Errata 01 for
MIPI CSI-2 Specification
(Camera Serial Interface 2)
Specification Version 2.1

Specification Dated 14 December 2017

Specification MIPI Board Adopted 09 April 2018

Errata 01 Dated 26 April 2018

Errata MIPI Board Approved 04 May 2018

* IMPORTANT NOTE TO IMPLEMENTERS *

- The issues(s) listed in this Errata document will be corrected in the next edition of this MIPI Specification.
- Implementations should observe all Corrections listed here.
- The location of each Correction is also marked in the attached copy of the MIPI Specification. To reduce the risk of incorrect implementations, we suggest you consider discarding any previous copies of this MIPI Specification not so marked.
- This MIPI Specification as modified by the corrections listed in this Errata document is also a MIPI Specification, as the MIPI Bylaws defines the term.
- **MIPI member companies' rights and obligations apply to the modified MIPI Specification as defined in the MIPI Membership Agreement and MIPI Bylaws.**

The WG has identified a contradiction in the CSI-2 Specification for descriptions of RAW6 and RAW7 data format (*Section 11.4.1* and *Section 11.4.2*) in the CSI-2 Specification since v1.0 (including CSI-2 v1.01, CSI-2 v1.1, CSI-2 v1.2, CSI-2 v1.3, CSI-2 v2.0, and CSI-2 v2.1).

The RAW6 and RAW7 data format descriptions describe the format as including LS/LE synchronization codes. The WG has confirmed that line synchronization packets are optional, and the correction is to remove the second sentence of each paragraph for RAW6 and RAW7 in order to remove the contradicting statements.

Item	Spec Page Number	PDF Page Number	Correction
1	152	172	**Editorial or Technical:** Technical **Location:** Line 1719-1720 **Correction:** Remove sentence "Each line is separated by line start / end synchronization codes." **Reason:** This sentence contradicts the specification statement that line synchronization packets (LS/LE short packets) are optional. For reference, see Section 9.8.2 Line Synchronization Packet, line 1165 stating "Line synchronization packets are optional on a per-image-frame basis." **Technical Impact:** Hardware implementations transmitting the additional LS/LE short packets in conjunction with this data type may wish to make them optional in the future for users not desiring them. Any CSI-2 verification IP reporting a non-conformance error when LS/LE short packets are not transmitted in conjunction with this data type must be changed in order to no longer report this error.
2	153	173	**Editorial or Technical:** Technical **Location:** Line 1728-1729 **Correction:** Remove sentence "Each line is separated by line start / end synchronization codes." **Reason:** This sentence contradicts the specification statement that line synchronization packets (LS/LE short packets) are optional. For reference, see Section 9.8.2 Line Synchronization Packet, line 1165 stating "Line synchronization packets are optional on a per-image-frame basis." **Technical Impact:** Hardware implementations transmitting the additional LS/LE short packets in conjunction with this data type may wish to make them optional in the future for users not desiring them. Any CSI-2 verification IP reporting a non-conformance error when LS/LE short packets are not transmitted in conjunction with this data type must be changed in order to no longer report this error.

Specification for
Camera Serial Interface 2
(CSI-2^{SM})

Version 2.1
14 December 2017

MIPI Board Adopted 9 April 2018

NOTICE OF DISCLAIMER

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

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

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

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

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

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

Contents

Confidential

Figures

Tables

Release History

Date	Version	Description
2005-11-29	v1.00	Initial Board-approved release.
2010-11-09	v1.01.00	Board-approved release.
2013-01-22	v1.1	Board approved release.
2014-09-10	v1.2	Board approved release.
2014-10-07	v1.3	Board approved release.
2017-03-28	v2.0	Board approved release.
2018-04-09	v2.1	Board approved release.

This page intentionally left blank.

1   Introduction

1.1   Scope

The Camera Serial Interface 2 Specification defines an interface between a peripheral device (camera) and a host processor (baseband, application engine). The purpose of this document is to specify a standard interface between a camera and a host processor for mobile applications.

This Revision of the Camera Serial Interface 2 Specification leverages C-PHY version 1.2 *[MIPI02]* and D-PHY version 2.1 *[MIPI01]*. These enhancements enable higher interface bandwidth and more flexibility in channel layout. The CSI-2 version 1.3 Specification was designed to ensure interoperability with CSI-2 version 1.2 when the former uses the D-PHY physical layer. If the C-PHY physical layer only is used, then backwards compatibility cannot be maintained.

In this document, the term 'host processor' refers to the hardware and software that performs essential core functions for telecommunication or application tasks. The engine of a mobile terminal includes hardware and the functions, which enable the basic operation of the mobile terminal. These include, for example, the printed circuit boards, RF components, basic electronics, and basic software, such as the digital signal processing software.

1.2   Purpose

Demand for increasingly higher image resolutions is pushing the bandwidth capacity of existing host processor-to-camera sensor interfaces. Common parallel interfaces are difficult to expand, require many interconnects, and consume relatively large amounts of power. Emerging serial interfaces address many of the shortcomings of parallel interfaces while introducing their own problems. Incompatible, proprietary interfaces prevent devices from different manufacturers from working together. This can raise system costs and reduce system reliability by requiring "hacks" to force the devices to interoperate. The lack of a clear industry standard can slow innovation and inhibit new product market entry.

CSI-2 provides the mobile industry a standard, robust, scalable, low-power, high-speed, cost-effective interface that supports a wide range of imaging solutions for mobile devices.

2   Terminology

2.1   Use of Special Terms

The MIPI Alliance has adopted Section 13.1 of the *IEEE Standards Style Manual*, which dictates use of the words "shall", "should", "may", and "can" in the development of documentation, as follows:

The word *shall* is used to indicate mandatory requirements strictly to be followed in order to conform to the Specification and from which no deviation is permitted (*shall* equals *is required to*).

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

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

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

The word *may* is used to indicate a course of action permissible within the limits of the Specification (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

All sections are normative, unless they are explicitly indicated to be informative.

2.2   Definitions

CCI (I²C): CCI supporting I²C.

CCI (I3C): CCI supporting I3C.

CCI (I3C SDR) means CCI supporting I3C SDR.

CCI (I3C DDR) means CCI supporting I3C DDR.

Lane: A unidirectional, point-to-point, 2- or 3-wire interface used for high-speed serial clock or data transmission; the number of wires is determined by the PHY specification in use (i.e. either D-PHY or C-PHY, respectively). A CSI-2 camera interface using the D-PHY physical layer consists of one clock Lane and one or more data Lanes. A CSI-2 camera interface using the C-PHY physical layer consists of one or more Lanes, each of which transmits both clock and data information. Note that when describing features or behavior applying to both D-PHY and C-PHY, this specification sometimes uses the term data Lane to refer to both a D-PHY data Lane and a C-PHY Lane.

Message: In CCI (I2C) or CCI (I3C SDR), a Message begins with a START or Repeated START condition, followed by the address of the targeted slave(s), R/W bit, other data, and ends with either a STOP or Repeated START condition. In the case of CCI (I3C SDR), a START or Repeated START condition followed by 7'h7E may be added to the beginning. In CCI (I3C DDR), a Message begins with either the I3C ENTHDR0 CCC or the I3C HDR Restart Pattern, followed by an HDR-DDR Command, HDR-DDR Data, and ends with either the I3C HDR Exit Pattern or the I3C HDR Restart Pattern.

Operation: An Operation is composed of one or more Messages in order to read or write.

Packet: A group of bytes organized in a specified way to transfer data across the interface. All packets have a minimum specified set of components. The byte is the fundamental unit of data from which packets are made.

Payload: Application data only – with all sync, header, ECC and checksum and other protocol-related information removed. This is the "core" of transmissions between application processor and peripheral.

Sleep Mode: Sleep mode (SLM) is a leakage level only power consumption mode.

Transmission: The time during which high-speed serial data is actively traversing the bus. A transmission is bounded by SoT (Start of Transmission) and EoT (End of Transmission) at beginning and end, respectively.

Virtual Channel: Multiple independent data streams for up to 32 peripherals are supported by this Specification. The data stream for each peripheral may be a Virtual Channel. These data streams may be interleaved and sent as sequential packets, with each packet dedicated to a particular peripheral or channel. Packet protocol includes information that links each packet to its intended peripheral.

2.3   Abbreviations

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

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

2.4   Acronyms

ALPS   Alternate Low Power State

BER   Bit Error Rate

CCI   Camera Control Interface

CIL   Control and Interface Logic

CRC   Cyclic Redundancy Check

CSI   Camera Serial Interface

CSPS   Chroma Shifted Pixel Sampling

DDR   Dual Data Rate

DI   Data Identifier

DT   Data Type

ECC   Error Correction Code

EoT   End of Transmission

EPD   Efficient Packet Delimiter (PHY and / or Protocol generated signaling used in LRTE)

EXIF   Exchangeable Image File Format

FE   Frame End

FS   Frame Start

HS   High Speed; identifier for operation mode

HS-LPS-LS   High speed to Low Power State to High speed switching (includes LPS entry and exit latencies)

HS-RX   High-Speed Receiver

HS-TX   High-Speed Transmitter

I2C   Inter-Integrated Circuit

96	ILR	Interpacket Latency Reduction
97	JFIF	JPEG File Interchange Format
98	JPEG	Joint Photographic Expert Group
99	LE	Line End
100	LFSR	Linear Feedback Shift Register
101	LLP	Low Level Protocol
102	LS	Line Start
103	LSB	Least Significant Bit
104	LSS	Least Significant Symbol
105	LP	Low-Power; identifier for operation mode
106	LP-RX	Low-Power Receiver (Large-Swing Single Ended)
107	LP-TX	Low-Power Transmitter (Large-Swing Single Ended)
108	LRTE	Latency Reduction Transport Efficiency
109	MSB	Most Significant Bit
110	MSS	Most Significant Symbol
111	PDQ	Packet Delimiter Quick (PHY generated and consumed signaling used in LRTE)
112	PF	Packet Footer
113	PH	Packet Header
114	PI	Packet Identifier
115	PT	Packet Type
116	PHY	Physical Layer
117	PPI	PHY Protocol Interface
118	PRBS	Pseudo-Random Binary Sequence
119	RGB	Color representation (Red, Green, Blue)
120	RX	Receiver
121	SCL	Serial Clock (for CCI)
122	SDA	Serial Data (for CCI)
123	SLM	Sleep Mode
124	SoT	Start of Transmission
125	TX	Transmitter
126	ULPS	Ultra Low Power State
127	VGA	Video Graphics Array
128	YUV	Color representation (Y for luminance, U & V for chrominance)

3   References

129   [NXP01]   UM10204, *I²C bus specification and user manual*, Revision 6, NXP Semiconductors
130   N.V., 4 April 2014.

131   [MIPI01]   *MIPI Alliance Specification for D-PHY*, version 2.1, MIPI Alliance, Inc., 28 March 2017.

132   [MIPI02]   *MIPI Alliance Specification for C-PHY*, version 1.2, MIPI Alliance, Inc., 28 March 2017.

133   [MIPI03]   *MIPI Alliance Specification for I3C (Improved Inter-Integrated Circuit)*, version 1.0,
134   MIPI Alliance, Inc., 31 December 2016.

135   [MIPI04]   *MIPI Alliance Specification for Camera Command Set (CCS)*, version 1.0, MIPI
136   Alliance, Inc., 24 October 2017.

137   [MIPI05]   *MIPI Alliance Specification for D-PHY*, version 2.0, MIPI Alliance, Inc., 8 March 2016.

This page intentionally left blank.

4   Overview of CSI-2

138 The CSI-2 Specification defines standard data transmission and control interfaces between transmitter and
139 receiver. Two high-speed serial data transmission interface options are defined.

140 The first option, referred to in this specification as the "D-PHY physical layer option," is a unidirectional
141 differential interface with one 2-wire clock Lane and one or more 2-wire data Lanes. The physical layer of
142 this interface is defined by the *MIPI Alliance Specification for D-PHY [MIPI01]*. ***Figure 1*** illustrates the
143 connections for this option between a CSI-2 transmitter and receiver, which typically are a camera module
144 and a receiver module, part of the mobile phone engine.

145 The second high-speed data transmission interface option, referred to in this specification as the "C-PHY
146 physical layer option," consists of one or more unidirectional 3-wire serial data Lanes, each of which has its
147 own embedded clock. The physical layer of this interface is defined by the *MIPI Alliance Specification for
148 C-PHY [MIPI02]*. ***Figure 2*** illustrates the CSI transmitter and receiver connections for this option.

149 The Camera Control Interface (CCI) for both physical layer options is a bi-directional control interface
150 compatible with the I2C standard *[NXP01]*.

151
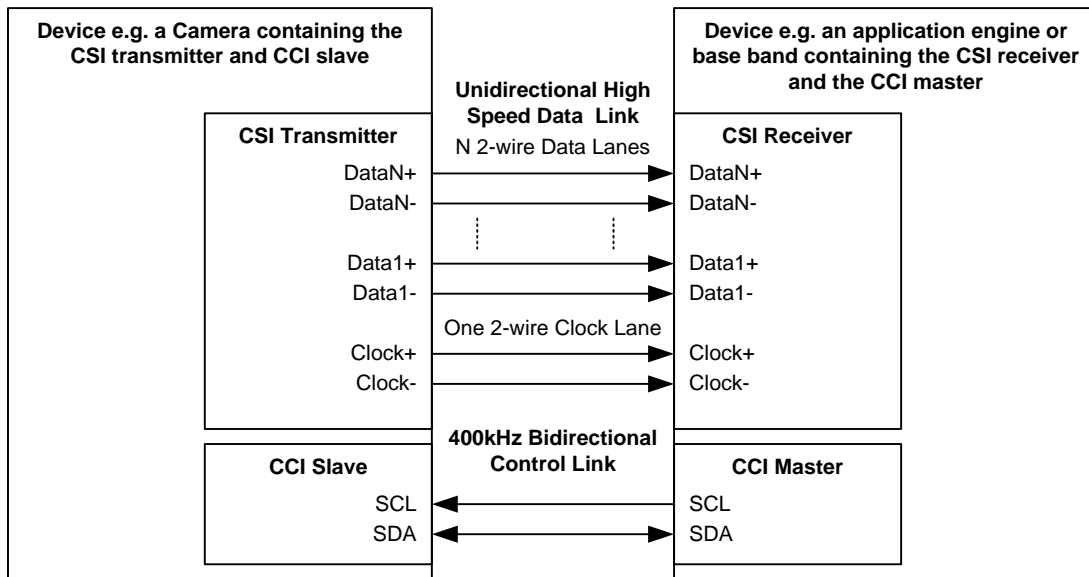
Figure 1 CSI-2 and CCI Transmitter and Receiver Interface for D-PHY
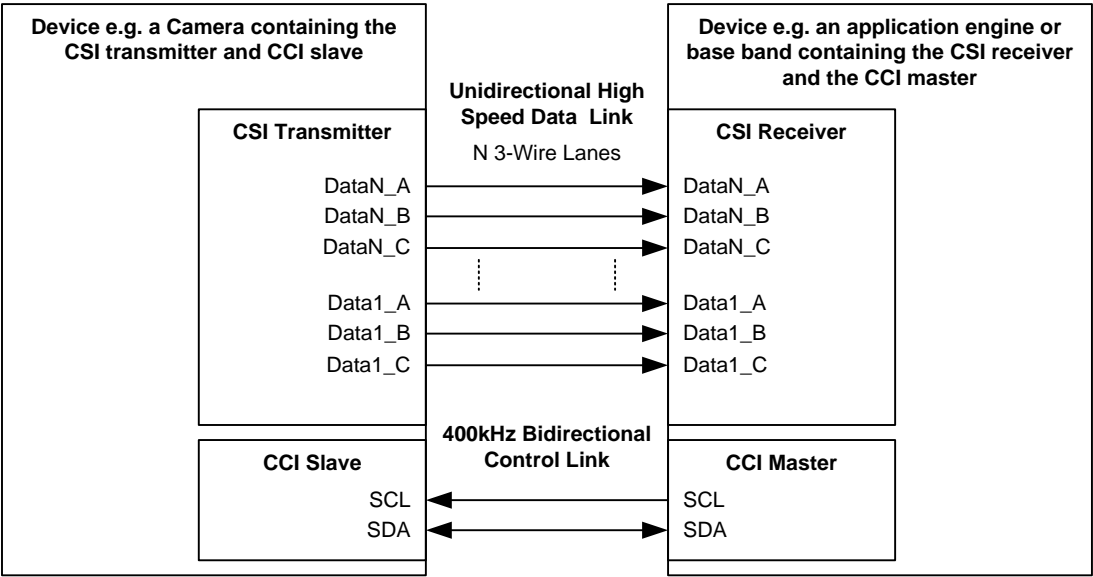
Figure 2 CSI-2 and CCI Transmitter and Receiver Interface for C-PHY

5   CSI-2 Layer Definitions

Transmitter   **Receiver**
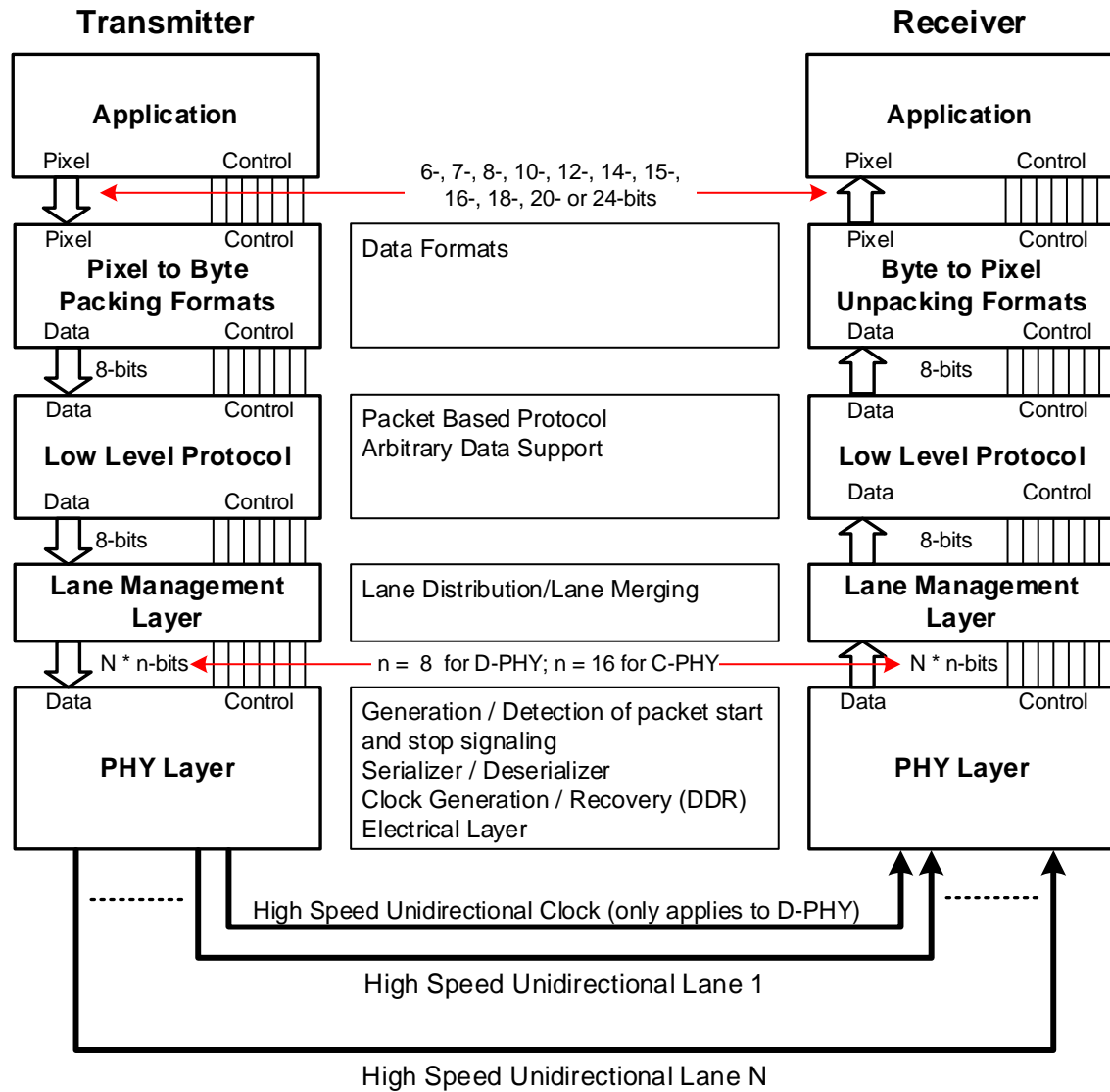
Figure 3 CSI-2 Layer Definitions

Figure 3 defines the conceptual layer structure used in CSI-2. The layers can be characterized as follows:

- **PHY Layer.** The PHY Layer specifies the transmission medium (electrical conductors), the input/output circuitry and the clocking mechanism that captures "ones" and "zeroes" from the serial bit stream. This part of the Specification documents the characteristics of the transmission medium, electrical parameters for signaling and for the D-PHY physical layer option, the timing relationship between clock and data Lanes.

  The mechanism for signaling Start of Transmission (SoT) and End of Transmission (EoT) is specified as well as other "out of band" information that can be conveyed between transmitting and receiving PHYs. Bit-level and byte-level synchronization mechanisms are included as part of the PHY.

  The PHY layer is described in *[MIPI01]* and *[MIPI02]*.

165   • **Protocol Layer.** The Protocol layer is composed of several layers, each with distinct
166   responsibilities. The CSI-2 protocol enables multiple data streams using a single interface on the
167   host processor. The Protocol layer specifies how multiple data streams may be tagged and
168   interleaved so each data stream can be properly reconstructed.

169   • **Pixel/Byte Packing/Unpacking Layer.** The CSI-2 specification supports image applications
170   with varying pixel formats. In the transmitter this layer packs pixels from the Application layer
171   into bytes before sending the data to the Low Level Protocol layer. In the receiver this layer
172   unpacks bytes from the Low Level Protocol layer into pixels before sending the data to the
173   Application layer. Eight bits per pixel data is transferred unchanged by this layer.

174   • **Low Level Protocol.** The Low Level Protocol (LLP) includes the means of establishing bit-
175   level and byte-level synchronization for serial data transferred between SoT (Start of
176   Transmission) and EoT (End of Transmission) events and for passing data to the next layer. The
177   minimum data granularity of the LLP is one byte. The LLP also includes assignment of bit-value
178   interpretation within the byte, i.e. the "Endian" assignment.

179   • **Lane Management.** CSI-2 is Lane-scalable for increased performance. The number of data
180   Lanes is not limited by this specification and may be chosen depending on the bandwidth
181   requirements of the application. The transmitting side of the interface distributes ("distributor"
182   function) bytes from the outgoing data stream to one or more Lanes. On the receiving side, the
183   interface collects bytes from the Lanes and merges ("merger" function) them together into a
184   recombined data stream that restores the original stream sequence. For the C-PHY physical
185   layer option, this layer exclusively distributes or collects byte pairs (i.e. 16-bits) to or from the
186   data Lanes. Scrambling on a per-Lane basis is an optional feature, which is specified in detail in
187   ***Section 9.15***.

188 Data within the Protocol layer is organized as packets. The transmitting side of the interface
189 appends header and error-checking information on to data to be transmitted at the Low Level
190 Protocol layer. On the receiving side, the header is stripped off at the Low Level Protocol layer
191 and interpreted by corresponding logic in the receiver. Error-checking information may be used to
192 test the integrity of incoming data.

193 • **Application Layer.** This layer describes higher-level encoding and interpretation of data
194   contained in the data stream and is beyond the scope of this specification. The CSI-2 Specification
195   describes the mapping of pixel values to bytes.

196 The normative sections of the Specification only relate to the external part of the Link, e.g. the data and bit
197 patterns that are transferred across the Link. All internal interfaces and layers are purely informative.

6   Camera Control Interface (CCI)

CCI is a two-wire, bi-directional, half duplex, serial interface for controlling the transmitter. CCI is compatible with I^{2}C Fast-mode (Fm) or Fast-mode Plus (Fm+) *[NXP01]* variants, and with the I3C *[MIPI03]* interface's Single Data Rate (SDR) or Double Data Rate (DDR) protocols. CCI shall support up to 400kbps (Fm) operation and 7-bit slave addressing. In addition, CCI can optionally support up to 1Mbps (Fm+), 12.5Mbps (SDR), or 25Mbps (DDR).

This Section uses the following terms:

- **CCI (I^{2}C)** means CCI supporting I^{2}C
- **CCI (I3C)** means CCI supporting I3C
- **CCI (I3C SDR)** means CCI supporting I3C SDR
- **CCI (I3C DDR)** means CCI supporting I3C DDR
- **CCI** alone (without following parentheses) means both **CCI (I^{2}C)** and **CCI (I3C)**.

CCI can be used with or without CSI-2 over C/D-PHY. When CCI is used as part of a CSI-2 bus, a CSI-2 receiver shall be configured as a master and a CSI-2 transmitter shall be configured as a slave. When CCI is used without CSI-2 over C/D-PHY, the host should be used as a master. CCI is capable of handling multiple slaves on the bus.

In **CCI (I^{2}C)**, multi-master mode is not supported. Any I^{2}C commands not described in this section shall be ignored, and shall not cause unintended device operation.

In **CCI (I3C)**, any I3C mandatory functions and 'Required' CCC commands shall be supported, and any I3C optional functions and commands may be supported (e.g. Multi-Master, In-Band Interrupt, Hot-Join).

Note:

Do not confuse the CCI terms master and slave with similar terms in the C-PHY or D-PHY Specifications; they are not related.

Typically, there is a dedicated CCI interface between the transmitter and the receiver.

CCI is a subset of the I^{2}C or I3C protocol that includes the minimum combination of obligatory features for I^{2}C/I3C slave devices specified in the I^{2}C or I3C specification. Therefore, transmitters complying with the CCI specification can also be connected to the system I^{2}C or I3C bus. However, care must be taken so that I^{2}C or I3C masters do not attempt to use I^{2}C or I3C features not supported by CCI masters or slaves.

A CCI transmitter may have additional features to support I^{2}C or I3C, but that is implementation-dependent. Further details can be found on a particular device's data sheet.

This specification does not attempt to define the contents of control Messages sent by the CCI master. Therefore, it is the responsibility of the implementer to define a set of control Messages and corresponding frame timing and any I^{2}C or I3C latency requirements that the CCI master must meet when sending such control Messages to the CCI slave.

CCI defines an additional data protocol layer on top of I^{2}C or I3C, as specified in the following sections.

6.1   CCI (I^{2}C) Data Transfer Protocol

232   The **CCI (I^{2}C)** data transfer protocol follows the I^{2}C specification. The START, REPEATED START, and
233   STOP conditions, and the data transfer protocol, are all specified in *[NXP01]*.

6.1.1   CCI (I^{2}C) Message Type

234   A basic **CCI (I^{2}C)** Message consists of:

235   • START or Repeated START condition

236   • Slave address with read/write bit

237   • Acknowledge from slave

238   • Sub address (INDEX) for pointing at a register inside the slave device (not used in Single Read
239   from Current Location)

240   • Acknowledge signal from slave (not used in Single Read from Current Location)

241   And then either:

242   • For a write operation:

243   • Data byte from master

244   • Acknowledge/negative acknowledge from slave, and

245   • STOP or Repeated START condition

246   Or:

247   • For a read operation:

248   • Repeated START condition (not used in Single Read from Current Location)

249   • slave address with read bit (not used in Single Read from Current Location)

250   • acknowledge signal from slave (not used in Single Read from Current Location)

251   • data byte from the slave

252   • acknowledge or negative acknowledge from the master, and

253   • STOP or Repeated START condition.

254   A CCI Slave may support back-to-back Messages by using Repeated START between CCI Messages
255   instead of START and/or STOP as shown in this Section.

256   The slave address in **CCI (I^{2}C)** is 7 bits long.

257   **CCI (I^{2}C)** supports an 8-bit INDEX with 8-bit data, or a 16-bit INDEX with 8-bit data. The slave device in
258   question defines what Message type is used.

6.1.2   CCI (I²C) Read/Write Operations

259   A **CCI (I²C)** compatible device shall support the four read operations and two write operations shown in
260   *Table 1*, as detailed in the following sub-sections:

261   **Table 1 CCI (I2C) Read/Write Operations**

Type	Operation	Section
Read	Single Read from Random Location	*6.1.2.1*
	Sequential Read from Random Location	*6.1.2.2*
	Single Read from Current Location	*6.1.2.3*
	Sequential Read from Current Location	*6.1.2.4*
Write	Single Write to Random Location	*6.1.2.5*
	Sequential Write Starting from Random Location	*6.1.2.6*

262   The INDEX in the slave device must be auto-incremented after each read/write operation. This is also
263   explained in the following sections.

6.1.2.1   CCI (I²C) Single Read from Random Location

264   In a single read from a random location (see *Figure 4*) the master does a dummy write operation to the
265   desired INDEX, issues a Repeated START condition, and then addresses the slave again with the read
266   operation. After acknowledging its slave address, the slave starts to output data onto the SDA line. The
267   master terminates the read operation by setting a negative acknowledge and a STOP or Repeated START
268   condition.

269

Figure 4 CCI (I²C) Single Read from Random Location

6.1.2.2 CCI (I²C) Single Read from Current Location

270 It is also possible to read from the last used INDEX, by addressing the slave with a read operation (see
271 *Figure 5*). The slave responds by sending the data from the last used INDEX to the SDA line. The master
272 terminates the read operation by setting a negative acknowledge and a STOP or Repeated START
273 condition.

274
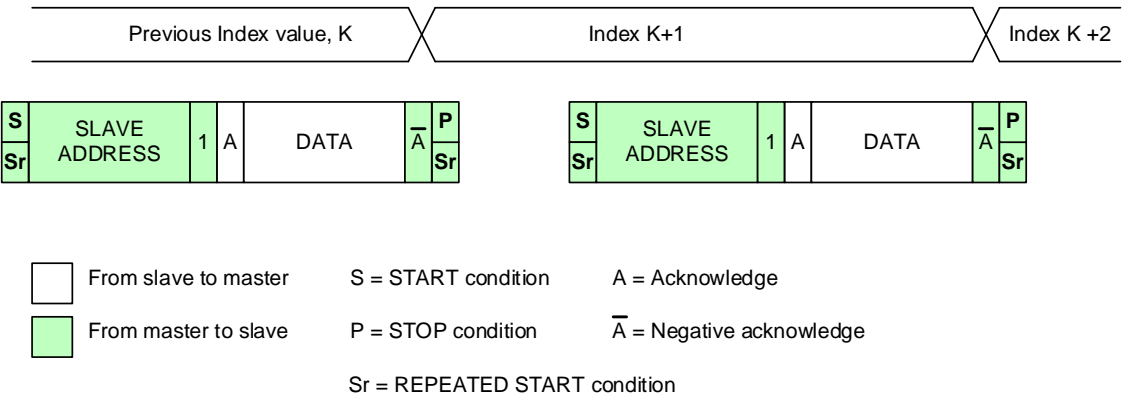
Figure 5 CCI (I²C) Single Read from Current Location

6.1.2.3 CCI (I²C) Sequential Read Starting from Random Location

275   Sequential read starting from a random location is illustrated in *Figure 6*. The master does a dummy write
276   to the desired INDEX, issues a Repeated START condition after an acknowledge from the slave, and then
277   addresses the slave again with a read operation. If a master issues an acknowledge after receiving data, this
278   acts as a signal to the slave that the read operation is to continue from the next INDEX. When the master
279   has read the last data byte, it issues a negative acknowledge and a STOP or Repeated START condition.

280
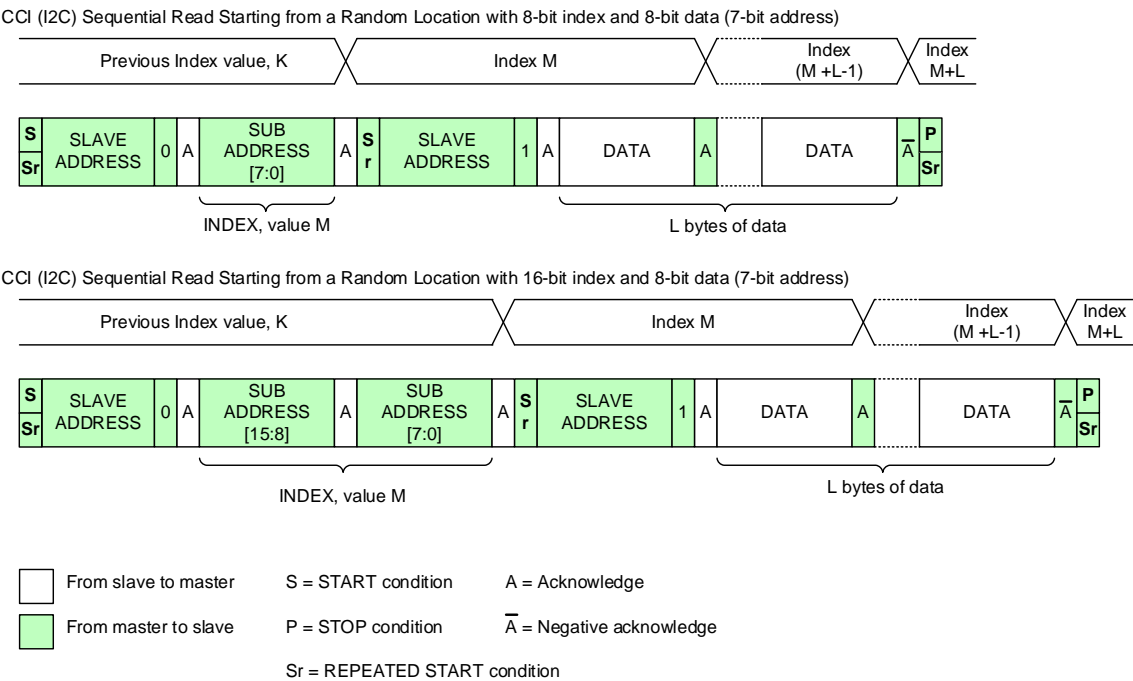
Figure 6 CCI (I²C) Sequential Read Starting from Random Location

6.1.2.4   CCI (I²C) Sequential Read Starting from Current Location

281   A sequential read starting from the current location (see **Figure 7**) is similar to a sequential read from a
282   random location. The only exception is there is no dummy write operation. The master terminates the read
283   operation by issuing a negative acknowledge, and a STOP or Repeated START condition.

284
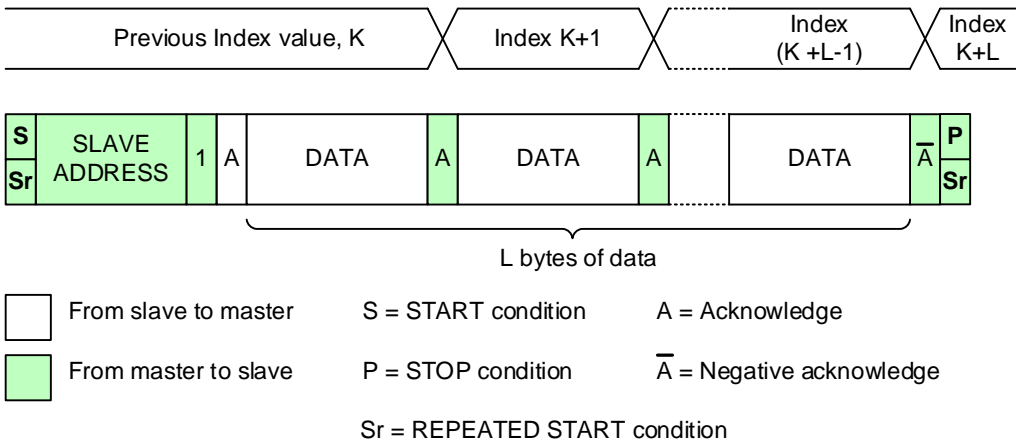
Figure 7 CCI (I²C) Sequential Read Starting from Current Location

Copyright © 2005-2018 MIPI Alliance, Inc.
All rights reserved.

6.1.2.5 CCI (I²C) Single Write to Random Location

285 A write operation to a random location is illustrated in *Figure 8*. The master issues a write operation to the
286 slave, then issues the INDEX and data after the slave has acknowledged the write operation. The write
287 operation is terminated with a stop or Repeated START condition from the master.

CCI (I2C) Single Write to a Random Location with 8-bit index and 8-bit data (7-bit address)

CCI (I2C) Single Write to a Random Location with 16-bit index and 8-bit data (7-bit address)
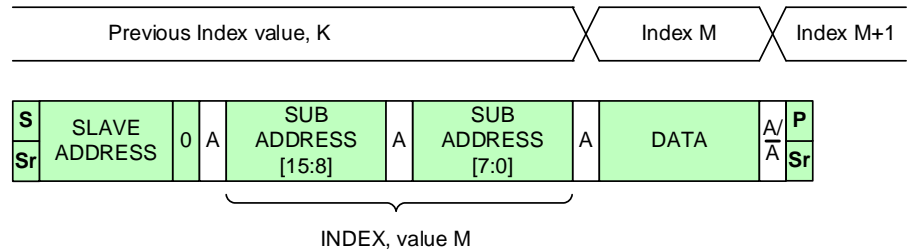
288

Figure 8 CCI (I²C) Single Write to Random Location

6.1.2.6 CCI (I²C) Sequential Write Starting from Random Location

289 The Sequential Write Starting from Random Location operation is illustrated in *Figure 9*. The slave auto-
290 increments the INDEX after each data byte is received. The Sequential Write Starting from Random
291 Location operation is terminated with a STOP or Repeated START condition from the master.

CCI (I2C) Sequential Write Starting from a Random Location with 8-bit index and 8-bit data (7-bit address)

CCI (I2C) Sequential Write Starting from a Random Location with 16-bit index and 8-bit data (7-bit address)

From slave to master	S = START condition
From master to slave	P = STOP condition
	Sr = REPEATED START condition

A = Acknowledge

$\overline{A}$ = Negative acknowledge

292
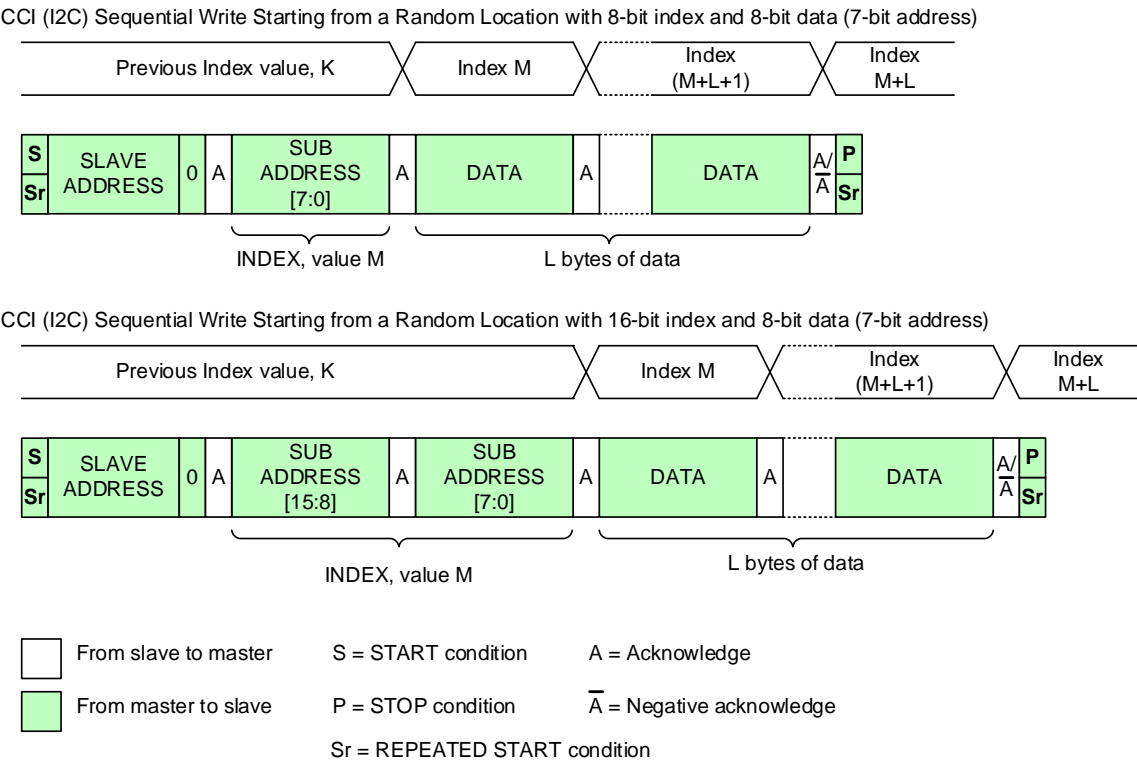
Figure 9 CCI (I²C) Sequential Write Starting from Random Location

6.2 CCI (I3C) Data Transfer Protocol

293 The **CCI (I3C)** data transfer protocol follows the I3C Specification. The START, Repeated START, and
294 STOP conditions, as well as data transfer protocol, are specified in *[MIPI03]*.

295 If **CCI (I3C)** is supported, then CCI **(I3C SDR)** shall be supported and **CCI (I3C DDR)** may be supported.
296 The master shall get the slave's Max Read Length (MRL) and Max Write Length (MWL) via transmitting
297 I3C CCCs GETMRL and GETMWL prior to **CCI (I3C)** data transfer.

6.2.1 CCI (I3C SDR) Data Transfer Protocol

6.2.1.1 CCI (I3C SDR) Message Type

298 The **CCI (I3C SDR)** master normally should start a Message with 7'h7E, and may choose to start a
299 Message with a slave address.

300 A basic **CCI (I3C SDR)** Message starting a Message with 7'h7E consists of:

301 • START condition
302 • 7'h7E with write bit
303 • Acknowledge from slave
304 • Repeated START condition
305 • Slave address with read/write bit
306 • Acknowledge from slave
307 • Sub-address (INDEX) of a register inside the slave device (not used in Single Read from Current
308   Location)
309 • Transition bit (Parity bit) from master (not used in Single Read from Current Location)
310   And then either:
311 • For a write operation:
312   • Data byte from master
313   • Transition bit (Parity bit) from master
314   • STOP or Repeated START condition;
315   Or
316 • For a read operation:
317   • Repeated START condition (not used in Single Read from Current Location)
318   • Slave address with read bit (not used in Single Read from Current Location)
319   • Acknowledge from slave (not used in Single Read from Current Location)
320   • Data byte from slave
321   • Transition bit (End-of-Data) from master or slave
322   • STOP or Repeated START condition.

Other **CCI (I3C SDR)** Messages starting a Message with a slave address consist of:

- START or Repeated START condition
- Slave address with read/write bit
- Acknowledge from slave
- Sub-address (INDEX) of a register inside the slave device (not used in Single Read from Current Location)
- Transition bit (Parity bit) from master (not used in Single Read from Current Location)

And then either:

- For a write operation:
  - Data byte from master
  - Transition bit (Parity bit) from master
  - STOP or Repeated START condition;

Or:

- For a read operation:
  - Repeated START condition (not used in Single Read from Current Location)
  - Slave address with read bit (not used in Single Read from Current Location)
  - Acknowledge from slave (not used in Single Read from Current Location)
  - Data byte from slave
  - Transition bit (End-of-Data) from master or slave
  - STOP or Repeated START condition.

The slave address in **CCI (I3C SDR)** is 7 bits long.

CCI (I3C SDR) supports an 8-bit INDEX with 8-bit data, or a 16-bit INDEX with 8-bit data. The slave device in question defines what Message type is used.

6.2.1.2 CCI (I3C SDR) Read/Write Operations

A **CCI (I3C SDR)** compatible device shall support the four read operations and two write operations shown in *Table 2*, as detailed in the following sub-sections:

Table 2 CCI (I3C SDR) Read/Write Operations

Type	Operation	Section
Read	Single Read from Random Location	*6.2.1.2.1*
	Single Read from Current Location	*6.2.1.2.2*
	Sequential Read from Random Location	*6.2.1.2.3*
	Sequential Read from Current Location	*6.2.1.2.4*
Write	Single Write to Random Location	*6.2.1.2.5*
	Sequential Write Starting from Random Location	*6.2.1.2.6*

The INDEX in the slave device must be auto-incremented after each read/write operation. This is also explained in the following sections.

6.2.1.2.1   CCI (I3C SDR) Single Read from Random Location

351 In a single read from a random location (*Figure 10*), the master does a dummy write operation to the
352 desired INDEX, issues a Repeated START condition, and then addresses the slave again with the read
353 operation. After acknowledging its slave address, the slave starts to output data onto the SDA line. The
354 master aborts the read operation by setting a Transition bit, and a STOP or Repeated START condition.

355
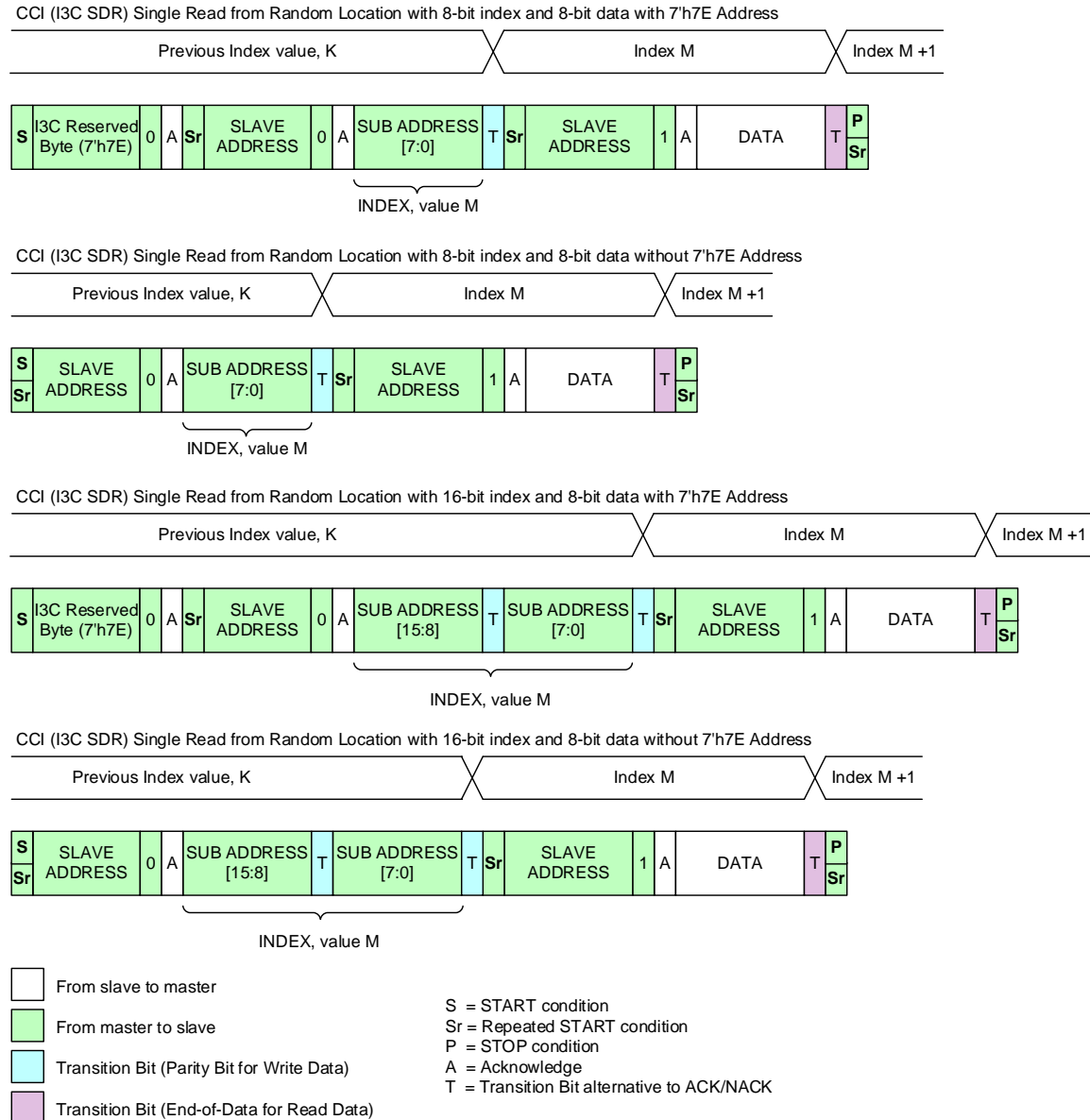
Figure 10 CCI (I3C SDR) Single Read from Random Location

6.2.1.2.2   CCI (I3C SDR) Single Read from Current Location

356   It is also possible to read from the last used INDEX by addressing the slave with a read operation (***Figure***
357   ***11***). The slave responds by setting the data from last used INDEX to SDA line. The master aborts the read
358   operation by setting a Transition bit, and a STOP or Repeated START condition.
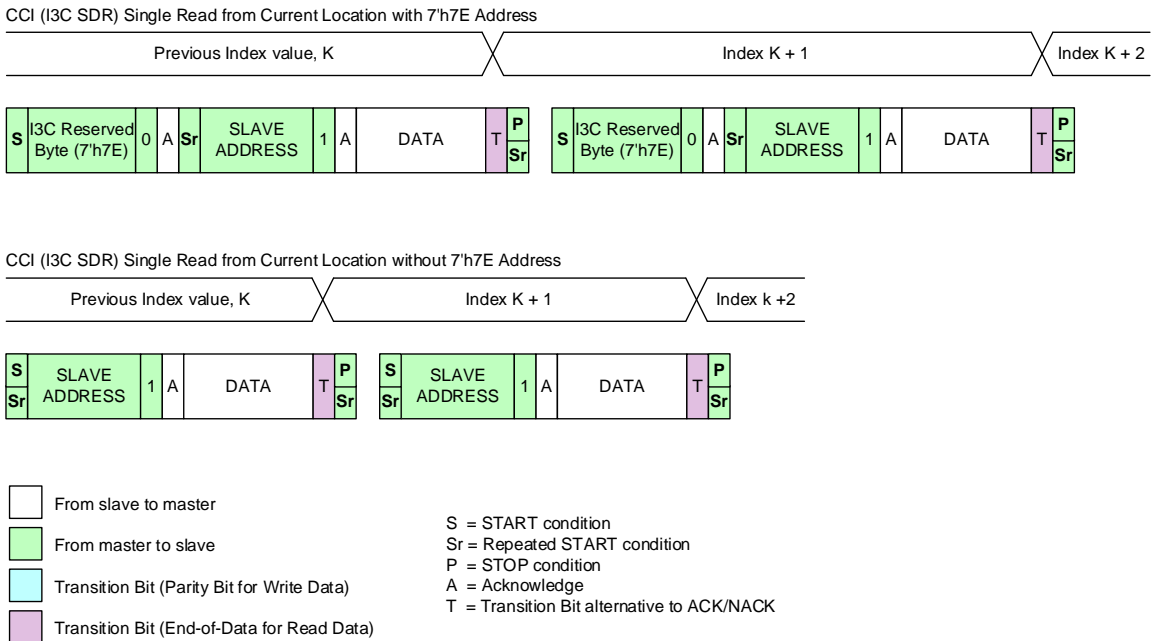
Figure 11 CCI (I3C SDR) Single Read from Current Location

6.2.1.2.3　CCI (I3C SDR) Sequential Read from Random Location

360　The sequential read starting from a random location is illustrated in *Figure 12*. The master does a dummy
361　write operation to the desired INDEX, issues a Repeated START condition, and then addresses the slave
362　again with the read operation. After acknowledging its slave address, the slave starts to output data onto the
363　SDA line. If a master doesn't abort the read transaction by using the transition bit, this acts as a signal for
364　the slave to continue a read operation from the next INDEX. When the master has read the last data byte, it
365　can abort a read transaction by setting the transition bit and then issuing a STOP or Repeated START
366　condition. Furthermore, when the master reads a large amount of data exceeding the Max Read Length
367　(MRL) limit (see the I3C Specification *[MIPI03]*), the slave can also terminate a read transaction by setting
368　the transition bit.

369　***Note:***

370　　　*When selecting a suitable value for MRL, the designer of the slave device and the system designer*
371　　　*should take into account the needs of the payload that the CCI will carry. For example, in the CCS*
372　　　*Data Transfer Interface **[MIPI04]**, it is beneficial to support an MRL of 64 bytes or larger (i.e. 64*
373　　　*bytes for Data payload).*

374
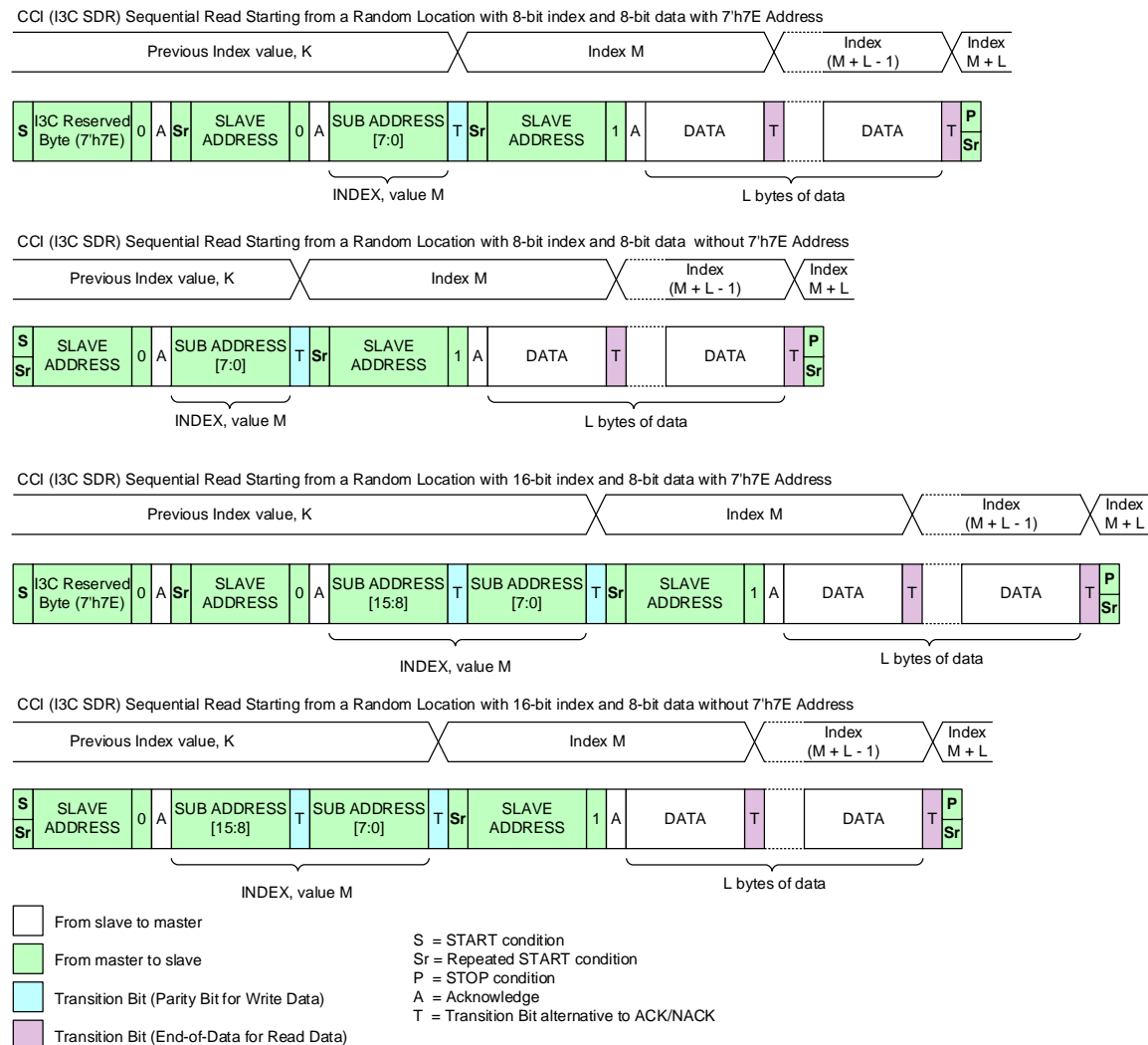
Figure 12 CCI (I3C SDR) Sequential Read Starting from Random Location

6.2.1.2.4 CCI (I3C SDR) Sequential Read from Current Location

A sequential read starting from the current location (*Figure 13*) is similar to a sequential read from a random location. The only exception is when there is no dummy write operation. The master or slave terminates a read transaction by setting the transition bit, and then issues a STOP or Repeated START condition.

