firmware download, key exchange)
  - For example, a custom CCC might act as a method for entering or exiting the modality in which the standard transfers are applied to a new purpose (such as transferring new firmware contents or key data). Upon exiting the modality, the new function of the Slave would be applied based on the data transferred during the modality, and the standard content protocol would be used for subsequent operations with the new function.

For other use cases that do not generally follow these guidelines, or use cases that rely on larger message lengths for data-intensive transactions, a content protocol that primarily uses standard transfer commands (i.e., not CCCs) is strongly recommended. Using custom CCCs for data-intensive transactions on the I3C Bus might cause implementation issues for various Slave Devices that are based on Peripheral logic and use "software" to handle the response for custom Direct GET CCCs. Additionally, using custom CCCs might introduce integration issues or other platform power concerns that might only appear on I3C Buses with other Slave Devices which would not have been fully optimized to ignore such custom CCCs, or with other Slave Devices that relied on different interpretations of custom CCCs and might not understand a particular use of SETBUSCON for the use case (as defined in ***Q1.31***). For such situations, it might not be possible to predict these issues in advance, until full system integration testing is performed.

Note that higher-level use cases could use a mix of Write/Read transfers for the content protocol, along with custom CCCs for targeted configuration and control functions, which would take advantage of the strengths of CCCs as well as the ways in which CCCs are well-suited for effective changes to control, modes or operational parameters of an I3C Slave Device. For some specific aspects custom CCCs might be recommended, whereas other use cases involving multiple endpoints might be better served with Virtual Slave capabilities (see ***Q1.34*** for more details) if such endpoints would need to be used simultaneously.

Implementers seeking more specific guidance or recommendations should contact the I3C WG within MIPI Alliance for assistance.

Q1.33 What is Offline, and what does it mean to be Offline Capable?

The Offline capability, which is now indicated in the Bus Control Register (BCR), allows a Slave to become inactive on the I3C Bus at some times, but then return to normal activity later.

There are two basic types of Offline-capable Slave:

1. A Slave that is fully inactive on the I3C Bus (e.g., is powered off) and only becomes active again as the result of some external event. Such a Slave will then Hot-Join to get a new Dynamic Address.
2. A Slave that is inactive on the I3C Bus, but where some portion of the Slave is monitoring the Bus either for Slave Reset, or for the use of its Dynamic Address. The Slave may be mostly powered-off, or in deepest sleep, but the monitoring portion retains the Slave's Dynamic Address.

   These Slaves can be awakened (i.e., can be re-activated) by a Slave Reset or by the use of their Dynamic Address. They will take some time to become active on the Bus again, such as the RSTACT CCC recovery from Full Reset time. During the time that they are offline, and while awakening, they will not be responsive to the Master, nor will they record CCCs, nor will they necessarily retain state (e.g., the ENEC/DISEC CCCs), so the Master will have to wait for them to become active, and then might also need to configure them again. This is all by private contract (agreement).

See details in ***Section 5.1.10.2.5*** for both I3C v1.0 (without Slave Reset) and I3C v1.1 (with Slave Reset).

Q1.34 What is a Virtual Slave?

Generally, a Virtual Slave is a function of a single physical Device that represents multiple I3C Slaves on the I3C Bus, such that the I3C Master can address each of those Slaves independently.

In the simplest form a Virtual Slave could be one of a set of several Slave Devices all integrated into the same physical package, sharing a common set of pins connecting them to an I3C Bus.

In a more advanced form, a Virtual Slave could act as one of several virtualized functions presented by a highly-integrated Device that stores a different Dynamic Address for each function. Depending on the implementation, such virtualized functions might share configuration information, and might return the same values for some CCCs.

Examples could include Bridging or Routing Devices, as well as other types of Devices that expose multiple functions and use shared Peripheral logic. I3C v1.1 defines several new capabilities and features for such Virtual Slaves.

See the I3C v1.1 Specification at ***Section 5.1.9.3.19***, and the ***System Integrators Application Note for I3C [MIPI05]*** at ***Section 5.7***.

Q1.35 Does the I3C v1.1 Bus enable 'Routing'?

Yes. I3C v1.1 defines requirements, expectations, and configuration for Routing Devices, which enable the creation of multiple Routes across I3C Buses. A Routing Device enables more advanced Bus topologies and requires buffers or queues to handle transactions across each Route.

The Routing Device contains a Control Function, presented on the I3C Bus as a Virtual Slave. The Master configures the Routing Device by sending the SETROUTE CCC to the Control Function. Routes to other I3C Buses are treated as downstream targets, each of which generally has a Target Function which is also presented as a Virtual Slave with its own Dynamic Address. Transactions are sent to the Route's Target Function via its Dynamic Address, and the Routing Device manages the communications on the downstream I3C Bus.

See the I3C v1.1 Specification at ***Section 5.1.9.3.20***, and the ***System Integrators Application Note for I3C [MIPI05]*** at ***Section 5.7***.

Q1.36 How are the following similar and/or different: In-Band Interrupt, Hot-Join, and Mastership Request (IBI / HJ / MR)?

All three are special, in-band methods that allow the Slave to notify the Master of a new request or state, without having to wait for the Master to query or poll the Slave(s). The term ‘in-band’ refers to doing this via the I3C Bus wires/pins themselves, rather than using methods requiring extra wires/pins.

- **In-Band Interrupt (IBI):** An IBI request is used by a Slave to notify the Master of a new state or event. If the Slave so indicates in the BCR, then an IBI may include one or more following data bytes. A Slave can only use IBI if it has indicated the intent to do in the BCR.

  If the Slave indicates it will send data with an IBI, then it is required to always send at least 1 byte, called the Mandatory Data Byte (MDB). Starting with I3C v1.1, the MDB is coded following certain rules. Any data after the MDB is a contract between Master and Slave.
- **Hot-Join (HJ):** An HJ request is used only by an I3C Slave that hasn’t yet been assigned a Dynamic Address and is attached or awakened on the I3C Bus after the Main Master has intialized it. HJ uses a fixed address reserved for this purpose only. The Master will recognize this fixed address and then initiate a new Dynamic Address Assignment procedure. Note that the HJ request cannot be used by a Slave until it has verified that the I3C Bus is in SDR Mode, as explained in the specification.
- **Mastership Request (MR):** A Secondary Master (including the Main Master, once it has given up initial Mastership) sends an MR when it wants to become Master of the I3C Bus. If the Current Master accepts the MR, then it will issue a GETACCMST CCC to hand control off to the requesting Secondary Master. It is also possible for the Current Master to initiate mastership handoff on its own, without any Slave initiating an MR, for example when a Seconary Master wants to return control.

Q1.37 Why does I3C allow more than one Master on the I3C Bus? What can a Secondary Master do that the Main Master can’t?