CCI (I3C SDR) Sequential Read Starting from the Current Location with 7'h7E Address

CCI (I3C SDR) Sequential Read Starting from the Current Location without 7'h7E Address
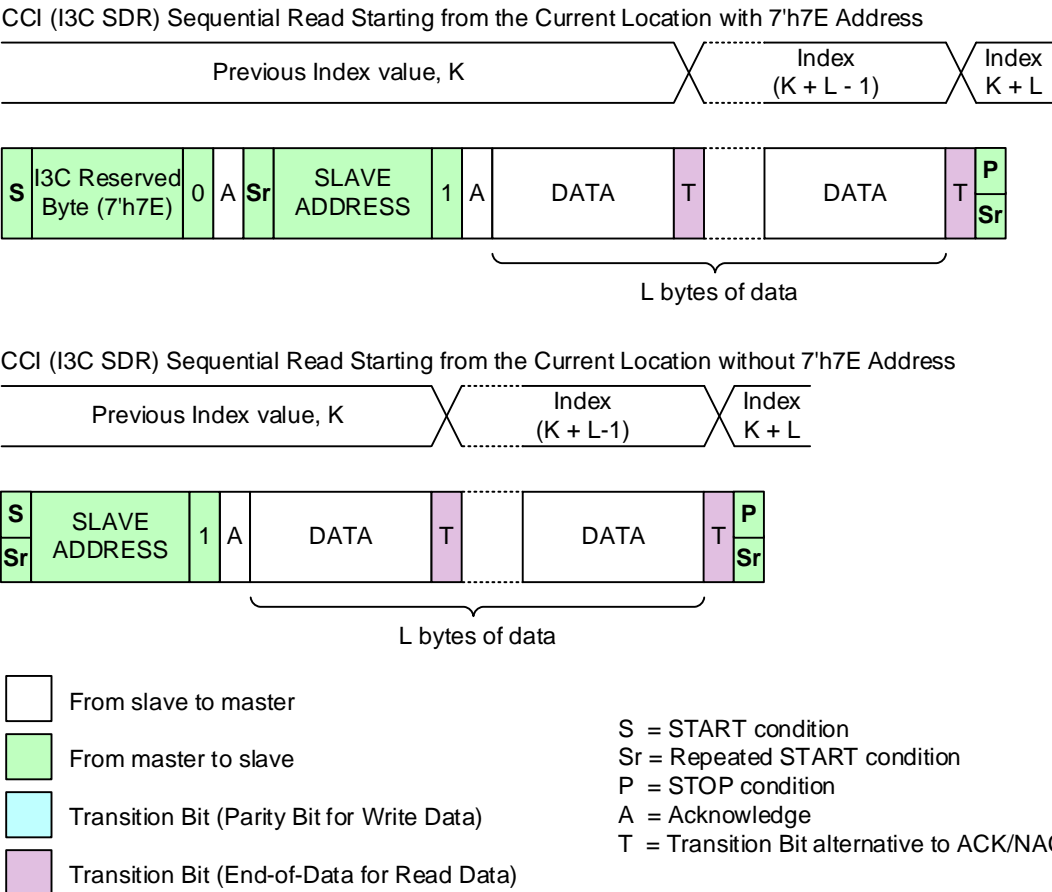
Figure 13 CCI (I3C SDR) Sequential Read Starting from Current Location

6.2.1.2.5 CCI (I3C SDR) Single Write to Random Location

380 A write operation to a random location is illustrated in *Figure 14*. The master issues a write operation to the
381 slave, then issues the INDEX and data after the slave has acknowledged the write operation. The write
382 operation is terminated with a STOP or Repeated START condition from the master.
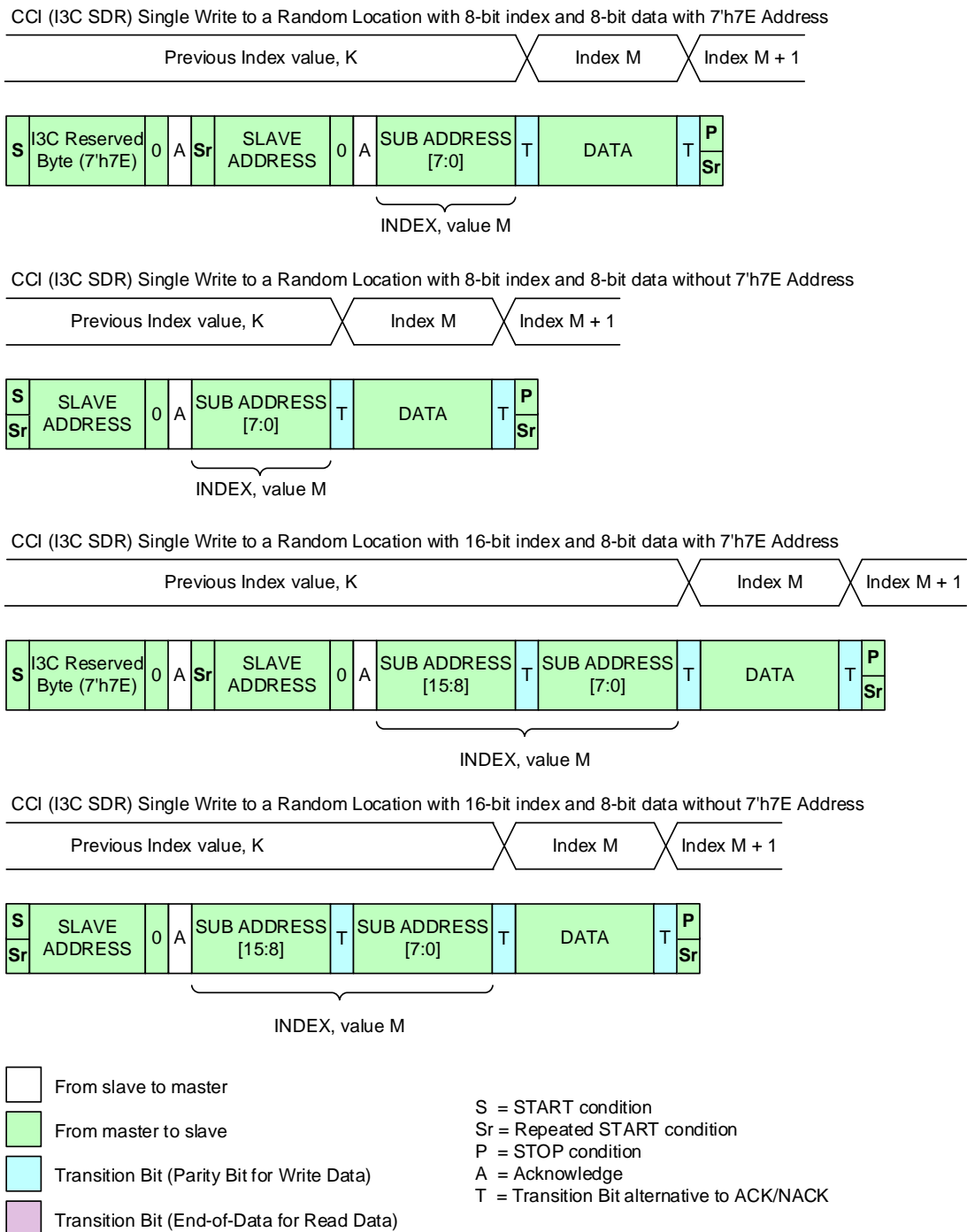
Figure 14 CCI (I3C SDR) Single Write to Random Location

6.2.1.2.6   CCI (I3C SDR) Sequential Write Starting from Random Location

384 The Sequential Write Starting from Random Location operation is illustrated in *Figure 15*. The slave auto-
385 increments the INDEX after each data byte is received. The Sequential Write Starting from Random
386 Location operation is terminated with a STOP or Repeated START condition from the master.

CCI (I3C SDR) Sequential Write Starting from a Random Location with 8-bit index and 8-bit data with 7'h7E Address

CCI (I3C SDR) Sequential Write Starting from a Random Location with 8-bit index and 8-bit data without 7'h7E Address

CCI (I3C SDR) Sequential Write Starting from a Random Location with 16-bit index and 8-bit data with 7'h7E Address

CCI (I3C SDR) Sequential Write Starting from a Random Location with 16-bit index and 8-bit data without 7'h7E Address

From slave to master

From master to slave

Transition Bit (Parity Bit for Write Data)

Transition Bit (End-of-Data for Read Data)

S = START condition
Sr = Repeated START condition
P = STOP condition
A = Acknowledge
T = Transition Bit alternative to ACK/NACK

387

Figure 15 CCI (I3C SDR) Sequential Write Starting from Random Location

6.2.2   CCI (I3C DDR) Data Transfer Protocol

6.2.2.1   CCI (I3C DDR) Message Type

388 The **CCI (I3C DDR)** master shall start a DDR Message with either the I3C ENTHDR0 CCC, or the I3C
389 HDR Restart Pattern. The **CCI (I3C DDR)** master shall end a DDR Message by issuing either the I3C
390 HDR Restart Pattern, or the I3C HDR Exit Pattern.

391 Two Message types are defined for DDR Messages: DDR Write Message and DDR Read Message.

392 **CCI (I3C DDR)** supports either:

393   • 8-bit LENGTH and 8-bit INDEX with 8-bit data

394   Both the LENGTH and the INDEX shall be included in the first data word of the DDR
395   Write Message.

396 or:

397   • 16-bit LENGTH and 16-bit INDEX with 8-bit data

398   The LENGTH shall be included in the first data word of the DDR Write Message, and the
399   INDEX shall be included in the second data word of the DDR Write Message.

400 The slave device in question defines what Message type is used.

401 The LENGTH field defines the number of 8-bit data bytes in the Read or Write Data Words. The LENGTH
402 field is zero-based, i.e. if the master wishes to read or write N bytes, then the value in the LENGTH field
403 must be N–1.

Examples:

404   • 0 LENGTH means 1 byte
405   • 255 LENGTH means 256 bytes

406 When a multi-byte register is accessed via **CCI (I3C DDR)**, the transmission byte order described in
407 *Section 6.6* shall be the same as for **CCI (I²C)** and **CCI (I3C SDR)**.

Example:

408   For the 16-bit register read shown in *Figure 17*, the DATA0 byte contains bits Data[15:8] and the
409   DATA1 byte contains bits Data[7:0].

6.2.2.2   CCI (I3C DDR) Read/Write Operations

410 A **CCI (I3C DDR)** compatible device shall support the two read operations and one write operation shown
411 in *Table 3*, as detailed in the following sub-sections:

412   **Table 3 CCI (I3C DDR) Read/Write Operations**

Type	Operation	Section
Read	Sequential Read from Random Location	*6.2.2.2.2*
	Concatenated Sequential Read from Random Location	*6.2.2.2.3*
Write	Sequential Write Starting from Random Location	*6.2.2.2.4*

413 The INDEX in the slave device must be auto-incremented after each read/write operation. This is also
414 explained in the following sections.

6.2.2.2.1   CCI (I3C DDR) Command Definitions

415 As defined in the I3C Specification *[MIPI03]*, bit[15] of the HDR-DDR Command Word is the R/W bit
416 and bits[14:8] contain the Command Code. Command Code values are reserved per application, and **CCI**
417 **(I3C DDR)** defines one such Command Code: 7'b0000000.

418 This single Command Code is sufficient, because the slave can still distinguish between three different
419 R/W operations. Consider the example of 16-bit LENGTH and 16-bit INDEX:

- If the slave receives a Data Word greater than 4 bytes, then the operation is "Sequential Write
  Starting from Random Location".
- If the slave receives a Data Word of 4 bytes before the HDR Restart Pattern, then there are two
  possibilities:
  - If the value of the LENGTH field is ≤ MRL–1, then the operation is "Sequential Read Starting
    from a Random Location".
  - If the value of the LENGTH field is > MRL–1, then the operation is "Concatenated Sequential
    Read Starting from a Random Location".

Copyright © 2005-2018 MIPI Alliance, Inc.
Confidential

428 ***Table 4*** defines the I3C HDR-DDR Command Codes (including R/W bit) for each **CCI (I3C DDR)**
429 Read/Write operation.

430 For **CCI (I3C DDR)**, the slave address is 7 bits long, and appears in bits[7:1] of the HDR-DDR Command
431 Word.

432 **Table 4 CCI (I3C DDR) Read/Write Operation Command Codes**

Type	Operation	Command Code Position	R/W Bit and Command Code *See Note 1*	Section
Write	Sequential Write Starting from Random Location	Command Word	0x00	***6.2.2.2.4***
Read	Sequential Read Starting from Random Location	Command Word for LENGTH & INDEX	0x00	***6.2.2.2.2***
		Command Word for ReadData	0x80	
	Concatenated Sequential Read Starting from Random Location	Command Word for LENGTH & INDEX	0x00	***6.2.2.2.3***
		Command Word for ReadData	0x80	

Note:

1. In all five cases, the 7-bit Command Code in the low seven bits is 7'b0000000. Only the R/W bit, which is the high bit of the byte, changes.

6.2.2.2.2 CCI (I3C DDR) Sequential Read From Random Location

433 In a sequential read from a random location (***Figure 16*** and ***Figure 17***):

434 • The master shall transmit:
435 • The HDR-DDR Command Word for LENGTH and INDEX
436 • The HDR-DDR Data Word, including LENGTH and INDEX
437 • The HDR-DDR CRC Word
438 • The HDR Restart Pattern
439 • The HDR-DDR Command Word for ReadData
440 • Then the slave shall send one or more HDR-DDR Read Data Words followed by the HDR-DDR
441 CRC Word
442 • Finally the master shall send either the HDR Restart Pattern or the HDR Exit Pattern.

443 If the number of 8-bit data words read is odd (i.e. the value in the LENGTH field is even), then the slave
444 shall insert one padding byte in the second byte of the last data word, with value 8'h00. The slave shall not
445 increment INDEX by the padding byte. The master shall take into account that the data includes the
446 padding byte in odd transfers, and that the INDEX is not incremented by the padding byte.

447 The master shall load the Sub Address into the INDEX and auto-increment the INDEX after each data byte
448 is received. The master can identify the padding byte from the value of the LENGTH field and the number
449 of the received 8-bit data words, and shall ignore the padding byte. Note that the INDEX is not incremented
450 by the padding byte.

CCI (I3C DDR) Sequential Read Starting from a Random Location with 8-bit length and 8-bit index (even byte read transfer)

CCI (I3C DDR) Sequential Read Starting from a Random Location with 8-bit length and 8-bit index (odd byte read transfer)

451
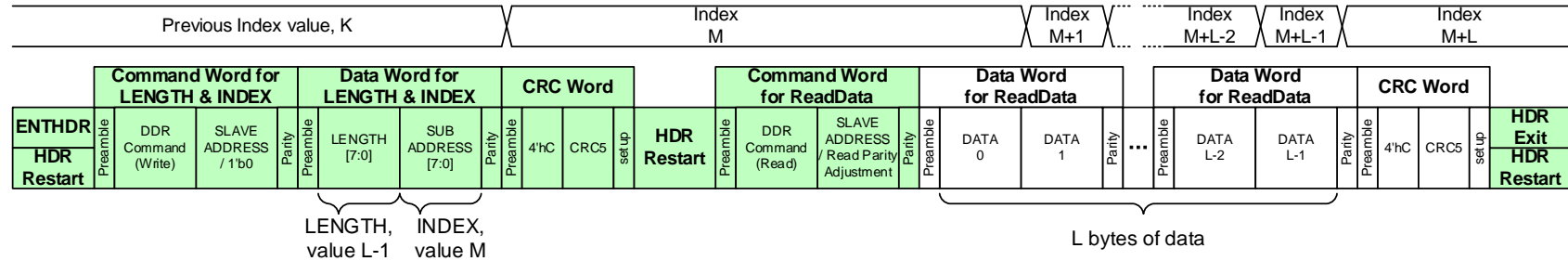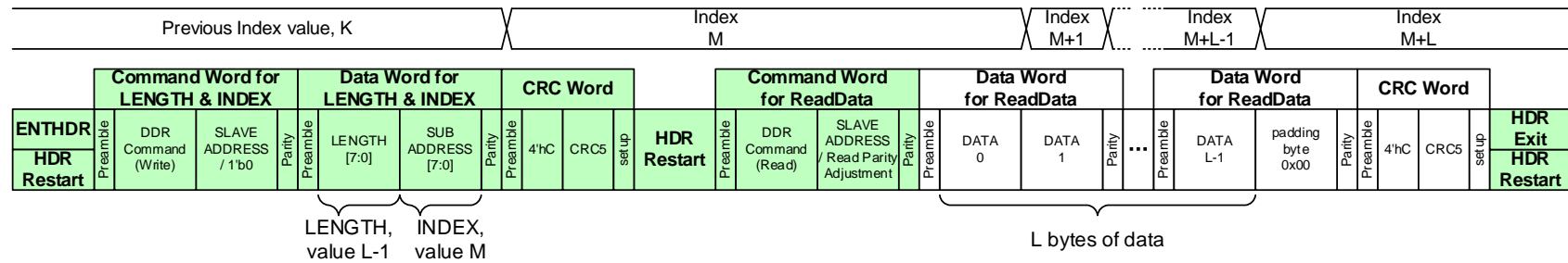
Figure 16 CCI (I3C DDR) Sequential Read from Random Location: 8-bit LENGTH & INDEX

CCI (I3C DDR) Sequential Read Starting from a Random Location with 16-bit length and 16-bit index (even byte read transfer)
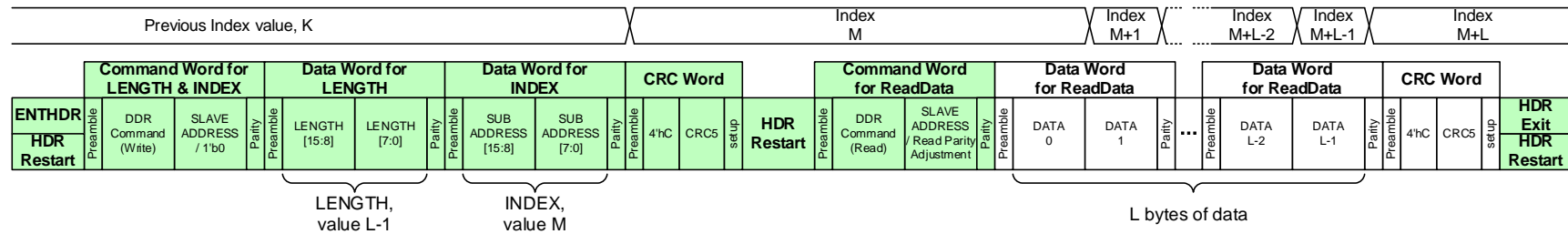
CCI (I3C DDR) Sequential Read Starting from a Random Location with 16-bit length and 16-bit index (odd byte read transfer)

452
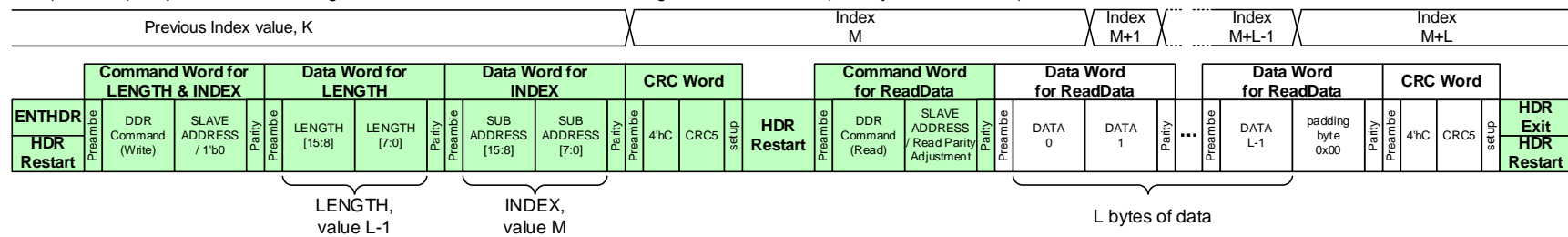
Figure 17 CCI (I3C DDR) Sequential Read from Random Location: 16-bit LENGTH & INDEX

6.2.2.2.3   CCI (I3C DDR) Concatenated Sequential Read from Random Location

When the master desires to read data longer than the slave's I3C Max Read Length (MRL) *[MIPI03]*, the master can divide the data into multiple units, and efficiently read the data using the Concatenated Sequential Read from Random Location operation (*Figure 18* and *Figure 19*). The master shall divide the data into multiple units, where all units except the last unit shall use the MRL size, and the last unit shall use a size less than or equal to the MRL. The MRL size is programmable.

In a Concatenated Sequential Read Starting from Random Location:

- The master shall first transmit the total LENGTH for the data to be read.

- The master shall use multiple read Messages. The slave shall transmit the initial read Messages to the master using the programmed MRL data bytes. And the slave may use no more than the programmed MRL data bytes to transfer the last Message.

  - If the full amount of requested data has not been received yet, then the master shall transmit another read Message, but without LENGTH and INDEX.

  - After receiving the read Message without LENGTH and INDEX, the slave shall continue transmission of the read data to the master, resuming from the previous LENGTH and INDEX.

The master shall continue to transmit read Messages without LENGTH and INDEX multiple times, until the last data is received.

Note:

When selecting a suitable value for MRL, the designer of the slave device and the system designer should take into account the needs of the payload that the CCI will carry. For example, in the CCS Data Transfer Interface [MIPI04], it is beneficial to support an MRL of 64 bytes or larger (i.e. 64 bytes for Data payload).

CCI (I3C DDR) Concatenated Sequential Read Starting from a Random Location with 8-bit length and 8-bit index
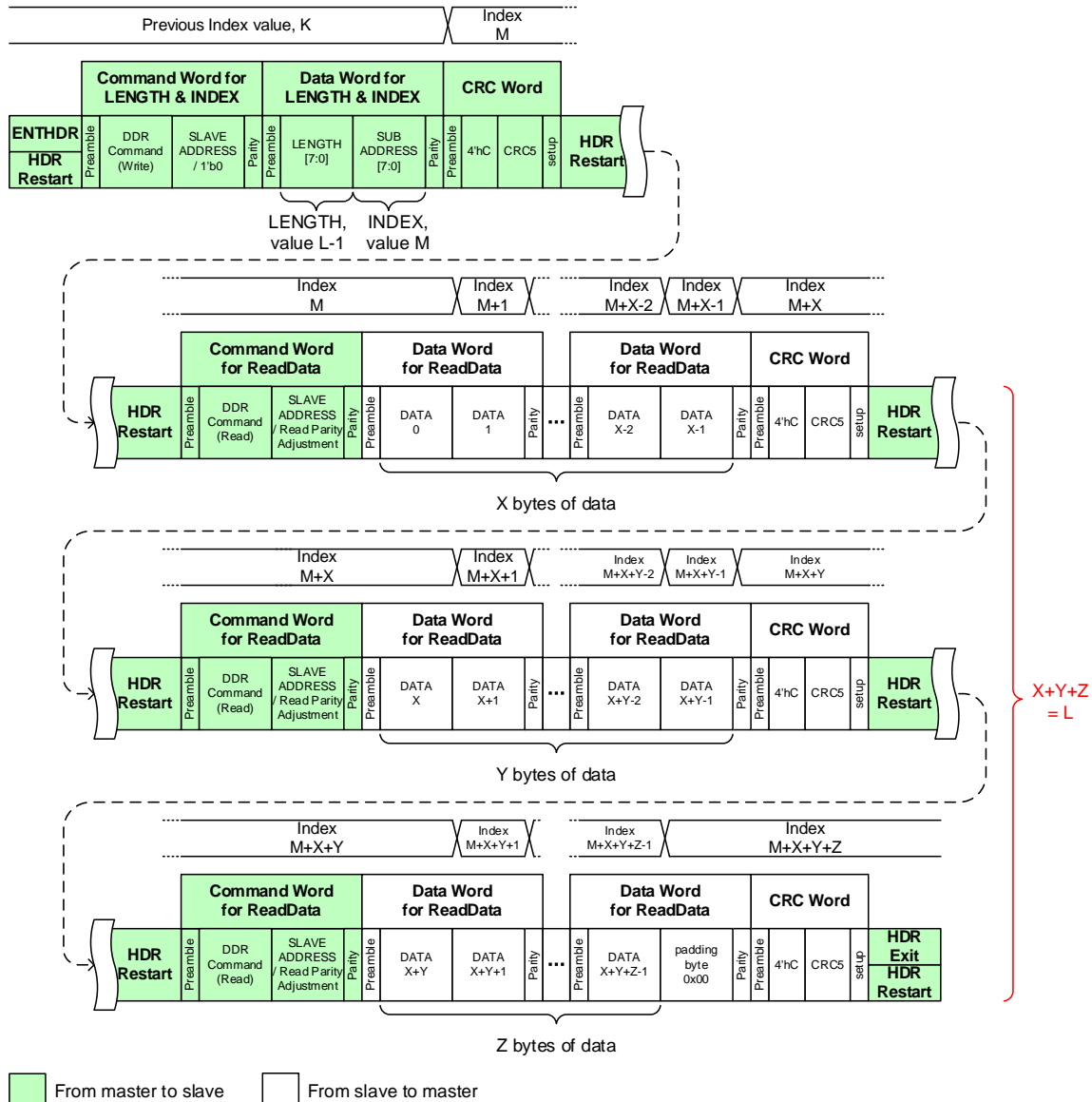
Figure 18 CCI (I3C DDR) Concatenated Sequential Read, Random Location: 8-bit LENGTH & INDEX

CCI (I3C DDR) Concatenated Sequential Read Starting from a Random Location with 16-bit length and 16-bit index

From master to slave   From slave to master
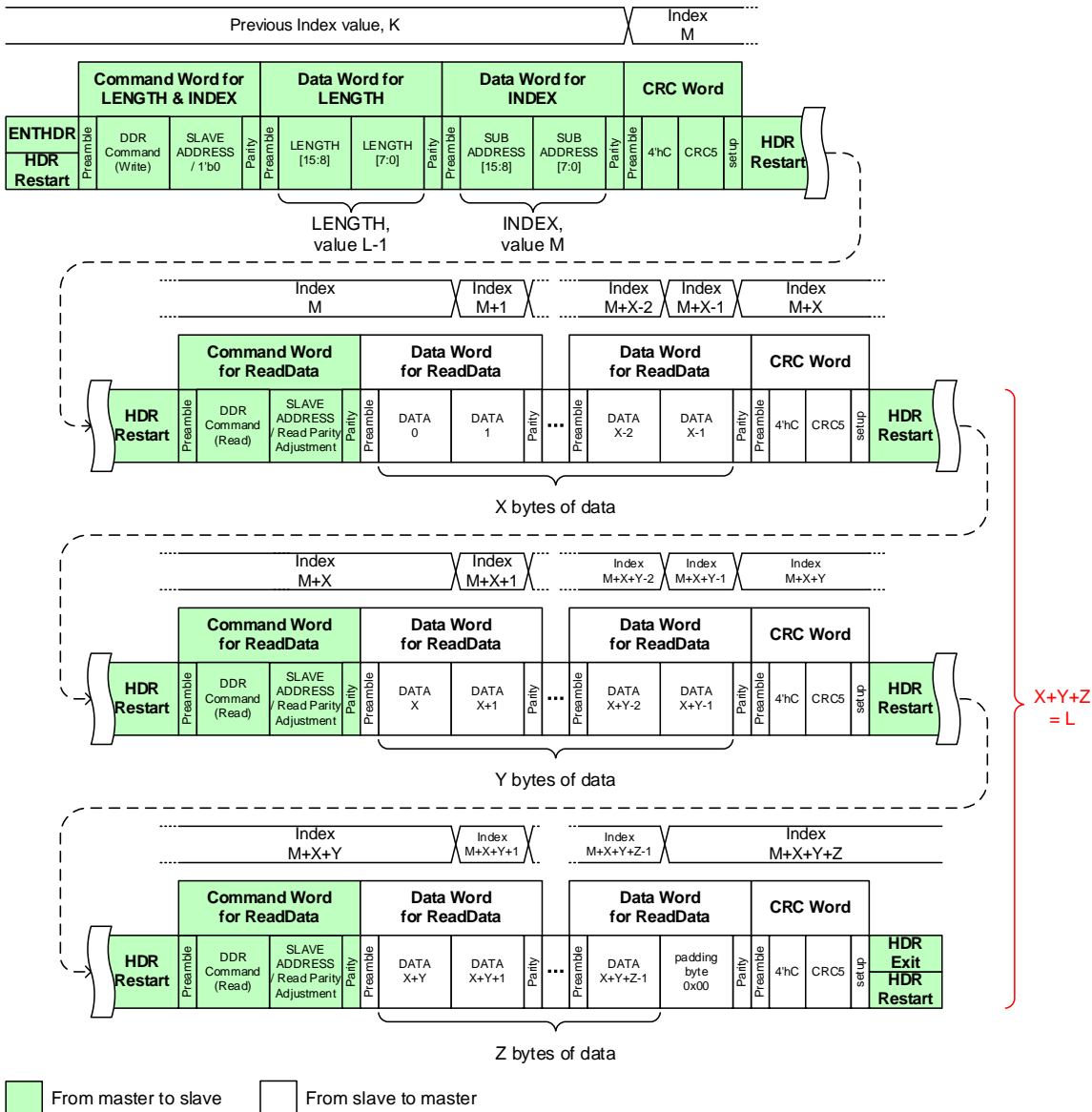
Figure 19 CCI (I3C DDR) Concatenated Sequential Read, Random Location: 16-bit LENGTH & INDEX

6.2.2.2.4 CCI (I3C DDR) Sequential Write Starting from Random Location

476 In a Sequential Write Starting from Random Location (*Figure 20*), the master shall transmit:

477 • The HDR-DDR Command Word

478 • The HDR-DDR Data Word including LENGTH and INDEX

479 • One or more HDR-DDR Write Data Words, and

480 • The HDR-DDR CRC Word.

481 If the number of 8-bit data words written is odd (i.e. the value in the LENGTH field is even), then the
482 master shall insert one padding byte in the second byte of the last data word, with value 8'h00. When the
483 slave receives the Sub Address, the slave loads it into the INDEX and auto-increments the INDEX after
484 each data byte is received.

485 The slave can identify the padding byte from the value of the LENGTH field and the number of 8-bit data
486 words received, and shall ignore the padding byte. Note that the INDEX is not incremented by the padding
487 byte.

488 In a Sequential Write Starting from Random Location, the value of LENGTH shall be set such that the
489 master does not exceed the maximum data byte length limit defined by the slave's I3C Max Write Length
490 (MWL) *[MIPI03]*. Note that the total number of bytes of "Data Word for INDEX", "Data Word for
491 LENGTH", and "Data Word for Write Data" shall not exceed MWL.

Example:

492 For a slave with MWL of 8 bytes, using 16-bit INDEX (so "Data Word for INDEX" is 2 bytes)
493 and 16-bit LENGTH (so "Data Word for LENGTH" is 2 bytes), the maximum number of "Data
494 Word for Write Data" is 8 – (2 + 2) bytes = 4 bytes. Since the LENGTH field is zero-based, it
495 would contain the value 3 (16'd3).

496 The slave cannot terminate the DDR Write Message, and shall receive all HDR-DDR Write Data sent by
497 the master.

498 ***Note:***

499 *When selecting a suitable value for MWL, the designer of the slave device and the system designer*
500 *should take into account the needs of the payload that the CCI will carry. For example, in the CCS*
501 *Data Transfer Interface **[MIPI04]**, it is beneficial to support an MWL of 68 bytes or larger (i.e. 64*
502 *bytes for Data payload + 2 bytes for a Data Word for INDEX + 2 bytes for a Data Word for*
503 *LENGTH).*

CCI (I3C DDR) Sequential Write to a Random Location with 8-bit length and 8-bit index (even byte write transfer)
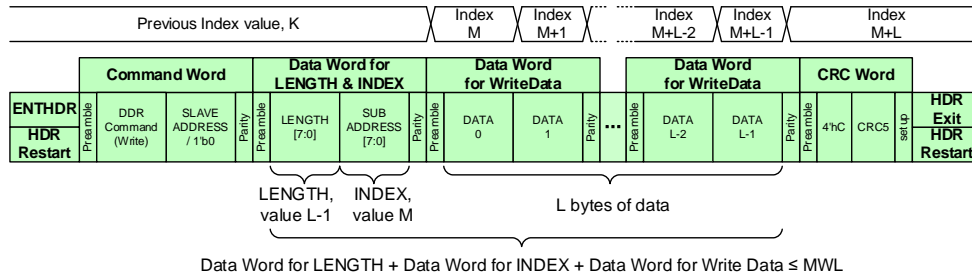
CCI (I3C DDR) Sequential Write to a Random Location with 8-bit length and 8-bit index (odd write byte transfer)
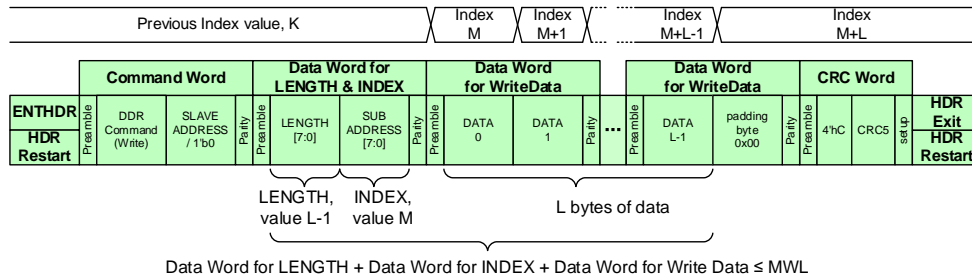
CCI (I3C DDR) Sequential Write to a Random Location with 16-bit length and 16-bit index (even byte write transfer)

CCI (I3C DDR) Sequential Write to a Random Location with 16-bit length and 16-bit index (odd write byte transfer)

504
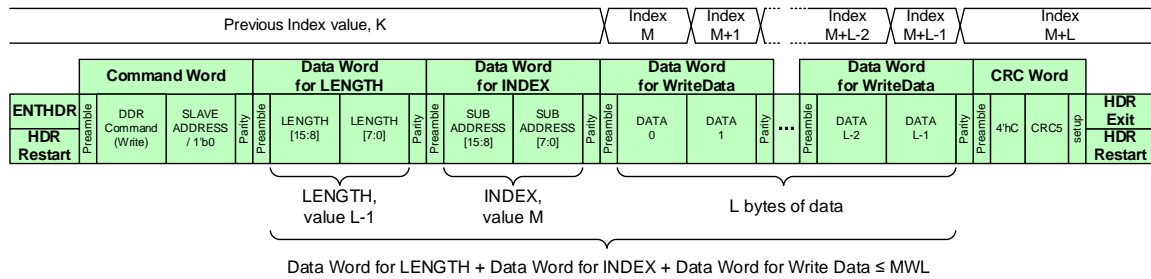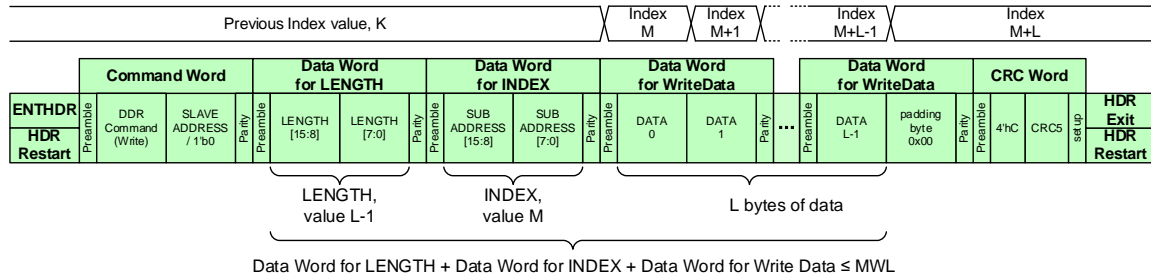
Figure 20 CCI (I3C DDR) Sequential Write Starting from Random Location

6.3 CCI (I3C) Error Detection and Recovery

6.3.1 CCI (I3C SDR) Error Detection and Recovery Method

The error detection and recovery methods specified in this Section are provided in order to avoid fatal conditions when errors occur. The CCI (I3C SDR) error detection and recovery methods follow the I3C Specification. The I3C error detection and recovery method for the Slave and Master are specified in *[MIPI03]*. A CCI (I3C SDR) compatible device shall support both the methods defined by I3C and the methods defined in this Section regarding CCI (I3C SDR), respectively.

6.3.1.1 Error Detection and Recovery Method for CCI (I3C SDR) Slave Devices

The SS0 error summarized in *Table 5* shall be supported for all CCI (I3C SDR) Slave Devices. If the CCI Slave detects the SS0 error, the CCI Slave shall set to 1'b1 in Protocol Error Flag of GETSTATUS. Details of the SS0 error are described in *Section 6.3.1.1.2*.

Table 5 CCI (I3C SDR) Slave Error Types

Error Type	Description	Error Detection Method	Error Recovery Method
SS0	**Read without INDEX Error**	Detect an error if the Slave receives the Slave's Dynamic Address (except 7'h7E) with a Read (R/W bit is 1) correctly but it does not have the INDEX.	Enable STOP or Repeated START detector and neglect other patterns.

6.3.1.1.1 Clearing the INDEX After Detecting I3C Error

The CCI (I3C SDR) Slave shall clear the INDEX value when the I3C Slave detects S2 *[MIPI03]* or S6 (*[MIPI03]*, optional) during the "CCI (I3C SDR) Read/Write Operations" in *Table 2*. Note that this rule shall not be applicable to other Operations (e.g., I3C CCC Transfers). As defined in the I3C specification, the I3C Slave sets to 1'b1 in the Protocol Error Flag of GETSTATUS (defined in the I3C specification) when the I3C Slave detects an error.

Clearing the INDEX due to S2 and S6 errors in the CCI (I3C SDR) Write Operations (Single Write to Random Location, Sequential Write Starting from Random Location) is described below:

- If an S2 error occurs in the CCI (I3C SDR) Write Operations, the CCI Slave cannot count up the INDEX because the CCI Slave cannot receive the correct write data. As a result, the INDEX in the CCI Slave may be different from the INDEX value that the Master is expecting. In order to avoid this situation, the CCI Slave shall clear the INDEX value.
- When the I3C Master doesn't have the collision detector and the I3C Slave has it, the INDEX in the CCI Slave may be different from the INDEX value that the Master is expecting in case of an S6 error. This is because the CCI Master assumes the INDEX counter in the Slave to be counting up, but the CCI Slave stops the counter. In order to avoid this situation, the CCI Slave shall clear the INDEX value.

Clearing the INDEX due to an S2 error in the CCI (I3C SDR) Read Operations (Single/Sequential Read to Random Location) is described below:

- If an S2 error occurs in the CCI (I3C SDR) Single/Sequential Read from Random Location during sub address, the CCI Slave cannot update the value of INDEX because the I3C Slave cannot get the correct sub address. This could cause slave to send undefined or wrong data. In order to avoid this situation, the CCI Slave shall clear the INDEX value.

6.3.1.1.2 SS0 Error

The CCI Slave shall detect an SS0 error if the CCI Slave receives the slave address (except 7'h7E) with a
Read (R/W bit is 1) correctly but it does not have the INDEX value. After detecting the SS0 error, the CCI
Slave shall replace ACK generated by the I3C Slave with NACK during SS0 error and then wait for STOP
or Repeated START. *Figure 21* illustrates how NACK is generated in CCI (I3C SDR) Sequential Read
from Random Location, when SS0 error occurs during this Message.
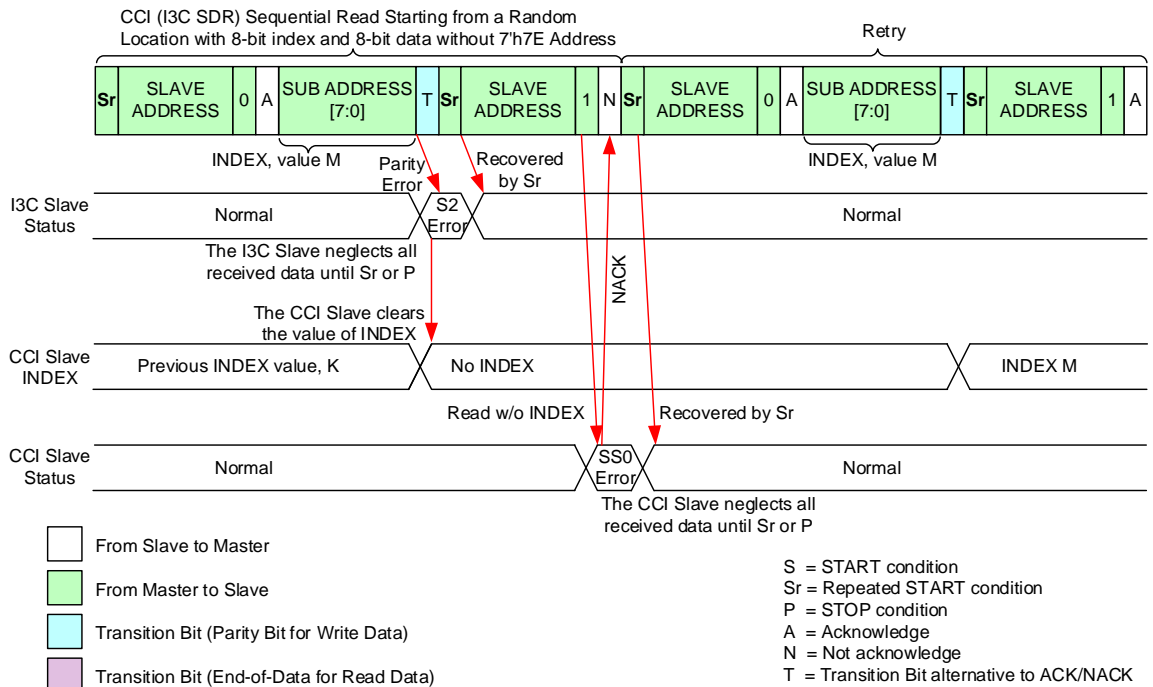
Figure 21 Example of SS0 Error Detection

6.3.2   CCI (I3C DDR) Error Detection and Recovery Method

542 The error detection and recovery methods specified in this Section are provided in order to avoid fatal
543 conditions when errors occur. The CCI (I3C DDR) error detection and recovery methods follow the I3C
544 Specification. The I3C error detection and recovery method for the Slave and Master are specified in
545 *[MIPI03]*. A CCI (I3C DDR) compatible device shall support both the methods defined by I3C and the
546 methods defined in this section regarding CCI (I3C DDR) respectively.

6.3.2.1   Error Detection and Recovery Method for CCI (I3C DDR) Slave Devices

547 The two Error Types summarized in *Table 6* shall be supported for all CCI (I3C DDR) Slave Devices. Each
548 Error Type is further explained below the table. If the Slave detects an SD0 or SD1 error, the Slave shall set
549 the Protocol Error Flag in GETSTATUS (defined in the I3C specification) to 1'b1. Details of the SD0 and
550 SD1 errors are described in *Section 6.3.2.1.2* and *Section 6.3.2.1.3*, respectively.

551   **Table 6 CCI (I3C DDR) Slave Error Types**

Error Type	Description	Error Detection Method	Error Recovery Method
SD0	Read without INDEX Error	Detect an error if the Slave receives the DDR command Word[15] = 1 (Read) correctly, but it does not have the INDEX	Enable HDR Exit or HDR Restart detector and neglect other patterns
SD1	Write over LENGTH Error	Detect an error if the value of Preamble following LENGTH +1 bytes of the Write Data is 2'b11	Clear INDEX value. Enable HDR Exit or HDR Restart detector and neglect other patterns

6.3.2.1.1   Clearing INDEX After Detecting I3C Error

552 The CCI Slave shall clear the INDEX value when the I3C Slave detects an I3C DDR error defined in the
553 I3C specification (Framing Error, Parity Error, CRC5 Error, or optional Monitoring Error) during the "CCI
554 (I3C DDR) Read/Write Operations" in *Table 3*. Note that this rule shall not be applicable to other
555 Operations (e.g., I3C CCC Transfers). As defined in the I3C specification, when the I3C Slave detects an
556 error it sets the Protocol Error Flag in GETSTATUS (defined in the I3C specification) to 1'b1.

557 If a parity error occurs during the sub address in a CCI (I3C DDR) Read Operation (i.e. Sequential or
558 Concatenated Sequential Read from Random Location), the CCI Slave cannot update the value of INDEX
559 because the I3C Slave cannot get the correct sub address. This could cause slave to send undefined or
560 wrong data. In order to avoid this situation, the CCI Slave shall clear the INDEX value.

6.3.2.1.2   SD0 Error

561 The CCI Slave shall detect an SD0 error if the CCI Slave receives a DDR command with Read (DDR
562 command Word[15] = 1) correctly, but no INDEX value. After detecting the SD0 error, the CCI Slave shall
563 replace the ACK generated by the I3C Slave with a NACK during SD0 error, and then wait for HDR Exit
564 or HDR Restart. *Figure 22* illustrates how NACK is generated in a CCI (I3C DDR) Sequential Read from
565 Random Location.

566
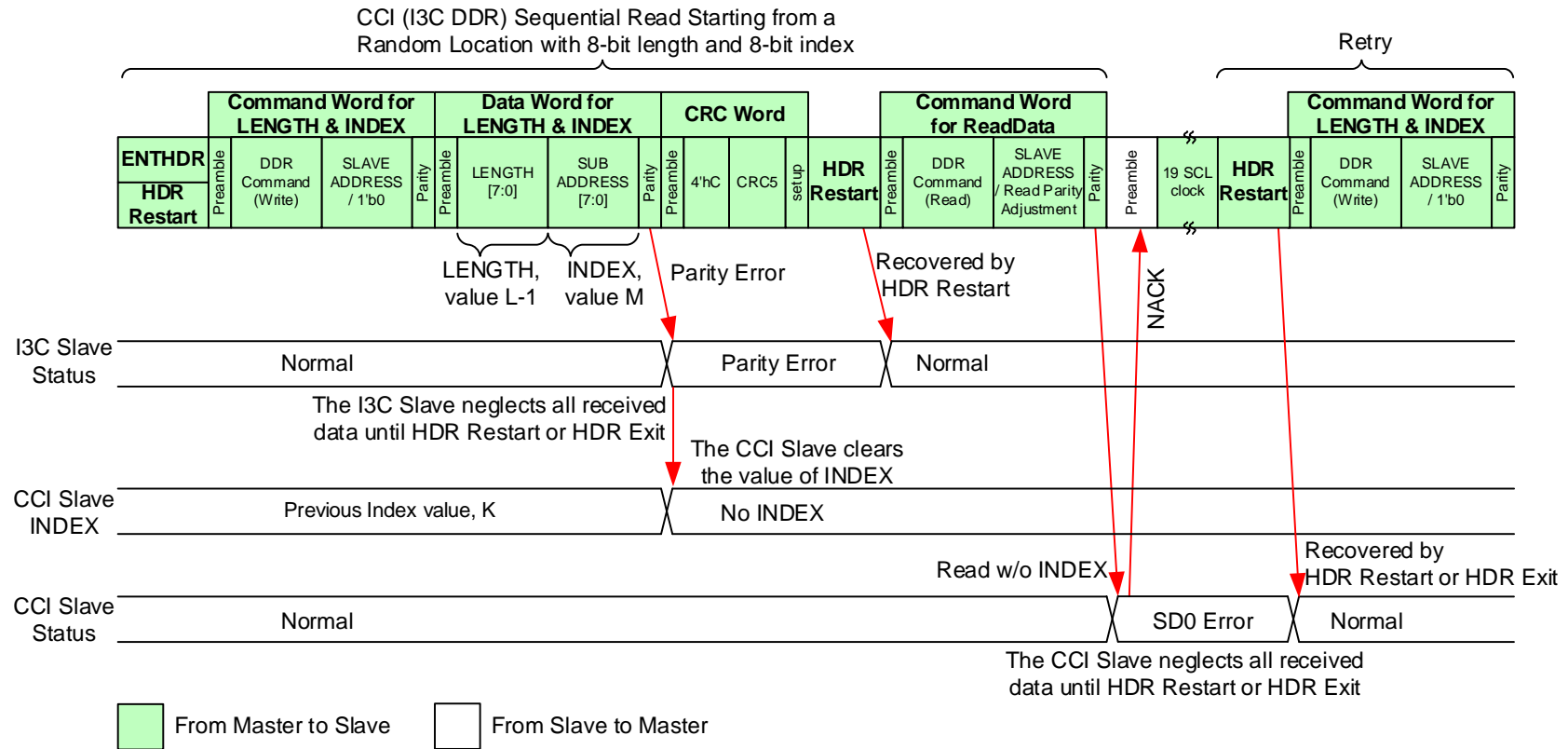
Figure 22 Example of SD0 Error Detection

567

6.3.2.1.3   SD1 Error

568   In CCI (I3C DDR), the LENGTH is included in the structure. If the CCI Slave receives data exceeding the
569   LENGTH, the Slave shall discard the extra data and detect this as an error condition.

570   In order to inform the Master of the error condition, the CCI Slave shall detect the SD1 error if the CCI
571   Slave receives a Preamble with value 2'b11 after receiving L bytes of WriteData. After detecting the SD1
572   error, the CCI Slave shall clear the value of INDEX and then wait for HDR Exit or HDR Restart. *Figure 23*
573   illustrates how INDEX is cleared in CCI (I3C DDR) Sequential Write to Random Location.

574
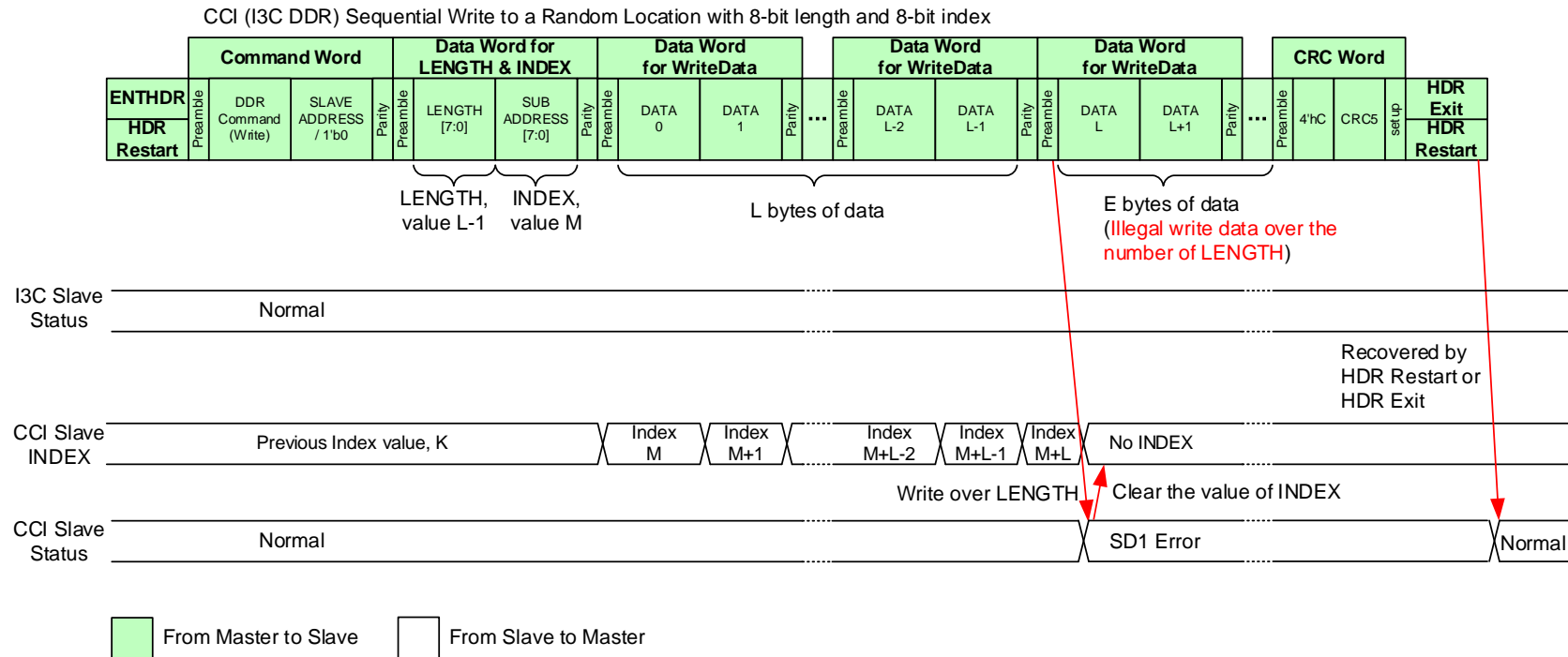
Figure 23 Example of SD1 Error Detection

575

6.3.2.2 Error Detection and Recovery Method for CCI (I3C DDR) Master Devices

576 The MD0 Error Type summarized in *Table 7* may be supported for all CCI (I3C DDR) Master Devices.
577 Each Error Type is further explained below *Table 7*. Details of MD0 error are described in
578 *Section 6.3.2.2.1*.

579 **Table 7 CCI (I3C DDR) Master Error Type**

Error Type	Description	Error Detection Method	Error Recovery Method
MD0 (optional)	Read over LENGTH Error	Detect an error if the value of Preamble[1] following LENGTH +1 bytes of the Read Data is 1'b1	Send Master Abort and then HDR Exit or HDR Restart

6.3.2.2.1 MD0 Error

580 In CCI (I3C DDR), the LENGTH is included in the structure. If the CCI Master receives read data
581 exceeding the LENGTH, it might cause big issues because memory leakage may occur, depending on the
582 implementation. In order to avoid fatal problems, the CCI Master may detect the MD0 error if the CCI
583 Master receives Preamble[1]=1'b1 after receiving LENGTH+1 bytes of ReadData. After detecting the MD0
584 Error, the CCI Master may send Master Abort, and then send HDR Exit or HDR Restart, as illustrated in
585 *Figure 24*.