The system designer decides whether their system needs more than one Master on the Bus. To provide that flexibility MIPI I3C allows this, but also does not require it. Master handoff is a well defined and controlled mechanism in I3C; if used, it can be relied on.

For most use cases, a Secondary Master is not a required component for an I3C Bus. If your Main Master has all the capabilities and features that you need, and if your use of the I3C Bus wouldn’t benefit from having multiple Master-capable Devices, then you might not need a Secondary Master.

Examples where Secondary Masters are useful:

1. Debug controller, if present, would be a Secondary Master
2. A sensor hub or other offload device can be used to continue operating during periods when the Host processor is in deep sleep (i.e., to save power)
3. The system can switch between standalone use (such as IoT devices) and connected uses (perhaps a USB-to-I3C cable is attached to take over temporarily)

Note that Devices such as MCUs will usually be able to operate as fully-fledged Slaves, as fully-fledged Main Masters, and as Secondary Masters (i.e., Devices that come up as Slaves but can become Masters), depending solely on the particular needs of the given system. They can simply be configured for the Bus they are on by the firmware.

2.2 MIPI I3C Specification

I3C v1.0 and I3C Basic v1.0 FAQs

Q2.1 How can the MIPI I3C v1.0 Specifications be obtained?

- **MIPI I3C Basic v1.0:** MIPI Alliance made the ***MIPI I3C Basic v1.0 Specification [MIPI10]*** publicly available for download in December 2018. Non-members may download a copyright-only version of the I3C Basic specification by visiting the MIPI I3C Basic page on the MIPI Alliance website: https://www.mipi.org/specifications/i3c-sensor-specification. MIPI Alliance members have access and rights to the I3C Basic specification through their MIPI membership and member website.
- **MIPI I3C v1.0:** MIPI Alliance members have access and rights to the ***MIPI I3C v1.0 Specification [MIPI01]*** through their MIPI membership and member website.

Q2.2 Does I3C support inclusion of I^2C devices on the bus, and at what speed?

I3C supports legacy I^2C devices using Fast-mode (Fm, 400 KHz) and FastMode+ (Fm+, 1 MHz) with the 50 ns spike filter, but not the other I^2C modes, and not devices lacking the spike filter, or that stretch the clock.

Q2.3 When is the pull-up resistor enabled?

Much of the activity on the I3C Bus is in push-pull mode (that is, with the pull-up resistor disabled) in order to achieve higher data rate. However for some Bus management activities, and for backwards compatibility with I^2C, pull-up-resistor-based open-drain mode is enabled. For example: arbitration during Dynamic Address Assignment, and In-Band Interrupt. Also, the ACK/NACK during the 9th bit is done using a pull-up resistor. With few exceptions, it is the responsibility of the I3C Master to provide an open-drain class pull-up resistor when the Bus is in the open drain mode.

Q2.4 Is a High-Keeper needed for the I3C Bus?

A high-keeper is used for Master-to-Slave and Slave-to-Master bus hand-off, as well as optionally when the Bus is idle. The high-keeper may be a passive weak pull-up resistor on the Bus, or an active weak pull-up or equivalent in the Master. The high-keeper only has to be strong enough to prevent system-leakage from pulling the Bus low. At the same time, the high-keeper has to be weak enough that a Slave with a normal I_{OL} driver is able to pull the Bus line low within the minimum period.

Q2.5 What is a Provisional ID, and why is it needed?

During bus initialization, the I3C Master assigns a 7-bit Dynamic Address to each Device on the I3C Bus. For this to happen, each Slave device must have a 48-bit Provisional ID (that is, provisioned with its ID). The Provisional ID has multiple fields, including MIPI Manufacturer ID and a vendor-defined part number. The I3C Slave may also have a static address, which if the Master knows it, allows for faster assignment of the Dynamic Address.

Q2.6 How do the first 32 bits of the Provisional ID work? Are they random or fixed?

The first part of the ID contains a unique manufacturer ID. Companies do not have to be MIPI Alliance members to be assigned a unique manufacturer ID.

The second part of the ID normally contains a part number (which is normally divided up into general and specific part info for that vendor), as well as possibly an instance number which allows for multiple instances of the same device on the same I3C Bus. The instance ID is usually fed from a pin-strap, or fuse(s), or non-volatile memory (NVM).

A random number may be used for the part number, although normally only for test mode, as set by the Master using the ENTTM (Enter Test Mode) CCC. When a Device that supports random values enters the test mode, the PID[31:0] bits are randomized. When the Master exits the test mode, the Devices reset bits PID[31:0] to their default value. The use of a random number allows for many instances of the same Device to be attached to a gang programmer/tester, relying on the random number to uniquely give each a Dynamic Address.

Q2.7 What if the Master detects a collision during Dynamic Address Assignment (DAA)?

With most configurations, this is not possible; each Device will have its own manufacturer ID and part number, so no collisions are possible. If more than one instance of the same Device is used on the same I3C Bus, then each instance must have a separate instance ID; otherwise there would be a collision. Likewise, if any Device is using a random number for its part number, then multiple instances from that manufacturer could collide (i.e., could have the same random value that time).

If the Master knows the number of Devices on the I3C Bus, then it can detect this condition: the number of Dynamic Addresses assigned would be less than the number of Devices. Once detected, the I3C Master can take steps to resolve such collisions, for example by resetting DAA and restarting the process, or by declaring a system error after a set maximum number (e.g., 3) of such attempts fail.

Q2.8 What CCCs must my Slave support before a Dynamic Address is assigned?

All I3C Devices must be able to process Broadcast CCCs at any time, whether or not they have been assigned a Dynamic Address (DA). For example, an I3C Device may act as an I²C device before it gets its DA assigned. However, it is still expected to ACK the START with 7'h7E; the only exception would be if this Device chooses to remain as an I²C-only device, in which case, it would leave any 50 ns spike filter enabled. For those Devices that do recognize START and address 7'h7E, those may see any CCC and not just ENTDAA (Enter DA Assignment) or SETDASA (Set Dynamic Address from Static Address). See the I3C Specification for the required CCCs. The Device can determine the effect of each CCC based on whether or not it has the Dynamic Address assigned (e.g., the RSTDAA CCC [Reset DA Address] will probably have no effect before the DA has been assigned).

Q2.9 What are some of the I3C Bus conditions when the Bus is considered inactive?

In addition to open drain, pull-up and high-keeper, the I3C Bus has three distinct conditions under which the Bus is considered inactive: Bus Free, Bus Available, and Bus Idle.

- **Bus Free** condition is defined as a period occurring after a STOP and before a START and for a given duration (e.g., t_{CAS} and t_{BUF} timing).
- **Bus Available** condition is defined as a Bus Free condition for at least t_{AVAL} duration. A Slave may only issue a START request (e.g., for In-Band Interrupt or Master Handoff) after a Bus Available condition.
- **Bus Idle** condition is defined to help ensure Bus stability during Hot-Join events. This condition is defined as a period during which the Bus Available condition is sustained continuously for a duration of at least t_{IDLE}.

Q2.10 When my Device drives the Bus, does it need to see a STOP before a Bus Idle?

For normal active I3C Slaves, yes. They should only drive the Bus for an In-Band Interrupt (IBI) when they have seen a STOP and the $t_{BusAvailable}$ time has elapsed (about 1 µs), and in response to a START (but not a Repeated START).

For Hot-Join devices, they do not know the Bus condition, so they wait until the bus is Idle, which means the SCL and SDA are both high for the t_{IDLE} period (about 1 ms).

Q2.11 When can an I3C Slave issue an In-Band Interrupt (IBI)?

An I3C Slave can issue the IBI in the following two ways:

- Following a START (but not a Repeated START)
- If no START is forthcoming within the Bus Available condition, then an I3C Slave can issue a START request by pulling the SDA line Low. The I3C Master would then complete the START condition by pulling the SCL clock line Low and taking over the SDA.

Q2.12 What is Hot-Join?

The I3C Bus protocol supports a mechanism for Slaves to join the I3C Bus after the Bus is already configured. This mechanism is called Hot-Join. The I3C Specification defines the conditions under which a Slave can do that, e.g., a Slave must wait for a Bus Idle condition.

Q2.13 Can I3C Hot-Join Slave Devices be used on a legacy I^{2}C bus?

Only if they have a way to turn off the Hot-Join feature. Hot-Join is not compatible with I^{2}C masters, so Hot-Join would have to be disabled for the Slave to be used on a legacy I^{2}C bus. The disabling of the Hot-Join feature should be done via some feature that is not part of the I3C protocol (i.e., not via the DISEC CCC), since an I^{2}C master does not support the I3C protocol.

Q2.14 What are the I3C Bus Activity States?

The I3C Bus Activity States provide a mechanism for the Master to inform the Slaves about the expected upcoming levels of activity or inactivity on the Bus, in order to help Slaves better manage their internal states (e.g., to save power).

The four activity states and their expected activity intervals are:

- **Activity State 0:** Normal activity
- **Activity State 1:** Expect quiet for at least 100 µs
- **Activity State 2:** Expect quiet for at least 2 ms
- **Activity State 3:** Expect quiet for at least 50 ms

Q2.15 Is there any time-stamping capability defined in the I3C Bus?

Note:

This question does not apply to I3C Basic v1.0.

Yes. The I3C Bus supports an optional Timing Control mechanism with multiple timing modes. One timing mode is synchronous (from the synchronized timing reference) and four modes are asynchronous (Slave provides timestamp data). All I3C Masters are expected to support at least Async Mode 0.

- **Synchronous:** The Master emits a periodic time sync that allows Slaves to set their sampling time relative to this sync. This may be used in conjunction with one of the Asynchronous modes.
- **Asynchronous:** The Slaves apply their own timestamps to the data at the time they acquire samples, permitting the Master to time-correlate samples received from multiple different Slaves or sensors.

  There are four types (timing modes) of asynchronous time controls:

  - **Async Mode 0:** Basic timing mode that assumes that a Slave has access to a reasonably accurate and stable clock source to drive the time stamping – at least accurate for the duration of the time it has to measure (i.e., from event to IBI). A set of counters, in conjunction with IBI, are used to communicate time stamping information to the Master.
  - **Async Mode 1:** Advanced timing mode extends the Basic mode by using some mutually identifiable bus events, like I3C START.
  - **Async Mode 2:** High-precision timing mode that uses SCL falling edges (for SDR and HDR-DDR modes) as a common timing reference for Master and Slave. A burst oscillator is used to interpolate the time between a detected event and next SCL falling edge. For HDR-TSL and HDR-TSP modes, this timing mode uses both SDA transitions and SCL transitions as timing references.
  - **Async Mode 3:** Highest-precision triggerable timing mode that supports precise time triggering and measurement across multiple transducers applications like beam forming.

Q2.16 Is there a maximum limit to I3C Bus payload length?

By default, there is no limit to the maximum message length. To reduce Bus availability latency across multiple Slaves, the I3C Bus allows negotiating for maximum message lengths between Master and Slave.

Further, Read terminate is possible from the Master, to allow regaining control of the Bus under a long message.

Q2.17 Does the I3C Bus enable 'Bridges'?

Bridge Devices are expected to enable an I3C Bus to be bridged to other protocols, such as SPI, UART, etc. A CCC is defined to enable bridging Devices, where the Master knows in advance that certain Devices are bridges.

Q2.18 Can a Slave indicate any speed limit that it might have?

All I3C Slaves must be tolerant of the 12.5 MHz maximum frequency, and all Slaves must be able to manage those speeds for CCCs. But Slaves may limit the maximum effective data rate for private message – either write, read, or both.

Q2.19 Is there any test mode in the I3C Bus?

Yes. A pair of Directed and Broadcast CCCs is available for the Master to enter/exit the test modes.

Q2.20 Are there any error detection and recovery methods in I3C?

Yes, the I3C Bus has elaborate error detection and recovery methods. Seven Slave error types (S0 to S6) and three Master error types (M0 to M2) are defined for SDR Mode, along with suggested recovery methods. Similarly, a set of errors are defined for each of the HDR Modes.

Q2.21 During HDR-DDR Mode CRC 5 transmission, how many clocks should I be looking for?

> ***Note:***
>
> *This question does not apply to I3C Basic v1.0.*

The CRC transmission ends at bit 6 (counting down from 15), but bit 5 allows High-Z.

Q2.22 Can a Master issue a STOP condition regardless of whether or not a Slave has issued an acknowledgment indicating a completed transaction?

The STOP can be issued anywhere the Slave is not driving the SDA during SCL High. It may not be appropriate to do so in terms of completion of a message. But ACK and completed transaction do not belong together in I3C.

I3C v1.1 FAQs

Q2.23 What errors are reported on the GETSTATUS Protocol Error bit?

The Protocol Error Report on the GETSTATUS CCC is intended to report on errors that have no other way of being detected unless the Device reports them. It is intended to cover parity error, CRC error, and anything else that means that a message from the Master was lost and the Master has no way of knowing it.

The Protocol Error Report on the GETSTATUS CCC would not cover the case of S5 errors, as they are more related to an unsupported CCC or Defining Byte which is not a protocol error *per se*. It is also not intended for situations when the Slave NACKs, because the Master will know that an error occurred when the Slave NACKs and recovers it. Errors like S0, S3, S4, and S5 are detected by the Master when the Slave sends a NACK response, and are recovered by using the appropriate recovery methods; as a result, they do not need to be reported via the GETSTATUS CCC.