586

CCI (I3C DDR) Sequential Read Starting from a Random Location with 8-bit length and 8-bit index

LENGTH, value L-1   INDEX, value M

L bytes of data

I3C Master Status   Normal

CCI Master Status   Normal

Read over LENGTH

Master Abort

Recovered by HDR Restart or HDR Exit

MD0 Error   Normal

587

From Master to Slave   From Slave to Master
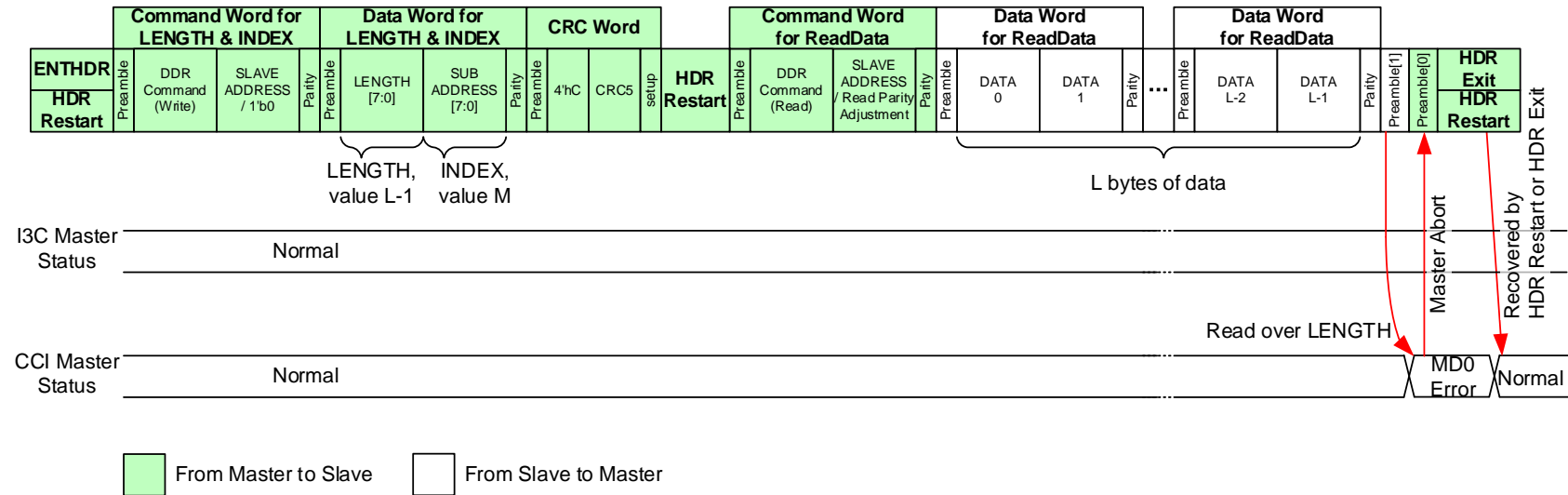
Figure 24 Example of MD0 Error Detection

6.3.3 Error Detection and Recovery for CCI (I3C) Master Devices

In many cases, the Master can detect an error inside the Slave by receiving NACK. However, for example in case of an S2 or S6 error in the CCI (I3C SDR) Write Operations, the Master cannot detect it by receiving NACK because there is no chance for the Slave to send NACK by the end of the operation (STOP or Repeated START). Therefore if high reliability is required, the Master may transmit GETSTATUS (defined in the I3C specification) at each important point.

Note:

> *E.g., the important point is that after critical CCI (I3C SDR) Write Operations, after CCI (I3C SDR) Write Operations before moving to CCI (I3C DDR), after multiple CCI (I3C SDR) Read/Write Operations before long pause if the last message is CCI (I3C SDR) Write Operations.*

As a result, the Master can detect each error by the following methods:

1. Slave's error by receiving NACK
2. Slave's error during CCI (I3C SDR) or CCI (I3C DDR) Write Operations by sending GETSTATUS
3. Master's I3C SDR Error defined in the I3C specification (M0, M1 or M2 error)
4. Master's I3C DDR Error defined in the I3C specification (including the Master sending HDR Exit or HDR Restart pattern)

After detecting an error, the Master should try the following error recovery method:

1. The Master may retry sending the same CCI (I3C SDR) Read/Write Operations or CCI (I3C DDR) Read/Write Operations again.
2. The Master may send certain other CCI (I3C SDR) Read/Write Operations or CCI (I3C DDR) Read/Write Operations, except CCI (I3C SDR) Single/Sequential Read From Current Location because the Slave would generate NACK again due to an SS0 or SD0 error.

In addition to, or instead of, a retry, the Master may read GETSTATUS, or try Escalation Handling as defined in the I3C specification.

6.4 CCI (I²C) Slave Addresses

For camera modules having only raw Bayer output the 7-bit slave address should be 7'b011011*X*, where *X* = either 1'b0 or 1'b1. For all other camera modules the 7-bit slave address should be 7'b011110*X*.

6.5 CCI (I3C) Slave Addresses

All camera modules shall use their own Dynamic Address as assigned by the I3C Master.

6.6   CCI Multi-Byte Registers

615   The description in this Section applies to both **CCI (I2C)** and **CCI (I3C)**.

6.6.1   Overview

616   Peripherals contain a wide range of different register widths for various control and setup purposes. This
617   Specification supports the following register widths:

618   • **8-bit:** Generic setup registers
619   • **16-bit:** Parameters like line-length, frame-length and exposure values
620   • **32-bit:** High precision setup values
621   • **64-bit:** For needs of future sensors

622   In general, the byte-oriented access protocols described in the previous sections provide an efficient means
623   to access multi-byte registers. However, the registers should reside in a byte-oriented address space, and the
624   address of a multi-byte register should be the address of its first byte. Thus, addresses of contiguous multi-
625   byte registers will not be contiguous. For example, a 32-bit register with its first byte at address 0x8000 can
626   be read by means of a sequential read of four bytes, starting at random address 0x8000. If there is an
627   additional 4-byte register with its first byte at 0x8004, then it could then be accessed using a four-byte
628   Sequential Read from the Current Location protocol.

629   The motivation for a generalized multi-byte protocol (rather than fixing register widths at 16 bits) is
630   flexibility. The protocol described below provides a way of transferring 16-bit, 32-bit, or 64-bit values over
631   a 16-bit INDEX, 8-bit data, two-wire serial link while ensuring that the bytes of data transferred for a
632   multi-byte register value are always consistent (temporally coherent).

633   Using this protocol, a single CCI Message can contain one, two, or all of the different register widths used
634   within a device.

635   The MS byte of a multi-byte register shall be located at the lowest address, and the LS byte shall be located
636   at the highest address.

637   The address of the first byte of a multi-byte register is not necessarily related to register size (i.e., not
638   required to be an integer multiple of register size in bytes). Register address alignment represents an
639   implementation choice between processing-optimized vs. bandwidth-optimized organizations. There are no
640   restrictions on the number or mix of multi-byte registers within the available 64K by 8-bit INDEX space,
641   with the exception of certain rules for the valid locations for the MS bytes and LS bytes of registers.

642   Partial access to multi-byte registers is not allowed. A multi-byte register shall only be accessed by a single
643   sequential Message. When a multi-byte register is accessed, its bytes shall be accessed in ascending address
644   order (i.e. first byte is accessed first, second byte is accessed second, etc.).

645   When a multi-byte register is accessed, the following re-timing rules shall be followed:

646   • For a Write operation, the updating of the register shall be deferred to a time when the last bit of
647   the last byte has been received.
648   • For a Read operation, the value read shall reflect the status of all bytes at the time that the first bit
649   of the first byte was read.

650   *Section 6.6.3* describes example re-timing behavior for multi-byte register accesses.

651   *Figure 25* and *Figure 26* illustrate that without re-timing, data could be corrupted.

Internal 32-bit Register Value (Locations M, M+1, M+2 and M+3)

| 0xFC FD FE FF | 0x01 02 03 04 |

Register Values updated by internal logic
(For example only)

Register Index

| Index M | Index M+1 | Index M+2 | Index M+3 |

| S | SLAVE ADDRESS | 1 | A | DATA = 0xFC | A | DATA=0xFD | A | DATA=0x03 | A | DATA=0x04 | Ā | P |

MS Data Byte *LS Data Byte*

| 0xFC | 0xFD | 0x03 | 0x04 |

| DATA[31:24] | DATA[23:16] | DATA[15:8] | DATA[7:0] |

DATA[31:0]

| | From slave to master | S = START condition | A = Acknowledge |
| | From master to slave | P = STOP condition | Ā = Negative acknowledge |

Figure 25 Corruption of 32-bit Register During Read Message

Internal 32-bit Register Value (Locations M, M+1, M+2 and M+3)

| 0xFC FD FE FF | 0x01 FD FE FF | 0x01 02 FE FF | 0x01 02 03 FF | 0x01 02 03 04 |

Internal logic
reads register
value
(For example
only)

Register Index

| Index M | Index M+1 | Index M+2 | Index M+3 |

| S | SLAVE ADDRESS | 0 | A | DATA=0x01 | A | DATA=0x02 | A | DATA=0x03 | A | DATA=0x04 | Ā | P |

MS Data Byte *LS Data Byte*

| 0x01 | 0x02 | 0x03 | 0x04 |

| DATA[31:24] | DATA[23:16] | DATA[15:8] | DATA[7:0] |

DATA[31:0]

| | From slave to master | S = START condition | A = Acknowledge |
| | From master to slave | P = STOP condition | Ā = Negative acknowledge |

Figure 26 Corruption of 32-bit Register During Write Message

6.6.2 Transmission Byte Order for Multi-Byte Register Values

654 *Figure 27*, *Figure 28*, and *Figure 29* illustrate the requirement that the first byte of a CCI Message shall
655 always be the MS byte of a multi-byte register, and the last byte of the CCI Message shall always be the LS
656 byte of the multi-byte register.
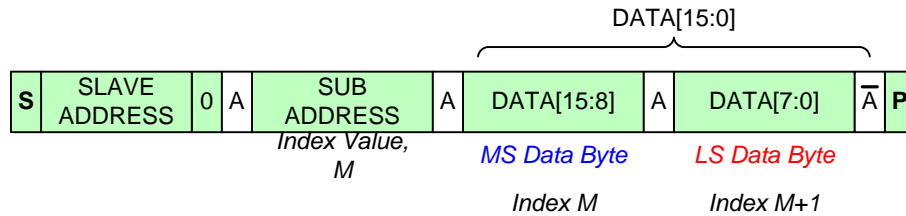
Figure 27 Example 16-bit Register Write
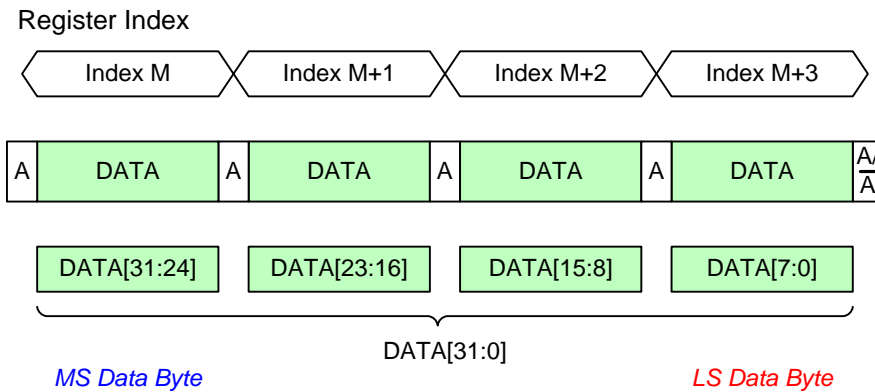
Figure 28 Example 32-bit Register Write (Address Not Shown)
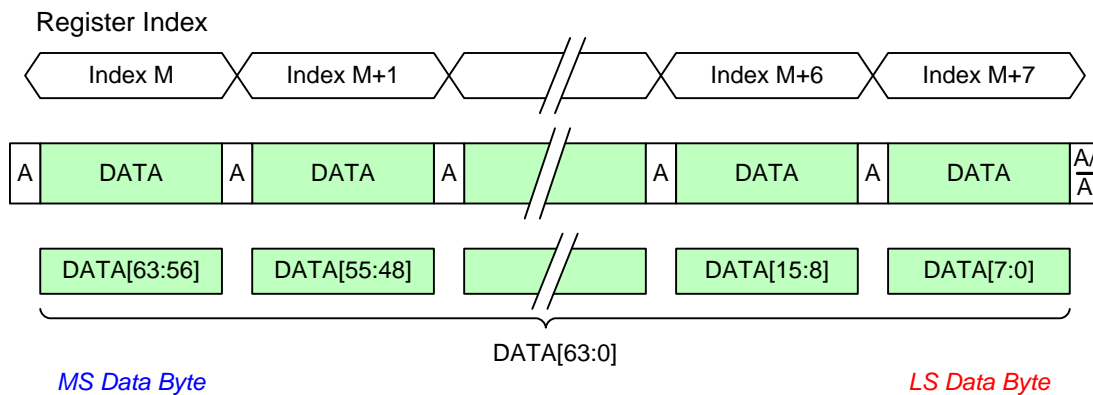
Figure 29 Example 64-bit Register Write (Address Not Shown)

6.6.3   Multi-Byte Register Protocol (Informative)

660 Each device may have both single-byte registers and multi-byte registers. Internally a device must
661 understand what addresses correspond to the different register widths.

6.6.3.1   Reading Multi-Byte Registers

662 To ensure that the value read from a multi-byte register is consistent (i.e., that all of the transmitted bytes
663 are temporally coherent), the device can internally transfer the register contents into a temporary buffer at
664 the time when the register's MS byte is read. The contents of the temporary buffer can then be sent out as a
665 sequence of bytes on the SDA line. *Figure 30* and *Figure 31* illustrate multi-byte register read operations.

666 The temporary buffer is always updated, except in the case of a read operation that is incremental within
667 the same multi-byte register.

Internal 16-bit Register Value (Locations M and M+1)

| 0xFC FD | 0x01 02 |

Internal 16-bit Register Value (Locations M+2 and M+3)

| 0xFE FF | 0x03 04 |

Register Values updated by internal logic
(For example only)

Register Index

| Index M | Index M+1 | Index M+2 | Index M+3 |

Temporary Buffer

| 0x00 00 | 0xFC FD | 0x03 04 |

A read from MS byte of the register causes the whole register value to be transferred into a temporary buffer

Incremental read within the same multi-byte register. Temporary Buffer not updated

| S | SLAVE ADDRESS | 1 | A | DATA = 0xFC | A | DATA=0xFD | A | DATA=0x03 | A | DATA=0x04 | Ā | P |

MS Data Byte   *LS Data Byte*   *MS Data Byte*   *LS Data Byte*

| 0xFC | 0xFD | 0x03 | 0x04 |

| DATA[15:8] | DATA[7:0] | DATA[15:8] | DATA[7:0] |

DATA[15:0]   DATA[15:0]

| ☐ | From slave to master | S = START condition | A = Acknowledge |
| 🟩 | From master to slave | P = STOP condition | Ā = Negative acknowledge |

668

Figure 30 Example 16-bit Register Read

669 In this definition no distinction is made between a register being accessed incrementally via multiple
670 separate single-byte read Messages with no intervening data writes, vs. a register being accessed via a
671 single multi-location read Message. This protocol purely relates to the behavior of the INDEX value.

672   Examples of when the temporary buffer is updated include:
673   • When the MS byte of a register is accessed
674   • When the INDEX has crossed a multi-byte register boundary
675   • Successive single-byte reads from the same INDEX location
676   • When the INDEX value for the byte about to be read is ≤ the previous INDEX

677   Note that the values read back are only guaranteed to be consistent if the contents (bytes) of the multi-byte
678   register are accessed in an incremental manner.

679   The contents of the temporary buffer are reset to zero by START and STOP conditions.

Internal 32-bit Register Value (Locations M, M+1, M+2 and M+3)

| 0xFC FD FE FF | 0x01 02 03 04 |

Register Values updated by internal logic (For example only)

Register Index

| Index M | Index M+1 | Index M+2 | Index M+3 |

Temporary Buffer

| 0x00 00 00 00 | 0xFC FD FE FF |

A read from MS byte of the register causes the whole register value to be transferred into a temporary buffer

Incremental read within the same multi-byte register. Temporary Buffer not updated

| S | SLAVE ADDRESS | 1 | A | DATA = 0xFC | A | DATA=0xFD | A | DATA=0xFE | A | DATA=0xFF | A̅ | P |

MS Data Byte   LS Data Byte

| 0xFC | 0xFD | 0xFE | 0xFF |
| DATA[31:24] | DATA[23:16] | DATA[15:8] | DATA[7:0] |

DATA[31:0]

From slave to master   S = START condition   A = Acknowledge
From master to slave   P = STOP condition   A̅ = Negative acknowledge

680

Figure 31 Example 32-bit Register Read

6.6.3.2   Writing Multi-Byte Registers

681 To ensure that the value written is consistent, the bytes of data from a multi-byte register are written into a
682 temporary buffer. Only after the LS byte of the register is written is the full multi-byte value transferred
683 into the internal register location.

684 *Figure 32* and *Figure 33* illustrate multi-byte register write operations.

685 CCI Messages that only write to the LS or MS byte of a multi-byte register are not allowed. Single byte
686 writes to a multi-byte register addresses may cause undesirable behavior in the device.

687
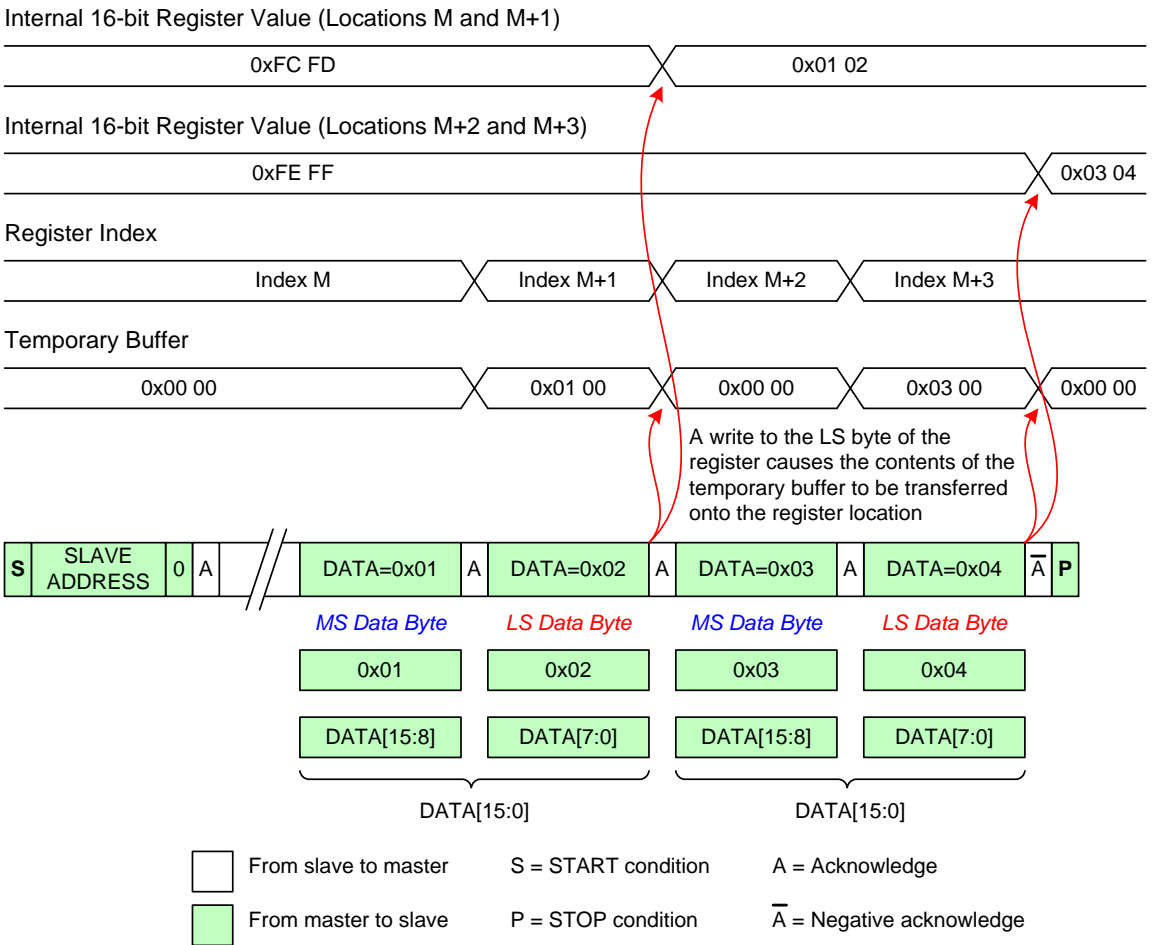
Figure 32 Example 16-bit Register Write

Internal 32-bit Register Value (Locations M, M+1, M+2 and M+3)

| 0xFC FD FE FF | 0x01 02 03 04 |

Register Index

| Index M | Index M+1 | Index M+2 | Index M+3 |

Temporary Buffer

| 0x00 00 00 00 | 0x01 00 00 00 | 0x01 02 00 00 | 0x01 02 03 00 | 0x00 00 00 00 |

A write to the LS byte of the register causes the contents of the temporary buffer to be transferred onto the register location

| S | SLAVE ADDRESS | 0 | A | | DATA=0x01 | A | DATA=0x02 | A | DATA=0x03 | A | DATA=0x04 | $\overline{A}$ | P |

MS Data Byte   *LS Data Byte*

| 0x01 | 0x02 | 0x03 | 0x04 |

| DATA[31:24] | DATA[23:16] | DATA[15:8] | DATA[7:0] |

DATA[31:0]

☐ From slave to master	S = START condition	A = Acknowledge
▦ From master to slave	P = STOP condition	$\overline{A}$ = Negative acknowledge

688

Figure 33 Example 32-bit Register Write

6.7   CCI I/O Electrical and Timing Specifications

689   The CCI I/O stages electrical specifications (*Table 8*) and timing specifications (*Table 9*) conform to I²C
690   Fast-mode and Fast-mode Plus devices. Information presented in *Table 8* is from *[NXP01]*.

691   The CCI timings specified in *Table 9* are illustrated in *Figure 34*.

692   **Table 8 CCI I/O Electrical Specifications**

Parameter	Symbol	Fast-mode		Fast-mode Plus		Unit
		Min.	Max.	Min.	Max.	
LOW level input voltage	V_{IL}	-0.5	0.3 V_{DD}	-0.5	0.3 V_{DD}	V
HIGH level input voltage	V_{IH}	0.7V_{DD}	Note 1	0.7V_{DD}	Note 1	V
Hysteresis of Schmitt trigger inputs	V_{HYS}	0.05V_{DD}	-	0.05V_{DD}	-	V
LOW level output voltage (open drain) at 2mA sink current V_{DD} > 2V V_{DD} < 2V	V_{OL1} V_{OL3}	0 0	0.4 0.2V_{DD}	0 0	0.4 0.2V_{DD}	V
Output fall time from V_{IHmin} to V_{ILmax} with bus capacitance from 10 pF to 400 pF	t_{OF}	20 x (V_{DD} / 5.5 V)	250	20 x (V_{DD} / 5.5 V)	120	ns
Pulse width of spikes which shall be suppressed by the input filter	t_{SP}	0	50	0	50	ns
Input current each I/O pin with an input voltage between 0.1 V_{DD} and 0.9 V_{DD}	I_I	-10 Note 2	10 Note 2	-10 Note 2	10 Note 2	μA
Input/Output capacitance (SDA)	$C_{I/O}$	-	10	-	10	pF
Input capacitance (SCL)	CI	-	10	-	10	pF

Note:

1. *Maximum VIH = V_{DDmax} + 0.5V*
2. *I/O pins of Fast-mode and Fast-mode Plus devices shall not obstruct the SDA and SCL line if V_{DD} is switched off*

693

Table 9 CCI I/O Timing Specifications

Parameter	Symbol	Fast-mode		Fast-mode Plus		Unit
		Min.	Max.	Min.	Max.	
SCL clock frequency	f_{SCL}	0	400	0	1000	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	0.6	-	0.26	-	µs
LOW period of the SCL clock	t_{LOW}	1.3	-	0.5	-	µs
HIGH period of the SCL clock	t_{HIGH}	0.6	-	0.26	-	µs
Setup time for a repeated START condition	$t_{SU;STA}$	0.6	-	0.26	-	µs
Data hold time	$t_{HD;DAT}$	0 Note 2	- Note 3	0	-	µs
Data set-up time	$t_{SU;DAT}$	100 Note 4	-	50	-	ns
Rise time of both SDA and SCL signals	t_R	20	300	-	120	ns
Fall time of both SDA and SCL signals	t_F	20 x (V_{DD} / 5.5 V)	300	20 x (V_{DD} / 5.5 V)	120	ns
Set-up time for STOP condition	$t_{SU;STO}$	0.6	-	0.26	-	µs
Bus free time between a STOP and START condition	t_{BUF}	1.3	-	0.5	-	µs
Capacitive load for each bus line	C_B	-	400	-	550	pF
Data valid time Note 5	$t_{VD;DAT}$	-	0.9 Note 3	-	0.45 Note 3	µs
Data valid acknowledge time Note 6	$t_{VD;ACK}$	-	0.9 Note 3	-	0.45 Note 3	µs
Noise margin at the LOW level for each connected device (including hysteresis)	V_{nL}	0.1 x V_{DD}	-	0.1 x V_{DD}	-	V
Noise margin at the HIGH level for each connected device (including hysteresis)	V_{nH}	0.2 x V_{DD}	-	0.2 x V_{DD}	-	V

Note:

1. *All values referred to $V_{IHmin} = 0.7V_{DD}$ and $V_{ILmax} = 0.3V_{DD}$*
2. *A device shall internally provide a hold time of at least 300 ns for the SDA signal (referred to the V_{IHmin} of the SCL signal) to bridge the undefined region of the falling edge of SCL*
3. *The maximum $t_{HD;DAT}$ could be 0.9 µs and 0.45 µs for Fast-mode and Fast-mode Plus, but must be less than the maximum of $t_{VD;DAT}$ or $t_{VD;ACK}$ by a transition time. This maximum must only be met if the device does not stretch the LOW period (t_{LOW}) of the SCL signal. If the clock stretches the SCL, then the data must be valid by the set-up time before it releases the clock.*
4. *A Fast-mode I2C-bus device can be used in a Standard-mode I2C-bus system, but the requirement $t_{SU;DAT} \geq 250$ ns shall be then met. This will be automatically the case if the device does not stretch the LOW period of the SCL signal. If such device does stretch the low period of SCL signal, it shall output the next data bit to the SDA line $t_{rMAX} + t_{SU;DAT} = 1000 + 250 = 1250$ ns (according to the Standard-mode I²C Bus specification [NXP01]) before the SCL line is released.*
5. *$t_{VD;DAT}$ = time for data signal from SCL LOW to SDA output (HIGH or LOW, whichever is worse).*
6. *$t_{VD;ACK}$ = time for Acknowledgement signal from SCL LOW to SDA output (HIGH or LOW, whichever is worse)*
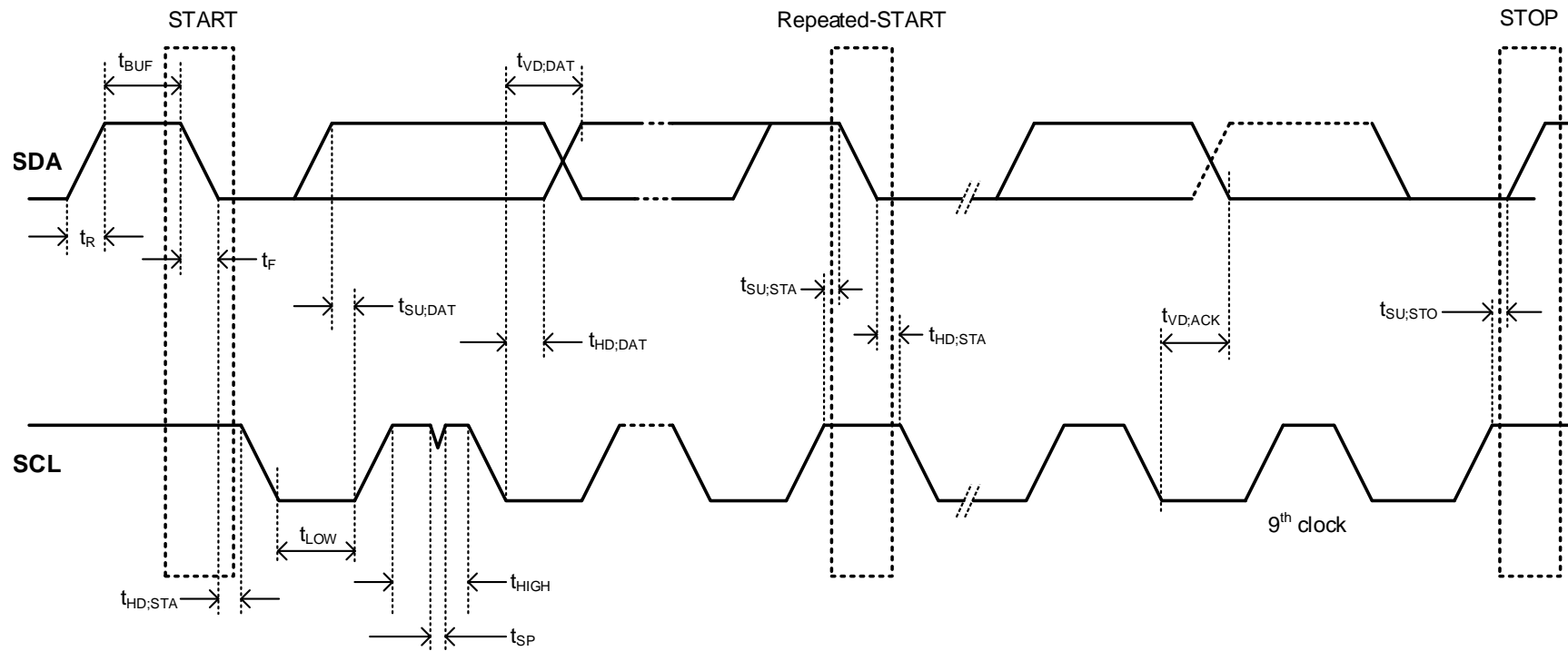
Figure 34 CCI I/O Timing

This page intentionally left blank.

7 Physical Layer

696 The CSI-2 lane management layer interfaces with the D-PHY and/or C-PHY physical layers described in
697 *[MIPI01]* and *[MIPI02]*, respectively. A device shall implement either the C-PHY 1.2 or the D-PHY 2.1
698 physical layer and may implement both. A practical constraint is that the PHY technologies used at both
699 ends of the Link need to match: a D-PHY transmitter cannot operate with a C-PHY receiver, or vice versa.

7.1 D-PHY Physical Layer Option

700 The D-PHY physical layer for a CSI-2 implementation is composed of a number of unidirectional data
701 Lanes and one clock Lane. All CSI-2 transmitters and receivers implementing the D-PHY physical layer
702 shall support continuous clock behavior on the Clock Lane, and optionally may support non-continuous
703 clock behavior.

704 For continuous clock behavior the Clock Lane remains in high-speed mode, generating active clock signals
705 between the transmission of data packets.

706 For non-continuous clock behavior the Clock Lane enters the LP-11 state between the transmission of data
707 packets.

708 The minimum D-PHY physical layer requirement for a CSI-2 transmitter is

709 • Data Lane Module: Unidirectional master, HS-TX, LP-TX and a CIL-MFEN function
710 • Clock Lane Module: Unidirectional master, HS-TX, LP-TX and a CIL-MCNN function

711 The minimum D-PHY physical layer requirement for a CSI-2 receiver is

712 • Data Lane Module: Unidirectional slave, HS-RX, LP-RX, and a CIL-SFEN function
713 • Clock Lane Module: Unidirectional slave, HS-RX, LP-RX, and a CIL-SCNN function

714 All CSI-2 implementations supporting the D-PHY physical layer option shall support forward escape ULPS
715 on all D-PHY Data Lanes.

716 To enable higher data rates and higher number of lanes the physical layer described in *[MIPI01]* includes
717 an independent deskew mechanism in the Receive Data Lane Module. To facilitate deskew calibration at
718 the receiver the transmitter Data Lane Module provides a deskew sequence pattern.

719 Since deskew calibration is only valid at a given transmit frequency:

720 For initial calibration sequence the Transmitter shall be programmed with the desired frequency for
721 calibration. It will then transmit the deskew calibration pattern and the Receiver will autonomously detect
722 this pattern and tune the deskew function to achieve optimum performance.

723 For any transmitter frequency changes the deskew calibration shall be rerun.

724 Some transmitters and/or receivers may require deskew calibration to be rerun periodically and it is
725 suggested that it can be optimally done within vertical or frame blanking periods.

726 For low transmit frequencies or when a receiver described in *[MIPI01]* is paired with a previous version
727 transmitter not supporting the deskew calibration pattern the receiver may be instructed to bypass the
728 deskew mechanism.

729 The D-PHY v2.1 physical layer *[MIPI05]* provides Alternate Low Power State (ALPS) using Low Voltage
730 Low Power (LVLP) signaling, which may optionally replace the legacy Low Power State (LPS). Use of
731 LVLP can help alleviate current leakage and electrical overstress issues with image sensors and applications
732 processors.

7.1.1 D-PHY v2.1 Compatibility with D-PHY v2.0 (Informative)

733   A D-PHY v2.0 *[MIPI05]* or earlier physical layer and a D-PHY v2.1 physical layer are fully interoperable.

734   For bit rates above 2.5 Gbps per Lane, a D-PHY v2.0 *[MIPI05]* or earlier physical layer and a D-PHY v2.1
735   physical layer are fully interoperable, provided certain new D-PHY v2.1 features are disabled as permitted
736   by the D-PHY v2.1 specification. Such features include the Alternate Calibration Sequence, Preamble
737   Sequence, and Extended Sync pattern as described in *Section 6.13* and *Section 6.14* of *[MIPI01]*.

738   These features allow system interfaces to more robustly compensate for variations such as temperature and
739   voltage when operating at bit rates above 2.5 Gbps but are not supported by D-PHY v2.0.

7.2 C-PHY Physical Layer Option

740   The C-PHY physical layer for a CSI-2 implementation is composed of one or more unidirectional Lanes.

741   The minimum C-PHY physical layer requirement for a CSI-2 transmitter Lane module is:

- Unidirectional master, HS-TX, LP-TX and a CIL-MFEN function
- Support for Sync Word insertion during data payload transmission

744   The minimum C-PHY physical layer requirement for a CSI-2 receiver Lane module is:

- Unidirectional slave, HS-RX, LP-RX, and a CIL-SFEN function
- Support for Sync Word detection during data payload reception

747   All CSI-2 implementations supporting the C-PHY physical layer option shall support forward escape ULPS
748   on all C-PHY Lanes.

749   The C-PHY Physical Layer provides Alternate Low Power State (ALPS) signaling using Low Voltage Low
750   Power (LVLP) signaling or Alternate Low Power (ALP) Embedded Codes, which may optionally replace
751   the legacy Low Power State (LPS). Use of ALPS can help alleviate current leakage and electrical overstress
752   issues with image sensors and applications processors. ALPS using the ALP Embedded Codes can also help
753   achieve longer reach for CSI-2 imaging interface channels before re-drivers and re-timers become
754   necessary.

8 Multi-Lane Distribution and Merging

755 CSI-2 is a Lane-scalable specification. Applications requiring more bandwidth than that provided by one
756 data Lane, or those trying to avoid high clock rates, can expand the data path to a higher number of Lanes
757 and obtain approximately linear increases in peak bus bandwidth. The mapping between data at higher
758 layers and the serial bit or symbol stream is explicitly defined to ensure compatibility between host
759 processors and peripherals that make use of multiple data Lanes.

760 Conceptually, between the PHY and higher functional layers is a layer that handles multi-Lane
761 configurations. As shown in *Figure 35* and *Figure 36* for the D-PHY and C-PHY physical layer options,
762 respectively, the CSI-2 transmitter incorporates a Lane Distribution Function (LDF) which accepts a
763 sequence of packet bytes from the low level protocol layer and distributes them across N Lanes, where each
764 Lane is an independent unit of physical-layer logic (serializers, etc.) and transmission circuitry. Similarly,
765 as shown in *Figure 37* and *Figure 38* for the D-PHY and C-PHY physical layer options, respectively,the
766 CSI-2 receiver incorporates a Lane Merging Function (LMF) which collects incoming bytes from N Lanes
767 and consolidates (merges) them into complete packets to pass into the packet decomposer in the receiver's
768 low level protocol layer.
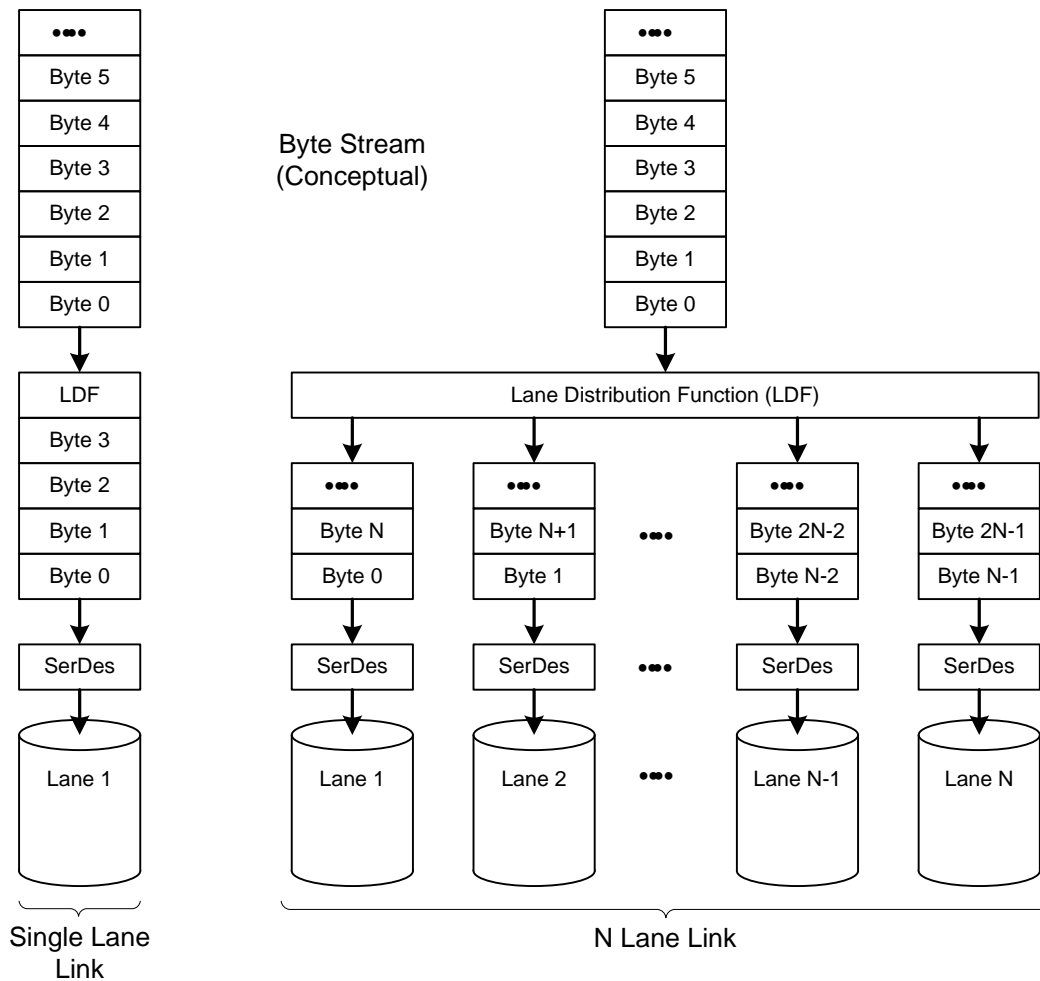
Figure 35 Conceptual Overview of the Lane Distributor Function for D-PHY

770
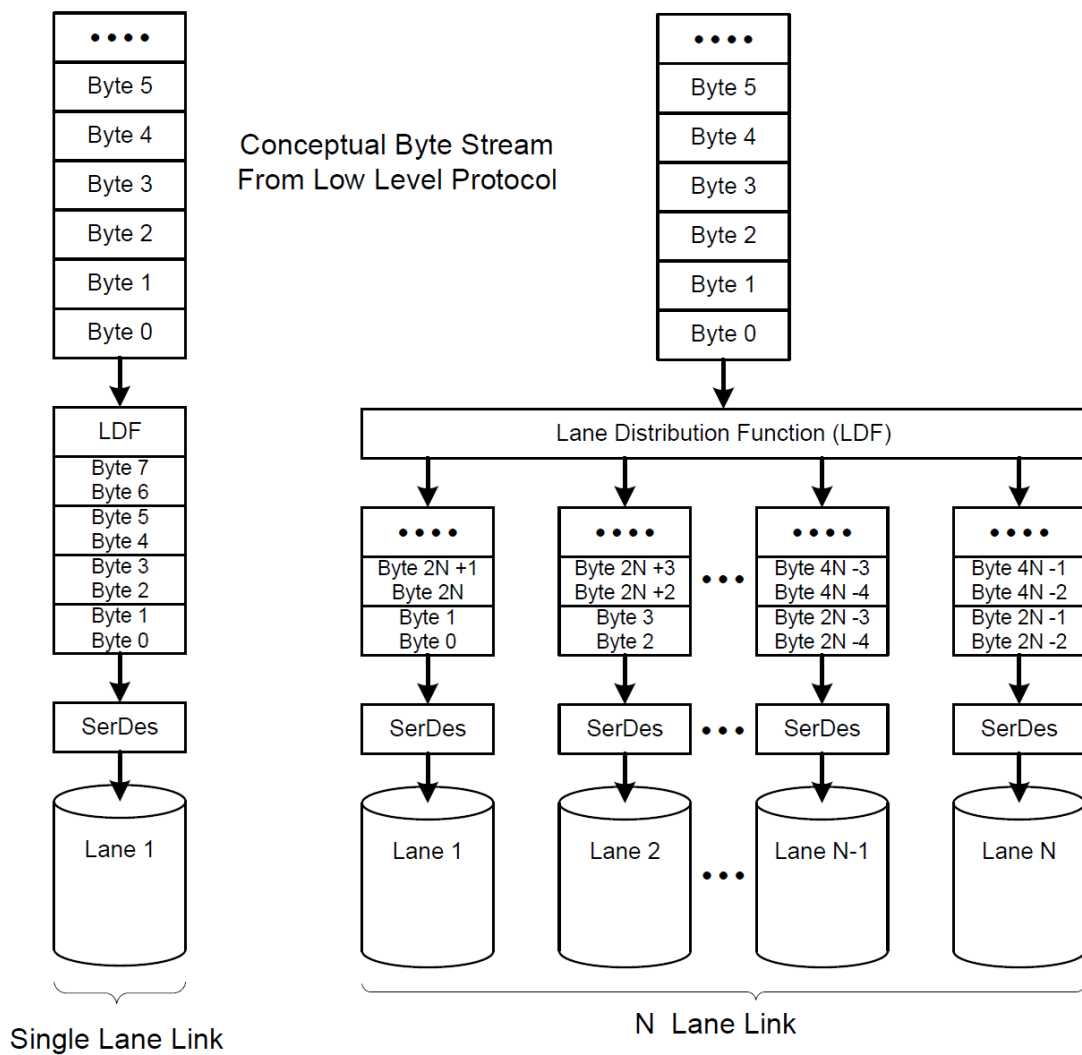
Figure 36 Conceptual Overview of the Lane Distributor Function for C-PHY
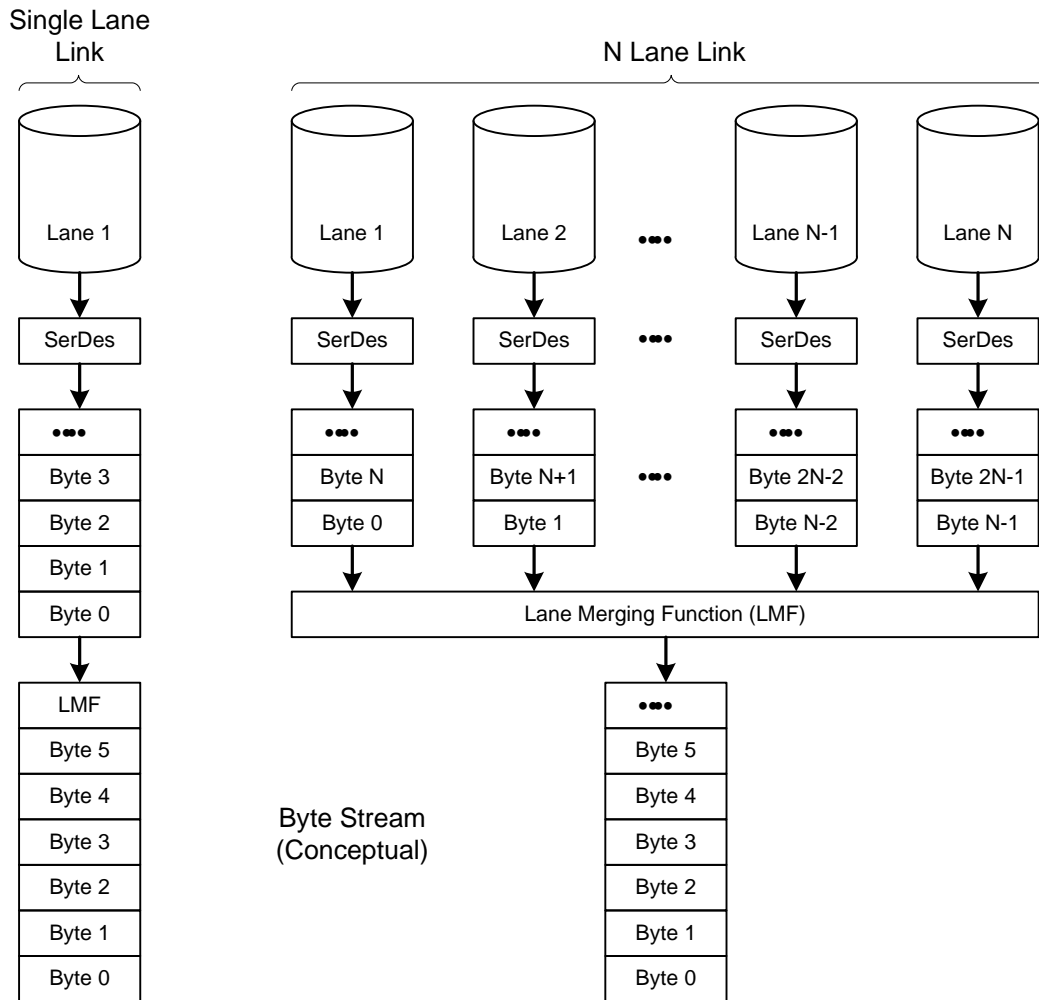
Figure 37 Conceptual Overview of the Lane Merging Function for D-PHY
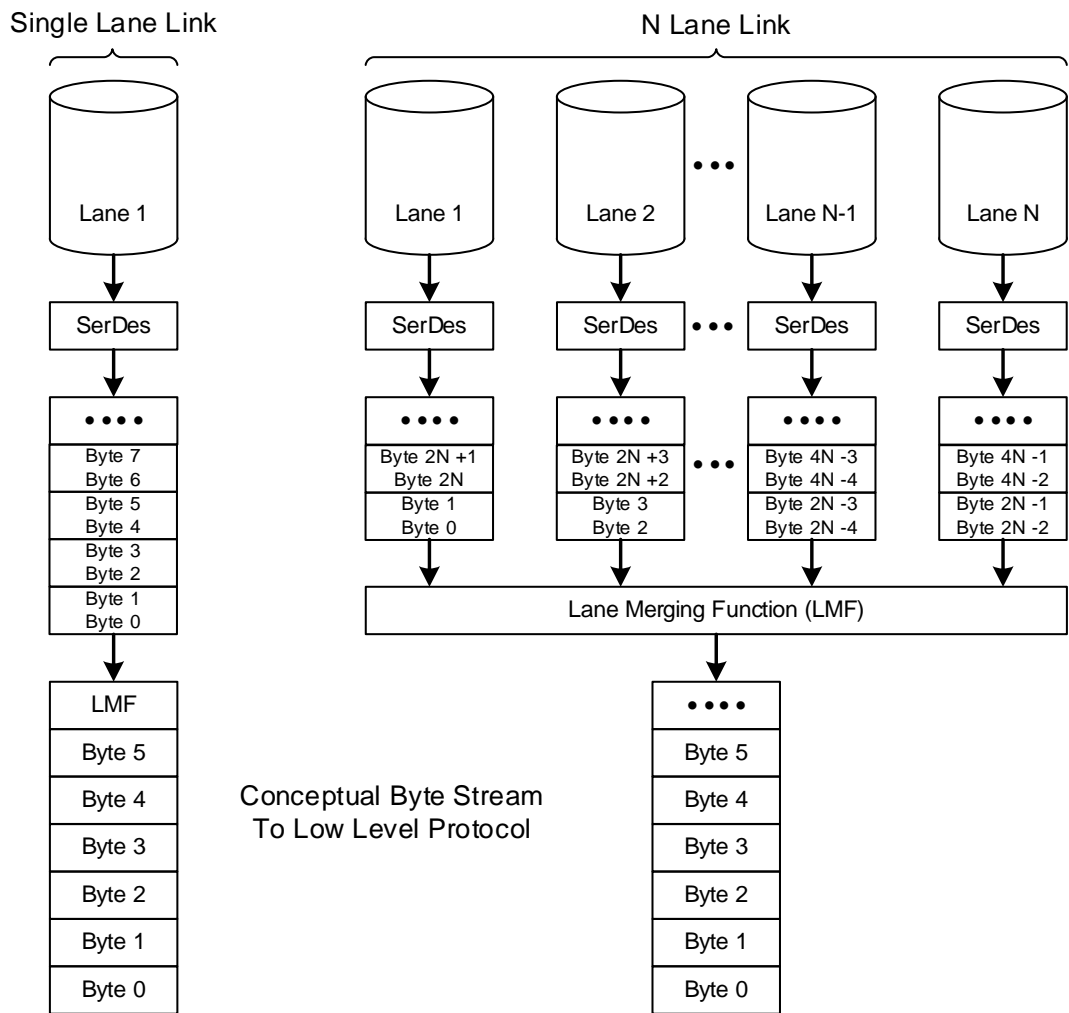
Figure 38 Conceptual Overview of the Lane Merging Function for C-PHY

The Lane distributor takes a transmission of arbitrary byte length, buffers up $N*b$ bytes (where N = number of Lanes and b = 1 or 2 for the D-PHY or C-PHY physical layer option, respectively), and then sends groups of $N*b$ bytes in parallel across N Lanes with each Lane receiving b bytes. Before sending data, all Lanes perform the SoT sequence in parallel to indicate to their corresponding receiving units that the first byte of a packet is beginning. After SoT, the Lanes send groups of successive bytes from the first packet in parallel, following a round-robin process.

8.1   Lane Distribution for the D-PHY Physical Layer Option

779   Examples are shown in *Figure 39*, *Figure 40*, *Figure 41*, and *Figure 42*:

780   • 2-Lane system (*Figure 39*): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to
781   Lane 1, byte 3 goes to Lane 2, byte 4 goes to Lane 1, and so on.

782   • 3-Lane system (*Figure 40*): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte 2 to
783   Lane 3, byte 3 goes to Lane 1, byte 4 goes to Lane 2, and so on.

784   • N-Lane system (*Figure 41*): byte 0 of the packet goes to Lane 1, byte 1 goes to Lane 2, byte N-1
785   goes to Lane N, byte N goes to Lane 1, byte N+1 goes to Lane 2, and so on.

786   • N-lane system (*Figure 42*) with N>4 short packet (4 bytes) transmission: byte 0 of the packet goes
787   to Lane 1, byte 1 goes to Lane 2, byte 2 goes to Lane 3, byte 3 goes to Lane 4, and Lanes 5 to N
788   do not receive bytes and stay in LPS state.

789   At the end of the transmission, there may be "extra" bytes since the total byte count may not be an integer
790   multiple of the number of Lanes, N. One or more Lanes may send their last bytes before the others. The
791   Lane distributor, as it buffers up the final set of less-than-N bytes in parallel for sending to N data Lanes,
792   de-asserts its "valid data" signal into all Lanes for which there is no further data. For systems with more
793   than 4 data Lanes sending a short packet constituted of 4 bytes the Lanes which do not receive a byte for
794   transmission shall stay in LPS state.

795   Each D-PHY data Lane operates autonomously.

796   Although multiple Lanes all start simultaneously with parallel "start packet" codes, they may complete the
797   transaction at different times, sending "end packet" codes one cycle (byte) apart.

798   The N PHYs on the receiving end of the link collect bytes in parallel, and feed them into the Lane-merging
799   layer. This reconstitutes the original sequence of bytes in the transmission, which can then be partitioned
800   into individual packets for the packet decoder layer.

Number of Bytes, B, transmitted is an integer multiple of the number of lanes:
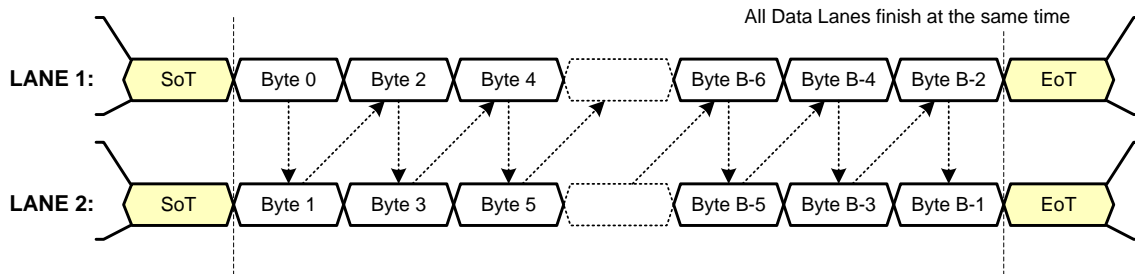
All Data Lanes finish at the same time

LANE 1: | SoT | Byte 0 | Byte 2 | Byte 4 | | Byte B-6 | Byte B-4 | Byte B-2 | EoT |

LANE 2: | SoT | Byte 1 | Byte 3 | Byte 5 | | Byte B-5 | Byte B-3 | Byte B-1 | EoT |

Number of Bytes, B, transmitted is NOT an integer multiple of the number of lanes:

Data Lane 2 finishes 1 byte earlier than Data Lane 1

LANE 1: | SoT | Byte 0 | Byte 2 | Byte 4 | | Byte B-5 | Byte B-3 | Byte B-1 | EoT |

LANE 2: | SoT | Byte 1 | Byte 3 | Byte 5 | | Byte B-4 | Byte B-2 | EoT | LPS |

KEY:

LPS – Low Power State   SoT – Start of Transmission   EoT – End of Transmission

801
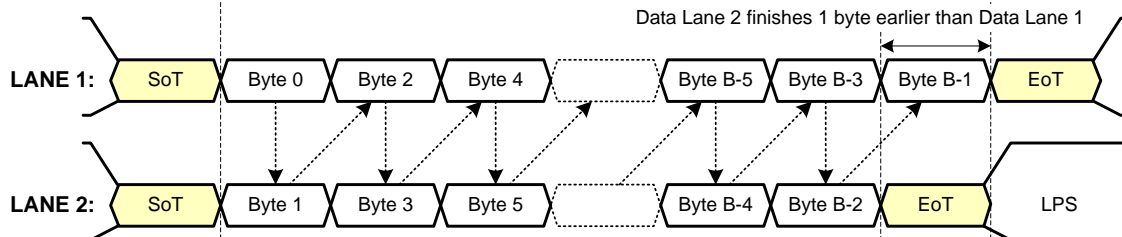
Figure 39 Two Lane Multi-Lane Example for D-PHY

Number of Bytes, B, transmitted is an integer multiple of the number of lanes:

All Data Lanes finish at the same time

LANE 1:	SoT	Byte 0	Byte 3	Byte 6		Byte B-9	Byte B-6	Byte B-3	EoT
LANE 2:	SoT	Byte 1	Byte 4	Byte 7		Byte B-8	Byte B-5	Byte B-2	EoT
LANE 3:	SoT	Byte 2	Byte 5	Byte 8		Byte B-7	Byte B-4	Byte B-1	EoT

Number of Bytes, B, transmitted is NOT an integer multiple of the number of lanes (Example 1):

Data Lanes 2 & 3 finish 1 byte earlier than Data Lane 1

LANE 1:	SoT	Byte 0	Byte 3	Byte 6		Byte B-7	Byte B-4	Byte B-1	EoT
LANE 2:	SoT	Byte 1	Byte 4	Byte 7		Byte B-6	Byte B-3	EoT	LPS
LANE 3:	SoT	Byte 2	Byte 5	Byte 8		Byte B-5	Byte B-2	EoT	LPS

Number of Bytes, B, transmitted is NOT an integer multiple of the number of lanes (Example 2):

Data Lane 3 finishes 1 byte earlier than Data Lanes 1 & 2

LANE 1:	SoT	Byte 0	Byte 3	Byte 6		Byte B-8	Byte B-5	Byte B-2	EoT
LANE 2:	SoT	Byte 1	Byte 4	Byte 7		Byte B-7	Byte B-4	Byte B-1	EoT
LANE 3:	SoT	Byte 2	Byte 5	Byte 8		Byte B-6	Byte B-3	EoT	LPS

KEY:
LPS – Low Power State   SoT – Start of Transmission   EoT – End of Transmission

802
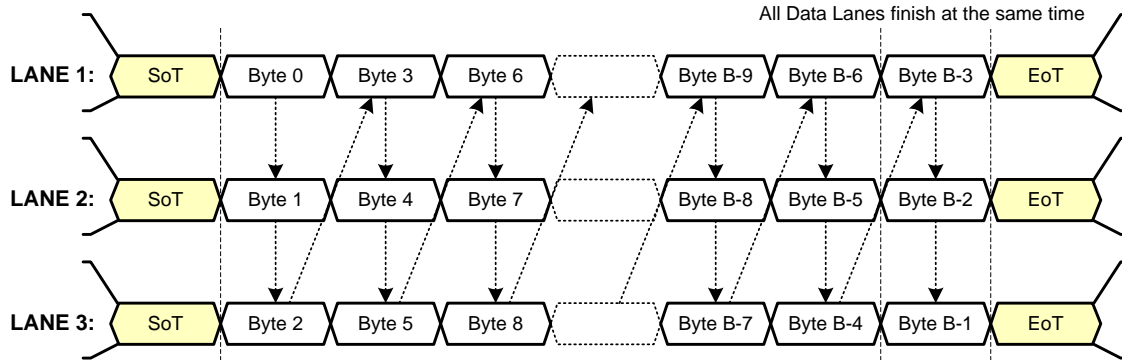
Figure 40 Three Lane Multi-Lane Example for D-PHY

Number of Bytes, B, transmitted is an integer multiple of the number of lanes, N:

All Data Lanes finish at the same time

LANE 1:	SoT	Byte 0	Byte N		Byte B-2N	Byte B-N	EoT
LANE 2:	SoT	Byte 1	Byte N+1		Byte B-2N+1	Byte B-N+1	EoT
LANE N-1:	SoT	Byte N-2	Byte 2N-2		Byte B-N-2	Byte B-2	EoT
LANE N:	SoT	Byte N-1	Byte 2N-1		Byte B-N-1	Byte B-1	EoT

Number of Bytes, B, transmitted is NOT an integer multiple of the number of lanes, N:

Data Lanes K+1 to N finish 1 byte earlier than Data Lanes 1 to K

LANE 1:	SoT	Byte 0	Byte N		Byte B-N-K	Byte B-K	EoT	
LANE K:	SoT	Byte K-1	Byte N+K-1		Byte B-N-1	Byte B-1	EoT	
LANE K+1:	SoT	Byte K	Byte N+K		Byte B-N	EoT		LPS
LANE N:	SoT	Byte N-1	Byte 2N-1		Byte B-K-1	EoT		LPS
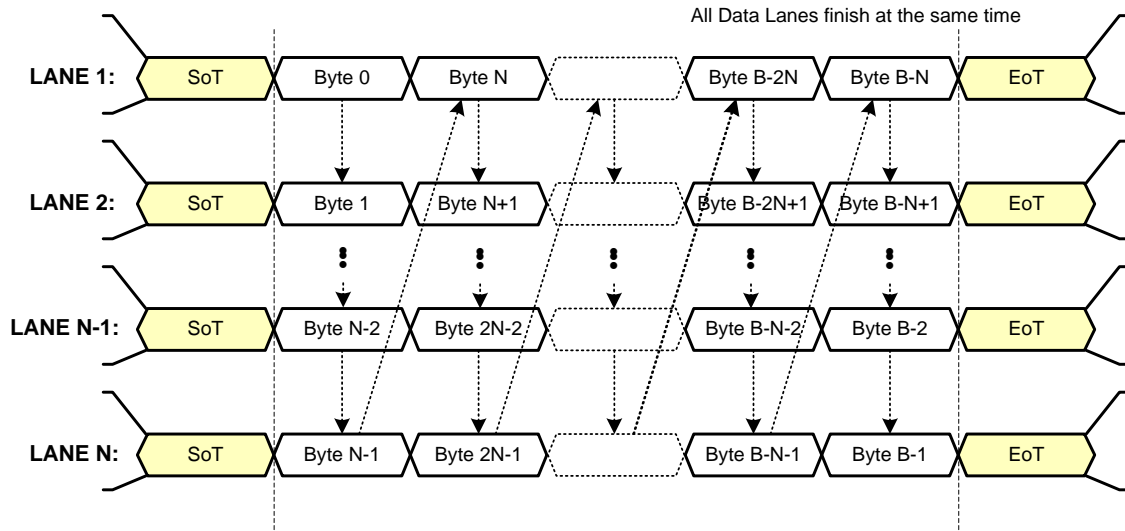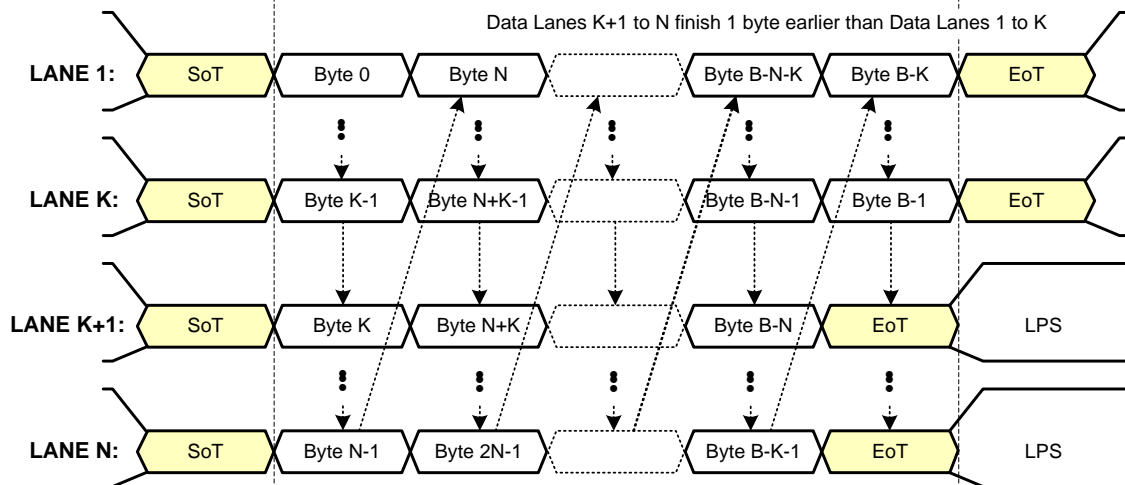
KEY:
LPS – Low Power State   SoT – Start of Transmission   EoT – End of Transmission

Figure 41 N-Lane Multi-Lane Example for D-PHY

Short packet of 4 bytes Transmitted on N lanes > 4

Bytes distributed on Lanes 1 to 4,
Lanes 5 to N stay in LPS

LANE 1: SoT | Byte 0 | EoT | LPS

LANE 4: SoT | Byte 3 | EoT | LPS

LANE 5: LPS

LANE N: LPS

KEY:
LPS – Low Power State   SoT – Start of Transmission   EoT – End of Transmission
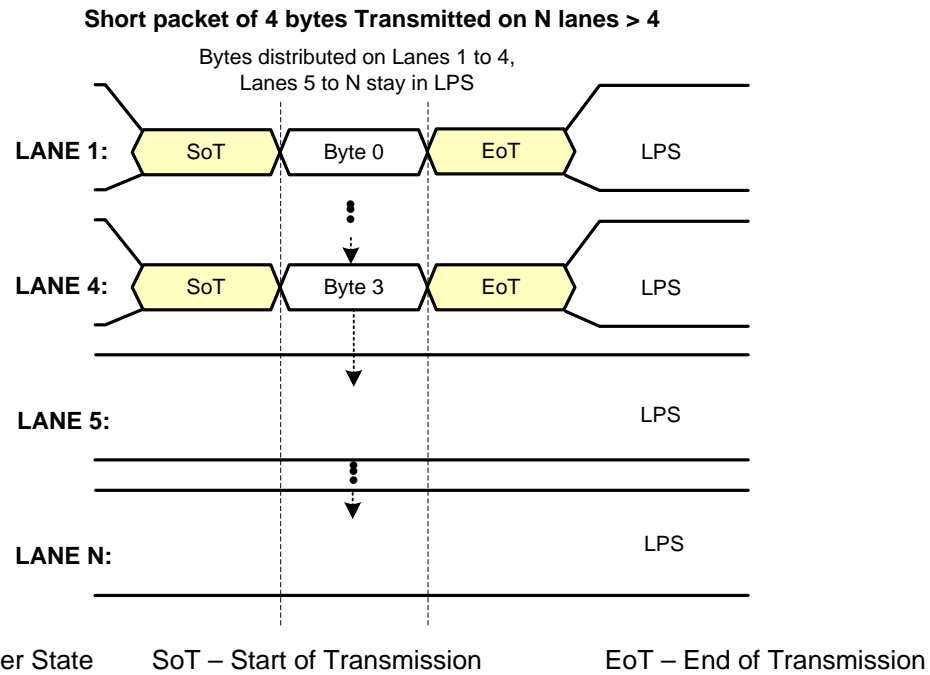
Figure 42 N-Lane Multi-Lane Example for D-PHY Short Packet Transmission

8.2   Lane Distribution for the C-PHY Physical Layer Option

Examples are shown in *Figure 43* and *Figure 44*:

- 2-Lane system (*Figure 43*): bytes 1 and 0 of the packet are sent as a 16-bit word to the Lane 1 C-PHY module, bytes 3 and 2 are sent to Lane 2, bytes 5 and 4 are sent to Lane 1, bytes 7 and 6 are sent to Lane 2, bytes 9 and 8 are sent to Lane 1, and so on.
- 3-Lane system (*Figure 44*): bytes 1 and 0 of the packet are sent as a 16-bit word to the Lane 1 C-PHY module, bytes 3 and 2 are sent to Lane 2, bytes 5 and 4 are sent to Lane 3, bytes 7 and 6 are sent to Lane 1, bytes 9 and 8 are sent to Lane 2, and so on.

Figure 45 illustrates normative behavior for an N-Lane system where $N \geq 1$: bytes 1 and 0 of the packet are sent as a 16-bit word to the Lane 1 C-PHY module, bytes 3 and 2 are sent to Lane 2, bytes 2N-1 and 2N-2 are sent to Lane N, bytes 2N+1 and 2N are sent to Lane 1, and so on. The last two bytes B-1 and B-2 are sent to Lane N, where B is the total number of bytes in the packet.

For an N-Lane transmitter, the C-PHY module for Lane n ($1 \leq n \leq N$) shall transmit the following sequence of {ms byte : ls byte} byte pairs from a B-byte packet generated by the low level protocol layer: {Byte 2*(k*N+n)-1 : Byte 2*(k*N+n)-2}, for k = 0, 1, 2, …, B/(2N) - 1, where Byte 0 is the first byte in the packet. The low level protocol shall guarantee that B is an integer multiple of 2N.

That is, at the end of the packet transmission, there shall be no "extra" bytes since the total byte count is always an even multiple of the number of Lanes, N. The Lane distributor, after sending the final set of 2N bytes in parallel to the N Lanes, simultaneously de-asserts its "valid data" signal to all Lanes, signaling to each C-PHY Lane module that it may start its EoT sequence.

Each C-PHY Lane module operates autonomously, but packet data transmission starts and stops at the same time on all Lanes.

The N C-PHY receiver modules on the receiving end of the link collect byte pairs in parallel, and feed them into the Lane-merging layer. This reconstitutes the original sequence of bytes in the transmission, which can then be partitioned into individual packets for the packet decoder layers.

All Lanes start and stop at the same time

| LANE 1: | SoT | Bytes 1:0 | Bytes 5:4 | Bytes 9:8 | | Bytes B-11: B-12 | Bytes B-7: B-8 | Bytes B-3: B-4 | EoT |

| LANE 2: | SoT | Bytes 3:2 | Bytes 7:6 | Bytes 11:10 | | Bytes B-9: B-10 | Bytes B-5: B-6 | Bytes B-1: B-2 | EoT |

KEY:

LPS – Low Power State   SoT – Start of Transmission   EoT – End of Transmission
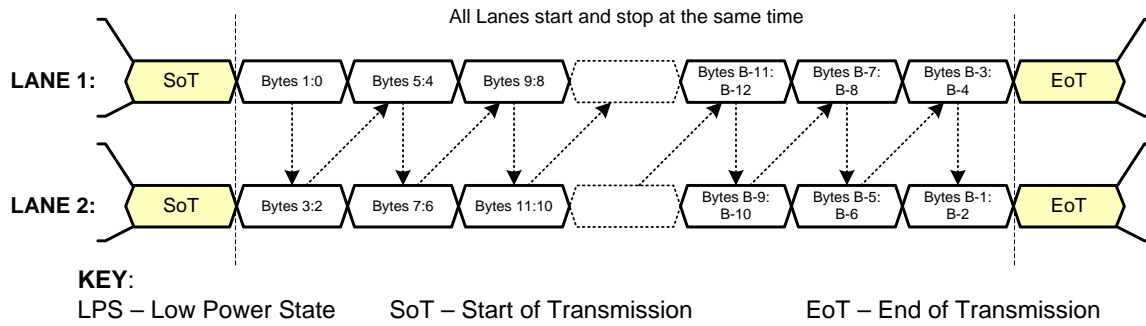
Figure 43 Two Lane Multi-Lane Example for C-PHY

All Lanes start and stop at the same time

| LANE 1: | SoT | Bytes 1:0 | Bytes 7:6 | Bytes 13:12 | | Bytes B-17: B-18 | Bytes B-11: B-12 | Bytes B-5: B-6 | EoT |

| LANE 2: | SoT | Bytes 3:2 | Bytes 9:8 | Bytes 15:14 | | Bytes B-15: B-16 | Bytes B-9: B-10 | Bytes B-3: B-4 | EoT |

| LANE 3: | SoT | Bytes 5:4 | Bytes 11:10 | Bytes 17:16 | | Bytes B-13: B-14 | Bytes B-7: B-8 | Bytes B-1: B-2 | EoT |

KEY:

LPS – Low Power State   SoT – Start of Transmission   EoT – End of Transmission
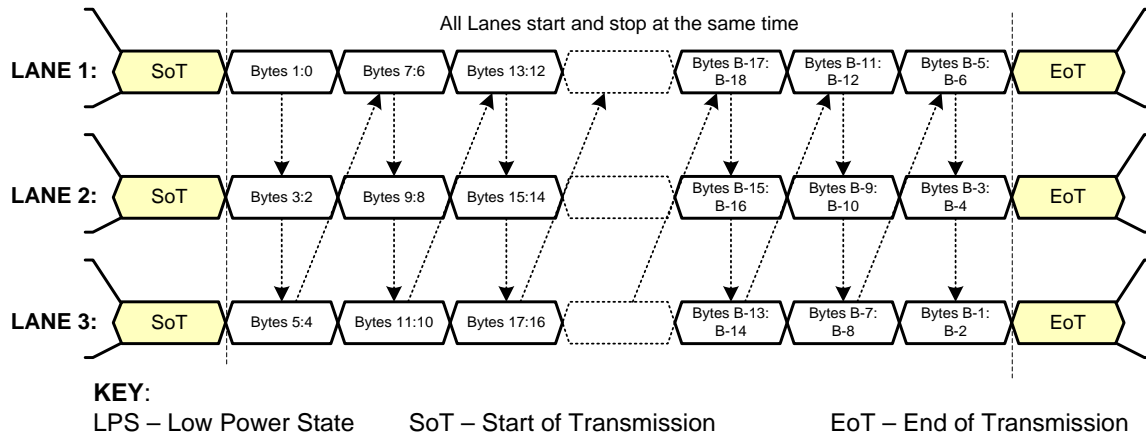
Figure 44 Three Lane Multi-Lane Example for C-PHY

All Lanes start and stop at the same time

| LANE 1: | SoT | Bytes 1:0 | Bytes 2N+1: 2N | Bytes 4N+1: 4N | | Bytes B-6N+1: B-6N | Bytes B-4N+1: B-4N | Bytes B-2N+1: B-2N | EoT |

| LANE 2: | SoT | Bytes 3:2 | Bytes 2N+3: 2N+2 | Bytes 4N+3: 4N+2 | | Bytes B-6N+3: B-6N+2 | Bytes B-4N+3: B-4N+2 | Bytes B-2N+3: B-2N+2 | EoT |

| LANE N-1: | SoT | Bytes 2N-3: 2N-4 | Bytes 4N-3: 4N-4 | Bytes 6N-3: 6N-4 | | Bytes B-4N-3: B-4N-4 | Bytes B-2N-3: B-2N-4 | Bytes B-3: B-4 | EoT |

| LANE N: | SoT | Bytes 2N-1: 2N-2 | Bytes 4N-1: 4N-2 | Bytes 6N-1: 6N-2 | | Bytes B-4N-1: B-4N-2 | Bytes B-2N-1: B-2N-2 | Bytes B-1: B-2 | EoT |

KEY:

LPS – Low Power State   SoT – Start of Transmission   EoT – End of Transmission

Figure 45 General N-Lane Multi-Lane Distribution for C-PHY

8.3   Multi-Lane Interoperability

832   The Lane distribution and merging layers shall be reconfigurable via the Camera Control Interface when
833   more than one data Lane is used.

834   An "N" data Lane receiver shall be connected with an "M" data Lane transmitter, by CCI configuration of
835   the Lane distribution and merging layers within the CSI-2 transmitter and receiver when more than one data
836   Lane is used. Thus, if M<=N a receiver with N data Lanes shall work with transmitters with M data Lanes.
837   Likewise, if M>=N a transmitter with M Lanes shall work with receivers with N data Lanes. Transmitter
838   Lanes 1 to M shall be connected to the receiver Lanes 1 to N.

839   Two cases:

840   • If M<=N then there is no loss of performance – the receiver has sufficient data Lanes to match the
841   transmitter (*Figure 46* and *Figure 47*).

842   • If M> N then there may be a loss of performance (e.g. frame rate) as the receiver has fewer data
843   Lanes than the transmitter (*Figure 48* and *Figure 49*).

844   • Note that while the examples shown are for the D-PHY physical layer option, the C-PHY physical
845   layer option is handled similarly, except there is no clock Lane.

846
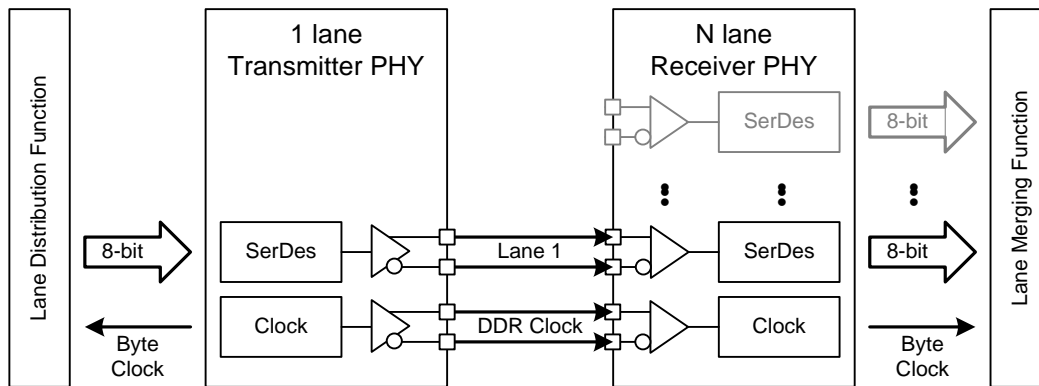
Figure 46 One Lane Transmitter and N-Lane Receiver Example for D-PHY

847
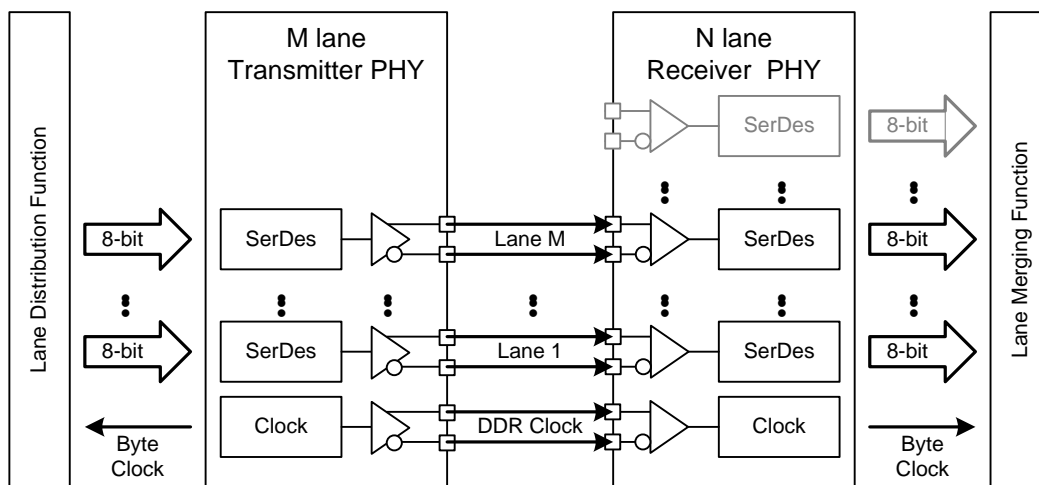
Figure 47 M-Lane Transmitter and N-Lane Receiver Example (M<N) for D-PHY
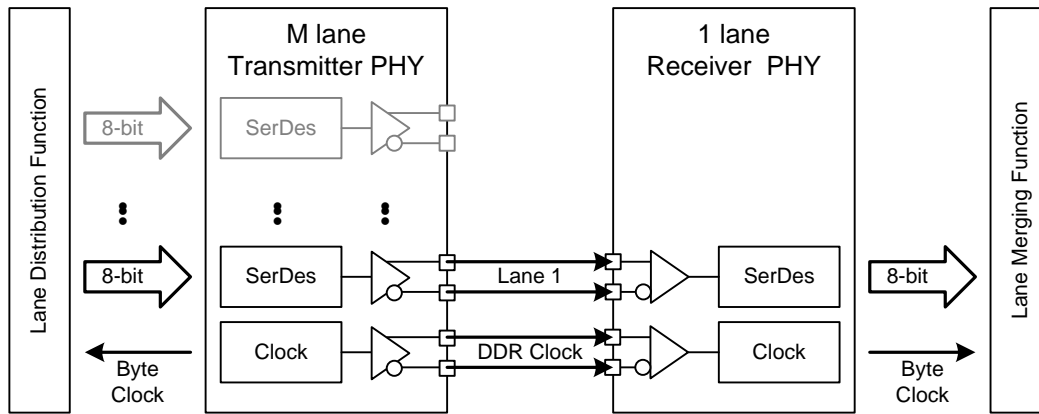
Figure 48 M-Lane Transmitter and One Lane Receiver Example for D-PHY
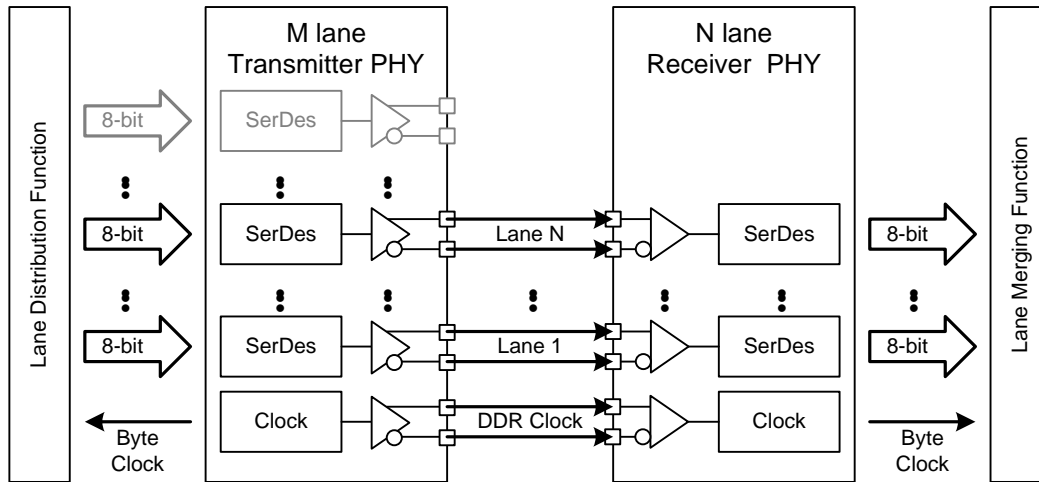
Figure 49 M-Lane Transmitter and N-Lane Receiver Example (N<M) for D-PHY

8.3.1   C-PHY Lane De-Skew

850   The PPI definition in the C-PHY Specification *[MIPI02]* defines one RxWordClkHS per Lane, and does
851   not address the use of a common receive RxWordClkHS for all Lanes within a Link. *Figure 50* shows a
852   mechanism for clocking data from the elastic buffers, in order to align (De-Skew) all RxDataHS to one
853   RxWordClkHS.

854
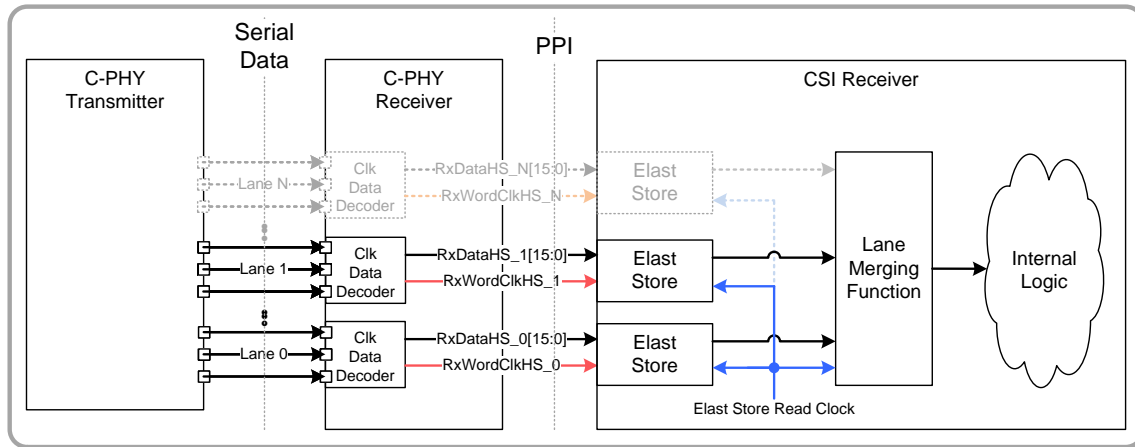
Figure 50 Example of Digital Logic to Align All RxDataHS

9 Low Level Protocol

855 The Low Level Protocol (LLP) is a byte orientated, packet based protocol that supports the transport of
856 arbitrary data using Short and Long packet formats. For simplicity, all examples in this section are single
857 Lane configurations unless specified otherwise.

858 Low Level Protocol Features:

859 • Transport of arbitrary data (Payload independent)

860 • 8-bit word size

861 • Support for up to sixteen interleaved virtual channels on the same D-PHY Link, or up to 32
862   interleaved virtual channels on the same C-PHY Link

863 • Special packets for frame start, frame end, line start and line end information

864 • Descriptor for the type, pixel depth and format of the Application Specific Payload data

865 • 16-bit Checksum Code for error detection.

866 • 6-bit Error Correction Code for error detection and correction (D-PHY physical layer only)

DATA:

KEY:
LPS – Low Power State   PH – Packet Header
ET – End of Transmission   PF – Packet Footer + Filler (if applicable)
ST – Start of Transmission

867
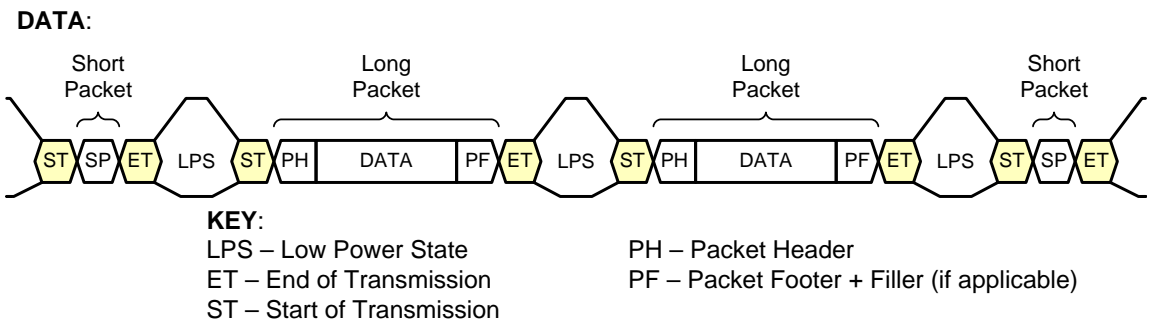
Figure 51 Low Level Protocol Packet Overview

9.1   Low Level Protocol Packet Format

868 As shown in ***Figure 51***, two packet structures are defined for low-level protocol communication: Long
869 packets and Short packets. The format and length of Short and Long Packets depends on the choice of
870 physical layer. For each packet structure, exit from the low power state followed by the Start of
871 Transmission (SoT) sequence indicates the start of the packet. The End of Transmission (EoT) sequence
872 followed by the low power state indicates the end of the packet.

9.1.1   Low Level Protocol Long Packet Format

873 ***Figure 52*** shows the structure of the Low Level Protocol Long Packet for the D-PHY physical layer option.
874 A Long Packet shall be identified by Data Types 0x10 to 0x37. See ***Table 10*** for a description of the Data
875 Types. A Long Packet for the D-PHY physical layer option shall consist of three elements: a 32-bit Packet
876 Header (PH), an application specific Data Payload with a variable number of 8-bit data words, and a 16-bit
877 Packet Footer (PF). The Packet Header is further composed of four elements: an 8-bit Data Identifier, a 16-
878 bit Word Count field, a 2-bit Virtual Channel Extension field, and a 6-bit ECC. The Packet footer has one
879 element, a 16-bit checksum (CRC). See ***Section 9.2*** through ***Section 9.5*** for further descriptions of the
880 packet elements.

881
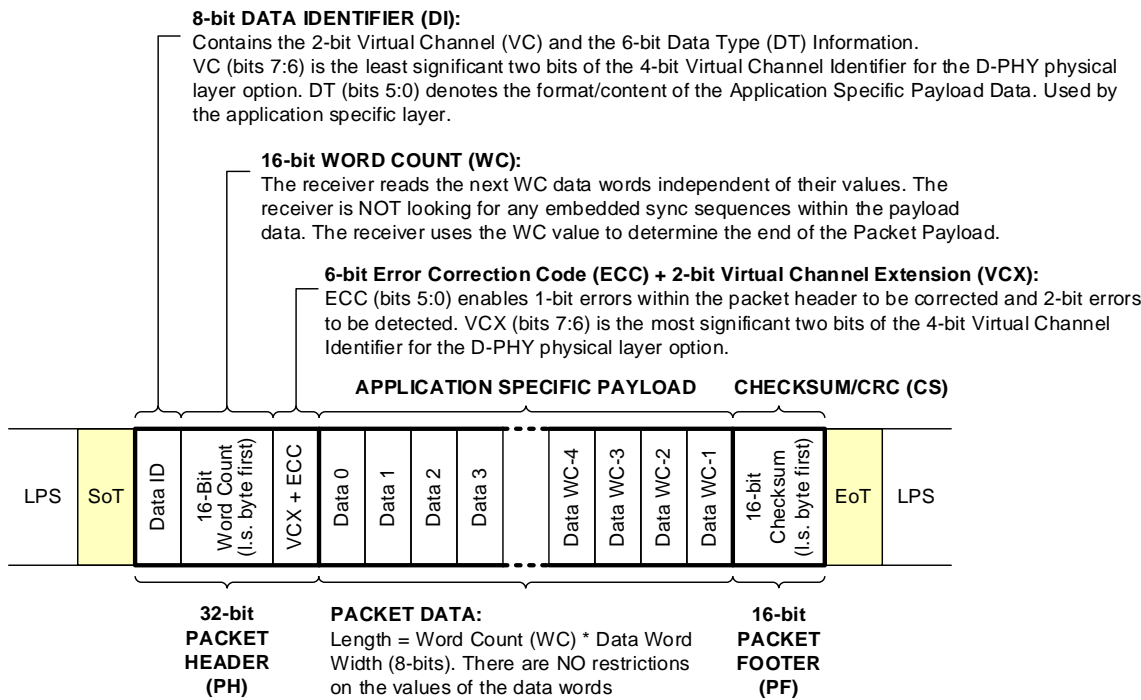
Figure 52 Long Packet Structure for D-PHY Physical Layer Option

882 ***Figure 53*** shows the Long Packet structure for the C-PHY physical layer option; it shall consist of four
883 elements: a Packet Header (PH), an application specific Data Payload with a variable number of 8-bit data
884 words, a 16-bit Packet Footer (PF), and zero or more Filler bytes (FILLER). The Packet Header is 6N x 16-
885 bits long, where N is the number of C-PHY physical layer Lanes. As shown in ***Figure 53***, the Packet
886 Header consists of two identical 6N-byte halves, where each half consists of N sequential copies of each of
887 the following fields: a 16-bit field containing five Reserved bits, a 3-bit Virtual Channel Extension (VCX)
888 field, and the 8-bit Data Identifier (DI); the 16-bit Packet Data Word Count (WC); and a 16-bit Packet
889 Header checksum (PH-CRC) which is computed over the previous four bytes. The value of each Reserved
890 bit shall be zero. The Packet Footer consists of a 16-bit checksum (CRC) computed over the Packet Data
891 using the same CRC polynomial as the Packet Header CRC and the Packet Footer used in the D-PHY
892 physical layer option. Packet Filler bytes are inserted after the Packet Footer, if needed, to ensure that the
893 Packet Footer ends on a 16-bit word boundary and that each C-PHY physical layer Lane transports the
894 same number of 16-bit words (i.e. byte pairs).
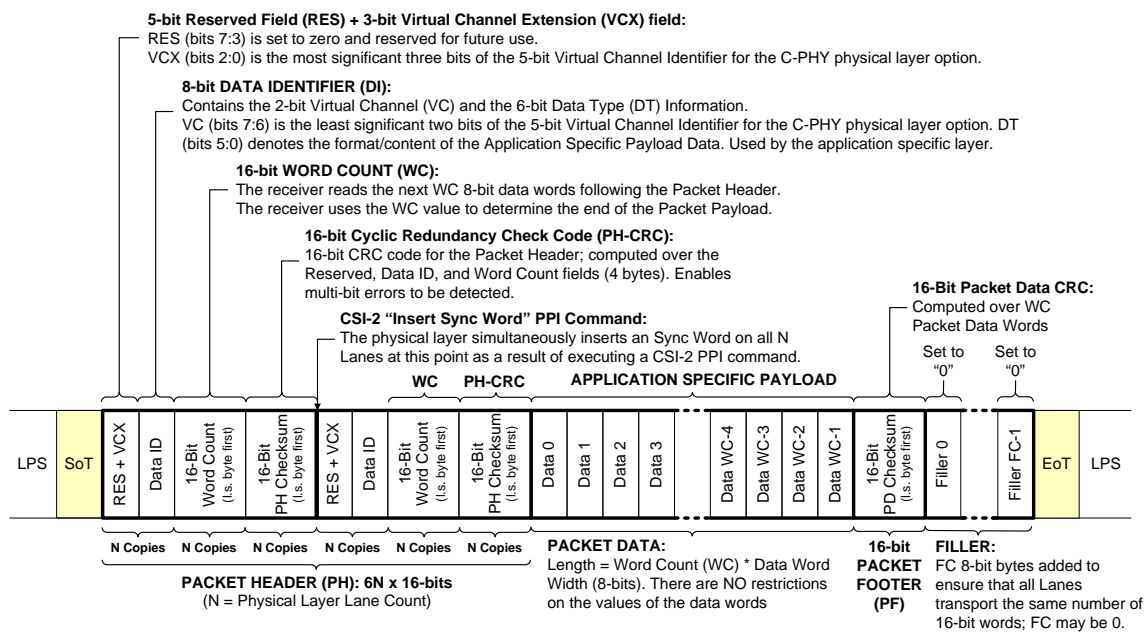
Figure 53 Long Packet Structure for C-PHY Physical Layer Option

896 As shown in *Figure 54*, the Packet Header structure depicted in *Figure 53* effectively results in the C-PHY
897 Lane Distributor broadcasting the same six 16-bit words to each of N Lanes. Furthermore, the six words per
898 Lane are split into two identical three-word groups which are separated by a mandatory C-PHY Sync Word
899 as described in *[MIPI02]*. The Sync Word is inserted by the C-PHY physical layer in response to a CSI-2
900 protocol transmitter PPI command.

901
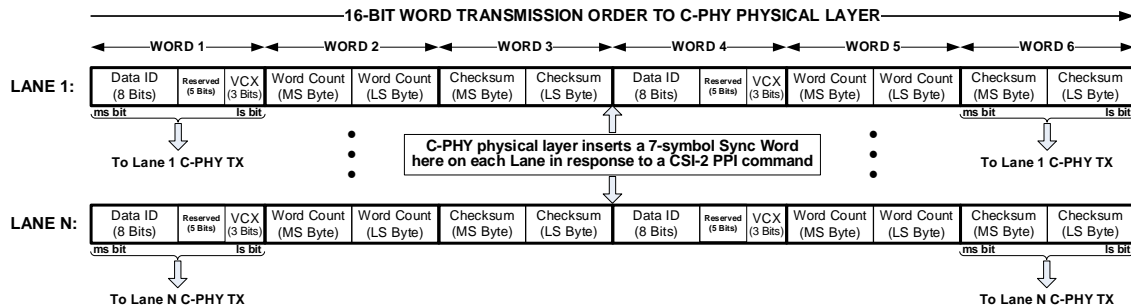
Figure 54 Packet Header Lane Distribution for C-PHY Physical Layer Option

902 For both physical layer options, the 8-bit Data Identifier field defines the 2-bit Virtual Channel (VC) and
903 the Data Type for the application specific payload data. The Virtual Channel Extension (VCX) field is also
904 common to both options, but is a 2-bit field for D-PHY and a 3-bit field for C-PHY. Together, the VC and
905 VCX fields comprise the 4- or 5-bit Virtual Channel Identifier field which determines the Virtual Channel
906 number associated with the packet (see *Section 9.3*).

907 For both physical layer options, the 16-bit Word Count (WC) field defines the number of 8-bit data words
908 in the Data Payload between the end of the Packet Header and the start of the Packet Footer. No Packet
909 Header, Packet Footer, or Packet Filler bytes shall be included in the Word Count.

910 For the D-PHY physical layer option, the 6-bit Error Correction Code (ECC) allows single-bit errors to be
911 corrected and 2-bit errors to be detected in the Packet Header. This includes the Data Identifier, Word
912 Count, and Virtual Channel Extension field values.

913 The ECC field is not used by the C-PHY physical layer option because a single symbol error on a C-PHY
914 physical link can cause multiple bit errors in the received CSI-2 Packet Header, rendering an ECC
915 ineffective. Instead, a CSI-2 protocol transmitter for the C-PHY physical layer option computes a 16-bit
916 CRC over the four bytes composing the Reserved, Virtual Channel Extension, Data Identifier, and Word
917 Count Packet Header fields and then transmits multiple copies of all these fields, including the CRC, to
918 facilitate their recovery by the CSI-2 protocol receiver in the event of one or more C-PHY physical link
919 errors. The multiple Sync Words inserted into the Packet Header by the C-PHY physical layer (as shown in
920 *Figure 54*) also facilitate Packet Header data recovery by enabling the C-PHY receiver to recover from lost
921 symbol clocks; see *[MIPI02]* for further information about the C-PHY Sync Word and symbol clock
922 recovery.

923 For both physical layer options, the CSI-2 receiver reads the next WC 8-bit data words of the Data Payload
924 following the Packet Header. While reading the Data Payload the receiver shall not look for any embedded
925 sync codes. Therefore, there are no limitations on the value of an 8-bit payload data word. In the generic
926 case, the length of the Data Payload shall always be a multiple of 8-bit data words. In addition, each Data
927 Type may impose additional restrictions on the length of the Data Payload, e.g. require a multiple of four
928 bytes.

929 For both physical layer options, once the CSI-2 receiver has read the Data Payload, it then reads the 16-bit
930 checksum (CRC) in the Packet Footer and compares it against its own calculated checksum to determine if
931 any Data Payload errors have occurred.

932 Filler bytes are only inserted by the CSI-2 transmitter's low level protocol layer in conjunction with the
933 C-PHY physical layer option. The value of any Filler byte shall be zero. If the Packet Data Word Count

934 (WC) is an odd number (i.e. LSB is "1"), the CSI-2 transmitter shall insert one Packet Filler byte after the
935 Packet Footer to ensure that the Packet Footer ends on a 16-bit word boundary. The CSI-2 transmitter shall
936 also insert additional Filler bytes, if needed, to ensure that each C-PHY Lane transports the same number of
937 16-bit words. The latter rules require the total number of Filler bytes, FC, to be greater than or equal to
938 (WC mod 2) + {{N - (([WC + 2 + (WC mod 2)] / 2) mod N)} mod N} * 2, where N is the number of
939 Lanes. Note that it is possible for FC to be zero.

940 *Figure 55* illustrates the Lane distribution of the minimal number of Filler bytes required for packets of
941 various lengths transmitted over three C-PHY Lanes. The total number of Filler bytes required per packet
942 ranges from 0 to 5, depending on the value of the Packet Data Word Count (WC). In general, the minimal
943 number of Filler bytes required per packet ranges from 0 to 2N-1 for an N-Lane C-PHY system.

944 For the D-PHY physical layer option, the CSI-2 Lane Distributor function shall pass each byte to the
945 physical layer which then serially transmits it least significant bit first.

946 For the C-PHY physical layer option, the Lane Distributor function shall group each pair of consecutive
947 bytes 2n and 2n+1 (for n ≥ 0) received from the Low Level Protocol into a 16-bit word (whose least
948 significant byte is byte 2n) and then pass this word to a physical layer Lane module. The C-PHY Lane
949 module maps each 16-bit word into a 7-symbol word which it then serially transmits least significant
950 symbol first.

951 For both physical layer options, payload data may be presented to the Lane Distributor function in any byte
952 order restricted only by data format requirements. Multi-byte protocol elements such as Word Count,
953 Checksum and the Short packet 16-bit Data Field shall be presented to the Lane Distributor function least
954 significant byte first.

955 After the EoT sequence the receiver begins looking for the next SoT sequence.
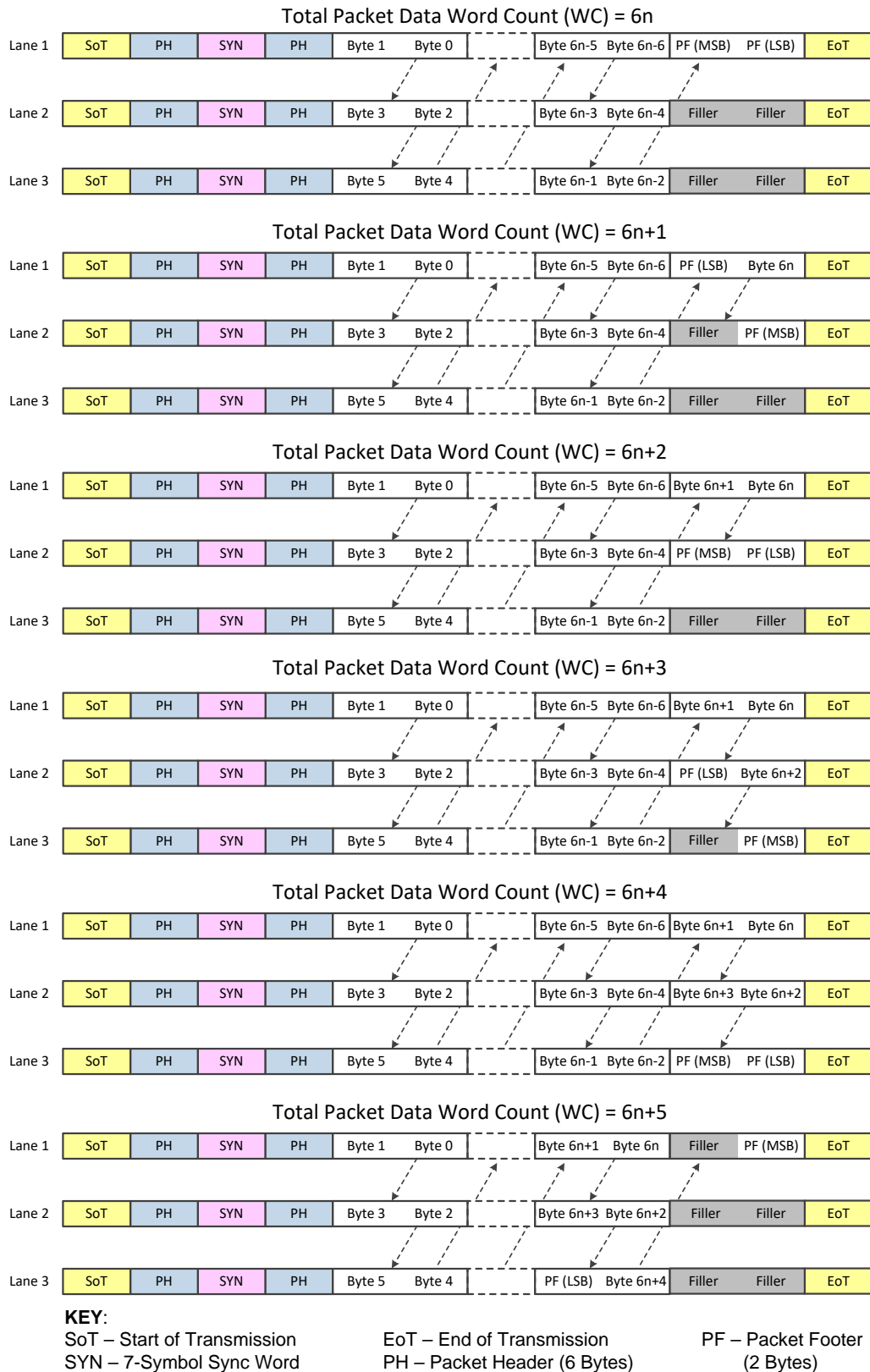
Figure 55 Minimal Filler Byte Insertion Requirements for Three Lane C-PHY

KEY:
SoT – Start of Transmission   EoT – End of Transmission   PF – Packet Footer
SYN – 7-Symbol Sync Word   PH – Packet Header (6 Bytes)   (2 Bytes)

9.1.2   Low Level Protocol Short Packet Format

957 *Figure 56* and *Figure 57* show the Low Level Protocol Short Packet structures for the D-PHY and C-PHY
958 physical layer options, respectively. For each option, the Short Packet structure matches the Packet Header
959 of the corresponding Low Level Protocol Long Packet structure with the exception that the Packet Header
960 Word Count (WC) field shall be replaced by the Short Packet Data Field. A Short Packet shall be identified
961 by Data Types 0x00 to 0x0F. See *Table 10* for a description of the Data Types. A Short Packet shall contain
962 only a Packet Header; neither Packet Footer nor Packet Filler bytes shall be present.

963 For Frame Synchronization Data Types the Short Packet Data Field shall be the frame number. For Line
964 Synchronization Data Types the Short Packet Data Field shall be the line number. See *Table 13* for a
965 description of the Frame and Line synchronization Data Types.

966 For Generic Short Packet Data Types the content of the Short Packet Data Field shall be user defined.

967 For the D-PHY physical layer option, the Error Correction Code (ECC) field allows single-bit errors to be
968 corrected and 2-bit errors to be detected in the Short Packet. For the C-PHY physical layer option, the 16-
969 bit Checksum (CRC) allows one or more bit errors to be detected in the Short Packet but does not support
970 error correction; the latter is facilitated by transmitting multiple copies of the various Short Packet fields
971 and by C-PHY Sync Word insertion on all Lanes.

972

32-bit SHORT PACKET (SH)
Data Type (DT) = 0x00 – 0x0F
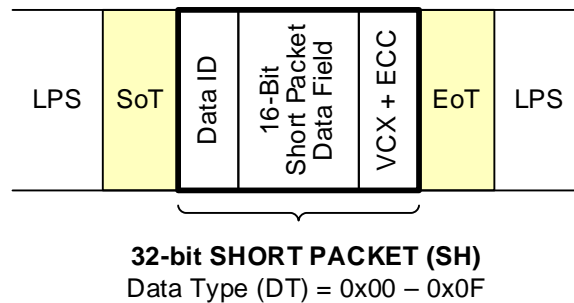
Figure 56 Short Packet Structure for D-PHY Physical Layer Option


973
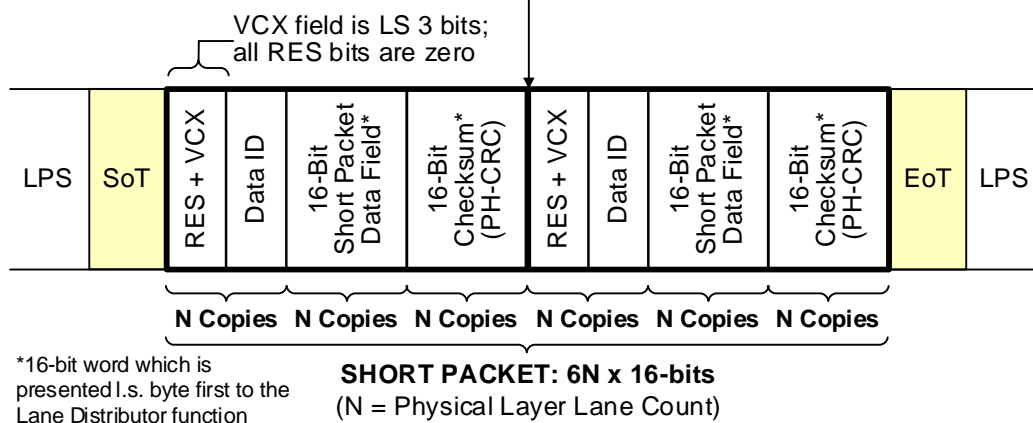
Figure 57 Short Packet Structure for C-PHY Physical Layer Option

9.2   Data Identifier (DI)

974   The Data Identifier byte contains the Virtual Channel (VC) and Data Type (DT) fields as illustrated in
975   *Figure 58*. The Virtual Channel field is contained in the two MS bits of the Data Identifier Byte. The Data
976   Type field is contained in the six LS bits of the Data Identifier Byte.

977

Data Identifier (DI) Byte

DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0
VC				DT			

Virtual Channel
(VC)   **Data Type**
(DT)
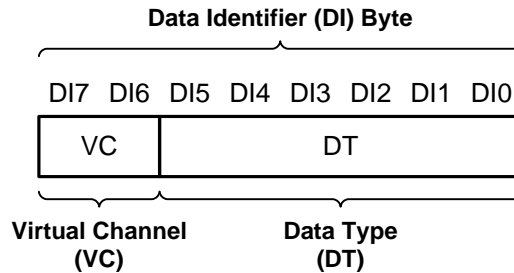
Figure 58 Data Identifier Byte

9.3   Virtual Channel Identifier

978   The purpose of the 4- or 5-bit Virtual Channel Identifier is to provide a means for designating separate
979   logical channels for different data flows that are interleaved in the data stream.

980   As shown in *Figure 59*, the least significant two bits of the Virtual Channel Identifier shall be copied from
981   the 2-bit VC field, and the most significant two or three bits shall be copied from the VCX field. The VCX
982   field is located in the Packet Header as shown in *Figure 52* and *Figure 53*, respectively, for the D-PHY and
983   C-PHY physical layer options. The Receiver shall extract the Virtual Channel Identifier from incoming
984   Packet Headers and de-multiplex the interleaved video data streams to their appropriate channel. A
985   maximum of N data streams is supported, where N = 16 or 32, respectively, for the D-PHY or C-PHY
986   physical layer option; valid channel identifiers are 0 to N-1. The Virtual Channel Identifiers in peripherals
987   should be programmable to allow the host processor to control how the data streams are de-multiplexed.

988   Host processors receiving packets from peripherals conforming to previous CSI-2 Specification versions
989   not supporting the VCX field shall treat the received value of VCX in all such packets as zero. Similarly,
990   peripherals conforming to this CSI-2 Specification version shall set the VCX field to zero in all packets
991   transmitted to host processors conforming with previous versions not supporting the VCX field. The means
992   by which host processors and peripherals meet these requirements are outside the scope of this
993   Specification.