Q2.24 What errors are covered by the S5 Error?

Due to confusion around the use of the words "illegally formatted" in I3C v1.0, it was critical in I3C v1.1 to more precisely define what errors are considered to be S5 Errors. The S5 Error covers only four cases of Errors, as follows.

The first two errors are Unsupported CCC and Unsupported Defining Byte, both of which are ignored by Slaves if not supported. The Slaves should wait for the appropriate exit condition in cases where these commands have an special exit condition.

The other two cases are when the master sends the Wrong RnW Bit for a Direct CCC command. This means that the Master sends a Dynamic Address and Read for a Direct Write CCC command, or vice versa. In these cases the Slaves send a NACK, notifying the Master that an error has occurred. The Master will then use the Retry and Escalation models.

Additional Data on CCC Payloads, and Additional Defining Bytes, are not error conditions. Slaves should ignore any additional unrecognized data bytes, per the I3C v1.1 Specification at ***Section 5.1.9.2.2***.

Q2.25 Do Devices have to wait for a Repeated START or STOP, or both, to recover from S2-S5 errors?

These errors should be recovered by STOP. But vendors can choose to recover from these errors when seeing a Repeated START. Note that STOP is the second step of escalation after Repeated START recovery.

Q2.26 When does the RSTACT CCC state clear in an I3C Slave?

The RSTACT state will be cleared back to default upon either:

1. Detection of a Slave Reset Pattern. This will enact the RSTACT action, and then clear the state.
2. Detection of a completed START (but not repeated START). This consists of the SDA line going from High to Low while the SCL line remains High, followed by the first falling edge on SCL. The RSTACT may be cleared either on that falling edge, or on the rising edge that follows.

Q2.27 What is the minimal Slave Reset support required in I3C v1.1?

Although MIPI Alliance strongly recommends true support of Slave Reset, such that a Master can fully reset a Slave chip when needed, it is not required. Full Chip Reset allows replacement of the SRSTn trace that would normally be used to reset a Device.

The minimal reset is a reset of the I3C peripheral, at least to a level that will allow a 'stuck' I3C peripheral to start working again. How much of a reset that requires is up to the Slave Vendor; for example, it could be handled by an internal interrupt which would allow firmware/software in the Slave to handle the reset.

Q2.28 When does a Slave escalate Slave Reset to Full-Chip reset?

If the Slave does not receieve a RSTACT CCC before seeing the Slave Reset Pattern, then it uses the default action of Peripheral Reset (i.e., reset just the I3C block). If the Slave again receives no RSTACT CCC before seeing a Slave Reset, then it then escalates to a Full-Chip Reset. It therefore holds the state after any default Peripheral Reset, so as to activate the escalation. That state is cleared if the Slave sees either an RSTACT CCC, or a GETSTATUS CCC addressed to it.

The purpose of this mechanism is to fix a broken system or setup. Because the Slave Reset Pattern Detector logic is normally separated both from the I3C block and from other parts of the main system, it will continue to work even if the rest of the chip has become broken (e.g., clocks stopped, Bus locked up, etc.). This means that not seeing a RSTACT CCC would be a symptom of a large problem, so the Slave escalates in order to recover from such a broken condition. The Slave first performs a Peripheral Reset, in case the fault condition exists only in the I3C block itself.

Q2.29 Are there any special timing requirements for sending the first START + 7’h7E + W?

Yes. The I3C Master must emit the first START, 7’h7E at Open-Drain speeds (usually I^{2}C Fm+ speeds) so that I3C Slaves with I^{2}C Spike Filters will be able to see the I3C Broadcast Address, and then as a result turn off the Spike Filter. If the Master knows that none of the I3C Slaves has a Spike Filter, then it may omit this. Similarly, if the Master knows that all of the I3C Slaves have accurate Spike Filters, then it may use the I3C 2.5 MHz open-drain speed (i.e., 200 ns Low, 200 ns High).

Q2.30 What is the I3C Open-Drain t_{High} Max? Table 10 shows it as 41 ns, but a Note says it may be longer

In I3C v1.1, ***Table 10*** (I3C Open Draining Timing parameters) lists the t_{High} symbol as having a maximum value of 41 ns, and then Note 4 says that “t_{High} may be exceeded when the signals can be safely seen by I^{2}C Slaves”. This means that a 41 ns t_{High} will ensure that no legacy I^{2}C Slave will see the SCL changes, and so will not interpret the START followed by the Address phase nor the ACK/NACK. The Master may use a much longer t_{High} if there is no harm with a legacy I^{2}C Slave on the I3C Bus seeing the clocks. For example, the legacy Slaves will see the STOP and then a START. It is acceptable for them to see the Address that follows (arbitrated or not), since none of those Addresses will match their own Address. However, as the t_{Low} may well be 200 ns, this may present problems for any legacy I^{2}C slaves not fast enough to correctly handle a 200 ns Low period.

Q2.31 Can you please explain how to interpret t_{SCO} timing?

This topic is extensively explained in the I3C v1.1 Specification at ***Section 5.1.9.3.18***.

Q2.32 Does as I3C Slave need to receive and process the Broadcast ENEC, DISEC, and other Bus-state CCCs before being assigned a Dynamic Address?

Yes. An I3C Slave is expected to monitor Broadcast CCCs; the special exception is for Hot-Join Slaves, as explained below.

The Slave should process all supported Broadcast CCCs, but it is required to process supported ones that affect Bus state. This includes the ENTHDRn CCCs (which turn the HDR Exit Detector on), as well as the ENEC and DISEC CCCs for whatever events the Slave supports (e.g., IBI).

As a practical matter the I3C Main Master will not generally use any CCCs other than Dynamic Address assignment while bringing up the I3C Bus, but it is allowed to use Bus state CCCs as needed.

For a Hot-Join Slave, this rule only applies once the Slave is safely on the I3C Bus and capable of starting a Hot-Join. So, as a general rule, the Slave will emit the Hot-Join request before the I3C Master is able to emit any Broadcast CCCs. However, after that, the Slave may see one or more Broadcast CCCs prior to being assigned a Dynamic Address.

Q2.33 When is the pull-up resister enabled?

Please see the I3C v1.1 Specification at ***Section 5.1.3.1***.

Q2.34 Does the mandated "single-retry model" apply to all Directed Read CCCs?

At ***Section 5.1.9.2.3*** the I3C v1.1 Specification states: “I3C mandates a single-retry model for Direct GET CCC Commands.” Based on this statement, adopters have a general notion that the Master is required to retry once for the Directed GET CCCs if the Slave NACKs for the first time. This may not be applicable for other “read” Directed CCCs (such as RSTACT, MLANE, vendor read, etc.) that are not defined for dedicated Read CCCs (i.e., CCCs that have names of the form GETxxx).

2.3 Implementation: Ecosystem

I3C v1.0 and I3C Basic v1.0 FAQs

Q3.1 Who is defining the MIPI I3C Specifications?

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

Q3.2 Which companies were part of the original MIPI Sensor Working Group?

Under the original name of "MIPI Sensor Working Group," representatives from AMD, Broadcom, Cadence, IDT, Intel, InvenSense, Knowles, Lattice Semiconductor, MediaTek, Mentor Graphics, Nvidia, NXP, Qualcomm, QuickLogic, Sony, STMicroelectronics, Synopsys, VLSI Plus, and others created the MIPI I3C v1.0 and MIPI I3C Basic v1.0 Specifications. The WG was (and is still) chaired by Ken Foust of Intel, and vice-chaired by Satwant Singh of Lattice Semiconductor.

Q3.3 What is the availability of development hardware for I3C prototyping, including FPGAs?

A few vendors have provided FPGA based design kits, including some low-cost FPGAs that might be good enough for smaller production runs.

Q3.4 What is the I3C IP core availability in the market?

Some vendors have started to offer Slave and/or Master IP cores for integration into ASIC devices and FPGAs, including a free-of-cost Slave IP available for prototyping and integration.

I3C v1.1 FAQs

Q3.5 What companies are part of the MIPI I3C Working Group?

In 2019 the Sensor Working Group was renamed to "I3C Working Group". Its member companies often change, but the following is a recent representation: Intel, Invensense, Lattice, NXP, Prodigy, Qualcomm, Robert Bosch, STMicroelectronics, Synopsys, and Sony.

2.4 Implementation: As a System Designer

I3C v1.0 and I3C Basic v1.0 FAQs

Q4.1 What is the maximum capacitance load allowed on the I3C Bus?

The I3C Specification lists the maximum per-Device capacitance on SCL and SDA, but the goal is that most or all Devices will be well below that. Capacitance alone is not sufficient to determine maximum frequency on the I3C Bus (as with any Bus). It is important to consider maximum propagation length, effect of stubs, internal clock-to-data (t_{SCO}) of the Slaves, as well as capacitive load.

Q4.2 What is the maximum wire length for I3C communication?

The maximum wire length would be a function of speed, as all the reflections and Bus turnaround must complete within one cycle. Larger distances can be achieved at the lower speeds than at the higher ones. For example at 1 meter (between Master and Slave), the maximum effective speed is around 6 MHz for read, to allow for clock propagation time to Slave and SDA return time to Master.

Q4.3 Can I^{2}C repeaters be used for I3C?

Not directly, for a couple of reasons:

1. The I3C Bus works with push-pull modes (in addition to the open drain for some transfers), and
2. Much higher speeds. Most such devices are quite limited in speed, because of the lag effect of changing states on SCL and SDA due to both series-resistance and assumptions about open-drain.

Long wire approaches are being evaluated for a future version of the I3C Specification.

Q4.4 Will the I^{2}C devices respond to I3C commands?

No. The I3C CCCs are always preceded by I3C Broadcast Address, 7'h7E. Since the I^{2}C specification reserves address 7'h7E, no legacy I^{2}C Slave will match the I3C Broadcast Address, and thus would not respond to the I3C commands. Likewise, the Dynamic Addresses assigned to I3C Devices would not overlap the I^{2}C static addresses, so an I^{2}C device would not respond to any I3C address (even if it could see it).

Q4.5 How are communication conflicts resolved on the I3C Bus?

I3C Slaves are only allowed to drive the Bus under certain situations. Besides during a read, and when ACKing their own address, they may also drive after a START (but not Repeated START). After a START, the I3C Bus reverts back to open-drain pull-up resistor mode; thus, the Slave that drives a low value (logic 0) would win.

Q4.6 Can I3C Devices cause the communication bus to 'hang'?

Unlike I²C, there is no natural way to 'hang' the I3C Bus. In I²C, clock stretching (where the Slave holds the clock low, stopping it from operating) often causes serious problems with no fix: there's simply no way to get the Slave's attention if it has hung the Bus. By contrast, in I3C only the Master drives the clock, and so the Slave performs all actions on SDA relative to that clock, thereby eliminating the normal causes of such hangs.

Further, since I3C is designed to ensure that I3C Slaves can operate their back-end I3C peripheral off the SCL clock (vs. oversampling), problems elsewhere in the Slave will not translate into Bus hangs.

If a system implementer is highly concerned about a Slave accidently locking itself, then a separate hard-reset line could be used. For the next revision of the I3C and I3C Basic Specifications, a feature called Slave Reset is being added to reset non-responsive I3C Slaves (i.e., if a Master emits a certain pattern that does not occur during regular communication, then the Devices on the Bus would treat it just like a hardwired reset line).

Q4.7 Will all I3C Devices be compatible with all CCCs?

No. Some CCCs are mandatory, whereas others are optional and a given I3C Device will either support them or not, depending upon the Device's capabilities.

I3C v1.1 FAQs (None)

2.5 Implementation: As a Software Developer

I3C v1.0, I3C v1.1, and I3C Basic v1.0 FAQs

Q5.1 Are there any companion MIPI I3C Specifications that enable software development?

Yes. The following MIPI Specifications are expected to help with software development:

- ***MIPI Specification for I3C Host Controller Interface (I3C HCI), v1.0 [MIPI02]***
  and
  MIPI Specification for I3C Host Controller Interface (I3C HCI), v1.1 [MIPI13] (In development)

  Creates a standard definition that allows a single OS driver (also known as 'in-box driver') to support I3C hardware from several vendors, while also allowing vendor-specific extensions or improvements. The target audience of the HCI Specification is application processor host controllers; in particular, developers of host controller (i.e., I3C Main Master) hardware, and developers of I3C host controller software.
- ***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, v1.0 [MIPI04]***

  Allows operating system software to use ACPI (Advanced Configuration and Power Interface) 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, a supporting document is also available. The ***System Integrators Application Note for I3C v1.0 and I3C Basic v1.0, App Note v1.0 [MIPI05]*** has been developed to help ASIC hardware developers, system designers, and others working in the more deeply embedded I3C Devices. The I3C WG plans to develop an updated revision of this Application Note that will cover new features and capabilities included in I3C v1.1.

Q5.2 Are there software libraries available (or about to be) for I3C?

Yes. Core I3C infrastructure is being added to the Linux Kernel via patchwork.kernel.org.

2.6 Interoperability Workshops

I3C v1.0 and I3C Basic v1.0 FAQs

Q6.1 What is a MIPI I3C Interoperability Workshop?

It is a MIPI Alliance sponsored event where different vendors bring their I3C implementations and check interoperation with other vendors.

Q6.2 What is the output from a MIPI I3C Interoperability Workshop?