994
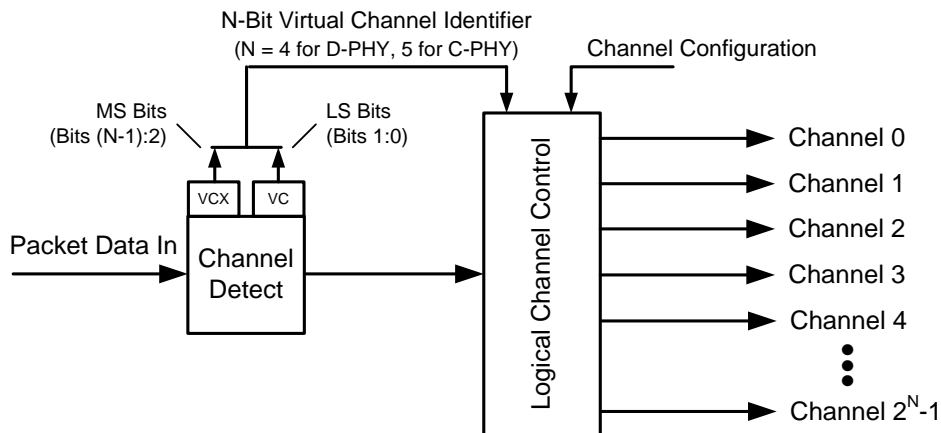
Figure 59 Logical Channel Block Diagram (Receiver)

995 *Figure 60* illustrates an example of data streams utilizing virtual channel support.

KEY:
LPS – Low Power State   PH – Packet Header
SoT – Start of Transmission   PF – Packet Footer + Filler (if applicable)
EoT – End of Transmission

996
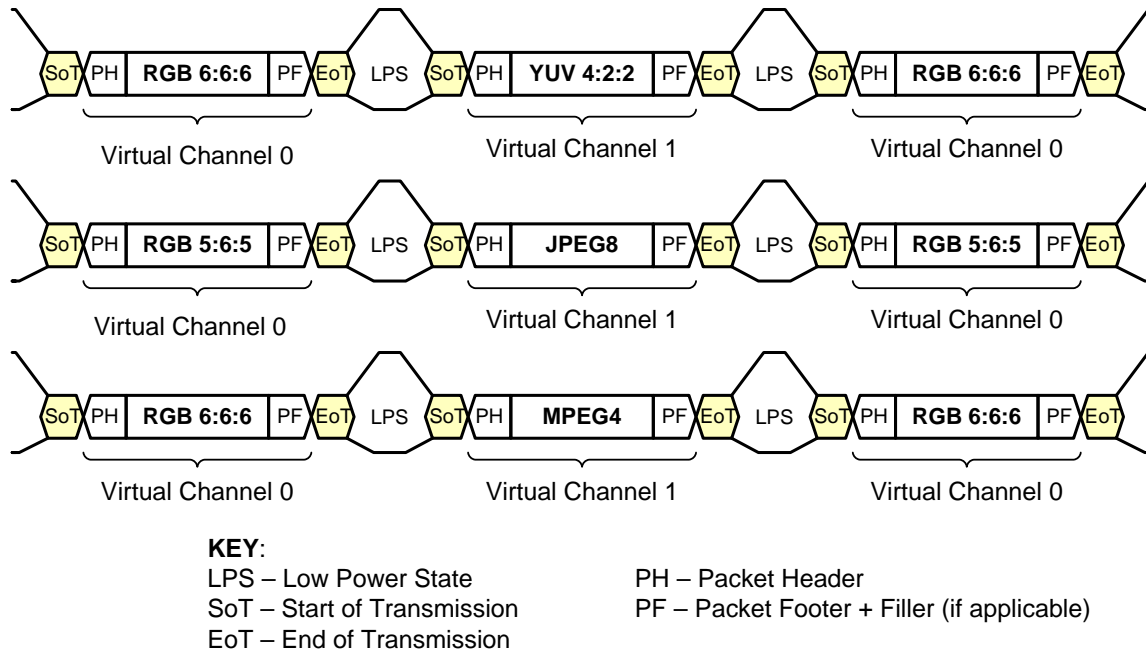
Figure 60 Interleaved Video Data Streams Examples

9.4   Data Type (DT)

997 The Data Type value specifies the format and content of the payload data. A maximum of sixty-four data
998 types are supported.

999 There are eight different data type classes as shown in *Table 10*. Within each class there are up to eight
1000 different data type definitions. The first two classes denote short packet data types. The remaining six
1001 classes denote long packet data types.

1002 For details on the short packet data type classes refer to *Section 9.8*.

1003 For details on the five long packet data type classes refer to *Section 11*.

1004   **Table 10 Data Type Classes**

Data Type	Description
0x00 to 0x07	Synchronization Short Packet Data Types
0x08 to 0x0F	Generic Short Packet Data Types
0x10 to 0x17	Generic Long Packet Data Types
0x18 to 0x1F	YUV Data
0x20 to 0x27	RGB Data
0x28 to 0x2F	RAW Data
0x30 to 0x37	User Defined Byte-based Data
0x38 to 0x3F	Reserved

9.5 Packet Header Error Correction Code for D-PHY Physical Layer Option

1005 The correct interpretation of the Data Identifier, Word Count, and Virtual Channel Extension fields is vital
1006 to the packet structure. The 6-bit Packet Header Error Correction Code (ECC) allows single-bit errors in the
1007 latter fields to be corrected, and two-bit errors to be detected for the D-PHY physical layer option; the ECC
1008 is not available for the C-PHY physical layer option. A 26-bit subset of the Hamming-Modified code
1009 described in *Section 9.5.2* shall be used. The error state resuts of ECC decoding shall be available at the
1010 Application layer in the receiver.

1011 The Data Identifier field DI[7:0] shall map to D[7:0] of the ECC input, the Word Count LS Byte (WC[7:0])
1012 to D[15:8], the Word Count MS Byte (WC[15:8]) to D[23:16], and the Virtual Channel Extension (VCX)
1013 field to D[25:24]. This mapping is shown in *Figure 61*, which also serves as an ECC calculation example.

1014
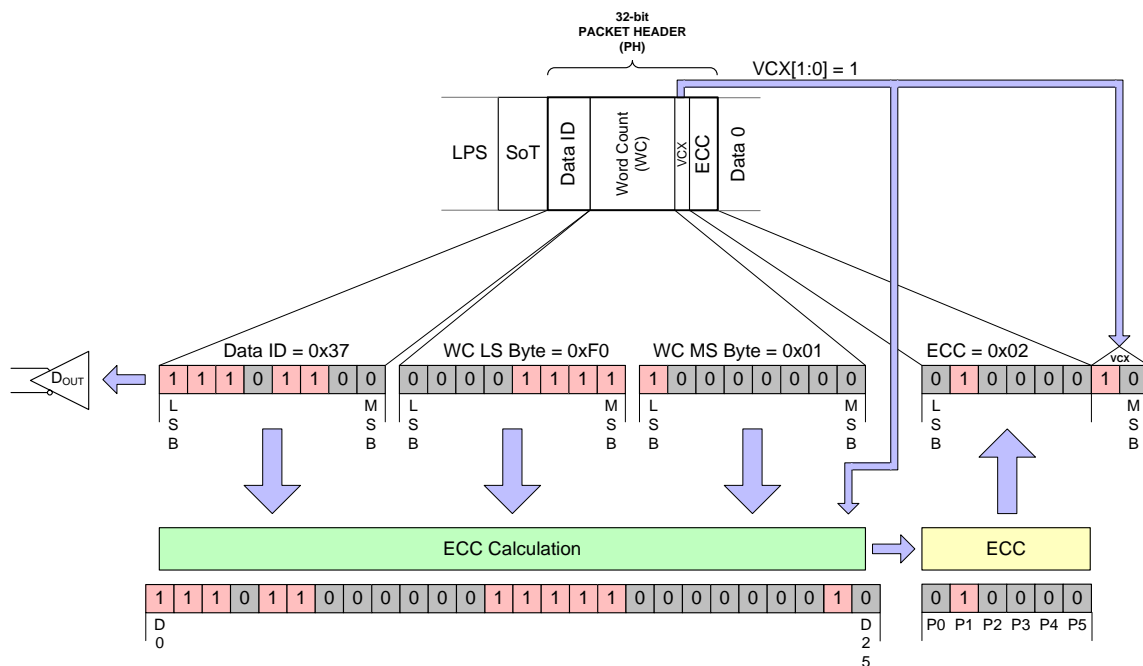
Figure 61 26-bit ECC Generation Example

9.5.1 General Hamming Code Applied to Packet Header

1015 The number of parity or error check bits required is given by the Hamming rule, and is a function of the
1016 number of bits of information transmitted. The Hamming rule is expressed by the following inequality:

1017 $d + p + 1 \leq 2^p$, where d is the number of data bits and p is the number of parity bits.

1018 The result of appending the computed parity bits to the data bits is called the Hamming code word. The size
1019 of the code word c is obviously $d + p$, and a Hamming code word is described by the ordered set (c, d). A
1020 Hamming code word is generated by multiplying the data bits by a generator matrix **G**. The resulting
1021 product is the code-word vector (c1, c2, c3 … cn), consisting of the original data bits and the calculated
1022 parity bits. The generator matrix **G** used in constructing Hamming codes consists of **I** (the identity matrix)
1023 and a parity generation matrix **A**:

1024 $\mathbf{G} = [\,\mathbf{I}\,|\,\mathbf{A}\,]$

1025 The packet header plus the ECC code can be obtained as: $PH = p*\mathbf{G}$ where p represents the header (26 or
1026 64 bits) and $\mathbf{G}$ is the corresponding generator matrix.

1027 Validating the received code word r, involves multiplying it by a parity check to form s, the syndrome or
1028 parity check vector: $s = \mathbf{H}*PH$ where PH is the received packet header and $\mathbf{H}$ is the parity check matrix:

1029 $\mathbf{H} = [\,\mathbf{A^T}\,|\,\mathbf{I}\,]$

1030 If all elements of s are zero, the code word was received correctly. If s contains non-zero elements, then at
1031 least one error is present. If a single bit error is encountered then the syndrome s is one of the elements of $\mathbf{H}$
1032 which will point to the bit in error. Further, in this case, if the bit in error is one of the parity bits, then the
1033 syndrome will be one of the elements on $\mathbf{I}$, else it will be the data bit identified by the position of the
1034 syndrome in $\mathbf{A^T}$.

9.5.2   Hamming-Modified Code

1035 The error correcting code used is a 7+1 bits Hamming-modified code (72,64) and the subset of it is 5+1 bits
1036 or (32,26). Hamming codes use parity to correct one error or detect two errors, but they are not capable of
1037 doing both simultaneously, thus one extra parity bit is added. The code used allows the same 6-bit
1038 syndromes to correct the first 26-bits of a 64-bit sequence. To specify a compact encoding of parity and
1039 decoding of syndromes, the matrix shown in *Table 11* is used:

1040 **Table 11 ECC Syndrome Association Matrix**

d5d4d3	d2d1d0							
	0b000	0b001	0b010	0b011	0b100	0b101	0b110	0b111
0b000	0x07	0x0B	0x0D	0x0E	0x13	0x15	0x16	0x19
0b001	0x1A	0x1C	0x23	0x25	0x26	0x29	0x2A	0x2C
0b010	0x31	0x32	0x34	0x38	0x1F	0x2F	0x37	0x3B
0b011	0x3D	0x3E	0x46	0x49	0x4A	0x4C	0x51	0x52
0b100	0x54	0x58	0x61	0x62	0x64	0x68	0x70	0x83
0b101	0x85	0x86	0x89	0x8A	0x43	0x45	0x4F	0x57
0b110	0x8C	0x91	0x92	0x94	0x98	0xA1	0xA2	0xA4
0b111	0xA8	0xB0	0xC1	0xC2	0xC4	0xC8	0xD0	0xE0

1041 Each cell in the matrix represents a syndrome, and the first 26 cells (the orange cells) use the first three or
1042 five bits to build the syndrome. Each syndrome in the matrix is MSB left aligned:

1043 e.g. 0x07 = 0b0000_0111 = P7 P6 P5 P4 P3 P2 P1 P0

1044 The top row defines the three LSB of data position bit, and the left column defines the three MSB of data
1045 position bit (there are 64-bit positions in total).

1046 e.g. 37th bit position is encoded 0b100_101 and has the syndrome 0x68.

1047 To derive the parity P0 for 26-bits, the P0's in the orange cells will define whether the corresponding bit
1048 position is used in P0 parity or not.

1049   e.g. $P0_{24\text{-bits}} = D0\wedge D1\wedge D2\wedge D4\wedge D5\wedge D7\wedge D10\wedge D11\wedge D13\wedge D16\wedge D20\wedge D21\wedge D22\wedge D23\wedge D24$

1050 Similarly, to derive the parity P0 for 64-bits, all P0's in *Table 12* will define the corresponding bit positions
1051 to be used.

1052 To correct a single data bit error, the syndrome must be one of the syndromes in *Table 11*. These syndromes
1053 identify the bit position in error. The syndrome is calculated as:

1054 $S = P_{SEND} \wedge P_{RECEIVED}$, where P_{SEND} is the 8/6-bit ECC field in the header and $P_{RECEIVED}$ is the
1055   calculated parity of the received header.

1056 *Table 12* represents the same information as the matrix in *Table 11*, organized so as to provide better insight
1057 into the way in which parity bits are formed out of data bits. The orange area of the table is used to form the
1058 ECC needed to protect a 26-bit header, whereas the whole table must be used to protect a 64-bit header.

1059 Previous CSI-2 specification versions not supporting the Virtual Channel Extension (VCX) field utilize a
1060 30-bit Hamming-modified code word with 24 data bits and 5+1 parity bits based on the first 24 bit
1061 positions of *Table 12* [i.e. a (30,24) ECC]. Packet Header bits 24 and 25 are set to zero by transmitters, and
1062 ignored by receivers conforming to such Specifications.

1063 When receiving Packet Headers with a (30,24) ECC, receivers conforming to this CSI-2 Specification
1064 version shall ignore the contents of bits 24 and 25 in such Packet Headers. The intent is for such receivers
1065 to ignore any errors occurring at these bit positions, in order to match the behavior of previous receivers.
1066 (See *Section 9.5.4* for implementation recommendations.)

1067

Table 12 ECC Parity Generation Rules

Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
0	0	0	0	0	0	1	1	1	0x07
1	0	0	0	0	1	0	1	1	0x0B
2	0	0	0	0	1	1	0	1	0x0D
3	0	0	0	0	1	1	1	0	0x0E
4	0	0	0	1	0	0	1	1	0x13
5	0	0	0	1	0	1	0	1	0x15
6	0	0	0	1	0	1	1	0	0x16
7	0	0	0	1	1	0	0	1	0x19
8	0	0	0	1	1	0	1	0	0x1A
9	0	0	0	1	1	1	0	0	0x1C
10	0	0	1	0	0	0	1	1	0x23
11	0	0	1	0	0	1	0	1	0x25
12	0	0	1	0	0	1	1	0	0x26
13	0	0	1	0	1	0	0	1	0x29
14	0	0	1	0	1	0	1	0	0x2A
15	0	0	1	0	1	1	0	0	0x2C
16	0	0	1	1	0	0	0	1	0x31
17	0	0	1	1	0	0	1	0	0x32
18	0	0	1	1	0	1	0	0	0x34
19	0	0	1	1	1	0	0	0	0x38
20	0	0	0	1	1	1	1	1	0x1F
21	0	0	1	0	1	1	1	1	0x2F
22	0	0	1	1	0	1	1	1	0x37
23	0	0	1	1	1	0	1	1	0x3B
24	0	0	1	1	1	1	0	1	0x3D
25	0	0	1	1	1	1	1	0	0x3E
26	0	1	0	0	0	1	1	0	0x46
27	0	1	0	0	1	0	0	1	0x49
28	0	1	0	0	1	0	1	0	0x4A
29	0	1	0	0	1	1	0	0	0x4C
30	0	1	0	1	0	0	0	1	0x51
31	0	1	0	1	0	0	1	0	0x52

Bit	P7	P6	P5	P4	P3	P2	P1	P0	Hex
32	0	1	0	1	0	1	0	0	0x54
33	0	1	0	1	1	0	0	0	0x58
34	0	1	1	0	0	0	0	1	0x61
35	0	1	1	0	0	0	1	0	0x62
36	0	1	1	0	0	1	0	0	0x64
37	0	1	1	0	1	0	0	0	0x68
38	0	1	1	1	0	0	0	0	0x70
39	1	0	0	0	0	0	1	1	0x83
40	1	0	0	0	0	1	0	1	0x85
41	1	0	0	0	0	1	1	0	0x86
42	1	0	0	0	1	0	0	1	0x89
43	1	0	0	0	1	0	1	0	0x8A
44	0	1	0	0	0	0	1	1	0x43
45	0	1	0	0	0	1	0	1	0x45
46	0	1	0	0	1	1	1	1	0x4F
47	0	1	0	1	0	1	1	1	0x57
48	1	0	0	0	1	1	0	0	0x8C
49	1	0	0	1	0	0	0	1	0x91
50	1	0	0	1	0	0	1	0	0x92
51	1	0	0	1	0	1	0	0	0x94
52	1	0	0	1	1	0	0	0	0x98
53	1	0	1	0	0	0	0	1	0xA1
54	1	0	1	0	0	0	1	0	0xA2
55	1	0	1	0	0	1	0	0	0xA4
56	1	0	1	0	1	0	0	0	0xA8
57	1	0	1	1	0	0	0	0	0xB0
58	1	1	0	0	0	0	0	1	0xC1
59	1	1	0	0	0	0	1	0	0xC2
60	1	1	0	0	0	1	0	0	0xC4
61	1	1	0	0	1	0	0	0	0xC8
62	1	1	0	1	0	0	0	0	0xD0
63	1	1	1	0	0	0	0	0	0xE0

9.5.3 ECC Generation on TX Side

1068   This is an informative section.

1069   The ECC can be easily implemented using a parallel approach as depicted in *Figure 62* for a 64-bit header.

1070
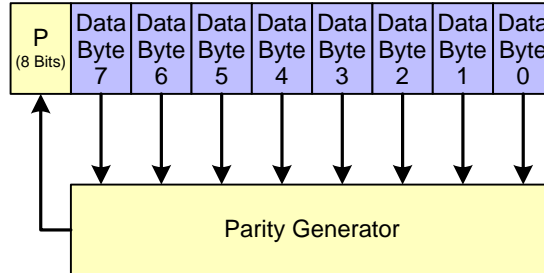
Figure 62 64-bit ECC Generation on TX Side

1071   And *Figure 63* for a 26-bit header:

1072
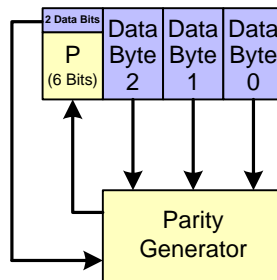
Figure 63 26-bit ECC Generation on TX Side

1073   The parity generators are based on *Table 12*.

1074   e.g. $P3_{26\text{-bit}} = D1 \wedge D2 \wedge D3 \wedge D7 \wedge D8 \wedge D9 \wedge D13 \wedge D14 \wedge D15 \wedge D19 \wedge D20 \wedge D21 \wedge D23 \wedge D24 \wedge D25$

1075   For backwards-compatibility, transmitters conforming to this CSI-2 Specification version should always set
1076   Packet Header bits 24 and 25 (the VCX field) to zero in any packets sent to receivers conforming to
1077   previous CSI-2 Specification versions incorporating a (30,24) ECC.

9.5.4   Applying ECC on RX Side (Informative)

Applying ECC on RX side involves generating a new ECC for the received Packet Header, computing the syndrome using the new ECC and the received ECC, decoding the syndrome to find if a single-error has occurred, and if so, correcting it. *Figure 64* depicts ECC processing for 64 received Packet Header data bits, using 8 parity bits.
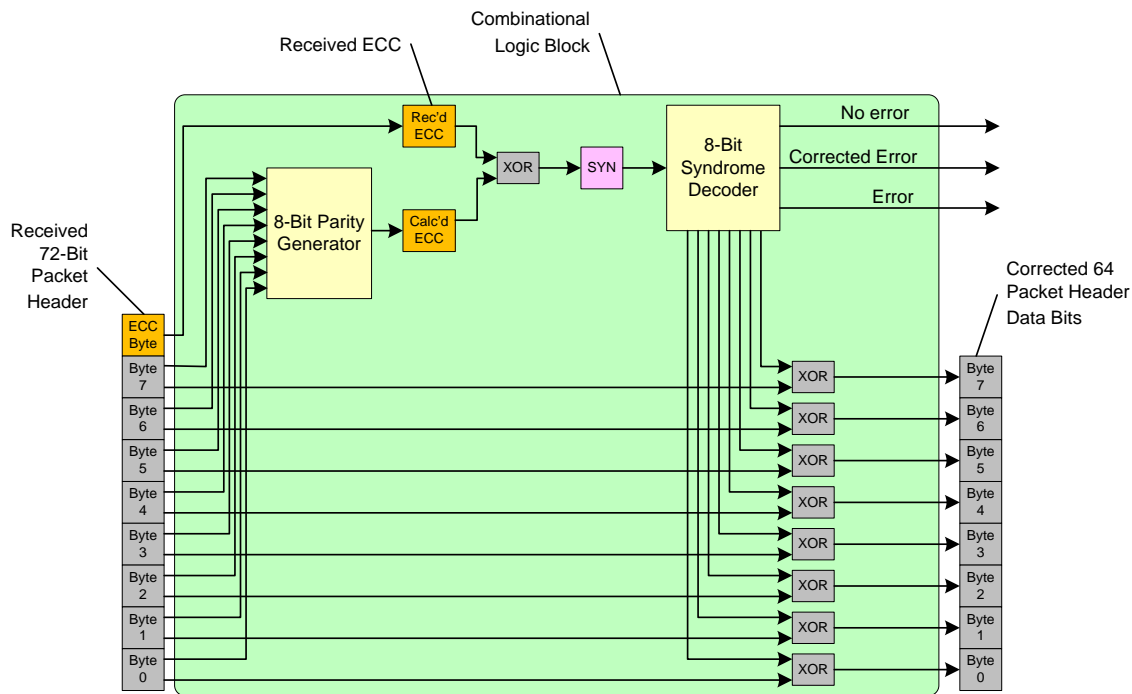
Figure 64 64-bit ECC on RX Side Including Error Correction

Decoding the syndrome has four possible outcomes:

3. If the syndrome is 0, no errors are present.

4. If the syndrome matches one of the matrix entries in the *Table 11*, then a single bit error has occurred and the corresponding bit position may be corrected by inverting it (e.g. by XORing with '1').

5. If the syndrome has only one bit set, then a single bit error has occurred at the parity bit located at that syndrome bit position, and the rest of the received packet header bits are error-free.

6. If the syndrome does not fit any of the other outcomes, then an uncorrectable error has occurred, and an error flag should be set (indicating that the Packet Header is corrupted).

The 26-bit implementation shown in *Figure 65* uses fewer terms to calculate the parity, and thus the syndrome decoding block is much simpler than the 64-bit implementation.

Receivers conforming to this CSI-2 Specification version that receive Packet Headers from transmitters without the VCX field should forcibly set received bits 24 and 25 to zero in such Packet Headers prior to any parity generation or syndrome decoding (this is the function of the "VCX Override" block shown in *Figure 65*). This guarantees that the receiver will properly ignore any errors occurring at bit positions 24 and 25, in order to match the behavior of receivers conforming to previous versions of this Specification.
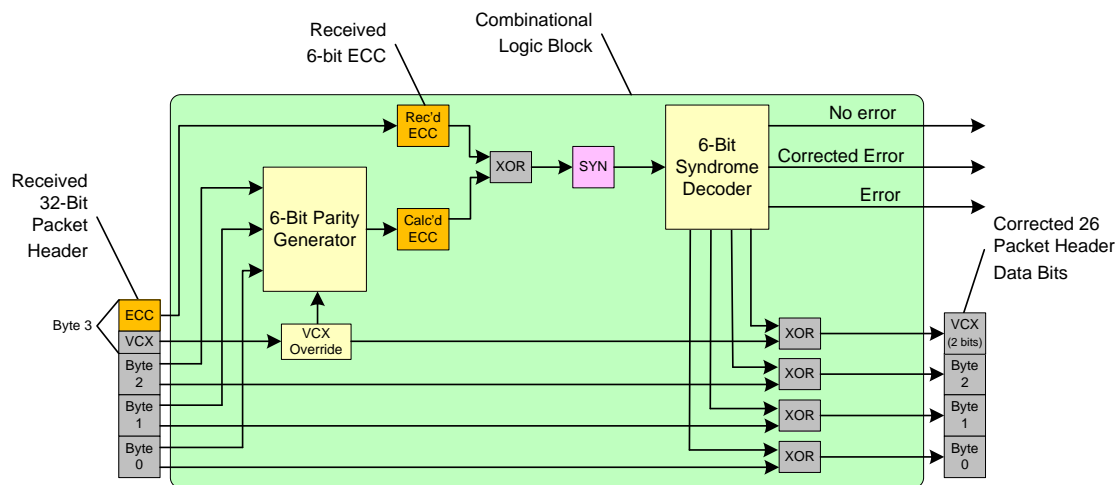
Figure 65 26-bit ECC on RX Side Including Error Correction

9.6   Checksum Generation

To detect possible errors in transmission, a checksum is calculated over the WC bytes composing the Packet Data of every Long Packet; a similar checksum is calculated over the four bytes composing the Reserved, Virtual Channel Extension, Data Identifier, and Word Count fields of every Packet Header for the C-PHY physical layer option. In all cases, the checksum is realized as 16-bit CRC based on the generator polynomial $x^{16}+x^{12}+x^5+x^0$ and is computed over bytes in the order in which they are presented to the Lane Distributor function by the low level protocol layer as shown in *Figure 52*, *Figure 53*, and *Figure 57*.

The order in which the checksum bytes are presented to the Lane Distributor function is illustrated in *Figure 66*.

16-bit Checksum

CRC LS Byte	CRC MS Byte
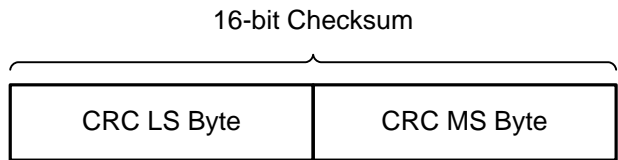
Figure 66 Checksum Transmission Byte Order

When computed over the Packet Data words of a Long Packet, the 16-bit checksum sequence is transmitted as part of the Packet Footer. When the Word Count is zero, the CRC shall be 0xFFFF. When computed over the Reserved, Virtual Channel Extension, Data Identifier, and Word Count fields of a Packet Header for the C-PHY physical layer option, the 16-bit checksum sequence is transmitted as part of the Packet Header CRC (PH-CRC) field.
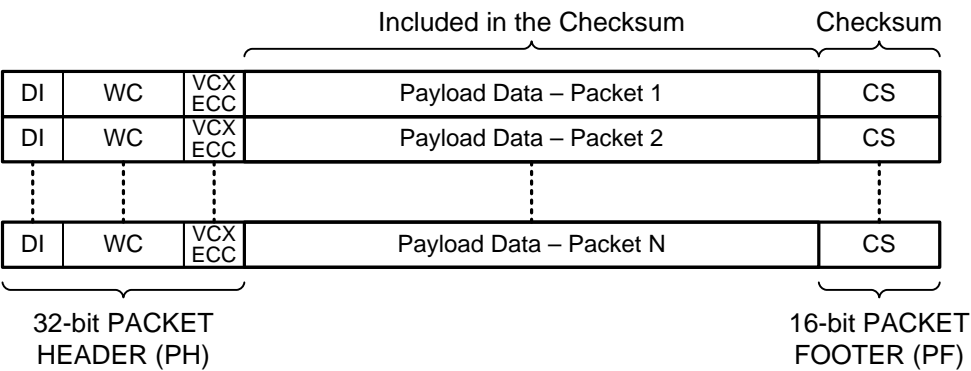
Figure 67 Checksum Generation for Long Packet Payload Data

The definition of a serial CRC implementation is presented in *Figure 68*. The CRC implementation shall be functionally equivalent with the C code presented in *Figure 69*. The CRC shift register is initialized to 0xFFFF at the beginning of each packet. Note that for the C-PHY physical layer option, if the same circuitry is used to compute both the Packet Header and Packet Footer CRC, the CRC shift register shall be initialized twice per packet, i.e. once at the beginning of the packet and then again following the computation of the Packet Header CRC. After all payload data has passed through the CRC circuitry, the CRC circuitry contains the checksum. The 16-bit checksum produced by the C code in *Figure 69* equals the final contents of the C[15:0] shift register shown in *Figure 68*. The checksum is then transmitted by the CSI-2 physical layer to the CSI-2 receiver to verify that no errors have occurred in the transmission.

Polynomial: $x^{16} + x^{12} + x^5 + x^0$

Note: C15 represents x^0, C0 represents x^{15}
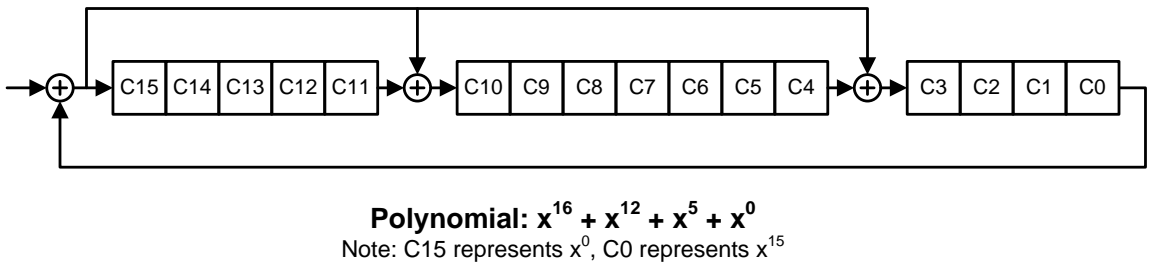
Figure 68 Definition of 16-bit CRC Shift Register

```
#define POLY 0x8408   /* 1021H bit reversed */

unsigned short crc16(char *data_p, unsigned short length)
{
   unsigned char i;
   unsigned int data;
   unsigned int crc = 0xffff;

   if (length == 0)
      return (unsigned short)(crc);
   do
   {
      for (i=0, data=(unsigned int)0xff & *data_p++;
        i < 8;i++, data >>= 1)
      {
         if ((crc & 0x0001) ^ (data & 0x0001))
            crc = (crc >> 1) ^ POLY;
         else
            crc >>= 1;
      }
   } while (--length);

   // Uncomment to change from little to big Endian
// crc = ((crc & 0xff) << 8) | ((crc & 0xff00) >> 8);

   return (unsigned short)(crc);
}
```

1125

Figure 69 16-bit CRC Software Implementation Example

1126 Beginning with index 0, the contents of the input data array in *Figure 69* are given by WC 8-bit payload
1127 data words for packet data CRC computations and by the four 8-bit [Reserved, VCX], Data Identifier, WC
1128 (LS byte), and WC (MS byte) fields for packet header CRC computations.

1129

1130 CRC computation examples:
1131 Input Data Bytes:
1132 FF 00 00 02 B9 DC F3 72 BB D4 B8 5A C8 75 C2 7C 81 F8 05 DF FF 00 00 01
1133 Checksum LS byte and MS byte:
1134 F0 00
1135
1136 Input Data Bytes:
1137 FF 00 00 00 1E F0 1E C7 4F 82 78 C5 82 E0 8C 70 D2 3C 78 E9 FF 00 00 01
1138 Checksum LS byte and MS byte:
1139 69 E5

9.7 Packet Spacing

All CSI-2 implementations shall support a transition into and out of the Low Power State (LPS) between Low Level Protocol packets; however, implementations may optionally remain in the High Speed State between packets as described in *Section 9.11*. *Figure 70* illustrates the packet spacing with the LPS.

The packet spacing illustrated in *Figure 70* does not have to be a multiple of 8-bit data words, as the receiver will resynchronize to the correct byte boundary during the SoT sequence prior to the Packet Header of the next packet.

SHORT / LONG PACKET SPACING:
Variable - always a LPS between packets
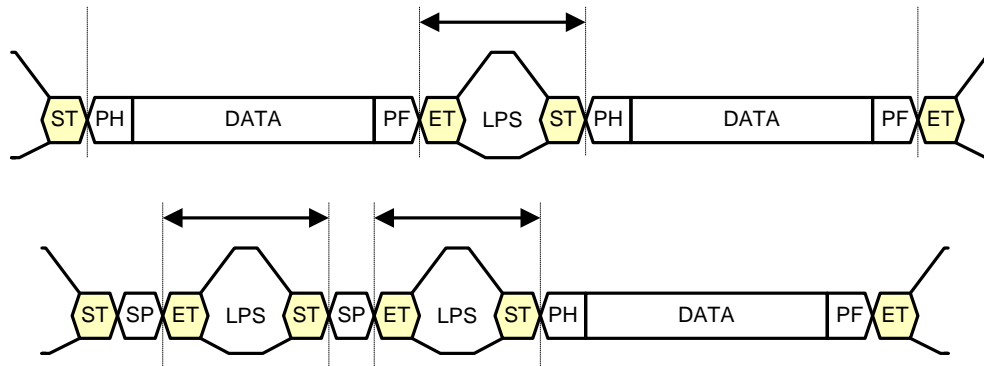
KEY:
LPS – Low Power State   PH – Packet Header
ST – Start of Transmission   PF – Packet Footer + Filler (if applicable)
ET – End of Transmission   SP – Short Packet

Figure 70 Packet Spacing

9.8 Synchronization Short Packet Data Type Codes

Short Packet Data Types shall be transmitted using only the Short Packet format. See *Section 9.1.2* for a format description.

Table 13 Synchronization Short Packet Data Type Codes

Data Type	Description
0x00	Frame Start Code
0x01	Frame End Code
0x02	Line Start Code (Optional)
0x03	Line End Code (Optional)
0x04 to 0x07	Reserved

9.8.1 Frame Synchronization Packets

Each image frame shall begin with a Frame Start (FS) Packet containing the Frame Start Code. The FS Packet shall be followed by one or more long packets containing image data and zero or more short packets containing synchronization codes. Each image frame shall end with a Frame End (FE) Packet containing the Frame End Code. See *Table 13* for a description of the synchronization code data types.

For FS and FE synchronization packets the Short Packet Data Field shall contain a 16-bit frame number. This frame number shall be the same for the FS and FE synchronization packets corresponding to a given frame.

The 16-bit frame number, when used, shall be non-zero to distinguish it from the use-case where frame number is inoperative and remains set to zero.

The behavior of the 16-bit frame number shall be one of the following:

- Frame number is always zero – frame number is inoperative.
- Frame number increments by 1 or 2 for every FS packet with the same Virtual Channel and is periodically reset to one; e.g. 1, 2, 1, 2, 1, 2, 1, 2 or 1, 2, 3, 4, 1, 2, 3, 4 or 1, 3, 5, 1, 3, 5 or 1, 2, 4, 1, 3, 4. Frame number may be incremented by 2 only when an image frame is masked (i.e. not transmitted) due to corruption. To accommodate such cases, increments by 1 or 2 may be freely intermixed within a sequence of frame numbers as needed.

9.8.2 Line Synchronization Packets

Line synchronization packets are optional on a per-image-frame basis. If an image frame includes line synchronization packets, it shall include both Line Start (LS) synchronization packets and Line End (LE) synchronization packets in each line of the frame.

For LS and LE synchronization packets, the Short Packet Data Field shall contain a 16-bit line number. This line number shall be the same for the LS and LE packets corresponding to a given line. Line numbers are logical line numbers and are not necessarily equal to the physical line numbers.

The 16-bit line number, when used, shall be non-zero to distinguish it from the case where line number is inoperative and remains set to zero.

The behavior of the 16-bit line number within the same Data Type and Virtual Channel shall be one of the following.

Either:

1. Line number is always zero – line number is inoperative.

Or:

2. Line number increments by one for every LS packet within the same Virtual Channel and the same Data Type. The line number is periodically reset to one for the first LS packet after a FS packet. The intended usage is for progressive scan (non- interlaced) video data streams. The line number must be a non-zero value.

Or:

3. Line number increments by the same arbitrary step value greater than one for every LS packet within the same Virtual Channel and the same Data Type. The line number is periodically reset to a non-zero arbitrary start value for the first LS packet after a FS packet. The arbitrary start value may be different between successive frames. The intended usage is for interlaced video data streams.

Figure 71 contains examples for the use of optional LS/LE packets within an interlaced frame with pixel data and additional embedded types. The Figure illustrates the use cases:

1. VC0 DT2 Interlaced frame with line counting incrementing by two. Frame1 starting at 1 and Frame2 starting at 2.

2. VC0 DT1 Progressive scan frame with line counting.

3. VC0 DT4 Progressive scan frame with non-operative line counting.

4. VC0 DT3 No LS/LE operation.
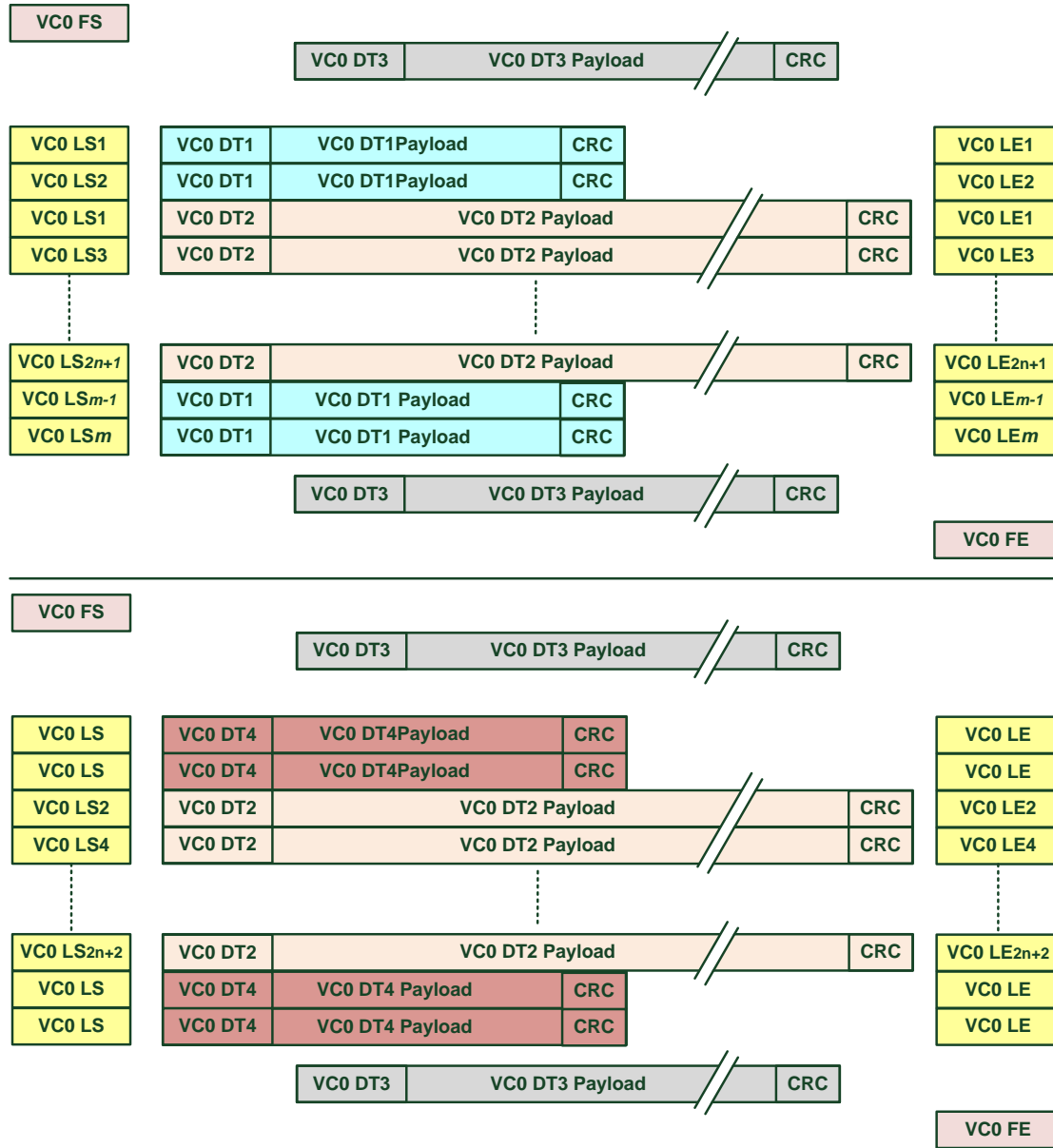
Note:
- For VC0 DT2 Odd Frames LS2n+1 and Even Frames LS2n+2 (where n=0,1,2,3...) the first line n=0
- For VC0 DT1 LSm+1(where m=0,1,2,3...) the first line m=0

Figure 71 Example Interlaced Frame Using LS/SE Short Packet and Line Counting

9.9   Generic Short Packet Data Type Codes

1197   *Table 14* lists the Generic Short Packet Data Types.

1198   **Table 14 Generic Short Packet Data Type Codes**

Data Type	Description
0x08	Generic Short Packet Code 1
0x09	Generic Short Packet Code 2
0x0A	Generic Short Packet Code 3
0x0B	Generic Short Packet Code 4
0x0C	Generic Short Packet Code 5
0x0D	Generic Short Packet Code 6
0x0E	Generic Short Packet Code 7
0x0F	Generic Short Packet Code 8

1199   The intention of the Generic Short Packet Data Types is to provide a mechanism for including timing
1200   information for the opening/closing of shutters, triggering of flashes, etc within the data stream. The intent
1201   of the 16-bit User defined data field in the generic short packets is to pass a data type value and a 16-bit
1202   data value from the transmitter to application layer in the receiver. The CSI-2 receiver shall pass the data
1203   type value and the associated 16-bit data value to the application layer.

9.10   Packet Spacing Examples Using the Low Power State

1204   Packets discussed in this section are separated by an EoT, LPS, SoT sequence as defined in *[MIPI01]* for
1205   the D-PHY physical layer option and *[MIPI02]* for the C-PHY physical layer option.

1206   *Figure 72* and *Figure 73* contain examples of data frames composed of multiple packets and a single
1207   packet, respectively.

1208   Note that the VVALID, HVALID and DVALID signals in the figures in this section are only concepts to
1209   help illustrate the behavior of the frame start/end and line start/end packets. The VVALID, HVALID and
1210   DVALID signals do not form part of the Specification.

1211
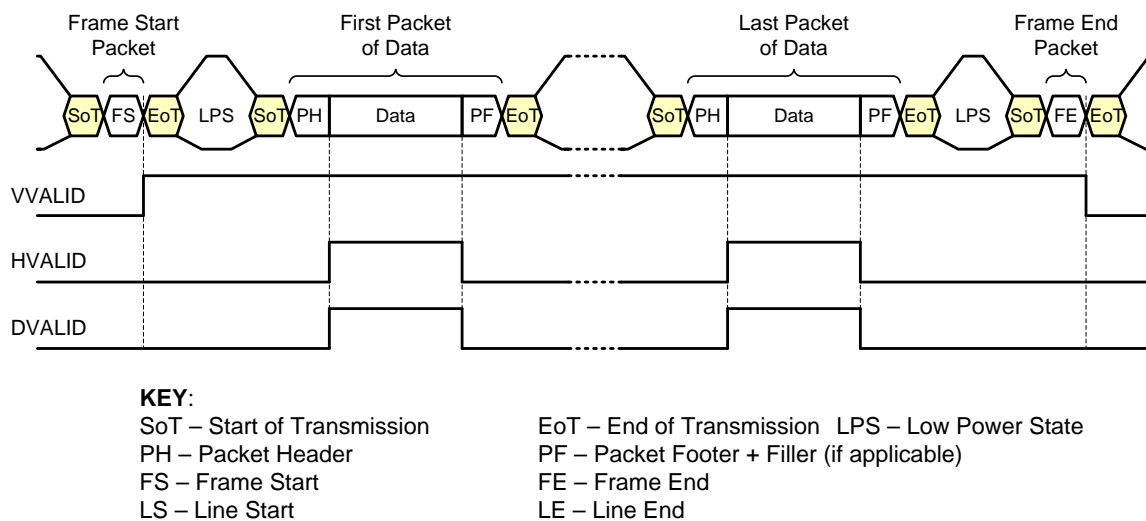
Figure 72 Multiple Packet Example

KEY:
SoT – Start of Transmission   EoT – End of Transmission   LPS – Low Power State
PH – Packet Header   PF – Packet Footer + Filler (if applicable)
FS – Frame Start   FE – Frame End
LS – Line Start   LE – Line End
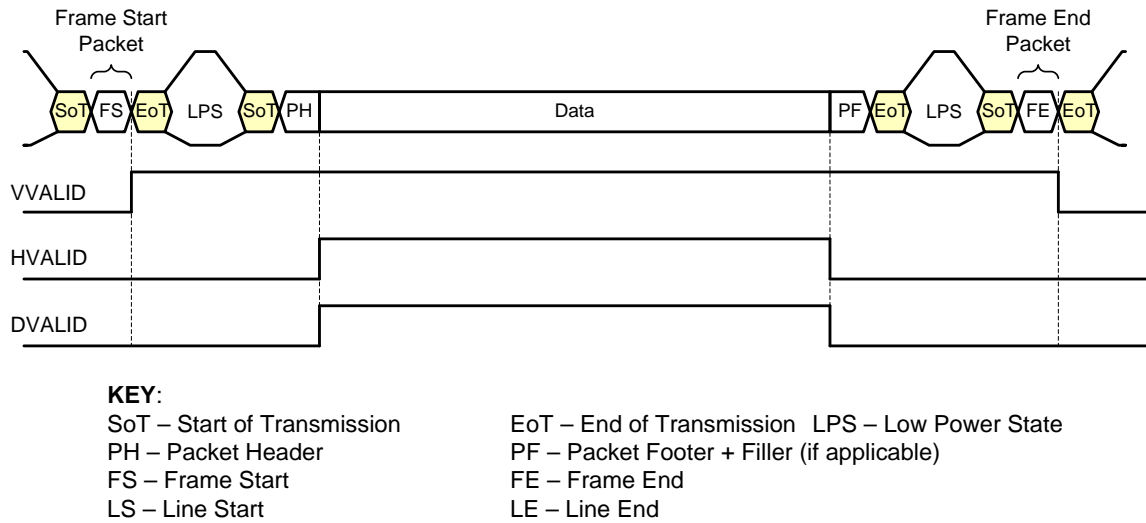
Figure 73 Single Packet Example

KEY:
SoT – Start of Transmission   EoT – End of Transmission   LPS – Low Power State
PH – Packet Header   PF – Packet Footer + Filler (if applicable)
FS – Frame Start   FE – Frame End
LS – Line Start   LE – Line End
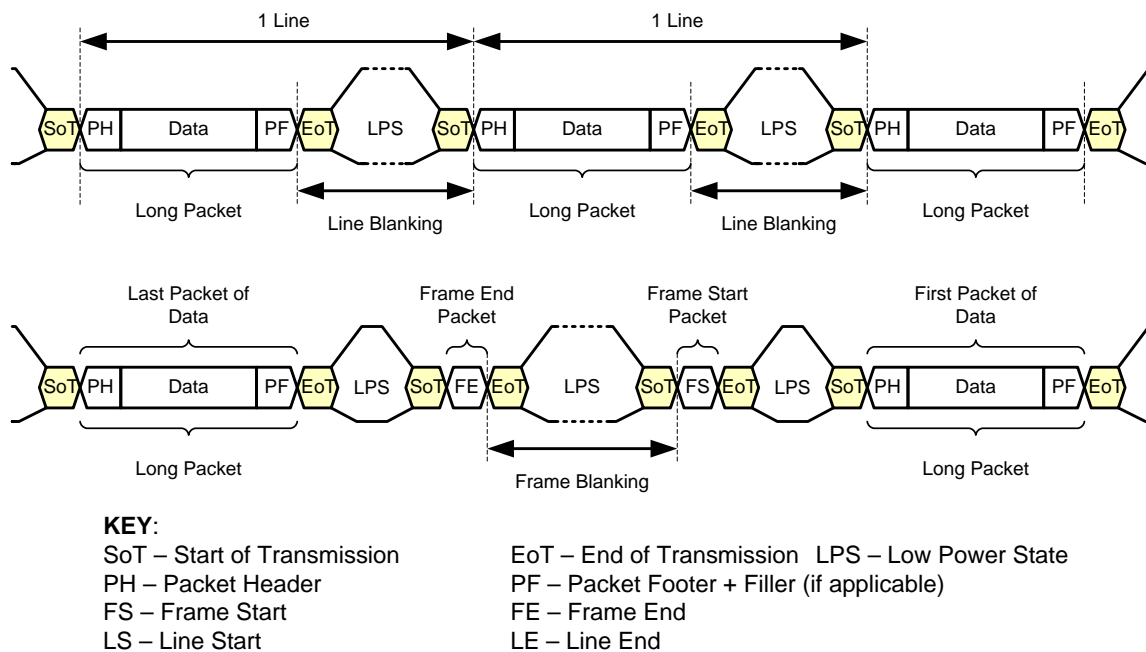
Figure 74 Line and Frame Blanking Definitions

The period between the end of the Packet Footer (or the Packet Filler, if present) of one long packet and the Packet Header of the next long packet is called the Line Blanking Period.

The period between the Frame End packet in frame N and the Frame Start packet in frame N+1 is called the Frame Blanking Period (*Figure 74*).

The Line Blanking Period is not fixed and may vary in length. The receiver should be able to cope with a near zero Line Blanking Period as defined by the minimum inter-packet spacing defined in *[MIPI01]* or *[MIPI02]*, as appropriate. The transmitter defines the minimum time for the Frame Blanking Period. The Frame Blanking Period duration should be programmable in the transmitter.

Frame Start and Frame End packets shall be used.

1223 Recommendations (informative) for frame start and end packet spacing:

- The Frame Start packet to first data packet spacing should be as close as possible to the minimum
packet spacing
- The last data packet to Frame End packet spacing should be as close as possible to the minimum
packet spacing

1228 The intention is to ensure that the Frame Start and Frame End packets accurately denote the start and end of
1229 a frame of image data. A valid exception is when the positions of the Frame Start and Frame End packets
1230 are being used to convey pixel level accurate vertical synchronization timing information.

1231 The positions of the Frame Start and Frame End packets can be varied within the Frame Blanking Period in
1232 order to provide pixel level accurate vertical synchronization timing information. See *Figure 75*.

1233 If pixel level accurate horizontal synchronization timing information is required, Line Start and Line End
1234 packets should be used to achieve it.

1235 The positions of the Line Start and Line End packets, if present, can be varied within the Line Blanking
1236 Period in order to provide pixel accurate horizontal synchronization timing information. See *Figure 76*.
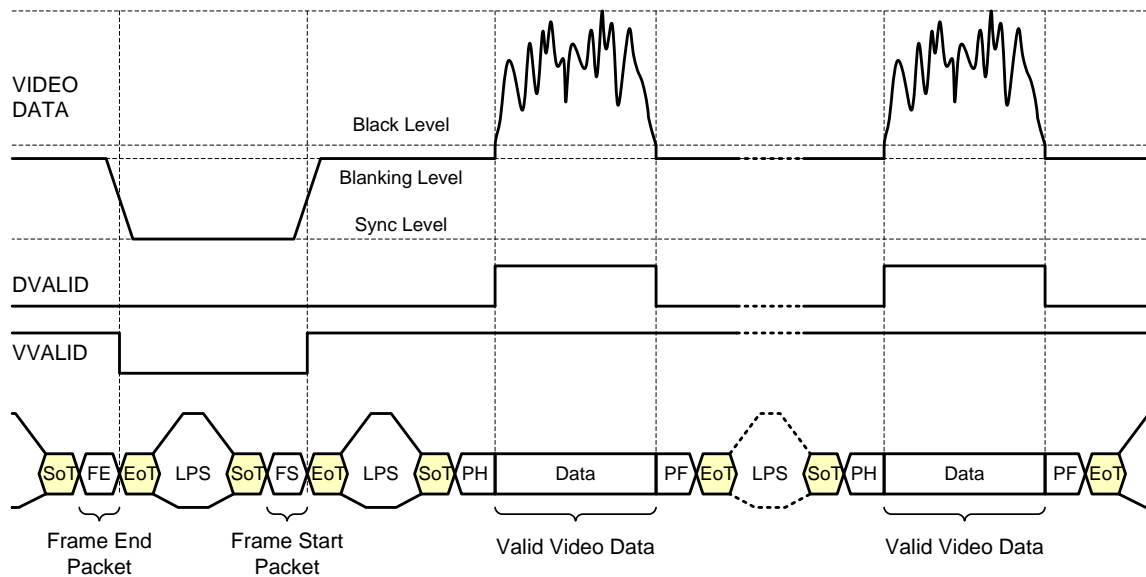
Figure 75 Vertical Sync Example
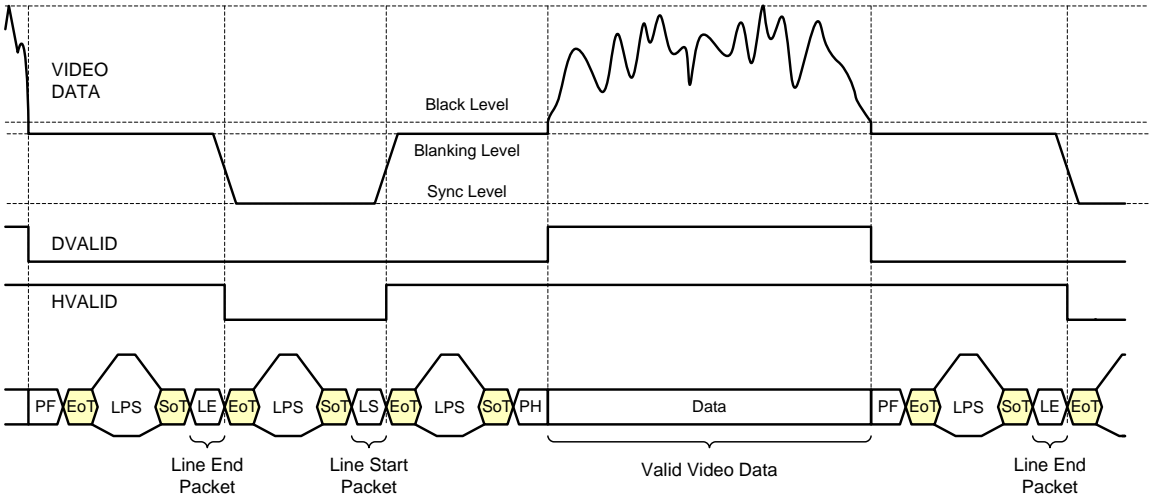
Figure 76 Horizontal Sync Example

9.11 Latency Reduction and Transport Efficiency (LRTE)

1239 Latency Reduction and Transport Efficiency (LRTE) is an optional CSI-2 feature that facilitates optimal
1240 transport, in order to support a number of emerging imaging applications.

1241 LRTE has two parts, further detailed in this Section:

1242 • Interpacket Latency Reduction (ILR)

1243 • Enhanced Transport Efficiency

9.11.1 Interpacket Latency Reduction (ILR)

1244 As per *[MIPI01]* for the D-PHY physical layer option, and *[MIPI02]* for the C-PHY physical layer option,
1245 CSI-2 Short Packets and Long Packets are separated by EoT, LPS, and SoT packet delimiters. Advanced
1246 imaging applications, PDAF (Phase Detection Auto Focus), Sensor Aggregation, and Machine Vision can
1247 substantially benefit from the effective speed increases produced by reducing the overhead of these
1248 delimiters.

1249 Interpacket latency reduction replaces legacy EoT, LPS, and SoT packet delimiters with a more Efficient
1250 Packet Delimiter (EPD) signaling mechanism that avoids the need for HS-LPS-HS transitions.

1251
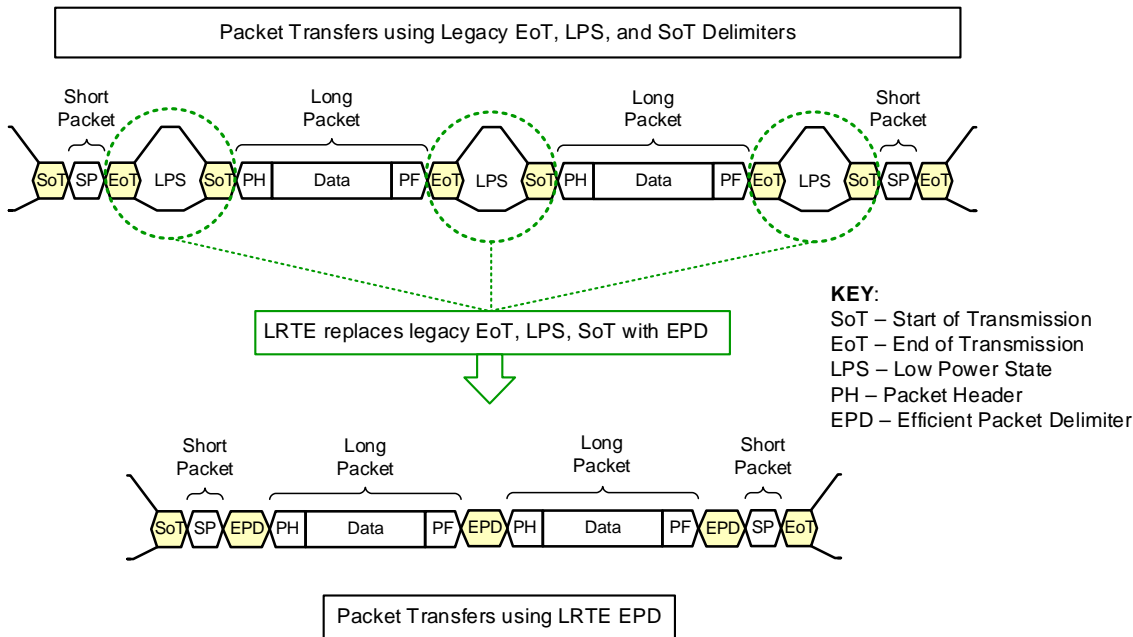
Figure 77 Interpacket Latency Reduction Using LRTE EPD

9.11.1.1   EPD for C-PHY Physical Layer Option

1252 The EPD for the C-PHY physical layer option uses one or more instances of the PHY-generated and PHY-
1253 consumed 7-UI Sync Word for the Packet Delimiter Quick (PDQ) signaling. The PDQ is generated and
1254 consumed by the transmitter and receiver physical layers, respectively, and as a result serves as a robust
1255 CSI-2 packet delimiter. An image sensor should reuse "TxSendSyncHS" at the PPI in order to generate the
1256 PDQ control code by the C-PHY transmitter. Upon reception of the PDQ control code by the C-PHY
1257 receiver, an application processor should reuse "RxSyncHS" at the PPI in order to notify the CSI-2 protocol
1258 layer. The duration of the 7-UI PDQ control code is directly proportional to the C-PHY Symbol rate.

1259 The EPD for C-PHY receivers can also benefit from optional CSI-2 protocol-generated and CSI-2 protocol-
1260 consumed Spacer insertion(s) prior to PDQ, because it facilitates optimal interpacket latency for imaging
1261 applications. The value of the Spacer Word for CSI-2 over C-PHY shall be 0xFFFF, and Spacer Words shall
1262 be generated across all Lanes within a Link.

1263 The image sensor (transmitter) shall include the following two 16-bit registers, in order to facilitate the
1264 optimal interpacket latency for imaging applications:

1265 **1.   TX_REG_CSI_EPD_EN_SSP** (EPD Enable and Short Packet Spacer) **Register**

1266 - The MS bit of this register shall be used to enable EPD with 7-UI PDQ (Sync Word) insertion
1267   between two CSI-2 packets and optional Spacer insertions for Short Packets and Long Packets.

1268   - 1'b0: C-PHY legacy EoT, LPS, SoT Packet Delimiter
1269   - 1'b1: C-PHY EPD (Efficient Packet Delimiter)

1270 - The remaining 15 bits of this register (bits [14:0]) shall be used to generate up to 32,767 Spacer
1271   insertions per Lane following CSI-2 Short Packets.

1272 **2.   TX_REG_CSI_EPD_OP_SLP** (Long Packet Spacer) **Register**

1273 - The MS bit of this register is reserved for future use.
1274 - The remaining 15 bits of this register (bits [14:0]) shall be used to generate up to 32,767 Spacer
1275   insertions per Lane following CSI-2 Long Packets.

1276 If the C-PHY EPD is enabled, then the following applies to the fifteen least significant bits of both EPD
1277 registers:

1278 - A register value of 15'd0 produces no Spacer generation (zero Spacers inserted).
1279 - A register value of 15'd5 generates five Spacers, resulting in a duration of 5 x 7 UI.
1280 - The maximum register value of 15'd32,767 generates 32,767 Spacers, resulting in a duration of
1281   32,767 x 7 UI.

1282 The transmitter shall support at least one non-zero value of the Spacer insertion count field in each of the
1283 **TX_REG_CSI_EPD_EN_SSP** and **TX_REG_CSI_EPD_OP_SLP** registers.
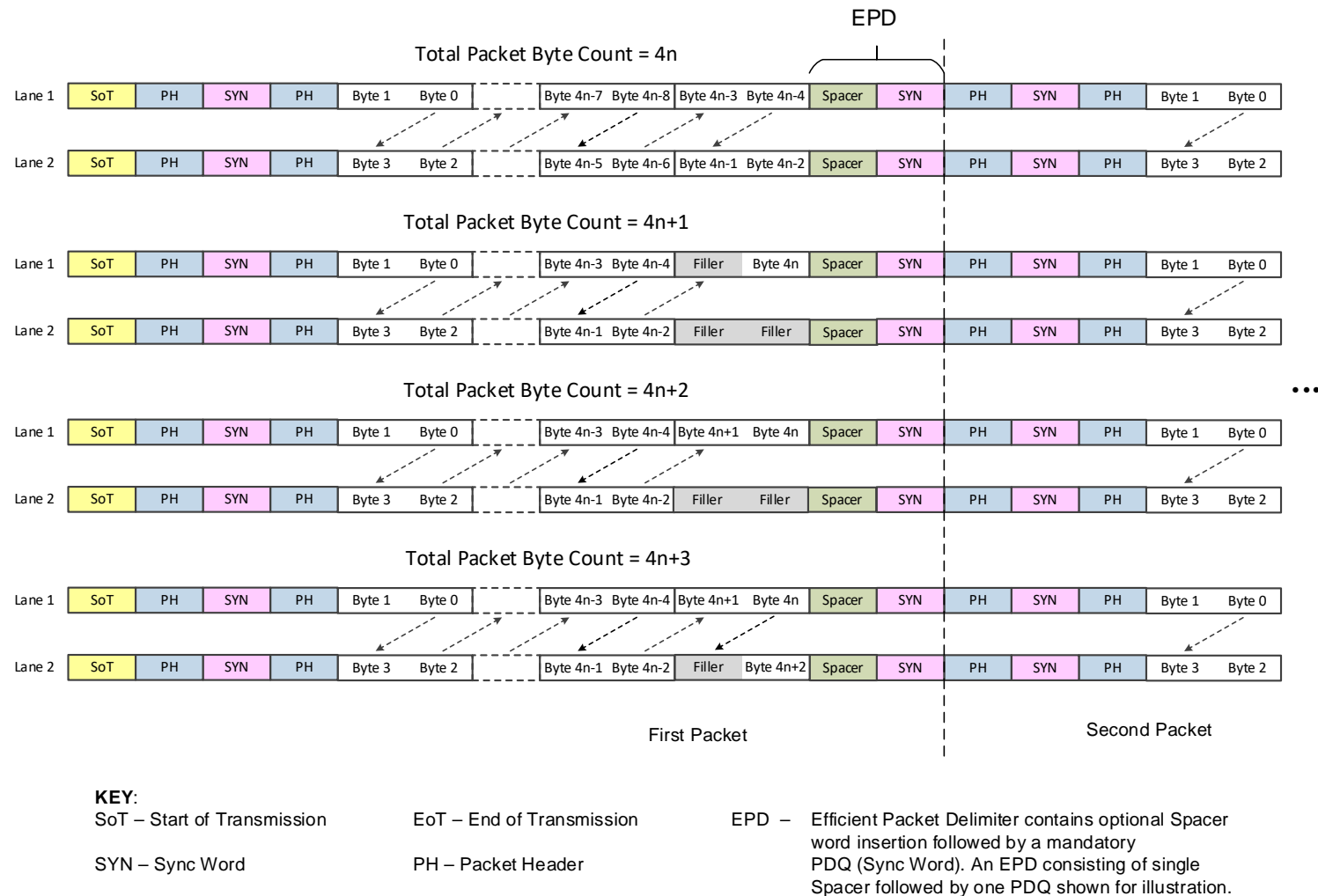
Figure 78 LRTE Efficient Packet Delimiter Example for CSI-2 Over C-PHY (2 Lanes)

9.11.1.2　EPD for D-PHY Physical Layer Option

There are two EPD options for CSI-2 over the D-PHY physical layer option, as detailed in the following sub-sections.

When EPD is enabled, CSI-2 over the D-PHY physical layer option shall align all Lanes corresponding to a Link using the minimum number of filler byte(s) for both options. The value of the filler byte shall be 0x00. The process of aligning Lanes within a Link through the use of filler bytes is similar to native EOT alignment of CSI-2 over C-PHY.

9.11.1.2.1　D-PHY EPD Option 1

The EPD for the D-PHY v2.1 physical layer option uses PHY-generated and PHY-consumed HS-Idle for the Packet Delimiter Quick (PDQ) signaling, with optional Spacer Byte insertions prior to PDQ. The value of the Spacer Byte for CSI-2 over D-PHY shall be 0xFF, and Spacer Bytes shall be generated across all Lanes within a Link. The PDQ is generated and consumed by the transmitter and receiver physical layers, respectively, and as a result serves as a robust CSI-2 packet delimiter. D-PHY receivers can benefit from protocol-generated and protocol-consumed Spacer(s), because additional clock cycles might be needed to flush the payload content through the pipelines before the forwarded clock is disabled for PDQ signaling.

The image sensor should use "TxHSIdleClkHS" at the PPI in order to generate the PDQ sequence by the D-PHY transmitter. Upon reception of the PDQ sequence by the D-PHY receiver, an application processor should use "RXSyncHS" at the PPI to notify the CSI-2 protocol layer. Additionally, "RxClkActiveHS" may also be used to provide an advance indication of the EPD.

Number of Bytes, B, transmitted is NOT an integer multiple of the number of lanes, N with alignment using Filler bytes for packet transfers using PHY generated and consumed PDQ. One optional Spacer byte insertion included for illustration.
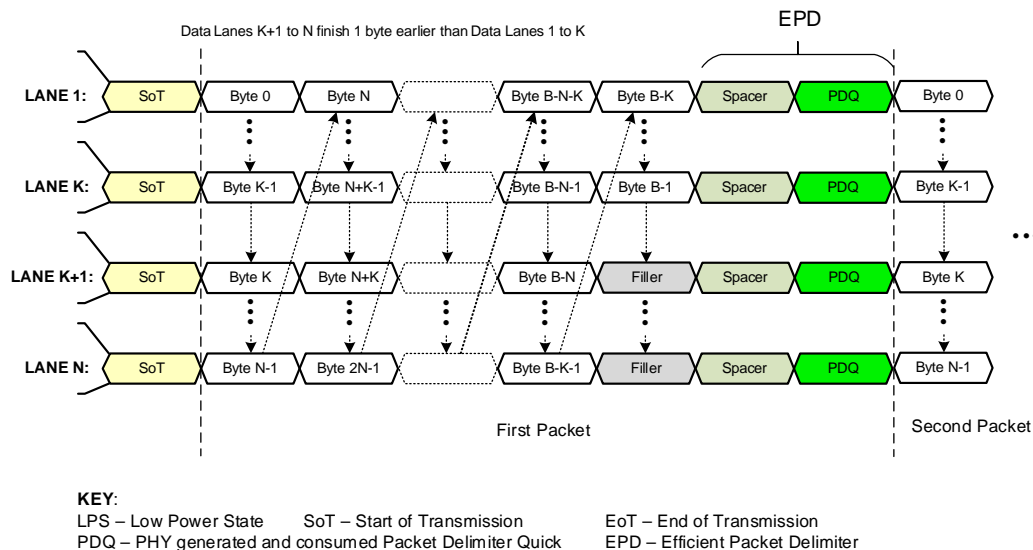
Figure 79 Example of LRTE EPD for CSI-2 Over D-PHY – Option 1

9.11.1.2.2 D-PHY EPD Option 2

1303 D-PHY EPD Option 2 is limited to optional CSI-2 protocol-generated and CSI-2 protocol-consumed
1304 Spacers for back-to-back transfers (i.e., there is no use of PHY-generated and PHY-consumed PDQ).
1305 Option 2 is primarily intended for use with legacy D-PHYs not supporting Option 1. Depending on the use
1306 case (i.e., the sizes and number of CSI-2 packets being concatenated), the lack of D-PHY-generated and
1307 D-PHY-consumed PDQ could compromise CSI-2 link integrity. Option 2 is not intended to completely
1308 replace the standard D-PHY-based LPS packet delimiters provided by legacy D-PHYs. It is recommended
1309 that one or more Spacers be included following a Short Packet or a Long Packet when using D-PHY EPD
1310 Option 2.

1311
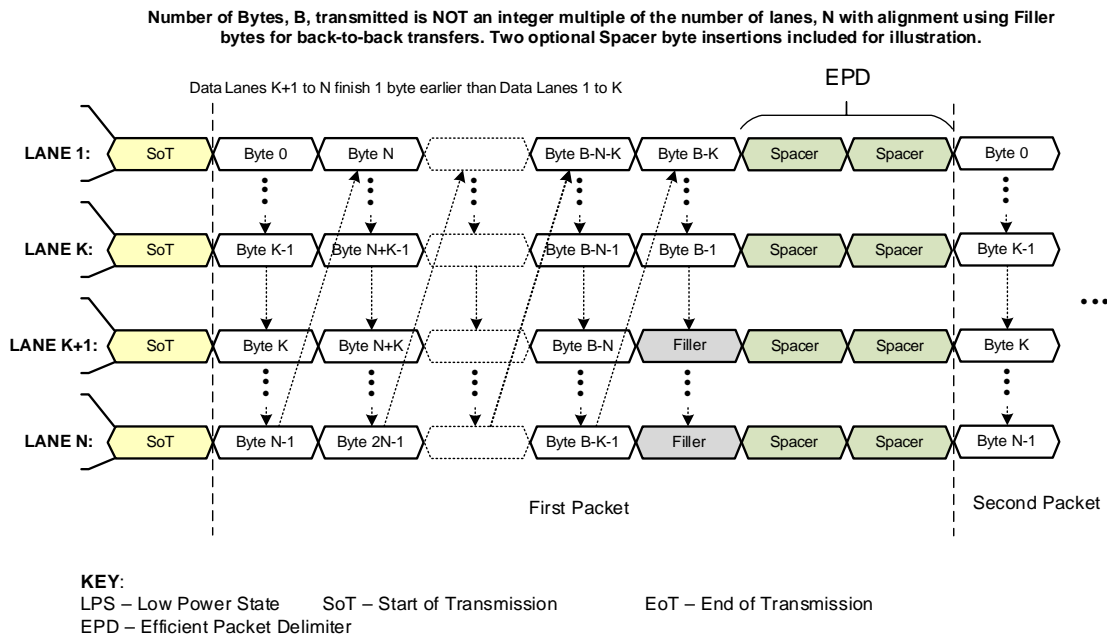
Figure 80 Example of LRTE EPD for CSI-2 Over D-PHY – Option 2

9.11.1.2.3 D-PHY EPD Specifications (for EPD Options 1 and 2)

1312 The image sensor (transmitter) shall include the following two 16-bit registers, in order to facilitate the
1313 optimal interpacket latency for imaging applications:

1314 **1. TX_REG_CSI_EPD_EN_SSP** (EPD Enable and Short Packet Spacer) **Register**

1315 • The MS bit of this register shall be used to enable EPD insertion between two CSI-2 packets.

1316   • 1'b1: Enable D-PHY EPD (Efficient Packet Delimiter)

1317 • If D-PHY EPD is enabled, then the remaining fifteen bits of this register (bits [14:0]) shall be used
1318   to generate up to 32,767 Spacer insertions per Lane following CSI-2 Short Packets. These Spacer
1319   insertions for CSI-2 Short Packets apply to both D-PHY EPD options.

1320 **2. TX_REG_CSI_EPD_OP_SLP** (EPD Option and Long Packet Spacer) **Register**

1321 • The MS bit of this register shall be used to select the D-PHY EPD option.

1322   • 1'b0: D-PHY EPD Option 1

1323   • 1'b1: D-PHY EPD Option 2

1324 • If D-PHY EPD is enabled, then the remaining fifteen bits of this register (bits [14:0]) shall be used
1325   to generate up to 32,767 optional Spacer insertions per Lane following CSI-2 Long Packets. These
1326   Spacer insertions for CSI-2 Long Packets apply to both D-PHY EPD options.

₁₃₂₇ The following applies to the least significant fifteen bits of the two EPD registers:

₁₃₂₈ • A register value of 15'd0 produces no Spacer generation (zero Spacers inserted).

₁₃₂₉ • A register value of 15'd5 generates 5 Spacers.

₁₃₃₀ • The maximum register value of 15'd32,767 generates 32,767 Spacers.

₁₃₃₁ The transmitter shall support at least one non-zero value of the Spacer insertion count field in each of the
₁₃₃₂ **TX_REG_CSI_EPD_EN_SSP** and **TX_REG_CSI_EPD_OP_SLP** registers. The duration of the PDQ
₁₃₃₃ sequence is directly proportional to the D-PHY Link rate, and is configured using register defined in
₁₃₃₄ *[MIPI01]* for the D-PHY physical layer option.

9.11.2   Using ILR and Enhanced Transport Efficiency Together

₁₃₃₅ EPD and ALPS, the two LRTE provisions referred to in *Section 7*, may be used together in many imaging
₁₃₃₆ applications in order to benefit from CSI-2 ILR and enhanced channel transport.

₁₃₃₇
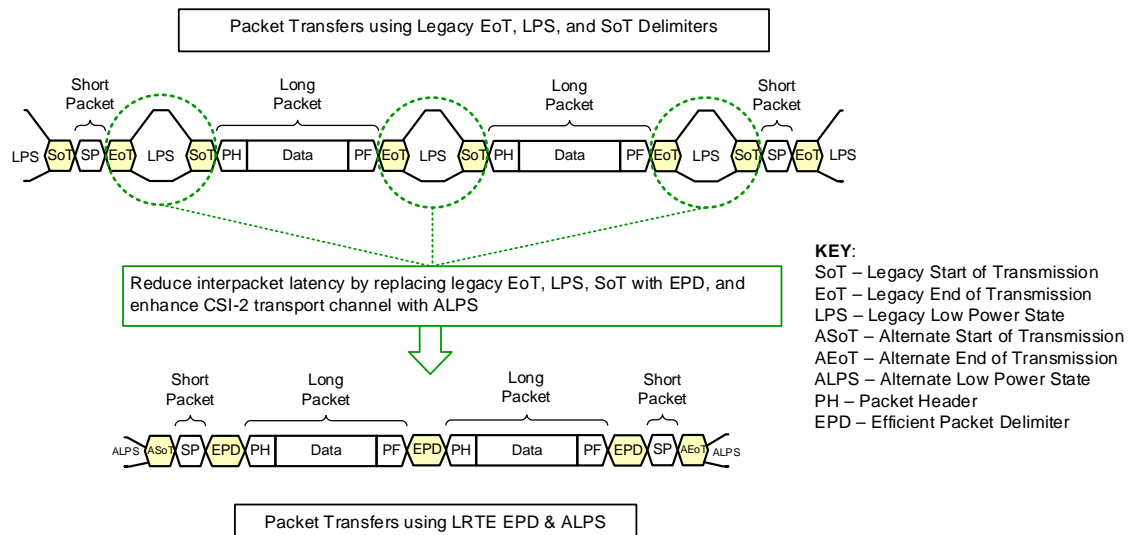
Figure 81 Using EPD and ALPS Together

9.11.3   LRTE Register Tables

1338   The CSI-2 over C-PHY Spacer Words and the CSI-2 over D-PHY Spacer Bytes shall be generated across
1339   all Lanes within a Link as specified in *Table 15* and *Table 16*.

1340   **Table 15 LRTE Transmitter Registers for CSI-2 Over C-PHY**

Transmitter Register		Description
TX_REG_CSI_EPD_EN_SSP [15:0]		Write-only. Required.
Bit [15]: Enable or disable Efficient Packet Delimiter using PHY-generated and PHY-consumed PDQ with optional minimum Spacer Insertion(s)	**Value 1'b0:** Disable Efficient Packet Delimiter **Value 1'b1:** Enable Efficient Packet Delimiter	CSI-2 over C-PHY EPD operation uses PHY-generated and PHY-consumed PDQ (7-UI Sync Word). Optional minimum Spacers may be Inserted for Short Packets and Long Packets. See *Figure 78*.
Bits [14:0]: EPD Short Packet Spacers	The minimum number of Spacer Words per Lane following a Short packet. **Examples:** **Value 15'd0:** No Spacer Words … **Value 15'd7:** Seven Spacer Words … **Value 15'd32767:** 32,767 Spacer Words	The Short Packet Spacers insertions are enabled by the C-PHY EPD (TX_REG_CSI_EPD_EN_SSP[15]). The Short Packet Spacers may range from 0 to 32,767 Words.
TX_REG_CSI_EPD_OP_SLP [15:0]		Write-only. Required
Bit [15]: Reserved	Reserved	Reserved for future use
Bits [14:0]: EPD Long Packet Spacers	The minimum number of Spacer Words per Lane following a Long packet. **Examples:** **Value 15'd0:** No Spacer Words … **Value 15'd7:** Seven Spacer Words … **Value 15'd32767:** 32,767 Spacer Words	The Long Packet Spacers insertions are enabled by the C-PHY EPD (TX_REG_CSI_EPD_EN_SSP[15]) The Long Packet Spacers may range from 0 to 32,767 Words.

1341

Table 16 LRTE Transmitter Registers for CSI-2 Over D-PHY

Transmitter Register		Description
TX_REG_CSI_EDP_EN_SSP [15:0]		Write-only. Required
Bit [15]: Enable or disable EPD (Efficient Packet Delimiter) operation	**Value 1'b0:** Disable EPD **Value 1'b1:** Enable EPD	See *Figure 79*. If EPD is enabled, the D-PHY EPD Options are determined by **TX_REG_CSI_EPD_OP_SLP[15]**.
Bits [14:0]: EPD Short Packet Spacers	**For D-PHY EPD Option 1:** Minimum number of Spacer Bytes per Lane following a Short packet. **For D-PHY EPD Option 2:** Fixed number of Spacer Bytes per Lane following a Short packet. **Examples:** **Value 15'd0:** No Spacer Bytes … **Value 15'd7:** Seven Spacer Bytes … **Value 15'd32767:** 32,767 Spacer Bytes	The Short Packet Spacers insertions are enabled by the D-PHY EPD (**TX_REG_CSI_EPD_EN_SSP[15]**). The Short Packet Spacers may range from 0 to 32,767 Bytes. See *Figure 79* and *Figure 80*.
TX_REG_CSI_EPD_OP_SLP [15:0]		Write-only. Required.
Bit [15]: D-PHY EPD Option Select	**Value 1'b0:** D-PHY EPD Option 1 **Value 1'b1:** D-PHY EPD Option 2	**D-PHY EPD Option 1:** CSI-2 over D-PHY EPD operation using PHY-generated and PHY-consumed PDQ (using forwarded clock signaling) and optional Spacer Insertion(s). See *Figure 79*. **D-PHY EPD Option 2:** CSI-2 over D-PHY EPD operation using optional Spacer Insertion(s). See *Figure 80*.
Bits [14:0]: Long Packet Spacers	**For D-PHY EPD Option 1:** Minimum number of Spacer Bytes per Lane following a Long packet. **For D-PHY EPD Option 2:** Fixed number of Spacer Bytes per Lane following a Long packet. **Examples:** **Value 15'd0:** No Spacer Bytes … **Value 15'd7:** Seven Spacer Bytes … **Value 15'd32767:** 32,767 Spacer Bytes	The Long Packet Spacers insertions are enabled by the D-PHY EPD (**TX_REG_CSI_EPD_EN_SSP[15]**). The Long Packet Spacers may range from 0 to 32,767 Bytes. See *Figure 79* and *Figure 80*.

9.12   Data Scrambling

1342 The purpose of Data Scrambling is to mitigate the effects of EMI and RF self-interference by spreading the
1343 information transmission energy of the Link over a possibly large frequency band, using a data
1344 randomization technique. The scrambling feature described in this Section is optional and normative: If a
1345 CSI-2 implementation includes support for scrambling, then the scrambling feature shall be implemented as
1346 described in this Section. The benefits of data scrambling are well-known, and it is strongly recommended
1347 to implement this data scrambling capability in order to minimize radiated emissions in the system.

1348 Data Scrambling shall be applied on a per-Lane basis, as illustrated in *Figure 82*. Each output of the Lane
1349 Distribution Function shall be individually scrambled by a separate scrambling function dedicated to that
1350 Lane, before the Lane data is sent to the PHY function over the Tx PPI.

1351
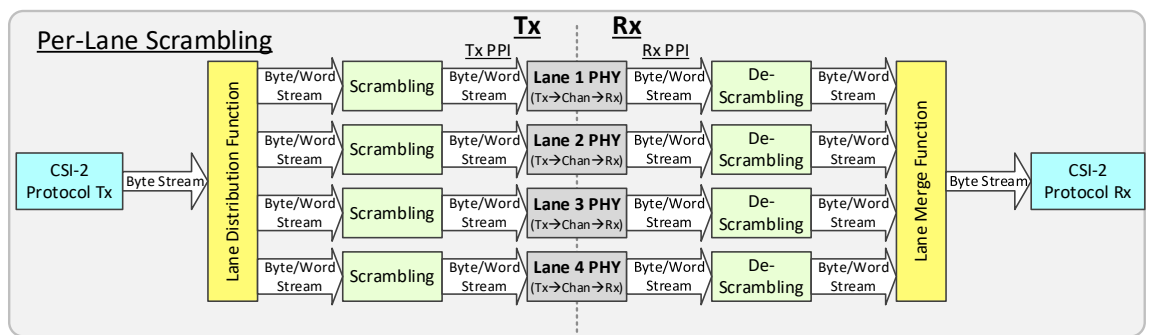
Figure 82 System Diagram Showing Per-Lane Scrambling

9.12.1   CSI-2 Scrambling for D-PHY

1352   *Figure 83* shows the format of a burst transmission of two packets over two Lanes when the D-PHY
1353   physical layer is used. After the Start of Transmission, HS-ZERO and HS-SYNC are transmitted, the
1354   Packet Header and data payload are distributed across the two Lanes.

1355   If the D-PHY physical layer is used, then the scrambler Linear Feedback Shift Register (LFSR) in each
1356   Lane shall be initialized with the Lane seed value under any of the following conditions:

1357   1.   At the beginning of the burst, which occurs immediately prior to the first byte transmitted
1358   following the HS-Sync that is generated by the D-PHY (applicable to both D-PHY EPD Option1
1359   and Option 2).

1360   2.   Prior to the first byte transmitted following the HS-Sync that is generated whenever the optional
1361   D-PHY EPD Option 1 HS-Idle is transmitted.

1362   The scrambler is not reinitialized between CSI-2 packets when using the optional D-PHY EPD Option 2.

1363   When the scrambler is initialized, the LFSR shall be initialized using the sixteen-bit seed value assigned to
1364   each Lane.

1365   Note: The Packet Footer at the end of every packet is scrambled.
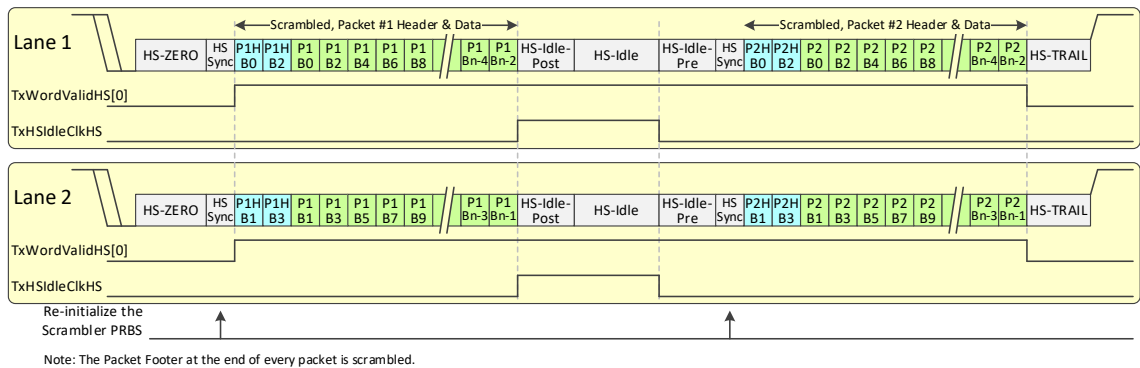
Figure 83 Example of Data Bursts in Two Lanes Using the D-PHY Physical Layer

Copyright © 2005-2018 MIPI Alliance, Inc.
                                 All rights reserved.
                                   Confidential

9.12.2   CSI-2 Scrambling for C-PHY

1366 *Figure 84* shows the format of a burst transmission of two packets over two Lanes when the C-PHY
1367 physical layer is used. After the Start of Transmission, Preamble, and Sync are transmitted, the Packet
1368 Header is replicated twice on each Lane, and data payloads of each packet are distributed across the two
1369 Lanes. If the C-PHY physical layer is used, then the scrambler LFSR in each Lane shall be initialized at the
1370 beginning of every Long Packet Header or Short Packet, using one of the sixteen-bit seed values assigned
1371 to each Lane. This initialization takes place each time the Sync Word is transmitted.

1372
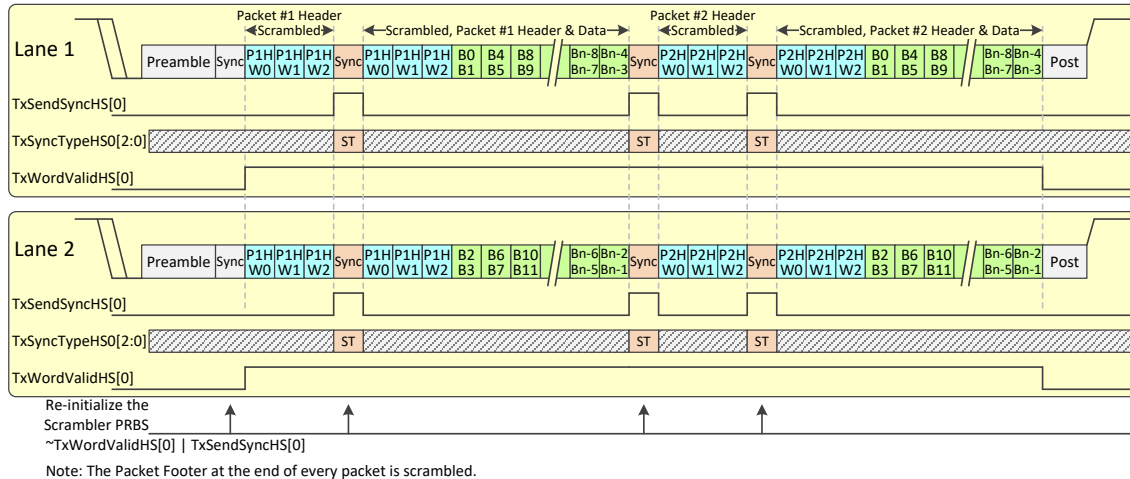
Figure 84 Example of Data Bursts in Two Lanes Using the C-PHY Physical Layer

1373 In some cases, images may cause repetitive transmission of Long Packets having the same or similar Long
1374 Packet Header and the same pixel data (for example: all dark pixels, or all white pixels). If the scrambler is
1375 initialized with the same seed value at the beginning of every packet, coinciding with the beginning of
1376 every pixel row, then the scrambled pseudo-random sequence will repeat at the rate that rows of identical
1377 image data are transmitted. This can cause the emissions to be less random, and instead have peaks at
1378 frequencies equivalent to the rate at which the image data rows are transmitted.

1379 To mitigate this issue, a different seed value is selected by the transmitter every time a Packet Header is
1380 transmitted. The Sync Word in the Packet Header encodes a small amount of data, so that the transmitter
1381 can inform the receiver which starting seed to use to descramble the packet. This small amount of data in
1382 the Sync Word is sent by transmitting a Sync Type that the CSI-2 protocol transmitter chooses. This Sync
1383 Type value is also used to select the starting seed in the scrambler and descrambler.

1384 **Table 17** shows the five possible Sync Types that the C-PHY supports. The Sync Word values are
1385 normatively specified in the C-PHY Specification and duplicated in **Table 17** for convenience. The CSI-2
1386 protocol uses only the first four out of the five possible Sync Types, which simplifies the implementation.

1387 **Table 17 Symbol Sequence Values Per Sync Type**

Sync Type	Sync Value	TxSyncTypeHS0[2:0], TxSyncTypeHS1[2:0]	Scrambler and Descrambler Seed Index
Type 0	3444440	0	0
Type 1	3444441	1	1
Type 2	3444442	2	2
Type 3	3444443	3	3
Type 4	3444444	4	N/A

1388 **Note:**

1389 *When a single seed value is used, Sync Type 3 is the default Sync Word value.*

1390 **Figure 85** shows the architecture of the scrambling in a single Lane. The pseudo-random number generated
1391 by the PRBS shall be used as the seed index to select the initial seed value from the seed list prior to
1392 sending the packet. This seed index shall also be sent to the C-PHY using the PPI signals
1393 TxSyncTypeHS0[1:0]. TxSyncTypeHS0[2] is always zero. TxSyncTypeHS1 [2:0] is used similarly for a
1394 32-bit data path. The C-PHY ensures that the very first packet in a burst begins with a Sync Word using
1395 Sync Type 3.
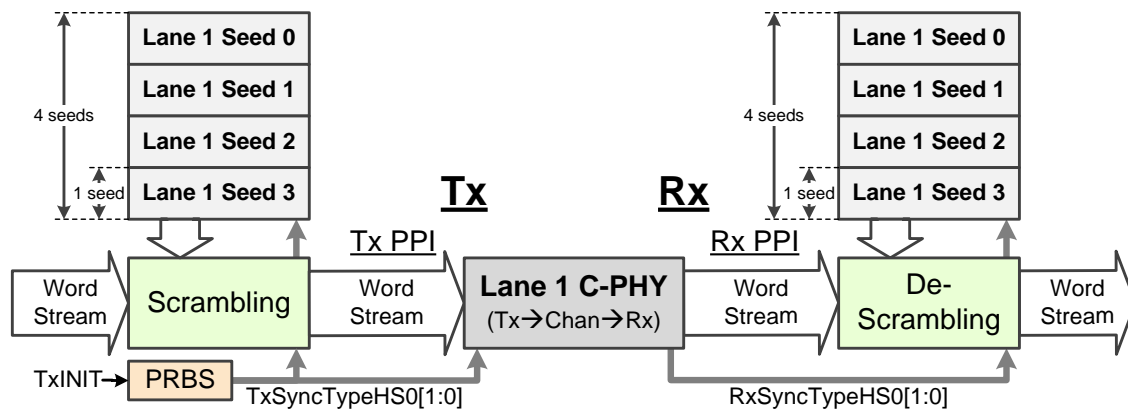
Figure 85 Generating Tx Sync Type as Seed Index (Single Lane View)

1397 The seed list may contain either one or four initial seed values. Transmitters and receivers shall have the
1398 capability to select exactly one seed value from a list of seeds. When a single seed value is used, that seed
1399 shall be identified as Seed 3 and the transmitter shall always transmit Sync Type 3. Transmitters and
1400 receivers should also have the capability to select a seed value from a list of four seed values, as shown in
1401 **Figure 85**. When a list of four seed values is used then Sync Type 0 through Sync Type 3 shall be used to
1402 convey the seed index value from the transmitter to the receiver.

1403 When the list of four seeds is used, the two-bit seed index shall be generated in the transmitter using a
1404 pseudo random generator (e.g., PRBS).

1405 Slight differences in the implementation of the PRBS generator will not affect the interoperability of the
1406 transmitter and receiver, because the receiver responds to the seed index chosen in the transmitter and
1407 conveyed to the receiver using the Sync Type.

1408 At the receiver, the C-PHY decodes the Sync Word and passes the 2-bit Sync Type value to the CSI-2
1409 protocol logic. The CSI-2 protocol logic uses the two-bit value as a seed index to select one of four seed
1410 values to initialize the descrambler. This concept is shown in the single Lane diagram in *Figure 85*. *Figure*
1411 *86* shows the use of the PPI signals to select which seed value was used to initialize the scrambler and
1412 descrambler. Since the seed selection field is transmitted via the Sync Word, no other mechanism is needed
1413 to coordinate the choice of specific descrambler initial seed values at the receiver.

1414
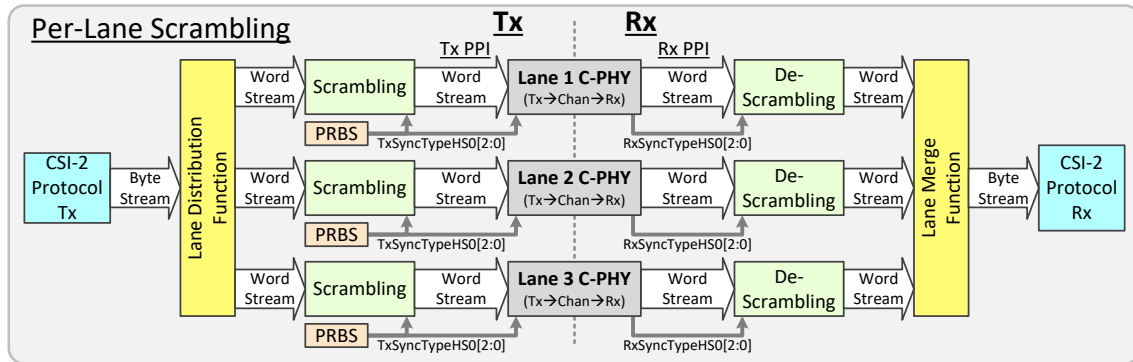
Figure 86 Generating Tx Sync Type Using the C-PHY Physical Layer

9.12.3   Scrambling Details

1415 The Long Packet Header, Data Payload, Long Packet Footer (which may include a Filler Byte), and Short
1416 Packets shall be scrambled. Special data fields generated by the PHY that are beyond the control of the
1417 CSI-2 protocol shall not be scrambled. For clarity, *Table 18* lists all of the fields that are not scrambled.

1418
Table 18 Fields That Are Not Scrambled

PHY	PHY-Generated	CSI-2-Protocol-Generated

D-PHY	• HS-Zero
	• Sync Word (aka Leader Sequence)
	• HS Trail
	• SoT
	• EoT
	• HS-Idle
	• All fields of the deskew sequence (aka deskew burst) including:
	  • HS-Zero
	  • Deskew sync pattern
	  • '01010101' data
	  • HS-Trail
		• LP Mode transactions for SoT, EoT and ULPS

C-PHY	• Preamble (including $t_{3\text{-PREBEGIN}}$ $t_{3\text{-PROGSEQ}}$ and $t_{3\text{-PREEND}}$) • Sync Word • Post • SoT • EoT	• Sync Word inserted via PPI command • LP Mode transactions for SoT, EoT and ULPS

1419 The data scrambler and descrambler pseudo-random binary sequence (PRBS) shall be generated using the
1420 Galois form of an LFSR implementing the generator polynomial:

1421 $$G(x) = x^{16} + x^5 + x^4 + x^3 + 1$$

1422 The initial D-PHY seed values in *Table 19* should be used to initialize the D-PHY scrambler LFSR in
1423 Lanes 1 through 8.

1424
Table 19 D-PHY Scrambler PRBS Initial Seed Values for Lanes 1 Through 8

Lane	Initial Seed Value
1	0x0810
2	0x0990
3	0x0a51
4	0x0bd0
5	0x0c30
6	0x0db0
7	0x0e70
8	0x0ff0

1425

1426 The initial C-PHY seed values in *Table 20* should be used to initialize the C-PHY scrambler LFSR in Lanes
1427 1 through 8. The table provides initial seed values for each of the four possible Sync Type values per Lane
1428 number. If only a single Sync Type is used, then it shall default to Sync Type 3.

1429 **Table 20 C-PHY Scrambler PRBS Initial Seed Values for Lanes 1 Through 8**

Lane	Initial Seed Value			
	Sync Type 0	Sync Type 1	Sync Type 2	Sync Type 3
1	0x0810	0x0001	0x1818	0x1008
2	0x0990	0x0180	0x1998	0x1188
3	0x0a51	0x0240	0x1a59	0x1248
4	0x0bd0	0x03c0	0x1bd8	0x13c8
5	0x0c30	0x0420	0x1c38	0x1428
6	0x0db0	0x05a0	0x1db8	0x15a8
7	0x0e70	0x0660	0x1e79	0x1668
8	0x0ff0	0x07e0	0x1ff8	0x17e8

1430 For D-PHY and C-PHY systems requiring more than eight Lanes, *Annex G* provides 24 additional seed
1431 values for Lanes 9 through 32, as well as a mechanism for finding seed values for Lanes 33 and higher. For
1432 each seed value, the LSB corresponds to scrambler PRBS register bit Q0 and the MSB corresponds to bit
1433 Q15.

1434 The LFSR shall generate an eight-bit sequence at G(x) for every byte of Payload data to be scrambled,
1435 starting from its initial seed value. The LFSR shall generate new bit sequences of G(x) by advancing eight
1436 bit cycles for each subsequent Payload data byte.

1437 Scrambling shall be achieved by modulo-2 bit-wise addition (X-OR) of a sequence of eight bits G(x) with
1438 the CSI-2 Payload data to be scrambled.

Implementation Tip: the 8-bit value from the PRBS is the flip of bits Q15:Q8 of the PRBS LFSR register on every 8th bit clock. The designer might choose to implement the PRBS LFSR in parallel form to shift the equivalent of 8 places in a single byte clock, or the PRBS LFSR might even be configured to shift a multiple of 8 places in a single word clock.

For the example shown in *Figure 87*, Q[15:8] are captured in a temporary register, then the PRBS LFSR is shifted eight times before Q[15:8] are captured again. The scrambling is performed as follows:

- TxD[7] = PktD[7] $\oplus$ Q'[8];
- TxD[6] = PktD[6] $\oplus$ Q'[9];
- TxD[5] = PktD[5] $\oplus$ Q'[10];
- TxD[4] = PktD[4] $\oplus$ Q'[11];
- TxD[3] = PktD[3] $\oplus$ Q'[12];
- TxD[2] = PktD[2] $\oplus$ Q'[13];
- TxD[1] = PktD[1] $\oplus$ Q'[14];
- TxD[0] = PktD[0] $\oplus$ Q'[15];
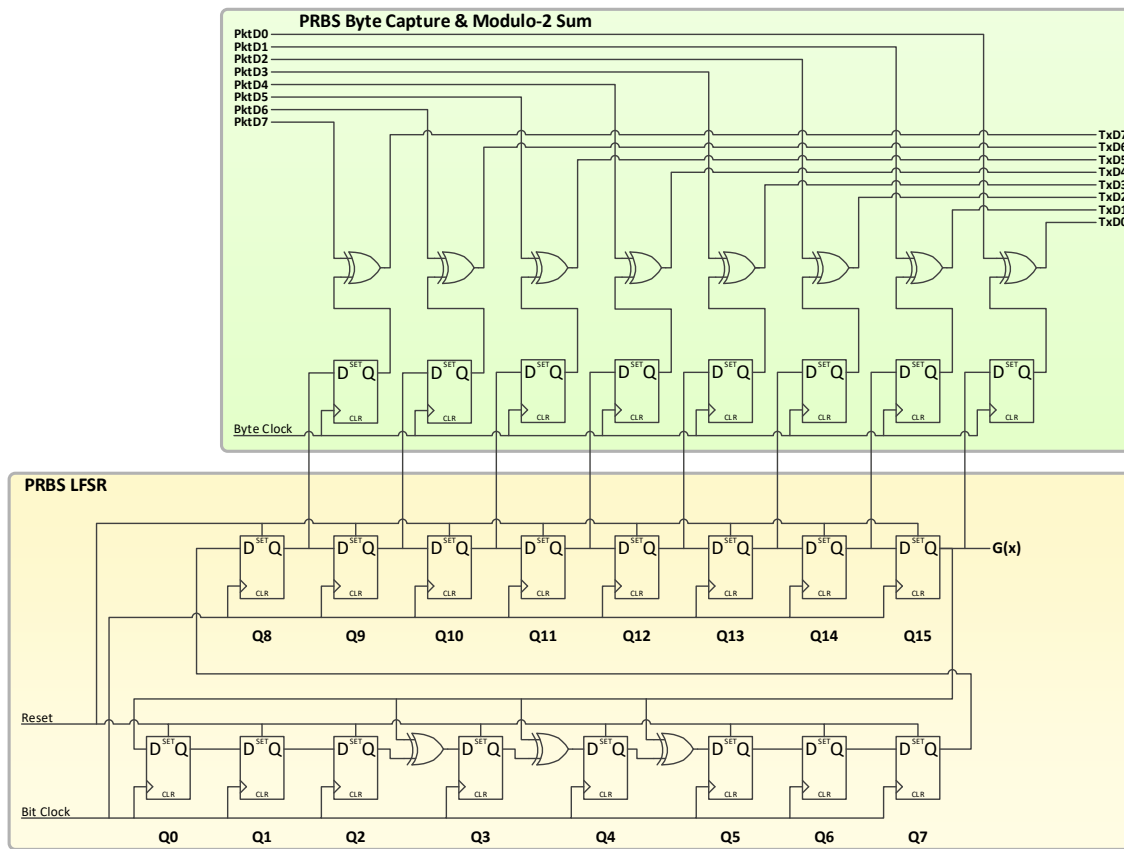
Figure 87 PRBS LFSR Serial Implementation Example

1454 *Table 21* illustrates the sequence of the PRBS register one bit at a time, starting with the initial seed value
1455 for Lane 2. The data scrambling sequence is the output G(x). The first bit output from the scrambler is the
1456 value output from G(x) (also Q15 of the register in *Figure 87*) when the register contains the initial seed
1457 value.

Table 21 Example of the PRBS Bit-at-a-Time Shift Sequence

t	Q15	Q14	Q13	Q12	Q11	Q10	Q9	Q8	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0	LFSR
0	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	0x1188
1	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0x2310
2	0	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0x4620
3	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0x8C40
4	0	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	0x18B9
5	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	0	0x3172
6	0	1	1	0	0	0	1	0	1	1	1	0	0	1	0	0	0x62E4
7	1	1	0	0	0	1	0	1	1	1	0	0	1	0	0	0	0xC5C8
8	1	0	0	0	1	0	1	1	1	0	1	0	1	0	0	1	0x8BA9
9	0	0	0	1	0	1	1	1	0	1	1	0	1	0	1	1	0x176B
10	0	0	1	0	1	1	1	0	1	1	0	1	0	1	1	0	0x2ED6
11	0	1	0	1	1	1	0	1	1	0	1	0	1	1	0	0	0x5DAC
12	1	0	1	1	1	0	1	1	0	1	0	1	1	0	0	0	0xBB58
13	0	1	1	1	0	1	1	0	1	0	0	0	1	0	0	1	0x7689
14	1	1	1	0	1	1	0	1	0	0	0	1	0	0	1	0	0xED12
15	1	1	0	1	1	0	1	0	0	0	0	1	1	1	0	1	0xDA1D
16	1	0	1	1	0	1	0	0	0	0	0	0	0	0	1	1	0xB403

1459 *Table 22* shows the first ten PRBS Byte Outputs produced by the PRBS LFSR in Lane 2 when the D-PHY
1460 physical layer is being used.

Table 22 Example PRBS LFSR Byte Sequence for D-PHY Physical Layer

Scrambling Sequence	PRBS Register	PRBS Byte	Input Byte	Output Byte
Initial Seed, Byte 0	0x0990	0x90	0x2b	0xbb
Byte 1	0x91f1	0x89	0x0d	0x84
Byte 2	0xee29	0x77	0x63	0x14
Byte 3	0x3dbe	0xbc	0x00	0xbc
Byte 4	0xbba5	0xdd	0x00	0xdd
Byte 5	0xbcb3	0x3d	0x00	0x3d
Byte 6	0xaa1c	0x55	0x19	0x4c
Byte 7	0x061a	0x60	0x41	0x21
Byte 8	0x1a96	0x58	0x22	0x7a
Byte 9	0x942a	0x29	0x53	0x7a

Table 23 shows an example of the PRBS Word Outputs at the beginning of a packet, that are produced by the PRBS LFSR in Lane 2 when the C-PHY physical layer is being used.

Table 23 Example PRBS LFSR Byte Sequence for C-PHY Physical Layer

Scrambling Sequence Word #	PRBS Register	PRBS Word	Input Word	Output Word
Initial Seed, Header[47:32]	0x0990	0x8990	0x2b00	0xa290
Header[31:16]	0xee29	0xbc77	0x13b0	0xafc7
Header[15:0]	0xbba5	0x3ddd	0x31c8	0x0c15
Sync Word	0xaa1c	0x6055	0xxxxxx	0xxxxxx
Re-initialized Seed, Header[47:32]	0x1188	0xd188	0x2b00	0xfa88
Header[31:16]	0xb403	0xd82d	0x13b0	0xcb9d
Header[15:0]	0xd613	0x406b	0x31c8	0x71a3
Word 0	0xc672	0x0663	0xd000	0xd663
Word 1	0x5f60	0x36fa	0x1360	0x259a
Word 2	0xbf4c	0xaafd	0x094c	0xa3b1
Word 3	0x5a0d	0x805a	0x100b	0x9051
Word 4	0x6a39	0x8c56	0x5fb8	0xd3ee
Word 5	0xde89	0x997b	0xd030	0x494b
Word 6	0x10e1	0x4708	0x0003	0x470b
Word 7	0x8592	0x71a1	0xd039	0xa198
Word 8	0x40de	0x0b02	0xa35b	0xa859
Word 9	0x5150	0xba8a	0x00ea	0xba60

9.13   Packet Data Payload Size Rules

For YUV, RGB or RAW data types, one long packet shall contain one line of image data. Each long packet of the same Data Type shall have equal length when packets are within the same Virtual Channel and when packets are within the same frame. An exception to this rule is the YUV420 data type which is defined in *Section 11.2.2*.

For User Defined Byte-based Data Types, long packets can have arbitrary length. The spacing between packets can also vary.

The total size of payload data within a long packet for all data types shall be a multiple of eight bits. However, it is also possible that a data type's payload data transmission format, as defined elsewhere in this Specification, imposes additional constraints on payload size. In order to meet these constraints it may sometimes be necessary to add some number of "padding" pixels to the end of a payload e.g., when a packet with the RAW10 data type contains an image line whose length is not a multiple of four pixels as required by the RAW10 transmission format as described in *Section 11.4.4*. The values of such padding pixels are not specified.

9.14   Frame Format Examples

This is an informative section.

This section contains three examples to illustrate how the CSI-2 features can be used.

- General Frame Format Example, *Figure 88*
- Digital Interlaced Video Example, *Figure 89*
- Digital Interlaced Video with accurate synchronization timing information, *Figure 90*

KEY:
PH – Packet Header   PF – Packet Footer + Filler (if applicable)
FS – Frame Start   FE – Frame End
LS – Line Start   LE – Line End

1483

Figure 88 General Frame Format Example

Data per line is a multiple of 16-bits (YUV422)

KEY:
PH – Packet Header   PF – Packet Footer + Filler (if applicable)
FS – Frame Start   FE – Frame End
LS – Line Start   LE – Line End

1484

Figure 89 Digital Interlaced Video Example

KEY:
PH – Packet Header   PF – Packet Footer + Filler (if applicable)
FS – Frame Start   FE – Frame End
LS – Line Start   LE – Line End
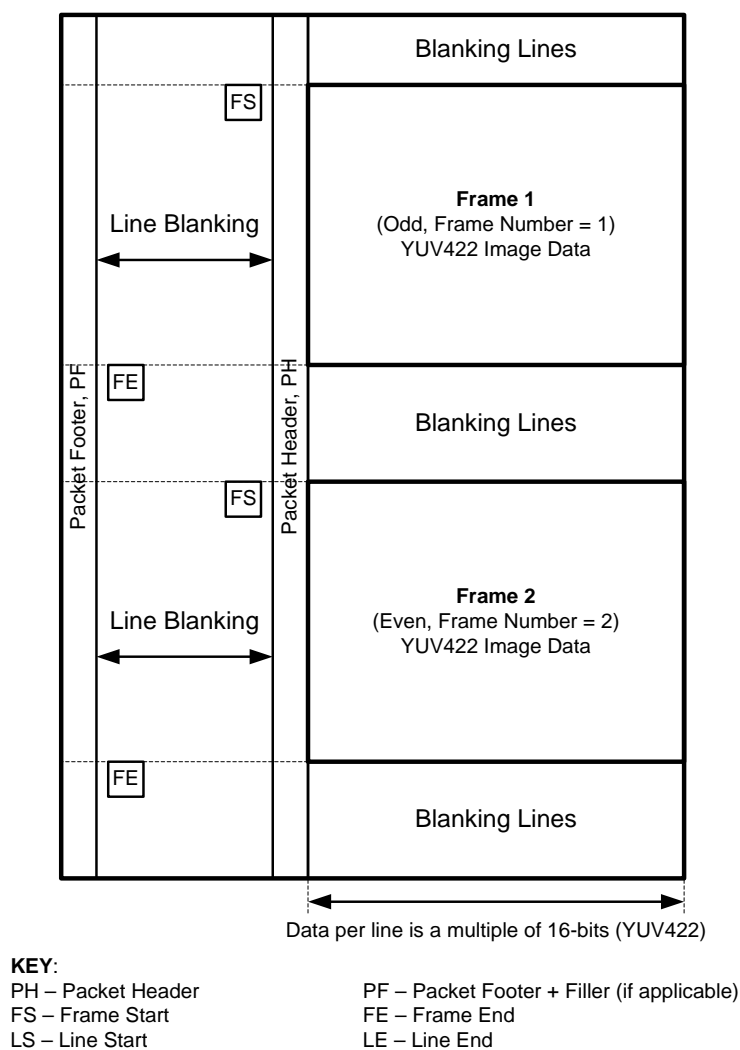
1485

Figure 90 Digital Interlaced Video with Accurate Synchronization Timing Information

9.15 Data Interleaving

1486 The CSI-2 supports the interleaved transmission of different image data formats within the same video data
1487 stream.