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

- The vendors can get detailed information about how well their I3C implementations interoperate with other vendors' implementation. Vendors can also compare their results with one another.
- MIPI Alliance can generate an overall picture that shows the industry state-of-the-I3C-implementaion. For example, how many vendors have implemented I3C, and how many implementations pass or fail against one another.
- The MIPI I3C Working Group gets better understanding about any major issues with the I3C Specification. The WG can then leverage that learning by adding to this FAQ, the ***System Integrators Application Note for I3C v1.0 and I3C Basic v1.0 [MIPI05]***, and the next revision of the MIPI I3C Specification ***[MIPI01][MIPI10][MIPI12]***.

Q6.3 Are MIPI I3C Interoperability Workshops an ongoing activity?

MIPI arranges a particular I3C Interoperability Workshop event in response to requests from its membership.

Q6.4 Who can attend or participate in a MIPI I3C Interoperability Workshop?

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

Q6.5 What HW/SW is typically needed to participate in a MIPI I3C Interoperability Workshop?

While this could change in future, the minimal requirements to date have been the availability of a board with an I3C Device that can connect to other Devices via the three wires SDA, SCL, and GND. It's also useful to have software (e.g., running on a laptop connected to the board and I3C Device) to interactively view transmitted and received bus communications, but this might not be required for Slaves.

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

I3C v1.1 FAQs

Q6.6 Are there any interoperability workshops planned for I3C v1.1?

Starting in 2020, MIPI Alliance plans to host interoperability workshops for I3C v1.1.

2.7 Up and Coming

Q7.1 What future MIPI Specifications will be leveraging I3C?

Many other MIPI Alliance Working Groups are in the process of leveraging the I3C Specification. As of the writing of this FAQ, the list includes:

- **Camera WG:** Camera Control Interface (CCI) chapter of the ***MIPI Specification for Camera Serial Interface 2 (CSI-2), v4.0 [MIPI06]*** (In development)
- **Debug WG:** ***MIPI Specification for Debug for I3C, v1.0 [MIPI07]*** (In development)
- **RIO WG** (Reduced I/O)**:** ***MIPI Specification for Virtual GPIO Interface (VGISM), v1.0 [MIPI08]***, reducing number of GPIOs used via I3C (In development)

I3C v1.0 and I3C Basic v1.0 FAQs

Q7.2 Are there any impending MIPI I3C fixes/errata that should be applied now?

Note:

*With the release of I3C v1.1, this question has been deprecated, but it is left here for reference. For example, see **Q1.25** for what's new in I3C v1.1.*

Based on learning from the early implementations, I3C Interoperability Workshops, queries from adopters, and reviews by the I3C WG, this FAQ represents clarifications, planned improvements that can be implemented by I3C v1.0 Devices, and other hints about what's coming next.

Q7.3 Are any revisions to MIPI I3C v1.0 expected?

No, there are no pending updates at this time.

I3C v1.1 FAQs

Q7.4 Are there any impending MIPI I3C fixes/errata that should be applied now?

No, there are no pending errata at this time.

Q7.5 Are any revisions to I3C v1.1 expected?

There are no immediate plans to revise I3C v1.1, however the MIPI I3C WG meets regularly and is considering proposals.

Q7.6 What new features, if any, are coming to MIPI I3C?

There are no new approved features, however the MIPI I3C WG is considering the following:

- Automotive-focused capabilities
- Security over I3C
- Improved reliability
- Speed increases
- New Multi-Lane uses
- Long Reach
- New HDR Modes
- Refining existing features

2.8 Clarification, Disambiguation, and Allowances

I3C v1.0 and I3C Basic v1.0 FAQs

Q8.1 Can Sync and Async Time Control be enabled at the same time?

Note:

This question does not apply to I3C Basic v1.0.

Yes. This is allowed such that the ODR (Output Data Rate) rate controls the In-Band Interrupt (IBI) rate, and the Async Mode timestamp on the IBI indicates how long ago the sample was collected.

Q8.2 Is there a way to turn off Time Control?

Note:

This question does not apply to I3C Basic v1.0.

Yes, the SETXTIME code of 0xFF may be used. This has been updated for I3C v1.1, please see ***Q8.10***.

Q8.3 Do I have to wait the full 1 ms for Hot-Join?

Note:

This question does not apply to I3C Basic v1.0 and I3C v1.1.

Newer versions of the I3C Specification define a new t_{IDLE} minimum value of 200 µs, since that is sufficient for all valid uses. I3C v1.0 devices may choose to support that smaller delay now. I3C Basic v1.0 and I3C v1.0 already support a t_{IDLE} minimum value of 200 µs.

Q8.4 Is there any way to exit from an S0/S1 error other than waiting for an Exit Pattern?

Yes. An I3C Slave may watch for 60 µs with no SCL or SDA changes to make sure that the Bus is not in HDR Mode (and therefore must be in SDR Mode). After that, it is appropriate to wait for START (assumed to be Repeated START) or STOP.

Q8.5 What happens if the Master crashes during a Read?

The I3C Slave may choose to detect 100 µs without an SCL edge. If that happens, it can abandon the Read and release SDA to avoid a Bus hang when the Master restarts.

Q8.6 If a Device has a t_{SCO} value greater than 12 ns, does that mean it doesn’t qualify as an I3C Device?

Note:

This question does not apply to I3C Basic v1.0.

No. The t_{SCO} (Clock to Data Turnaround delay time) is information provided by Slaves so that system designers can properly compute the maximum effective frequency for reads on the Bus. The t_{SCO} number is meant to be used together with the line capacitance (trace length) and number of Slaves and stubs (if present).

However, I3C Slaves with t_{SCO} delay greater than 12 ns must:

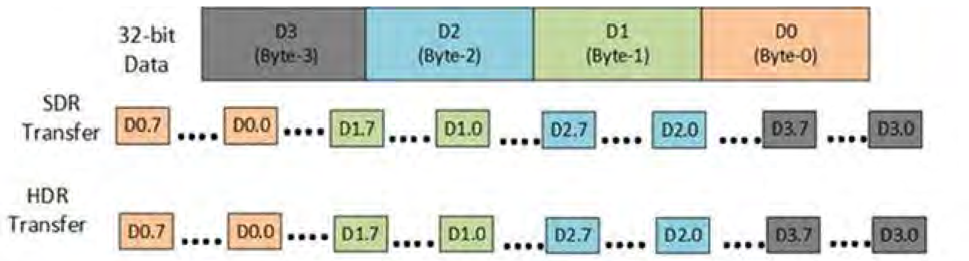
- Set the Limitation bit in the Bus Characteristics Register (BCR) to 1’b1,
- Set the Clock-to-Data Turnaround field of the maxRD Byte to 3’b111, and
- Communicate the t_{SCO} value to the Master by private agreement (i.e., product datasheet).
- I3C Basic v1.0 already clarifies this.