1488 There are two methods to interleave the transmission of different image data formats:

1489 • Data Type

1490 • Virtual Channel Identifier

1491 The preceding methods of interleaved data transmission can be combined in any manner.

9.15.1 Data Type Interleaving

1492 The Data Type value uniquely defines the data format for that packet of data. The receiver uses the Data
1493 Type value in the packet header to de-multiplex data packets containing different data formats as illustrated
1494 in *Figure 91*. Note, in the figure the Virtual Channel Identifier is the same in each Packet Header.

1495 The packet payload data format shall agree with the Data Type code in the Packet Header as follows:

1496 • For defined image data types – any non-reserved codes in the range 0x18 to 0x3F – only the single
1497 corresponding MIPI-defined packet payload data format shall be considered correct

1498 • Reserved image data types – any reserved codes in the range 0x18 to 0x3F – shall not be used. No
1499 packet payload data format shall be considered correct for reserved image data types

1500 • For generic long packet data types (codes 0x10 thru 0x17) and user-defined, byte-based (codes
1501 0x30 – 0x37), any packet payload data format shall be considered correct

1502 • Generic long packet data types (codes 0x10 thru 0x17) and user-defined, byte-based (codes 0x30 –
1503 0x37), should not be used with packet payloads that meet any MIPI image data format definition

1504 • Synchronization short packet data types (codes 0x00 thru 0x07) shall consist of only the header
1505 and shall not include payload data bytes

1506 • Generic short packet data types (codes 0x08 thru 0x0F) shall consist of only the header and shall
1507 not include payload data bytes

1508 Data formats are defined further in *Section 11*.

1509
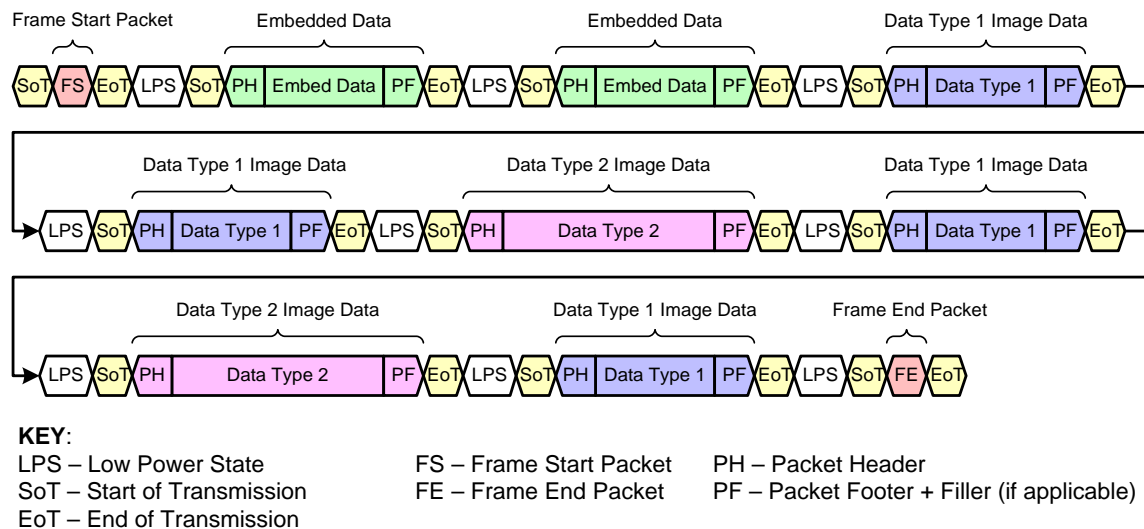
Figure 91 Interleaved Data Transmission using Data Type Value

1510 All of the packets within the same virtual channel, independent of the Data Type value, share the same
1511 frame start/end and line start/end synchronization information. By definition, all of the packets,

1512 independent of data type, between a Frame Start and a Frame End packet within the same virtual channel
1513 belong to the same frame.

1514 Packets of different data types may be interleaved at either the packet level as illustrated in *Figure 92* or
1515 the frame level as illustrated in *Figure 93*. Data formats are defined in *Section 11*.

1516
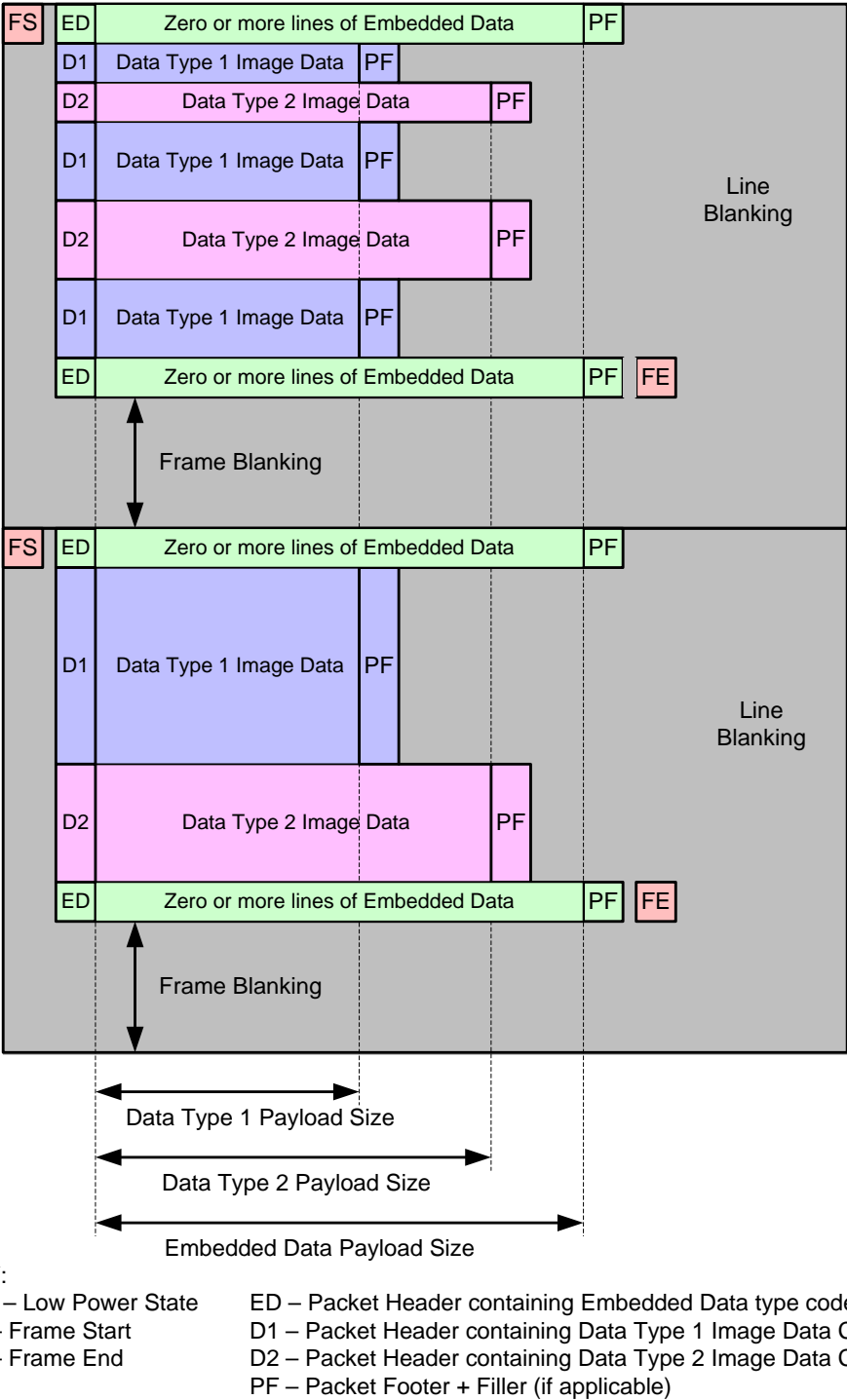
Figure 92 Packet Level Interleaved Data Transmission

Figure 93 Frame Level Interleaved Data Transmission
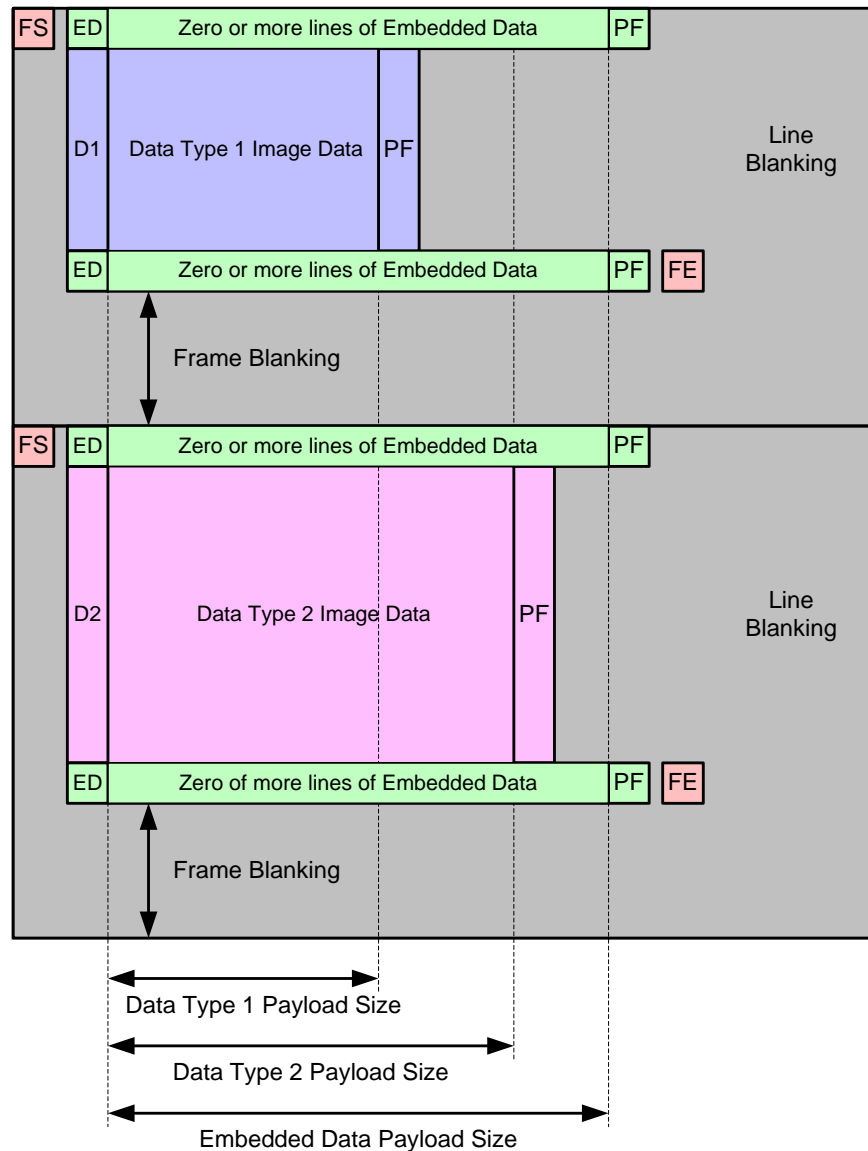
KEY:
LPS – Low Power State   ED – Packet Header containing Embedded Data type code
FS – Frame Start   D1 – Packet Header containing Data Type 1 Image Data Code
FE – Frame End   D2 – Packet Header containing Data Type 2 Image Data Code
                             PF – Packet Footer + Filler (if applicable)

9.15.2   Virtual Channel Identifier Interleaving

1518   The Virtual Channel Identifier allows different data types within a single data stream to be logically
1519   separated from each other. *Figure 94* illustrates data interleaving using the Virtual Channel Identifier.

1520   Each virtual channel has its own Frame Start and Frame End packet. Therefore, it is possible for different
1521   virtual channels to have different frame rates, though the data rate for both channels would remain the
1522   same.

1523   In addition, Data Type value Interleaving can be used for each virtual channel, allowing different data types
1524   within a virtual channel and a second level of data interleaving.

1525   Therefore, receivers should be able to de-multiplex different data packets based on the combination of the
1526   Virtual Channel Identifier and the Data Type value. For example, data packets containing the same Data
1527   Type value but transmitted on different virtual channels are considered to belong to different frames
1528   (streams) of image data.

1529
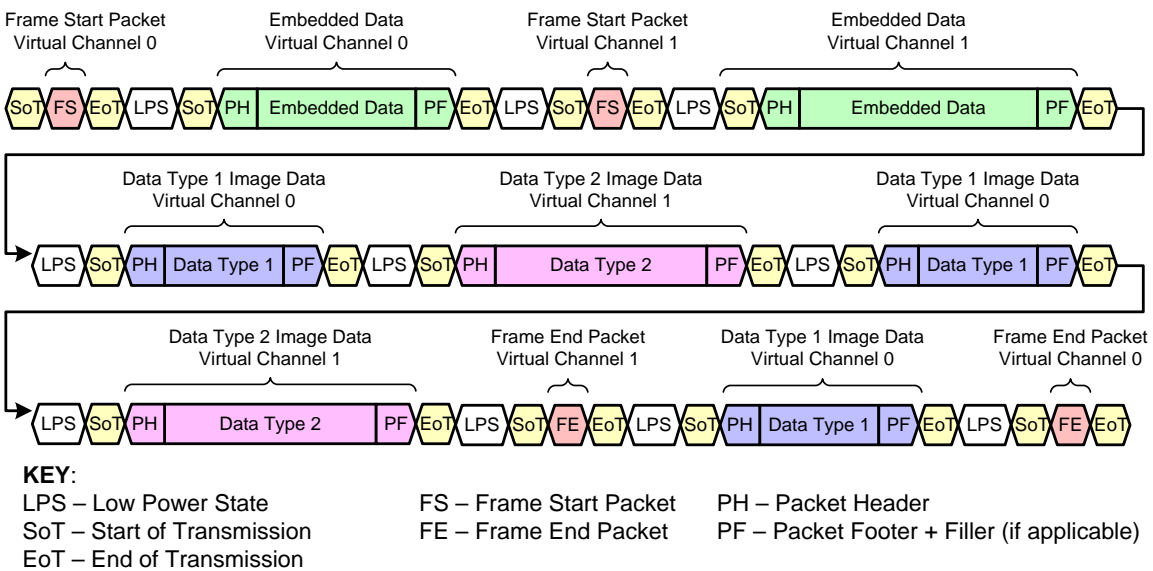
Figure 94 Interleaved Data Transmission using Virtual Channels

This page intentionally left blank

10   Color Spaces

1530 The color space definitions in this section are simply references to other standards. The references are
1531 included only for informative purposes and not for compliance. The color space used is not limited to the
1532 references given.

10.1   RGB Color Space Definition

1533 In this Specification, the abbreviation RGB means the nonlinear sR'G'B' color space in 8-bit representation
1534 based on the definition of sRGB in IEC 61966.

1535 The 8-bit representation results as RGB888. The conversion to the more commonly used RGB565 format is
1536 achieved by scaling the 8-bit values to five bits (blue and red) and six bits (green). The scaling can be done
1537 either by simply dropping the LSBs or rounding.

10.2   YUV Color Space Definition

1538 In this Specification, the abbreviation YUV refers to the 8-bit gamma corrected Y'CBCR color space
1539 defined in ITU-R BT601.4.

This page intentionally left blank.

11 Data Formats

1540 The intent of this section is to provide a definitive reference for data formats typically used in CSI-2
1541 applications. *Table 24* summarizes the formats, followed by individual definitions for each format. Generic
1542 data types not shown in the table are described in *Section 11.1*. For simplicity, all examples are single Lane
1543 configurations.

1544 The formats most widely used in CSI-2 applications are distinguished by a "primary" designation in *Table
1545 24*. Transmitter implementations of CSI-2 should support at least one of these primary formats. Receiver
1546 implementations of CSI-2 should support all of the primary formats.

1547 The packet payload data format shall agree with the Data Type value in the Packet Header. See *Section 9.4*
1548 for a description of the Data Type values.

1549 **Table 24 Primary and Secondary Data Formats Definitions**

Data Format	Primary	Secondary
YUV420 8-bit (legacy)		S
YUV420 8-bit		S
YUV420 10-bit		S
YUV420 8-bit (CSPS)		S
YUV420 10-bit (CSPS)		S
YUV422 8-bit	P	
YUV422 10-bit		S
RGB888	P	
RGB666		S
RGB565	P	
RGB555		S
RGB444		S
RAW6		S
RAW7		S
RAW8	P	
RAW10	P	
RAW12		S
RAW14		S
RAW16		S
RAW20		S
Generic 8-bit Long Packet Data Types	P	
User Defined Byte-based Data (Note 1)	P	

Note:

1. *Compressed image data should use the user defined, byte-based data type codes*

1550 For clarity the Start of Transmission and End of Transmission sequences in the figures in this section have
1551 been omitted.

1552 The balance of this section details how sequences of pixels and other application data conforming to each
1553 of the data types listed in *Table 24* are converted into equivalent byte sequences by the CSI-2 Pixel to Byte
1554 Packing Formats layer shown in *Figure 3*.

1555 Various figures in this section depict these byte sequences as shown at the top of *Figure 95*, where Byte n
1556 always precedes Byte m for n < m. Also note that even though each byte is shown in LSB-first order, this is
1557 not meant to imply that the bytes themselves are bit-reversed by the Pixel to Byte Packing Formats layer
1558 prior to output.

1559 For the D-PHY physical layer option, each byte in the sequence is serially transmitted LSB-first, whereas
1560 for the C-PHY physical layer option, successive byte pairs in the sequence are encoded and then serially
1561 transmitted LSS-first. *Figure 95* illustrates these options for a single-Lane system.
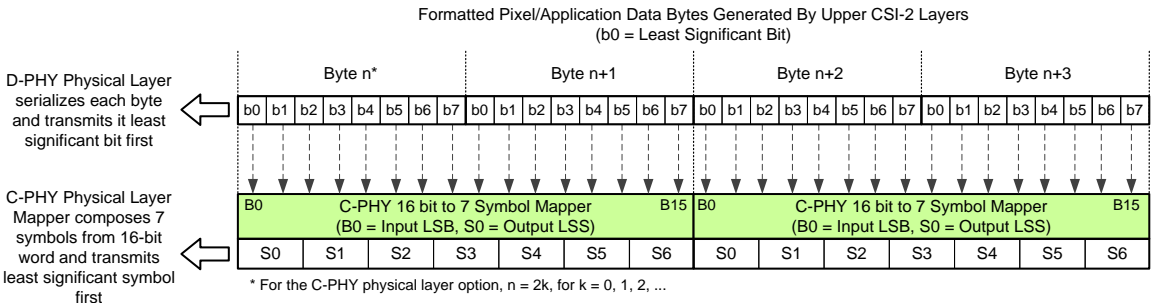
Figure 95 Byte Packing Pixel Data to C-PHY Symbol Illustration

11.1 Generic 8-bit Long Packet Data Types

1563   *Table 25* defines the generic 8-bit Long packet data types.

1564   **Table 25 Generic 8-bit Long Packet Data Types**

Data Type	Description	See Section
0x10	Null	*11.1.1*
0x11	Blanking Data	
0x12	Embedded 8-bit non Image Data	*11.1.2*
0x13	Generic long packet data type 1	*11.1.3*
0x14	Generic long packet data type 2	
0x15	Generic long packet data type 3	
0x16	Generic long packet data type 4	
0x17	Reserved	–

11.1.1 Null and Blanking Data

1565   For both the null and blanking data types the receiver must ignore the content of the packet payload data.

1566   A blanking packet differs from a null packet in terms of its significance within a video data stream. A null
1567 packet has no meaning whereas the blanking packet may be used, for example, as the blanking lines
1568 between frames in an ITU-R BT.656 style video stream.

11.1.2 Embedded Information

1569   It is possible to embed extra lines containing additional information to the beginning and to the end of each
1570 picture frame as presented in the *Figure 96*. If embedded information exists, then the lines containing the
1571 embedded data must use the embedded data code in the data identifier.

1572   There may be zero or more lines of embedded data at the start of the frame. These lines are termed the
1573 frame header.

1574   There may be zero or more line of embedded data at the end of the frame. These lines are termed the frame
1575 footer.

Payload Data per packet must be a multiple of 8-bits

KEY:
LPS – Low Power State	DI – Data Identifier	WC – Word Count
ECC – Error Correction Code	CS – Checksum	FS – Frame Start
FE – Frame End	LS – Line Start	LE – Line End
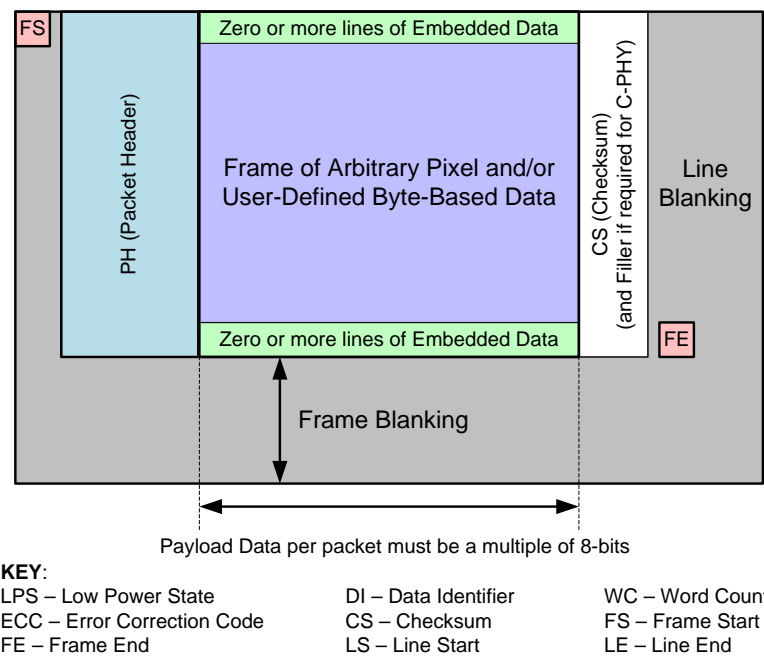
Figure 96 Frame Structure with Embedded Data at the Beginning and End of the Frame

11.1.3   Generic Long Packet Data Types 1 Through 4

These codes have no specific definitions and may be used, for example, to identify various types of vendor-specific metadata packets transmitted within an image frame.

11.2 YUV Image Data

1579 *Table 26* defines the data type codes for YUV data formats described in this section. The number of lines
1580 transmitted for the YUV420 data type shall be even.

1581 YUV420 data formats are divided into legacy and non-legacy data formats. The legacy YUV420 data
1582 format is for compatibility with existing systems. The non-legacy YUV420 data formats enable lower cost
1583 implementations.

1584 **Table 26 YUV Image Data Types**

Data Type	Description
0x18	YUV420 8-bit
0x19	YUV420 10-bit
0x1A	Legacy YUV420 8-bit
0x1B	Reserved
0x1C	YUV420 8-bit (Chroma Shifted Pixel Sampling)
0x1D	YUV420 10-bit (Chroma Shifted Pixel Sampling)
0x1E	YUV422 8-bit
0x1F	YUV422 10-bit

11.2.1 Legacy YUV420 8-bit

1585 Legacy YUV420 8-bit data transmission is performed by transmitting UYY… / VYY… sequences in odd /
1586 even lines. U component is transferred in odd lines (1, 3, 5 …) and V component is transferred in even lines
1587 (2, 4, 6 …). This sequence is illustrated in *Figure 97*.

1588 *Table 27* specifies the packet size constraints for YUV420 8-bit packets. Each packet must be a multiple of
1589 the values in the table.

1590 **Table 27 Legacy YUV420 8-bit Packet Data Size Constraints**

Pixels	Bytes	Bits
2	3	24

1591 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated
1592 in *Figure 98*.

1593
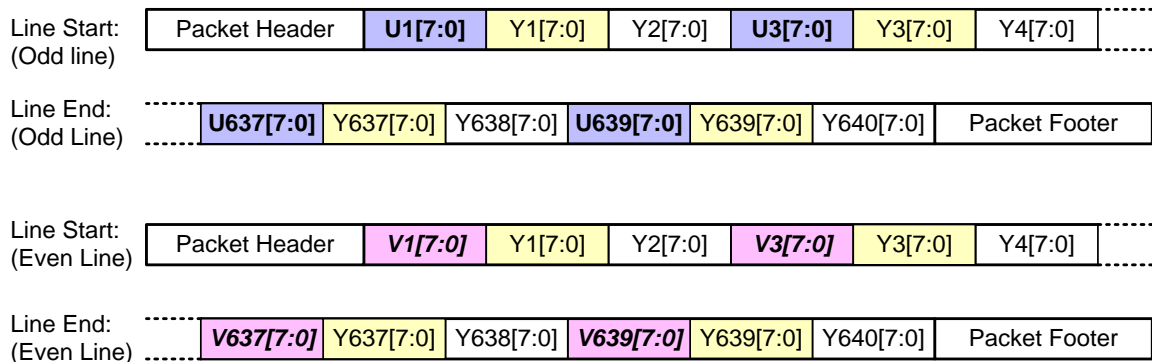
Figure 97 Legacy YUV420 8-bit Transmission
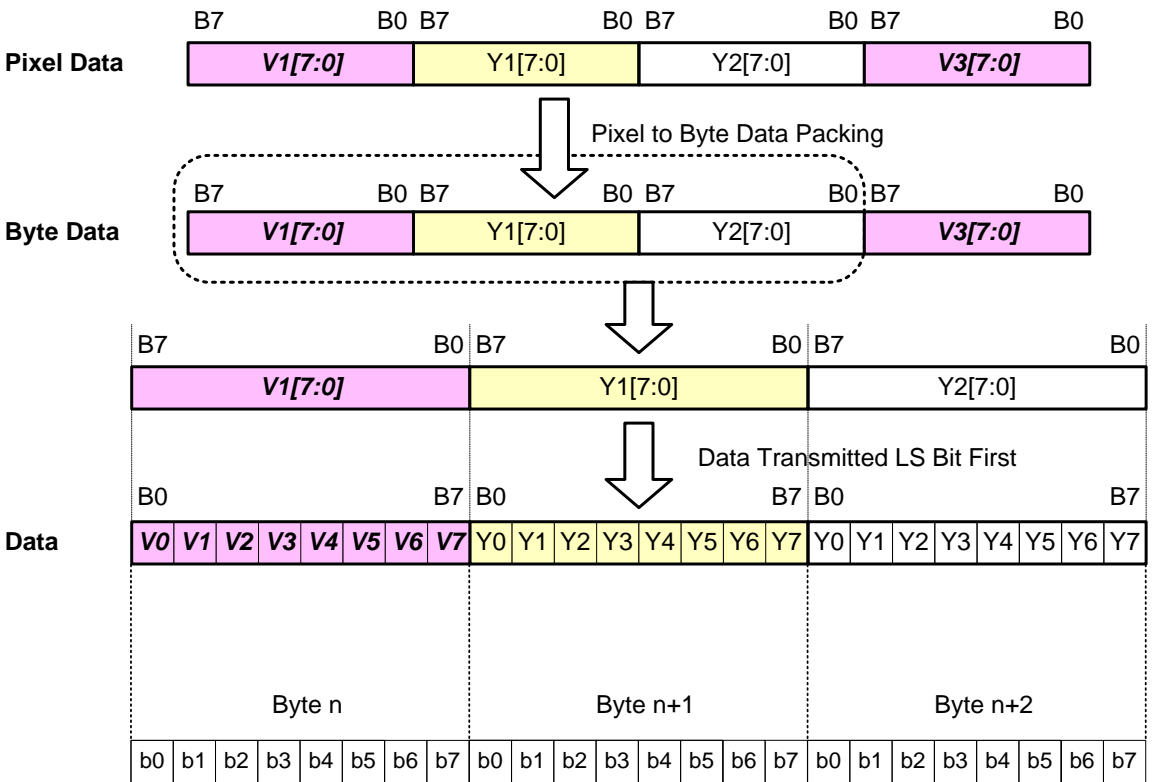
Figure 98 Legacy YUV420 8-bit Pixel to Byte Packing Bitwise Illustration

There is one spatial sampling option

- H.261, H.263 and MPEG1 Spatial Sampling (*Figure 99*).
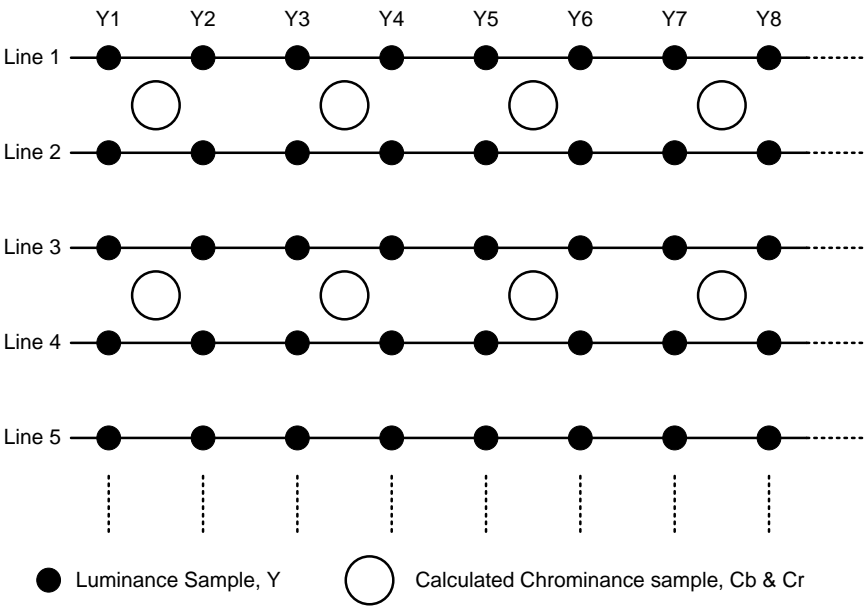
Figure 99 Legacy YUV420 Spatial Sampling for H.261, H.263 and MPEG 1
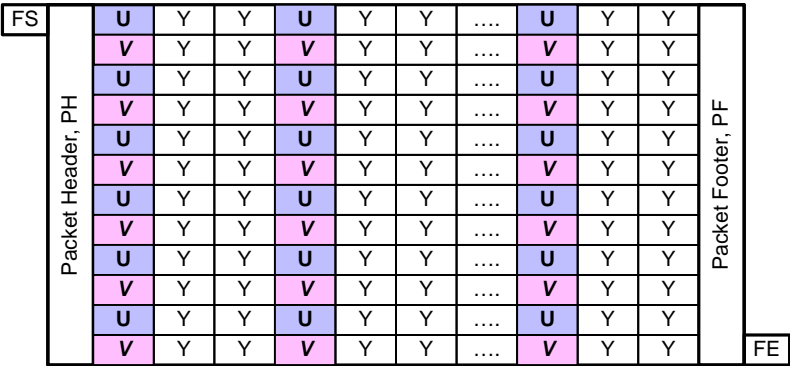
Figure 100 Legacy YUV420 8-bit Frame Format

11.2.2   YUV420 8-bit

1599 YUV420 8-bit data transmission is performed by transmitting YYYY… / UYVYUYVY… sequences in
1600 odd / even lines. Only the luminance component (Y) is transferred for odd lines (1, 3, 5…) and both
1601 luminance (Y) and chrominance (U and V) components are transferred for even lines (2, 4, 6…). The
1602 format for the even lines (UYVY) is identical to the YUV422 8-bit data format. The data transmission
1603 sequence is illustrated in *Figure 101*.

1604 The payload data size, in bytes, for even lines (UYVY) is double the payload data size for odd lines (Y).
1605 This is exception to the general CSI-2 rule that each line shall have an equal length.

1606 *Table 28* specifies the packet size constraints for YUV420 8-bit packets. Each packet must be a multiple of
1607 the values in the table.

1608 **Table 28 YUV420 8-bit Packet Data Size Constraints**

Odd Lines (1, 3, 5...) Luminance Only, Y			Even Lines (2, 4, 6…) Luminance and Chrominance, UYVY		
Pixels	**Bytes**	**Bits**	**Pixels**	**Bytes**	**Bits**
2	2	16	2	4	32

1609 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated
1610 in *Figure 102*.

1611
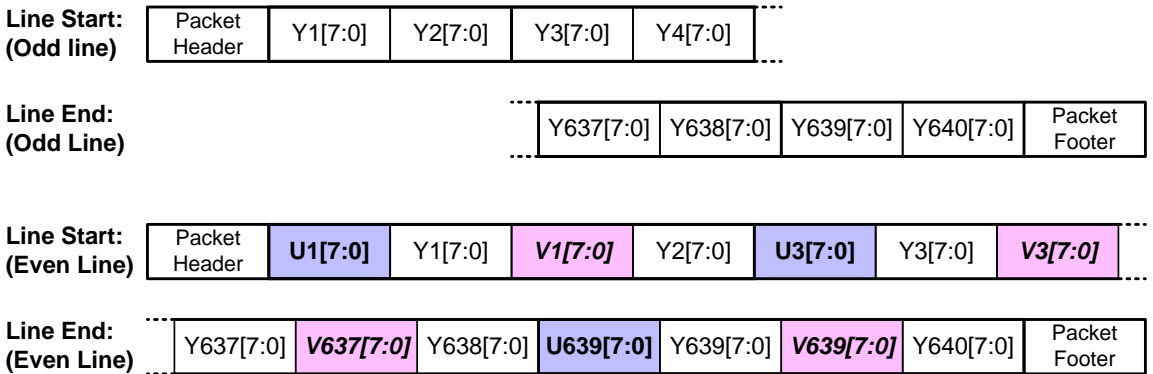
Figure 101 YUV420 8-bit Data Transmission Sequence

Odd lines:

Even lines:

1612
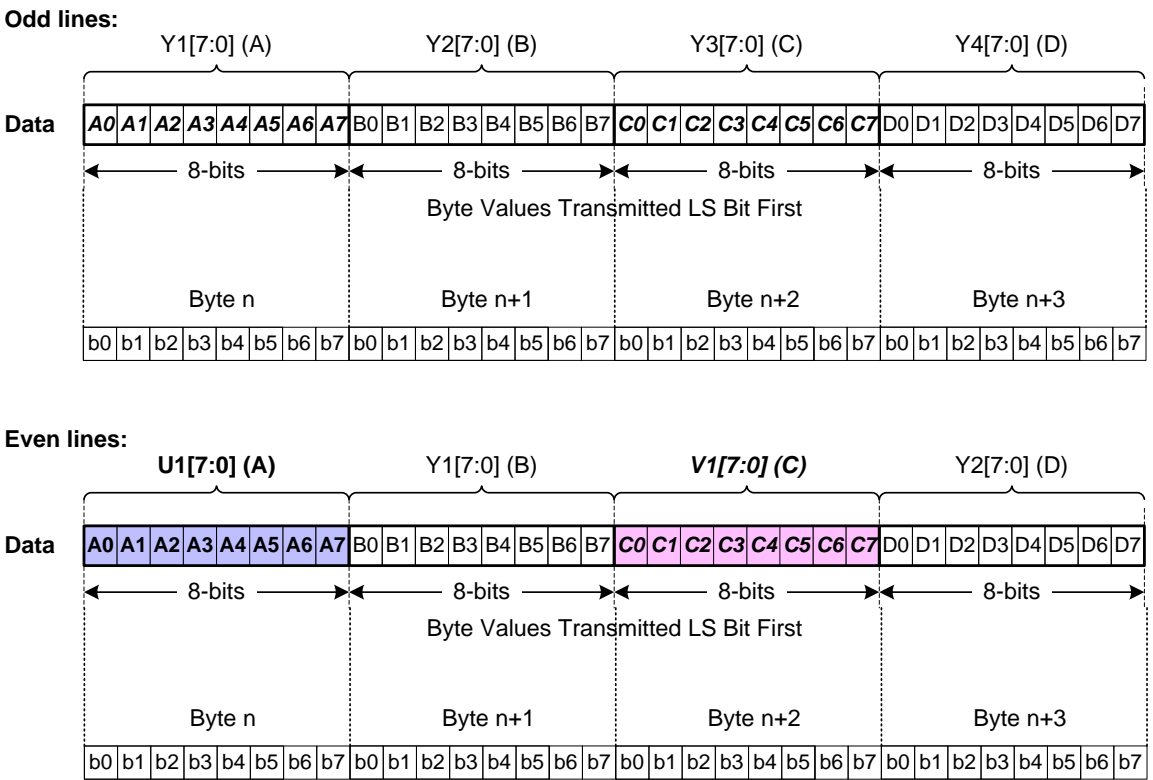
Figure 102 YUV420 8-bit Pixel to Byte Packing Bitwise Illustration

1613   There are two spatial sampling options

1614   • H.261, H.263 and MPEG1 Spatial Sampling (*Figure 103*).

1615   • Chroma Shifted Pixel Sampling (CSPS) for MPEG2, MPEG4 (*Figure 104*).

1616   *Figure 105* shows the YUV420 frame format.
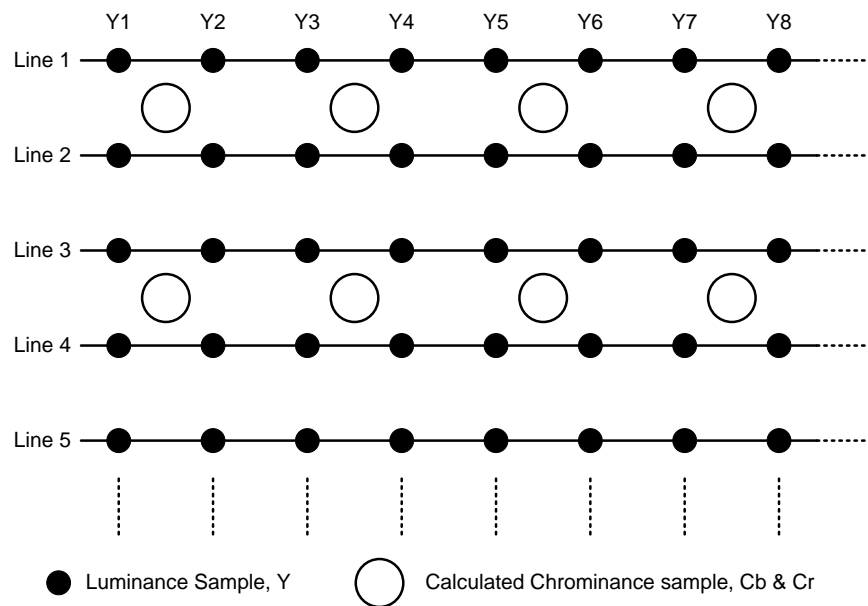
Figure 103 YUV420 Spatial Sampling for H.261, H.263 and MPEG 1
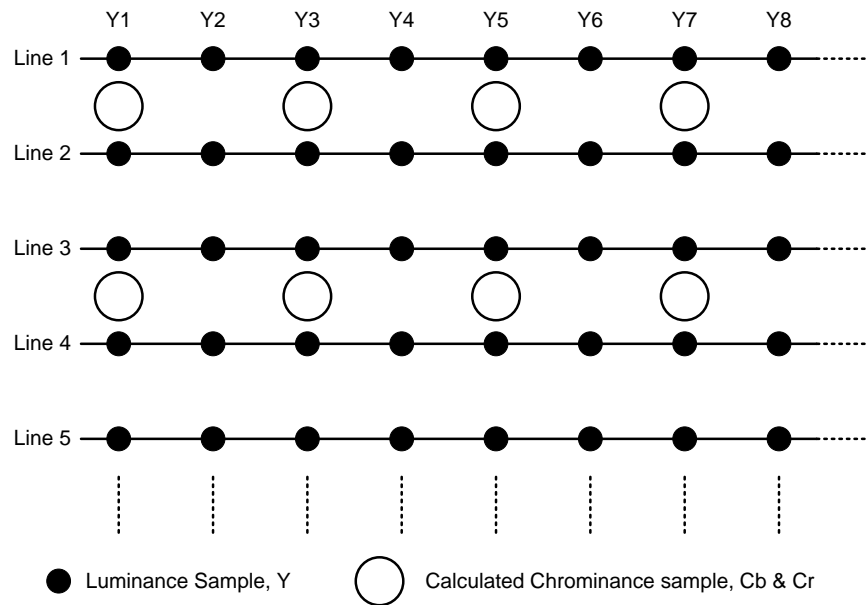
Figure 104 YUV420 Spatial Sampling for MPEG 2 and MPEG 4
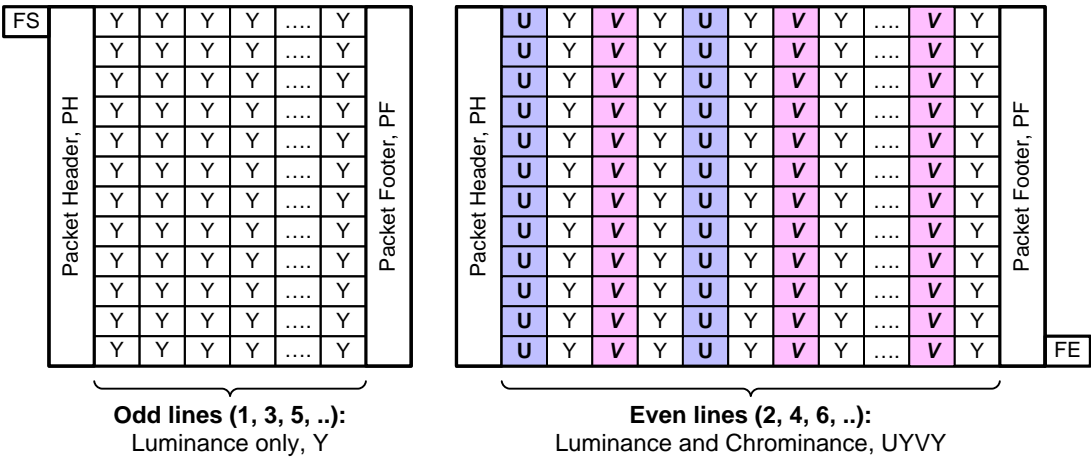
Figure 105 YUV420 8-bit Frame Format

11.2.3 YUV420 10-bit

1620 YUV420 10-bit data transmission is performed by transmitting YYYY… / UYVYUYVY… sequences in
1621 odd / even lines. Only the luminance component (Y) is transferred in odd lines (1, 3, 5…) and both
1622 luminance (Y) and chrominance (U and V) components transferred in even lines (2, 4, 6…). The format for
1623 the even lines (UYVY) is identical to the YUV422 –10-bit data format. The sequence is illustrated in
1624 *Figure 106*.

1625 The payload data size, in bytes, for even lines (UYVY) is double the payload data size for odd lines (Y).
1626 This is exception to the general CSI-2 rule that each line shall have an equal length.

1627 *Table 29* specifies the packet size constraints for YUV420 10-bit packets. The length of each packet must
1628 be a multiple of the values in the table.

1629

Table 29 YUV420 10-bit Packet Data Size Constraints

Odd Lines (1, 3, 5...) Luminance Only, Y			Even Lines (2, 4, 6…) Luminance and Chrominance, UYVY		
Pixels	**Bytes**	**Bits**	**Pixels**	**Bytes**	**Bits**
4	5	40	4	10	80

1630 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel-to-byte mapping is illustrated
1631 in *Figure 107*.

1632

**Figure 106 YUV420 10-bit Transmission**

Odd lines:

Even lines:
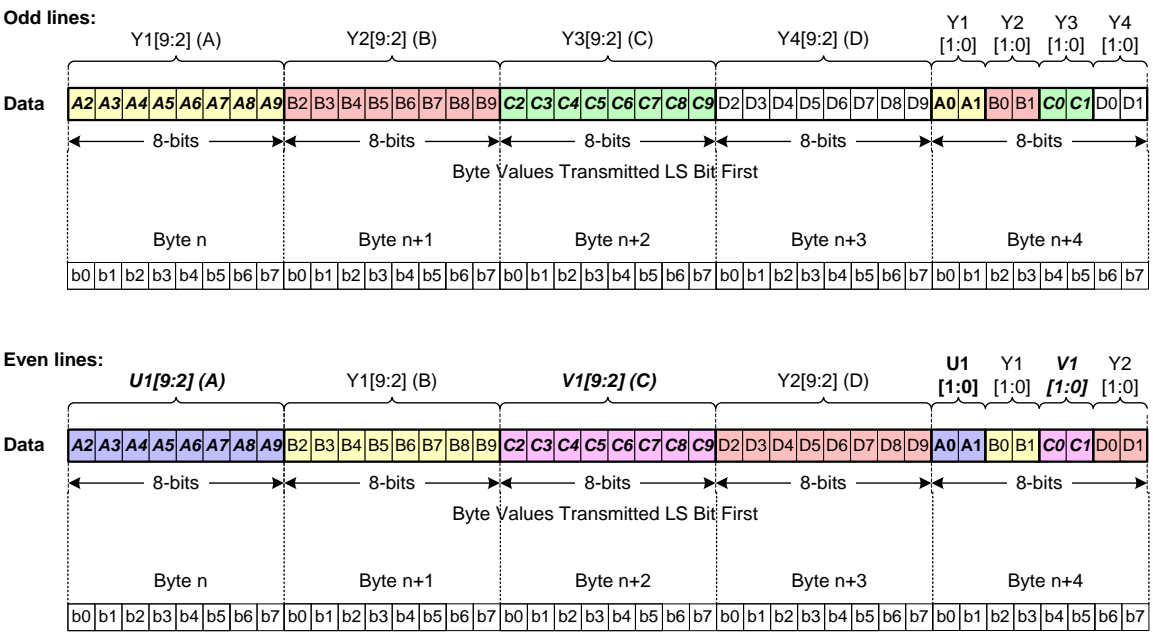
Figure 107 YUV420 10-bit Pixel to Byte Packing Bitwise Illustration

The pixel spatial sampling options are the same as for the YUV420 8-bit data format.

Odd lines (1, 3, 5, ..):
Luminance only, Y

Even lines (2, 4, 6, ..):
Luminance and Chrominance, UYVY
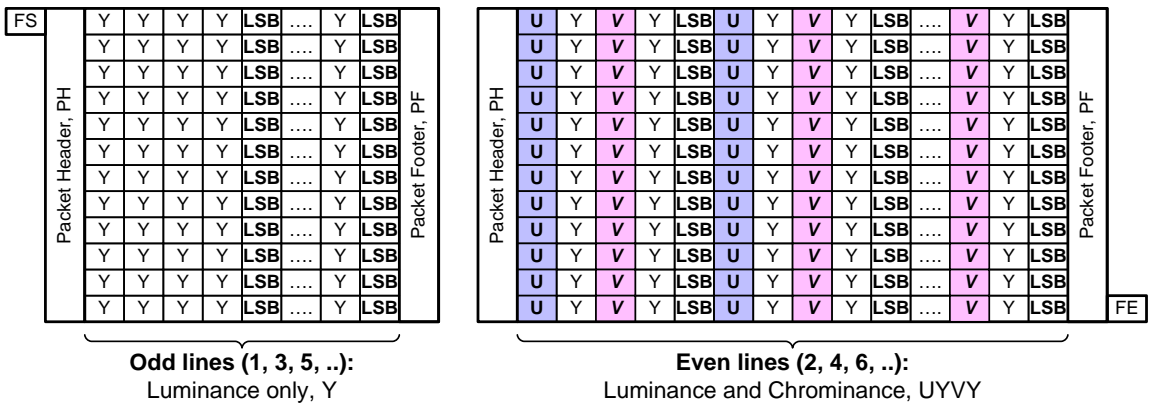
Figure 108 YUV420 10-bit Frame Format

11.2.4 YUV422 8-bit

1636 YUV422 8-bit data transmission is performed by transmitting a UYVY sequence. This sequence is
1637 illustrated in *Figure 109*.

1638 *Table 30* specifies the packet size constraints for YUV422 8-bit packet. The length of each packet must be a
1639 multiple of the values in the table.

1640

Table 30 YUV422 8-bit Packet Data Size Constraints

Pixels	Bytes	Bits
2	4	32

1641 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated
1642 in *Figure 110*.

1643
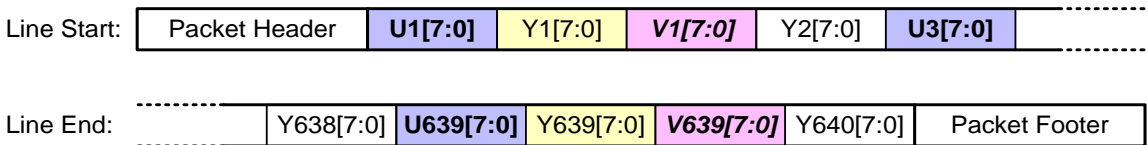
Figure 109 YUV422 8-bit Transmission

1644
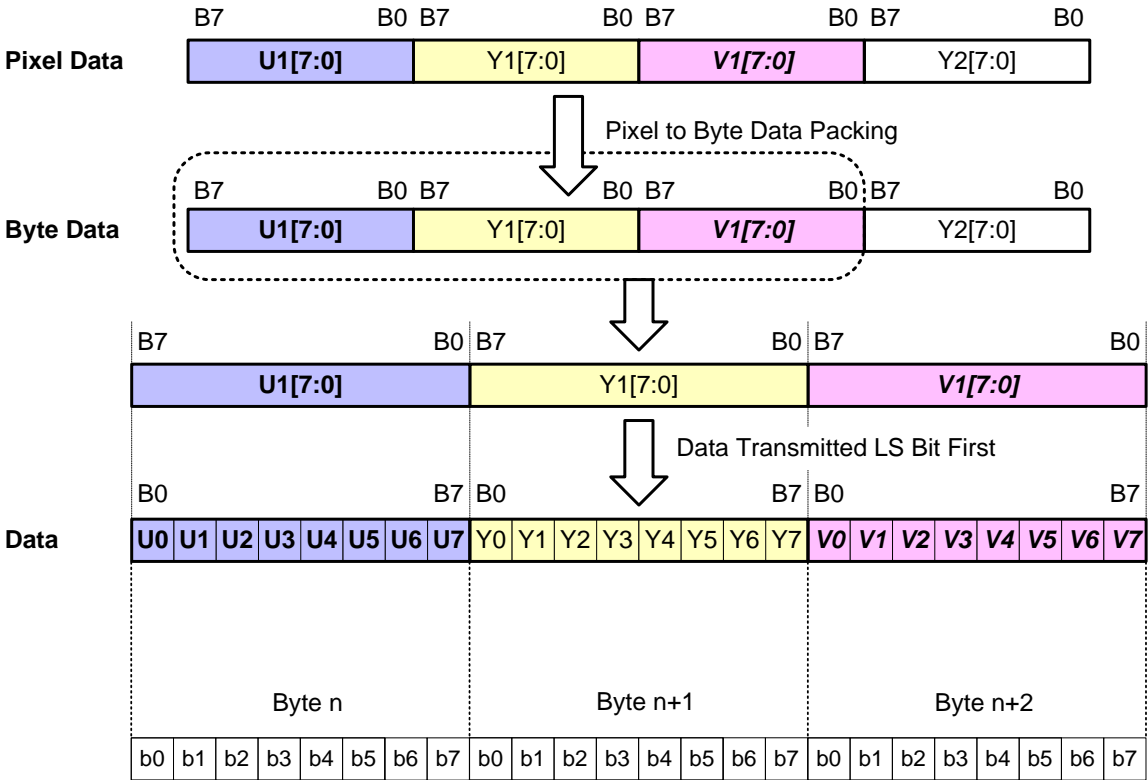
Figure 110 YUV422 8-bit Pixel to Byte Packing Bitwise Illustration
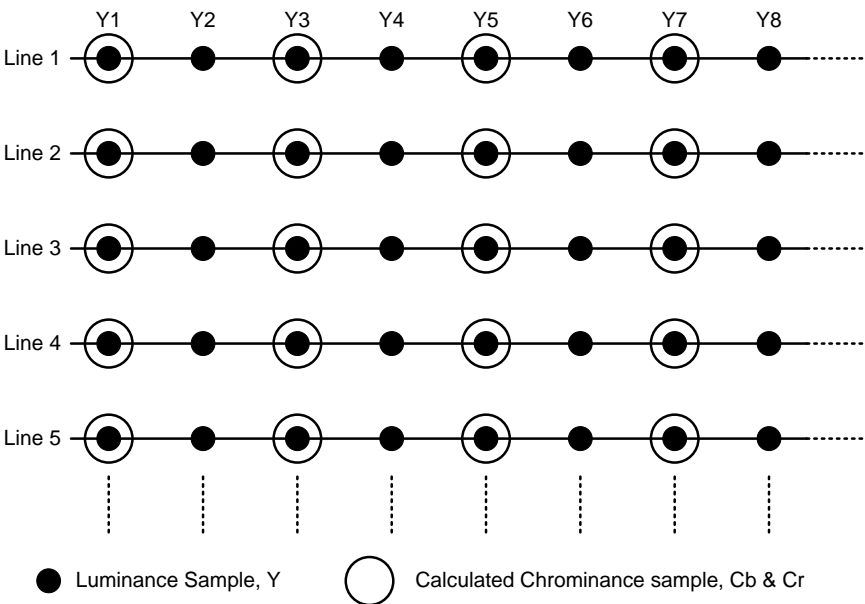
Figure 111 YUV422 Co-sited Spatial Sampling

The pixel spatial alignment is the same as in CCIR-656 standard. The frame format for YUV422 is presented in *Figure 112*.
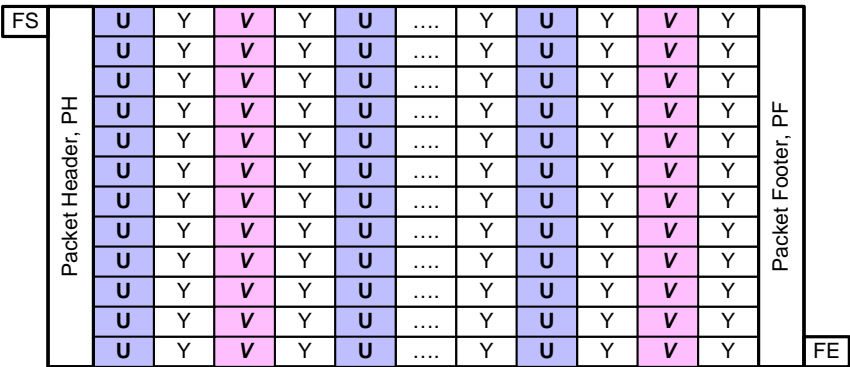
Figure 112 YUV422 8-bit Frame Format

11.2.5 YUV422 10-bit

1649
1650

YUV422 10-bit data transmission is performed by transmitting a UYVY sequence. This sequence is illustrated in *Figure 113*.

1651
1652

Table 31 specifies the packet size constraints for YUV422 10-bit packet. The length of each packet must be a multiple of the values in the table.