Q8.7 Does Figure 44 HDR-DDR Format apply for Command, Data, and CRC? Or only for Data?

Note:

This question does not apply to I3C Basic v1.0.

Only for Data. The Data bytes are sent the same way as in SDR Mode. The following figure shows how Data is transferred from memory to the I3C Data Line:


32-bit Data
D3 (Byte-3)
D2 (Byte-2)
D1 (Byte-1)
D0 (Byte-0)
SDR Transfer
D0.7 .... D0.0 .... D1.7 .... D1.0 .... D2.7 .... D2.0 .... D3.7 .... D3.0
HDR Transfer
D0.7 .... D0.0 .... D1.7 .... D1.0 .... D2.7 .... D2.0 .... D3.7 .... D3.0

The Command and CRC Byte are each transferred as a packet instead:

Command (MSb to LSb):
- Preamble (2 bits)
- Command (8 bits)
- Slave address (7 bits)
- Reserved bit (1 bit)

CRC (MSb to LSb):
- Preamble (2 bits)
- Token (4 bits)
- CRC Byte (5 bits)
- Setup bits (2 bits)

Q8.8 Does the Master provide an additional CLK after the Terminate Condition and before the Restart/Exit Pattern, as shown in Figure 52 Master Terminates Read?

Note:

This question does not apply to I3C Basic v1.0.

No, there is an error in ***Figure 52*** in the I3C v1.0 Specification. The beginning of the Restart/Exit Pattern should show SCL Low and SDA changing.

Q8.9 In a Hot-Join, when should the DISEC CCC be sent? After ACK, or after NACK?

The Hot-Join mechanism allows the Master to first NACK, and then send the DISEC CCC to stop the Hot-Join. If the Master ACKs, that is interpreted as a promise that it will eventually send the ENTDAA CCC.

I3C v1.1 FAQs

Q8.10 Is there a way to turn off Timing Control?

Yes. The SETXTIME CCC allows the code 8’hFF to be sent, and this turns off Timing. This can be used by v1.0 Slaves as well.

Q8.11 What happens if the Master crashes during a Read?

The I3C Slave should time out if it experiences more than 100 µs with no SCL, and abandon the read. It does not have to be precise, but since I3C’s minimum frequency is 10 KHz, anything longer than 100 µs means that the Master has failed to complete the Read, so the Slave should High-Z the SDA and abandon the Read.

Note that if the Slave is in legacy I^{2}C mode, then it would not normally abandon the read, since there is no minimum frequency, and because 9 clocks by a Master will be enough to abort the Read (since the 9th bit of data is ACK/NACK).

Q8.12 Do I have to wait the full 1 ms for Hot-Join? How does I3C v1.1 handle this?

In I3C v1.1, this time was reduced since no HDR Mode can exceed 100 µs due to the minimum frequency being 10 KHz. As a result, t_{IDLE} is now 200 µs to provide an adequate timing margin.

Q8.13 Why was the RSTDAA Directed CCC deprecated, and why is it being removed?

Because the RSTDAA CCC should not be directed to just one Slave. If the Master needs to reassign a Slave address, then it can use the SETNEWDA CCC.

Q8.14 How does the Set Bridge Targets (SETBRGTGT) CCC differ between I3C v1.0 and I3C v1.1?

In I3C v1.1, use of this CCC is expanded to support Bridging Devices with Dynamic Addresses, enabling multiple Bridging Devices on the same I3C Bus. The context of this CCC is also redefined, such that a Bridging Device is now one of several types of Devices that expose or present Virtual Slaves. Bit[4] of the Bus Configuration Register (BCR) now indicates Virtual Slave capabilities.

For bridged Slaves that are enabled by a Bridging Device, I3C v1.1 clarifies the use of other CCCs (such as SETMRL) that address a bridged Slave. I3C v1.1 also clarifies that the SETBRGTGT CCC must be used to change the Dynamic Address of a bridged Slave (not the SETNEWDA CCC).

See the I3C v1.1 Specification at ***Sections 5.1.1.2.1***, ***5.1.9.3.17***, and ***5.1.9.3.19***; and the ***System Integrators Application Note for I3C [MIPI05]*** at ***Section 5.7***.

Q8.15 How do t_{CBSr} and t_{CASr} timing differ between I3C v1.1 and I3C v1.0?

In I3C v1.1, t_{CASr}’s minimum value is reduced to $t_{CASmin/2}$ (whereas in I3C v1.0, it is t_{CASmin}). This reduced duration might be a challenge for some Slaves. In order to prevent such situations, the Master of the Bus shall provide suitable timing, accommodating the slowest Devices on the Bus.

In the I3C v1.1 Specification, ***Table 111*** “I3C Push-Pull Timing Parameters for SDR, ML, HDR-DDR, and HDR-BT Modes” includes a clarifying note:

> *9) Slaves with speed limitations inform the Master via the Bus Characteristics Register that the minimum may not be acceptable. As a result, if the given SCL HIGH period is 50 ns or greater, the Master needs to accommodate for Legacy I^{2}C Devices that might see it.*

Q8.16 How is the GETMXDS CCC (maximum data speed) updated in I3C v1.1?

The standard GETMXDS CCC itself was not changed in I3C v1.1, though the definition of t_{SCO} was clarified. However, the GETMXDS CCC now supports a Defining Byte value that allows the return of other forms of information. With no Defining Byte (and with a Defining Byte with value 0), the GETMXDS CCC behaves exactly as the original GETMXDS CCC did in I3C v1.0.

2.9 Conformance Testing

I3C v1.0 and I3C Basic v1.0 FAQs

Q9.1 What is a MIPI Conformance Test Suite (CTS)?

A MIPI WG develops a Conformance Test Suite (CTS) document in order 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.

Q9.2 Is there a CTS for I3C v1.0?

Yes. A CTS for I3C v1.0 is being drafted by the MIPI Alliance I3C WG ***[MIPI09]***.

Q9.3 What is the scope of tests for the I3C v1.0 CTS?

The CTS tests are designed to determine whether a given product conforms to a subset of the requirements defined in the I3C Specification v1.0. The scope of this first version of the CTS is intentionally limited, in order to meet time-to-market requirements imposed by the rapid adoption of I3C in the marketplace, focusing on:

1. SDR-only Devices without optional I3C capabilities,
2. All Master and Slave Error Detection and Recovery methods, and
3. Basic HDR Enter/tolerance/Restart/Exit are in scope, but HDR-DDR is under consideration.

Considering the CTS a living document, the I3C WG plans to continue expanding the scope of the CTS through future revisions that eventually encompass all required and optional features of the I3C Specification.

The CTS tests are organized as Master device under test (DUT) and Slave DUT. Tests for each are presented in the order in which they appear in the I3C Specification, to simplify identification of pertinent detail between the two documents.

Q9.4 Does the I3C Interoperability Workshop follow the I3C CTS?

Interoperability Workshops will ultimately follow the tests identified in the I3C CTS, as it nears completion.

Q9.5 What details are provided for each I3C CTS test case?

Each test in the I3C CTS will contain:

- A clear purpose
- References
- Resource requirements
- Tracked last technical modification
- Discussion
- All test case detail (i.e., setup, procedure, results, and problems).

DC/AC parametric requirements will be embedded in each test, not split out into a separate PHY-related CTS or subsection.

I3C v1.1 FAQs

Q9.6 Is there a CTS for I3C v1.1?

The I3C WG is currently developing a CTS for I3C v1.1, for early 2020 release.

2.10 Legal & Intellectual Property Related Questions

I3C v1.0 and I3C Basic v1.0 FAQs

Q10.1 Is MIPI I3C Basic royalty free?

The parties that directly developed the MIPI I3C Basic specification have agreed to license all implementers on royalty free terms, as further described in the I3C Basic specification document ***[MIPI10]***. Further, all implementers of the I3C Basic specification must commit to grant a reciprocal royalty free license to all other implementers if they wish to benefit from these royalty free license commitments. And, of course, MIPI itself does not charge royalties in connection with its specifications. MIPI's intent is to create a robust royalty free environment for all implementers of I3C Basic.

No set of IPR terms can comprehensively address all potential risks, however. The terms apply only to those parties that agree to them, for example, and the scope of application is limited to what is described in the terms. Implementers must ultimately make their own risk assessment.

Q10.2 What license terms apply to MIPI I3C v1.x?

MIPI's regular IPR terms apply to the full MIPI I3C specification. MIPI's terms require that members make licenses available only to other members, as described in the MIPI Membership Agreement and MIPI Bylaws. To benefit from the license commitments, a party must be a MIPI member.

For features that are included in MIPI I3C Basic, a MIPI member can implement under the regular IPR terms, or can opt to implement the feature under the I3C Basic framework. If a member opts in to the I3C Basic framework, then they must grant the reciprocal licenses required under that framework. MIPI Alliance members are not required to participate in the I3C Basic license framework, however. Features of I3C 1.x that are not included in I3C Basic are subject only to MIPI's regular IPR terms.

Prior to the release of I3C Basic, MIPI had made certain versions of the full MIPI I3C specification available for public review, under "copyright only" terms – that is, MIPI published the specification, but noted that no rights to implement the specification were granted under any party's patent rights. MIPI no longer publishes the full I3C specification publicly. A non-member is not granted any right to implement the full MIPI I3C 1.x specification, either by MIPI or any MIPI member.

I3C Basic is available to non-members, as described in question ***Q2.1 How can the MIPI I3C v1.0 Specifications be obtained?***.

I3C v1.1 FAQs (None)

3 Terminology

See also ***Section 2*** in the MIPI I3C Specification ***[MIPI01][MIPI10][MIPI12]***.

3.1 Definitions

ACK: Short for "acknowledge" (an I3C Bus operation)

Bus Available: I3C Bus condition in which a Device is able to initiate a transaction on the Bus.

Bus Free: I3C Bus condition after a STOP and before a START with a duration of at least t_{CAS}.

Bus Idle: An extended duration of the Bus Free condition in which Devices may attempt to Hot-Join the I3C Bus.

High-Keeper: A weak Pull-Up type Device used when SDA, and sometimes SCL, is in High-Z with respect to all Devices.

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

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

Main Master: Master that has initial control of the I3C Bus.

Master: The I3C Bus Device that is controlling the Bus.

Slave: An I3C Slave Device can only respond to either Common or individual commands from a Master. A Slave Device cannot generate a clock.

3.2 Abbreviations

DisCo	Discovery and Configuration (family of MIPI Alliance interface Specifications)
e.g.	For example (Latin: exempli gratia)
i.e.	That is (Latin: id est)

3.3 Acronyms

See also the acronyms defined in the MIPI I3C Specification ***[MIPI01][MIPI10][MIPI12]***.

CCC	Common Command Code (an I3C common command or its unique code number)
CTS	Conformance Test Suite
DAA	Dynamic Address Assignment (an I3C Bus operation)
FAQ	Frequently Asked Questions
HCI	Host Controller Interface (a MIPI Alliance interface Specification ***[MIPI02]***)
HDR	High Data Rate (a set of I3C Bus Modes)
HDR-DDR	HDR Double Data Rate (an I3C Bus Mode)
HDR-TSP	HDR Ternary Symbol for Pure Bus (an I3C Bus Mode)
I3C	Improved Inter Integrated Circuit (a MIPI Alliance interface Specification ***[MIPI01][MIPI10][MIPI12]***)
IBI	In-Band Interrupt (an I3C Bus feature)
ODR	Output Data Rate
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)

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 1.0, MIPI Alliance, Inc., 25 January 2019 (Adopted 18 June 2019).

[MIPI05] *MIPI Alliance System Integrators Application Note for I3C® v1.0 and I3C Basic ^{SM} v1.0*, App Note version 1.0, MIPI Alliance, Inc., 8 December 2018 (approved 8 December 2018).

[MIPI06] *MIPI Alliance Specification for Camera Serial Interface 2 (CSI-2^{SM}*), version 4.0, MIPI Alliance, Inc., In press.

[MIPI07] *MIPI Alliance Specification for Debug for I3C^{SM}*, version 1.0, MIPI Alliance, Inc., In press.

[MIPI08] *MIPI Alliance Specification for Virtual GPIO Interface (VGI^{SM}*), version 1.0, MIPI Alliance, Inc., In press.

[MIPI09] *MIPI Alliance Conformance Test Suite (CTS) for I3C^{SM} v1.0*, CTS version 1.0, MIPI Alliance, Inc., In press.

[MIPI10] *MIPI Alliance Specification for I3C Basic^{SM} (Improved Inter Integrated Circuit)*, version 1.0, MIPI Alliance, Inc., 19 July 2018 (Adopted 8 October 2018).

[MIPI11] MIPI Alliance, Inc., "I3C SETBUSCON Table", https://www.mipi.org/MIPI_I3C_bus_context_byte_values_public.html, last accessed 26 February 2020.

[MIPI12] *MIPI Alliance Specification for I3C® (Improved Inter Integrated Circuit)*, version 1.1, MIPI Alliance, Inc., 27 November 2019 (Adopted 11 December 2019).

[MIPI13] *MIPI Alliance Specification for I3C Host Controller Interface (I3C HCI^{SM})*, version 1.1, MIPI Alliance, Inc., In press.

This page intentionally left blank.