1653

Table 31 YUV422 10-bit Packet Data Size Constraints

Pixels	Bytes	Bits
2	5	40

1654
1655

Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated in *Figure 114*.

1656
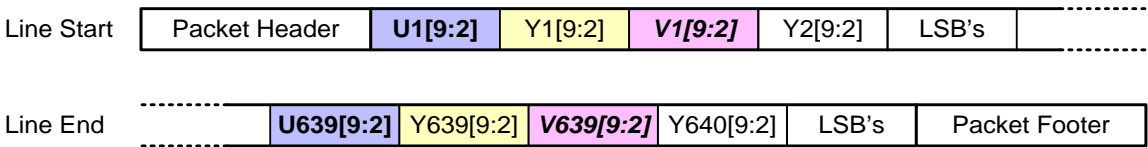
Figure 113 YUV422 10-bit Transmitted Bytes

1657
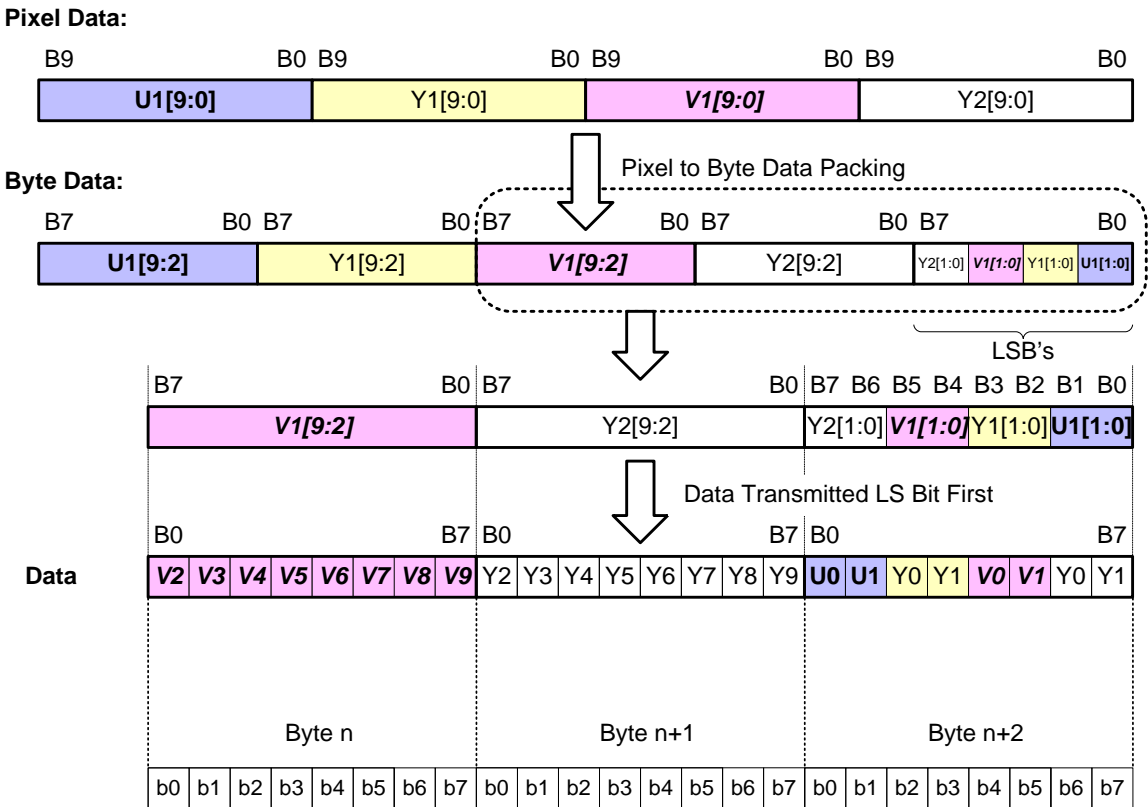
Figure 114 YUV422 10-bit Pixel to Byte Packing Bitwise Illustration

1658
1659

The pixel spatial alignment is the same as in the YUV422 8-bit data case. The frame format for YUV422 is presented in the *Figure 115*.
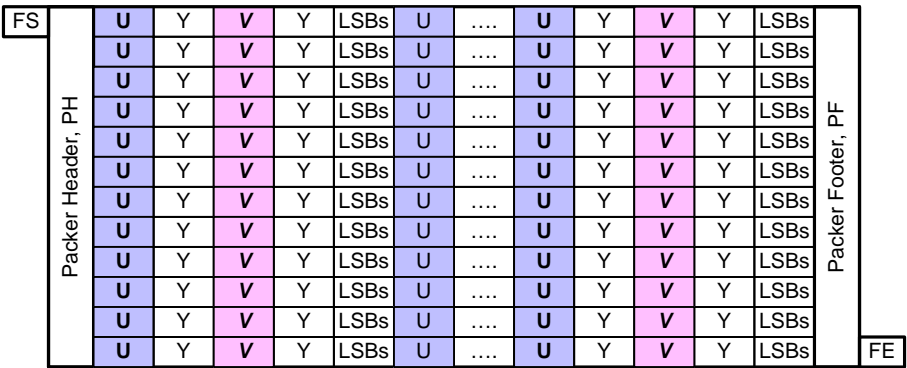
Figure 115 YUV422 10-bit Frame Format

11.3   RGB Image Data

1661   *Table 32* defines the data type codes for RGB data formats described in this section.

1662   **Table 32 RGB Image Data Types**

Data Type	Description
0x20	RGB444
0x21	RGB555
0x22	RGB565
0x23	RGB666
0x24	RGB888
0x25	Reserved
0x26	Reserved
0x27	Reserved

11.3.1 RGB888

1663 RGB888 data transmission is performed by transmitting a BGR byte sequence. This sequence is illustrated
1664 in *Figure 116*. The RGB888 frame format is illustrated in *Figure 118*.

1665 *Table 33* specifies the packet size constraints for RGB888 packets. The length of each packet must be a
1666 multiple of the values in the table.

1667 **Table 33 RGB888 Packet Data Size Constraints**

Pixels	Bytes	Bits
1	3	24

1668 Bit order in transmission follows the general CSI-2 rule, LSB first. The pixel to byte mapping is illustrated
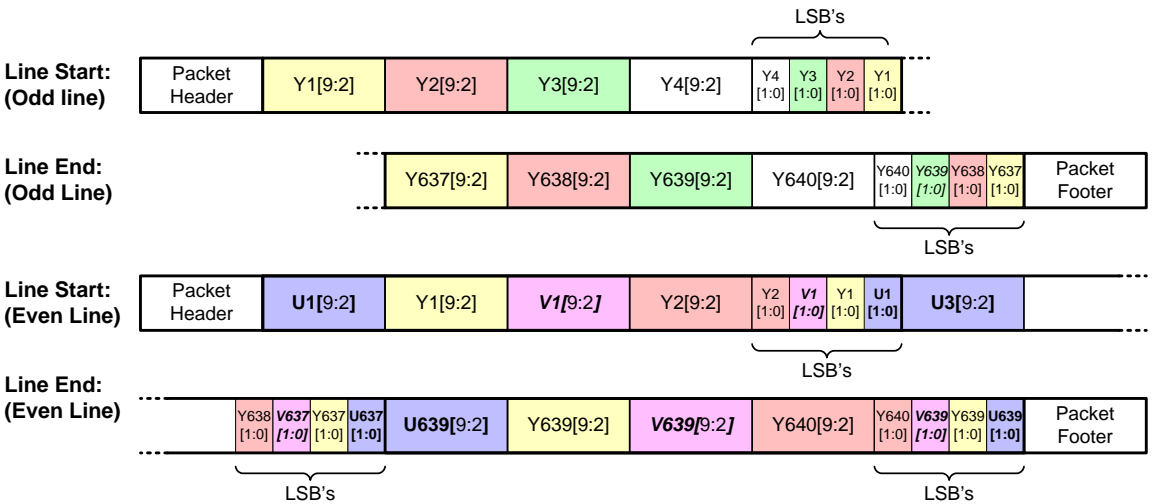1669 in *Figure 117*.
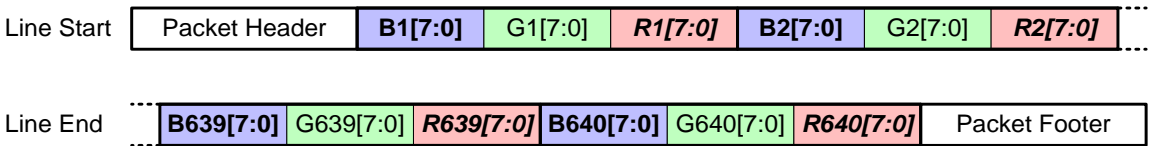
Figure 116 RGB888 Transmission
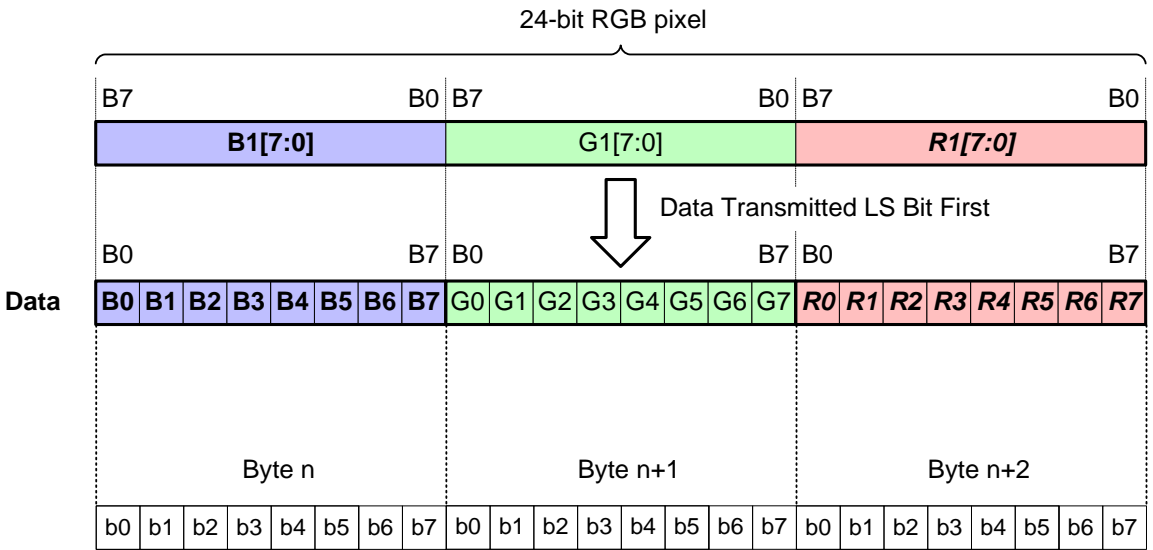
Figure 117 RGB888 Transmission in CSI-2 Bus Bitwise Illustration
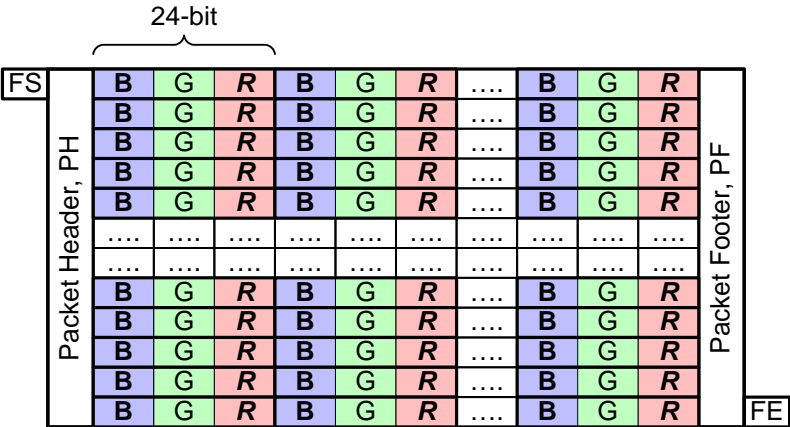
Figure 118 RGB888 Frame Format

11.3.2 RGB666

1673 RGB666 data transmission is performed by transmitting a B0…5, G0…5, and R0…5 (18-bit) sequence.
1674 This sequence is illustrated in *Figure 119*. The frame format for RGB666 is presented in the *Figure 121*.

1675 *Table 34* specifies the packet size constraints for RGB666 packets. The length of each packet must be a
1676 multiple of the values in the table.

1677 **Table 34 RGB666 Packet Data Size Constraints**

Pixels	Bytes	Bits
4	9	72

1678 Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB666 case the length of one data
1679 word is 18-bits, not eight bits. The word-wise flip is done for 18-bit BGR words; i.e. instead of flipping
1680 each byte (8-bits), each 18-bits pixel value is flipped. This is illustrated in *Figure 120*.

1681
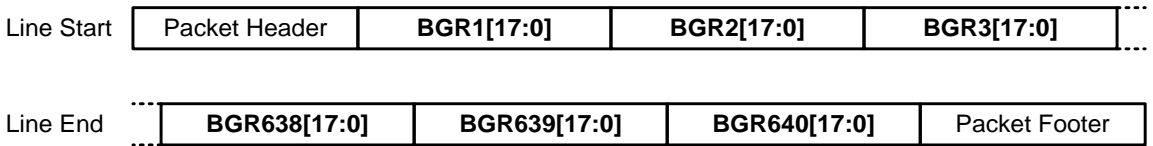
Figure 119 RGB666 Transmission with 18-bit BGR Words

1682
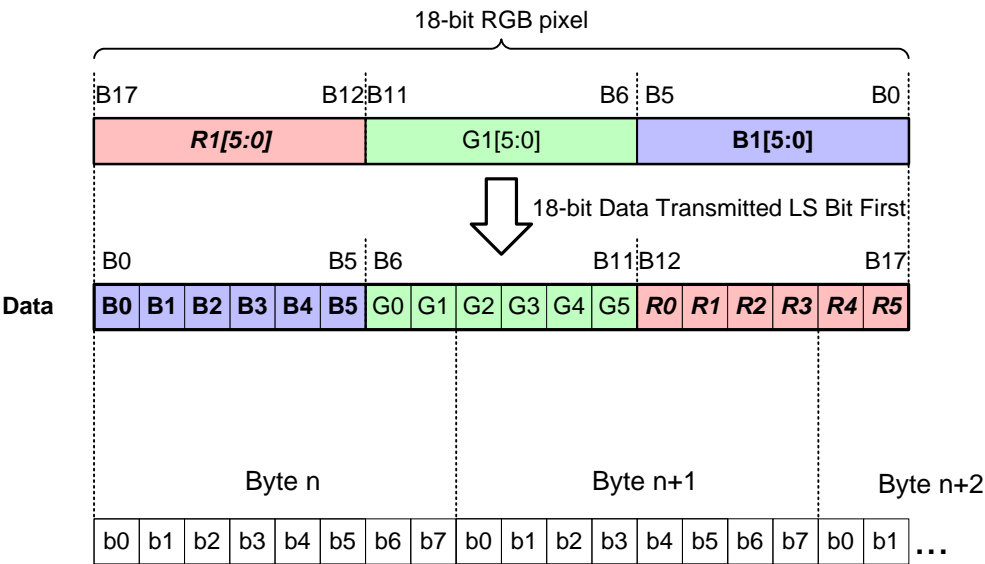
Figure 120 RGB666 Transmission on CSI-2 Bus Bitwise Illustration
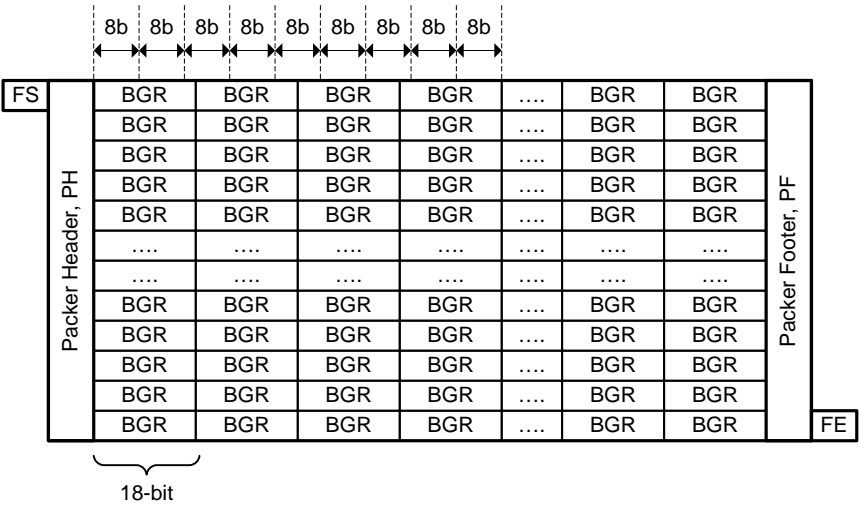
Figure 121 RGB666 Frame Format

11.3.3 RGB565

1684 RGB565 data transmission is performed by transmitting B0…B4, G0…G5, R0…R4 in a 16-bit sequence.
1685 This sequence is illustrated in *Figure 122*. The frame format for RGB565 is presented in the *Figure 124*.

1686 *Table 35* specifies the packet size constraints for RGB565 packets. The length of each packet must be a
1687 multiple of the values in the table.

1688 **Table 35 RGB565 Packet Data Size Constraints**

Pixels	Bytes	Bits
1	2	16

1689 Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB565 case the length of one data
1690 word is 16-bits, not eight bits. The word-wise flip is done for 16-bit BGR words; i.e. instead of flipping
1691 each byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in *Figure 123*.

1692
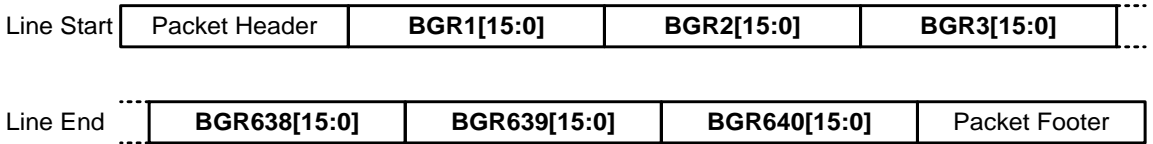
Figure 122 RGB565 Transmission with 16-bit BGR Words

1693

Figure 123 RGB565 Transmission on CSI-2 Bus Bitwise Illustration

16-bit

	BGR	BGR	BGR	....	BGR	BGR	
	BGR	BGR	BGR	....	BGR	BGR	
	BGR	BGR	BGR	....	BGR	BGR	
Packer Header, PH	BGR	BGR	BGR	....	BGR	BGR	Packer Footer, PF
	BGR	BGR	BGR	....	BGR	BGR	
	....	....	....	....	....	....	
	....	....	....	....	....	....	
	BGR	BGR	BGR	....	BGR	BGR	
	BGR	BGR	BGR	....	BGR	BGR	
	BGR	BGR	BGR	....	BGR	BGR	
	BGR	BGR	BGR	....	BGR	BGR	
	BGR	BGR	BGR	....	BGR	BGR	

FS (top left), FE (bottom right)
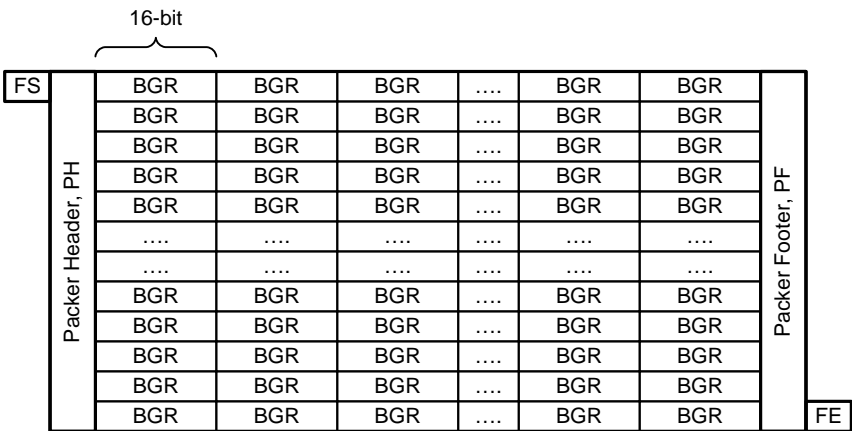
1694

Figure 124 RGB565 Frame Format

11.3.4 RGB555

1695 RGB555 data can be transmitted over a CSI-2 bus with some special arrangements. The RGB555 data
1696 should be made to look like RGB565 data. This can be accomplished by inserting padding bits to the LSBs
1697 of the green color component as illustrated in *Figure 125*.

1698 Both the frame format and the package size constraints are the same as the RGB565 case.

1699 Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB555 case the length of one data
1700 word is 16-bits, not eight bits. The word-wise flip is done for 16-bit BGR words; i.e. instead of flipping
1701 each byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in *Figure 125*.

1702
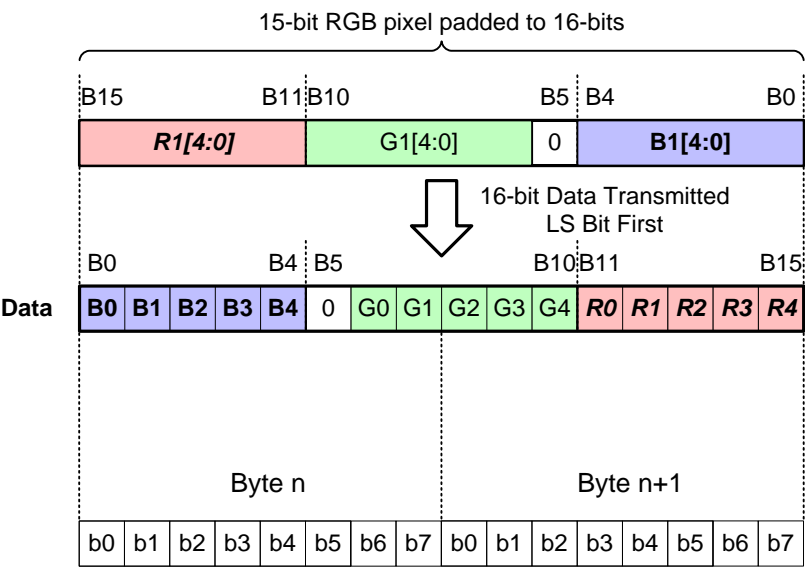
Figure 125 RGB555 Transmission on CSI-2 Bus Bitwise Illustration

11.3.5 RGB444

1703　RGB444 data can be transmitted over a CSI-2 bus with some special arrangements. The RGB444 data
1704　should be made to look like RGB565 data. This can be accomplished by inserting padding bits to the LSBs
1705　of each color component as illustrated in **Figure 126**.

1706　Both the frame format and the package size constraints are the same as the RGB565 case.

1707　Bit order in transmission follows the general CSI-2 rule, LSB first. In RGB444 case the length of one data
1708　word is 16-bits, not eight bits. The word-wise flip is done for 16-bit BGR words; i.e. instead of flipping
1709　each byte (8-bits), each two bytes (16-bits) are flipped. This is illustrated in **Figure 126**.

1710
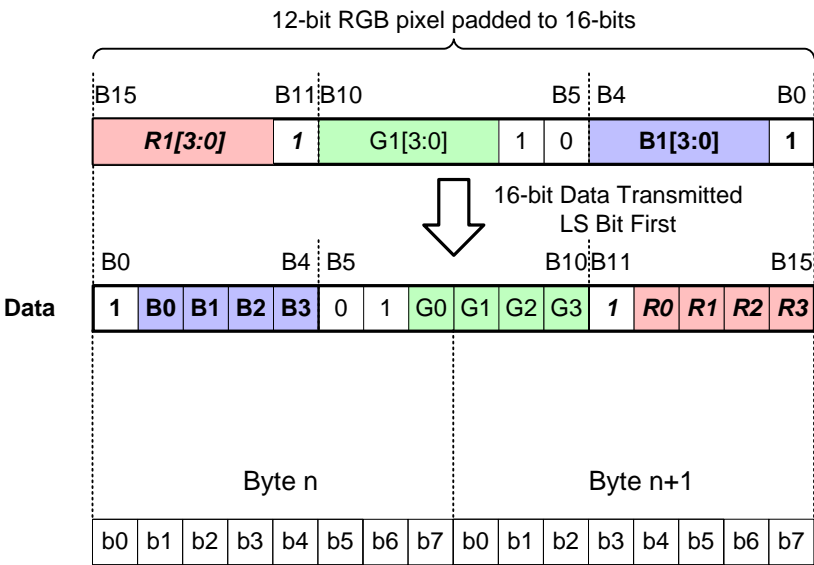
Figure 126 RGB444 Transmission on CSI-2 Bus Bitwise Illustration

11.4   RAW Image Data

1711   The RAW 6/7/8/10/12/14/16/20 modes are used for transmitting Raw image data from the image sensor.

1712   The intent is that Raw image data is unprocessed image data (i.e. Raw Bayer data) or complementary color
1713   data, but RAW image data is not limited to these data types.

1714   It is possible to transmit e.g. light shielded pixels in addition to effective pixels. This leads to a situation
1715   where the line length is longer than sum of effective pixels per line. The line length, if not specified
1716   otherwise, has to be a multiple of word (32 bits).

1717   *Table 36* defines the data type codes for RAW data formats described in this section.

1718   **Table 36 RAW Image Data Types**

Data Type	Description
0x28	RAW6
0x29	RAW7
0x2A	RAW8
0x2B	RAW10
0x2C	RAW12
0x2D	RAW14
0x2E	RAW16
0x2F	RAW20

11.4.1   RAW6

1719 The 6-bit Raw data transmission is done by transmitting the pixel data over CSI-2 bus. Each line is
1720 separated by line start / end synchronization codes. This sequence is illustrated in *Figure 127* (VGA case).
1721 *Table 37* specifies the packet size constraints for RAW6 packets. The length of each packet must be a
1722 multiple of the values in the table.

1723

Table 37 RAW6 Packet Data Size Constraints

Pixels	Bytes	Bits
4	3	24

1724 Each 6-bit pixel is sent LSB first. This is an exception to general CSI-2 rule byte wise LSB first.
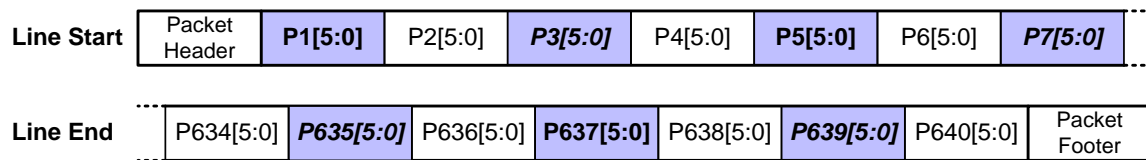
Figure 127 RAW6 Transmission

1726
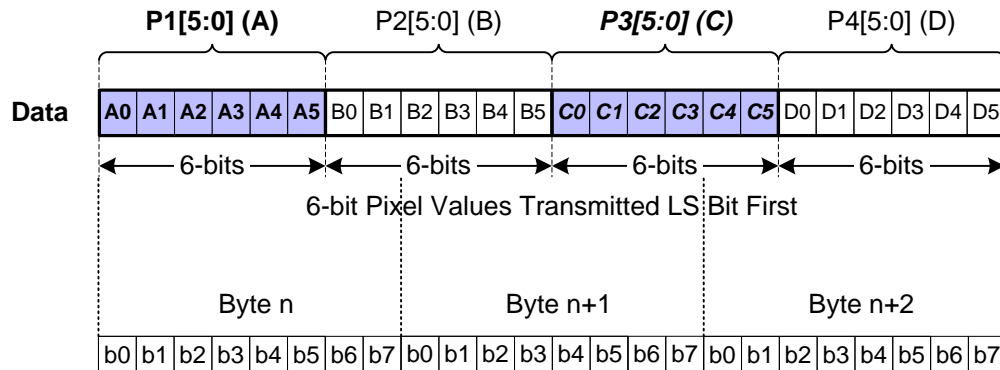
Figure 128 RAW6 Data Transmission on CSI-2 Bus Bitwise Illustration

1727
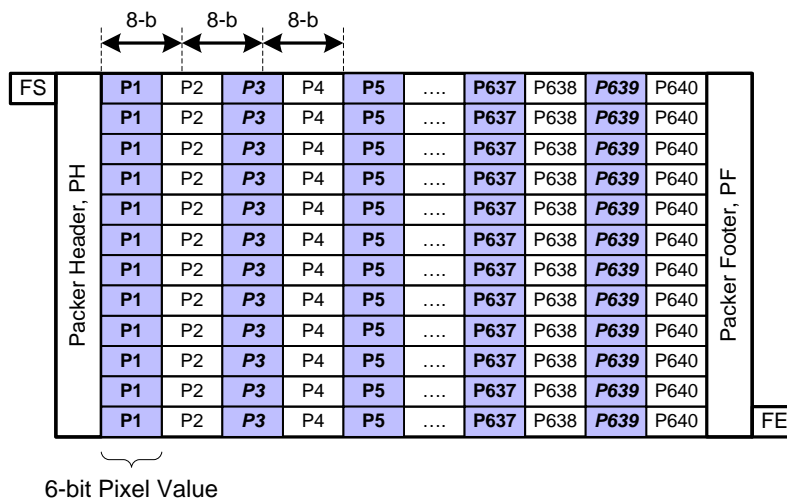
Figure 129 RAW6 Frame Format

11.4.2   RAW7   See Errata 01, Item 2

1728   The 7-bit Raw data transmission is done by transmitting the pixel data over CSI-2 bus. Each line is
1729   separated by line start / end synchronization codes. This sequence is illustrated in *Figure 130* (VGA case).
1730   Table 38 specifies the packet size constraints for RAW7 packets. The length of each packet must be a
1731   multiple of the values in the table.

1732   **Table 38 RAW7 Packet Data Size Constraints**

Pixels	Bytes	Bits
8	7	56

1733   Each 7-bit pixel is sent LSB first. This is an exception to general CSI-2 rule byte-wise LSB first.
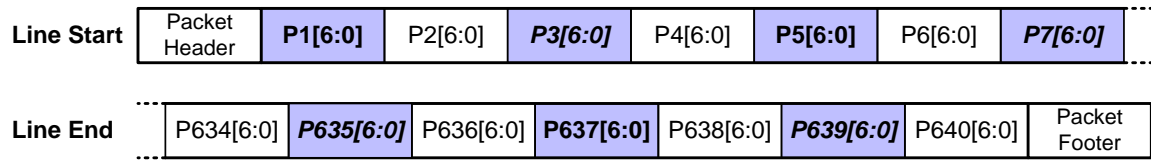
Figure 130 RAW7 Transmission

Figure 131 RAW7 Data Transmission on CSI-2 Bus Bitwise Illustration

Figure 132 RAW7 Frame Format

11.4.3   RAW8

1737 The 8-bit Raw data transmission is done by transmitting the pixel data over a CSI-2 bus. *Table 39* specifies
1738 the packet size constraints for RAW8 packets. The length of each packet must be a multiple of the values in
1739 the table.

1740 **Table 39 RAW8 Packet Data Size Constraints**

Pixels	Bytes	Bits
1	1	8

1741 This sequence is illustrated in *Figure 133* (VGA case).

1742 Bit order in transmission follows the general CSI-2 rule, LSB first.

1743
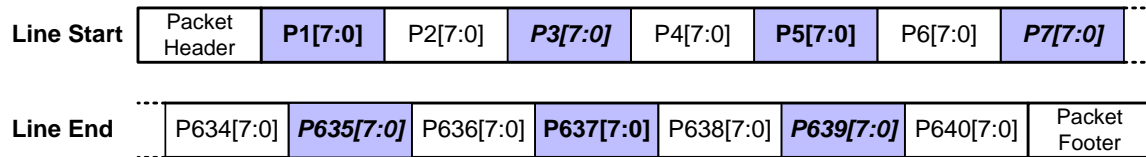
Figure 133 RAW8 Transmission

1744
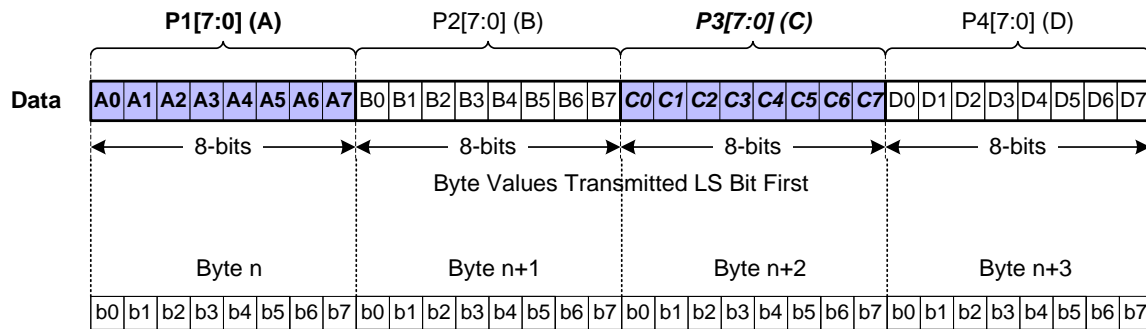
Figure 134 RAW8 Data Transmission on CSI-2 Bus Bitwise Illustration

1745
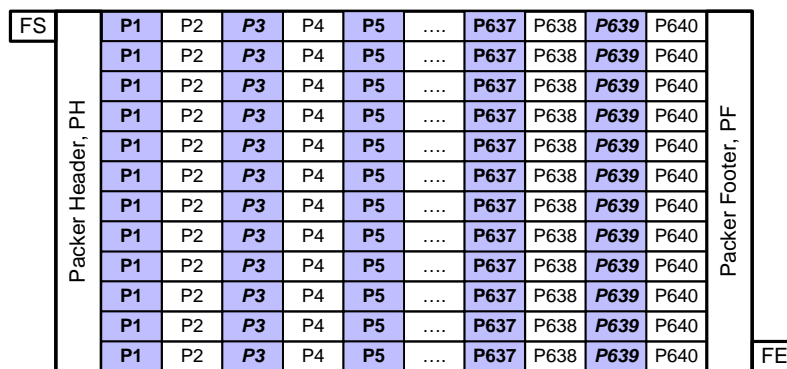
Figure 135 RAW8 Frame Format

11.4.4 RAW10

1746 The transmission of 10-bit Raw data is done by packing the 10-bit pixel data to look like 8-bit data format.
1747 *Table 40* specifies the packet size constraints for RAW10 packets. The length of each packet must be a
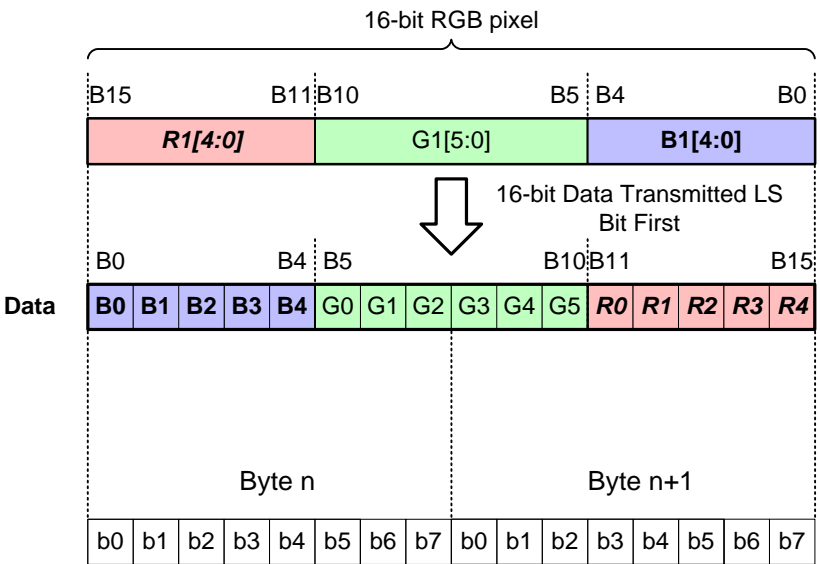1748 multiple of the values in the table.

1749 **Table 40 RAW10 Packet Data Size Constraints**

Pixels	Bytes	Bits
4	5	40

1750 This sequence is illustrated in *Figure 136* (VGA case).

1751 Bit order in transmission follows the general CSI-2 rule: LSB first.
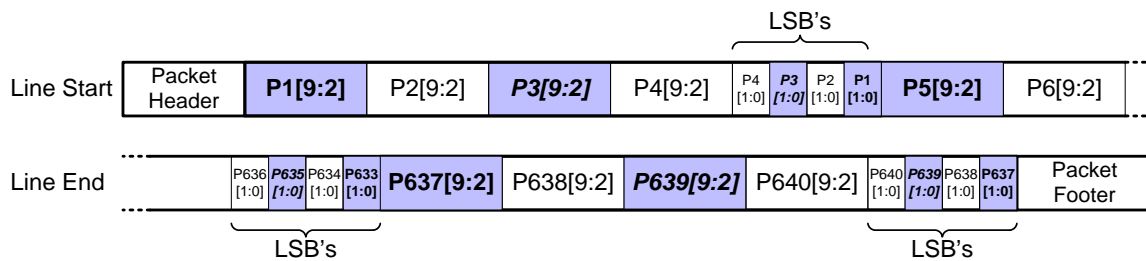
Figure 136 RAW10 Transmission
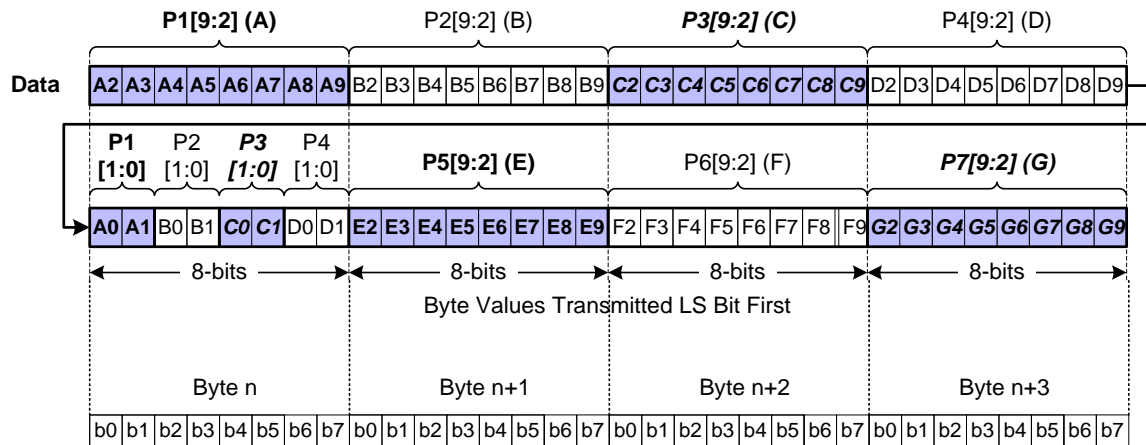
Figure 137 RAW10 Data Transmission on CSI-2 Bus Bitwise Illustration

	P1	P2	*P3*	P4	**LSBs**	**P5**	....	**P637**	P638	*P639*	P640	**LSBs**	
	P1	P2	*P3*	P4	**LSBs**	**P5**	....	**P637**	P638	*P639*	P640	**LSBs**	
	P1	P2	*P3*	P4	**LSBs**	**P5**	....	**P637**	P638	*P639*	P640	**LSBs**	
	P1	P2	*P3*	P4	**LSBs**	**P5**	....	**P637**	P638	*P639*	P640	**LSBs**	
	P1	P2	*P3*	P4	**LSBs**	**P5**	....	**P637**	P638	*P639*	P640	**LSBs**	
	P1	P2	*P3*	P4	**LSBs**	**P5**	....	**P637**	P638	*P639*	P640	**LSBs**	
	P1	P2	*P3*	P4	**LSBs**	**P5**	....	**P637**	P638	*P639*	P640	**LSBs**	
	P1	P2	*P3*	P4	**LSBs**	**P5**	....	**P637**	P638	*P639*	P640	**LSBs**	
	P1	P2	*P3*	P4	**LSBs**	**P5**	....	**P637**	P638	*P639*	P640	**LSBs**	
	P1	P2	*P3*	P4	**LSBs**	**P5**	....	**P637**	P638	*P639*	P640	**LSBs**	
	P1	P2	*P3*	P4	**LSBs**	**P5**	....	**P637**	P638	*P639*	P640	**LSBs**	
	P1	P2	*P3*	P4	**LSBs**	**P5**	....	**P637**	P638	*P639*	P640	**LSBs**	

FS — Packer Header, PH … Packer Footer, PF — FE

1754

Figure 138 RAW10 Frame Format

11.4.5 RAW12

1755 The transmission of 12-bit Raw data is done by packing the 12-bit pixel data to look like 8-bit data format.
1756 *Table 41* specifies the packet size constraints for RAW12 packets. The length of each packet must be a
1757 multiple of the values in the table.

1758 **Table 41 RAW12 Packet Data Size Constraints**

Pixels	Bytes	Bits
2	3	24

1759 This sequence is illustrated in *Figure 139* (VGA case).

1760 Bit order in transmission follows the general CSI-2 rule: LSB first.
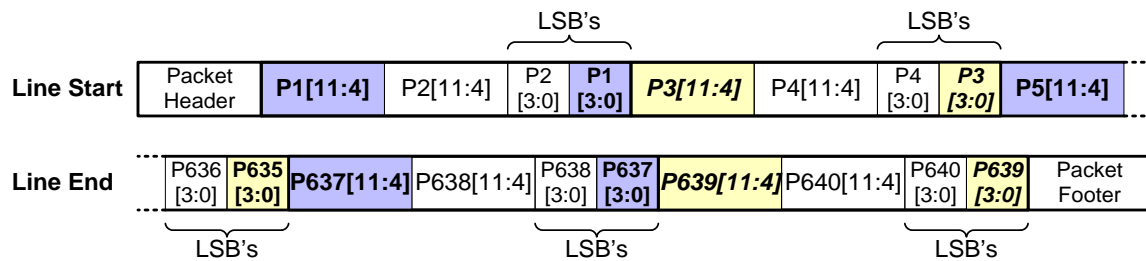
1761 **Figure 139 RAW12 Transmission**
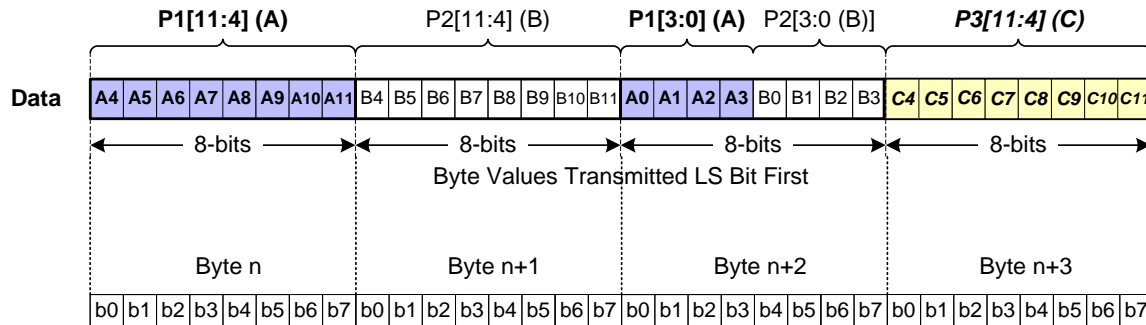
1762 **Figure 140 RAW12 Transmission on CSI-2 Bus Bitwise Illustration**
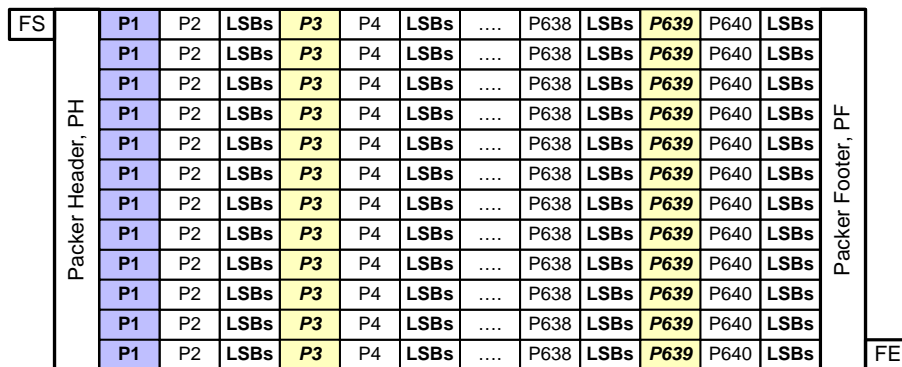
1763 **Figure 141 RAW12 Frame Format**

11.4.6   RAW14

1764   The transmission of 14-bit Raw data is done by packing the 14-bit pixel data in 8-bit slices. For every four
1765   pixels, seven bytes of data is generated. *Table 42* specifies the packet size constraints for RAW14 packets.
1766   The length of each packet must be a multiple of the values in the table.

1767   **Table 42 RAW14 Packet Data Size Constraints**

Pixels	Bytes	Bits
4	7	56

1768   The sequence is illustrated in *Figure 142* (VGA case).

1769   The LS bits for P1, P2, P3, and P4 are distributed in three bytes as shown in *Figure 142* and *Figure 143*.
1770   The same is true for the LS bits for P637, P638, P639, and P640. The bit order during byte transmission
1771   follows the general CSI-2 rule, i.e. LSB first.

1772   **Note:**

1773   *Figure 142 has been modified relative to the figures shown in the CSI-2 Specification version 2.0*
1774   *and earlier, in order to more clearly correspond with Figure 143. The RAW14 byte packing and*
1775   *transmission formats themselves have not changed relative to earlier CSI-2 Specification versions.*

1776
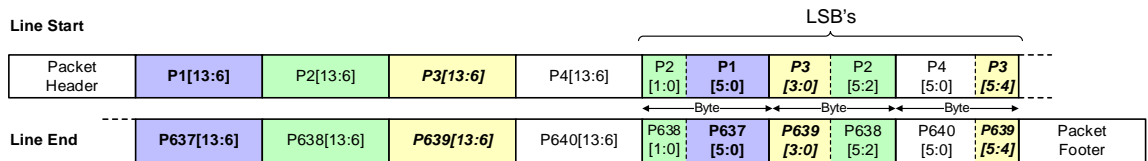
Figure 142 RAW14 Transmission

1777

Figure 143 RAW14 Transmission on CSI-2 Bus Bitwise Illustration
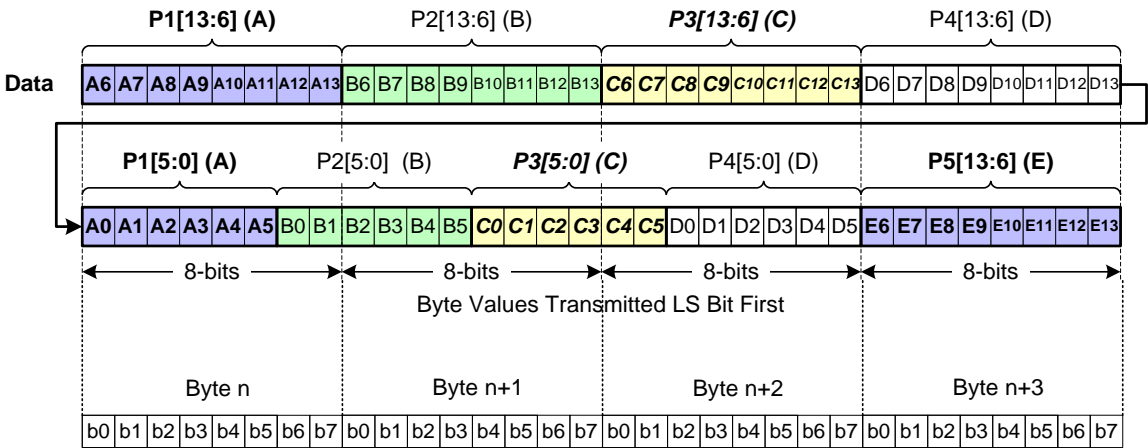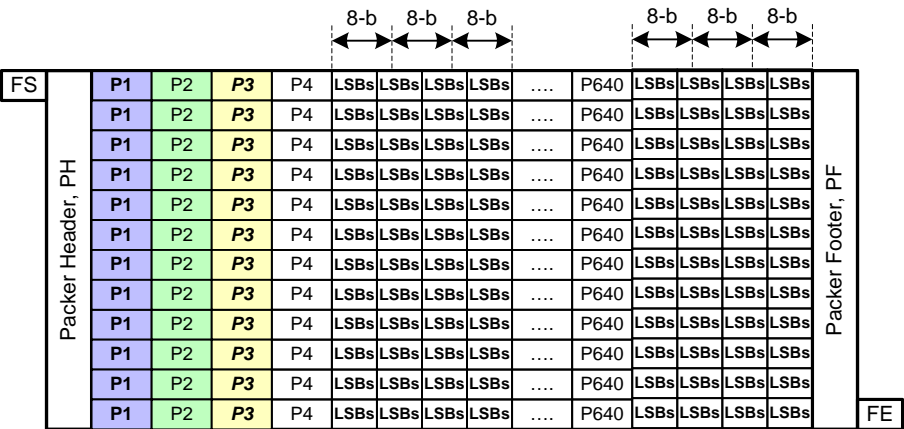
Figure 144 RAW14 Frame Format

1778

11.4.7 RAW16

1779 The transmission of 16-bit Raw data is done by packing the 16-bit pixel data to look like the 8-bit data
1780 format. *Table 43* specifies the packet size constraints for RAW16 packets. The length of each packet must
1781 be a multiple of the values in the table.

1782 **Table 43 RAW16 Packet Data Size Constraints**

Pixels	Bytes	Bits
1	2	16

1783 This sequence is illustrated in *Figure 145* (VGA case).

1784 Bit order in transmission follows the general CSI-2 rule: LSB first.

1785
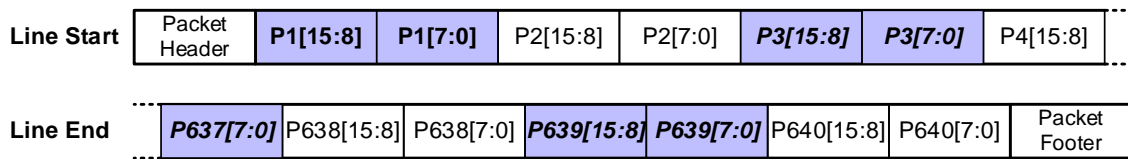
Figure 145 RAW16 Transmission

1786
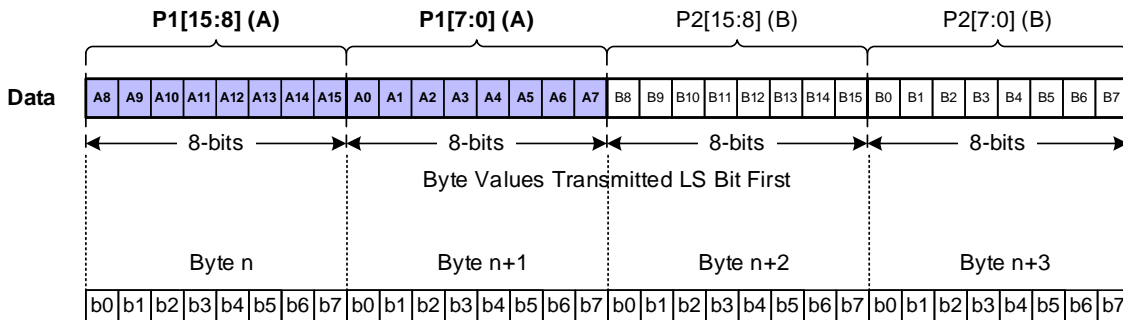
Figure 146 RAW16 Transmission on CSI-2 Bus Bitwise Illustration

1787
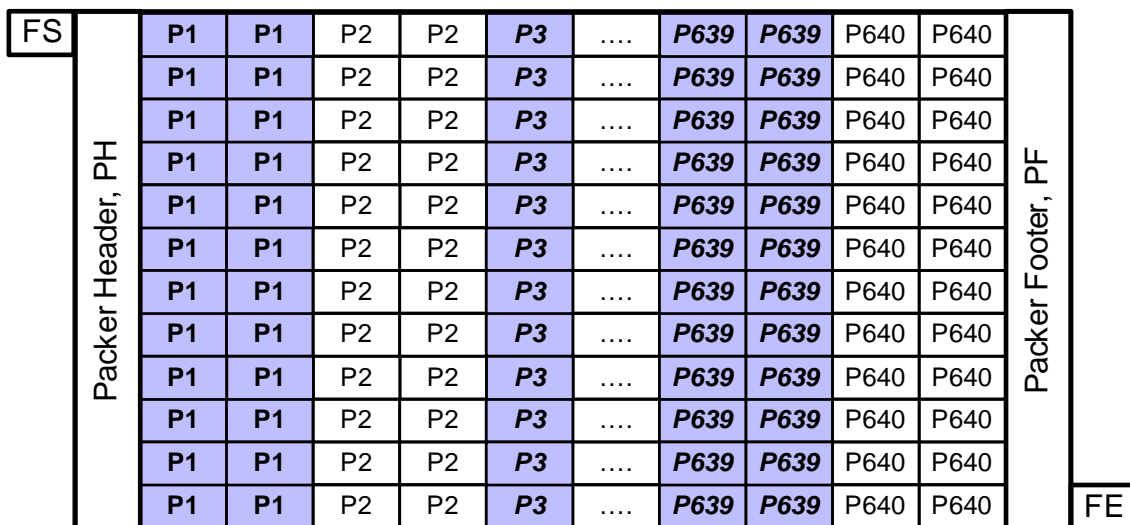
Figure 147 RAW16 Frame Format

11.4.8   RAW20

1788 The transmission of 20-bit Raw data is done by packing the 20-bit pixel data to look like the 10-bit data
1789 format. *Table 44* specifies the packet size constraints for RAW20 packets. The length of each packet must
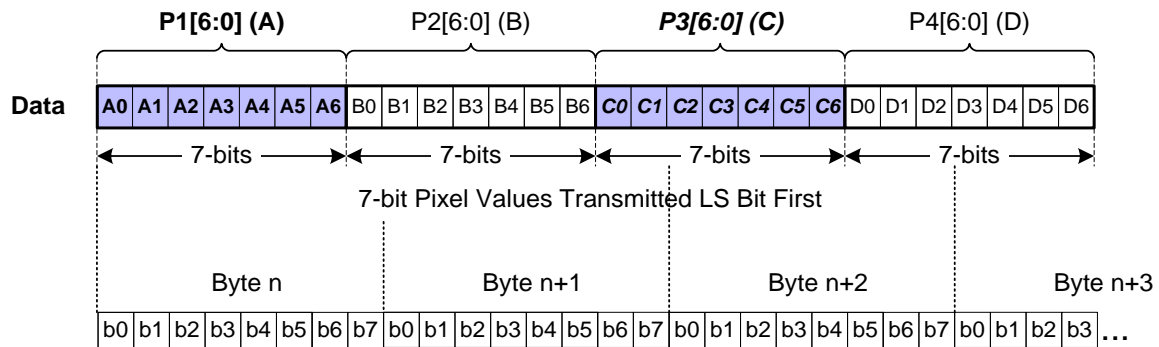1790 be a multiple of the values in the table.

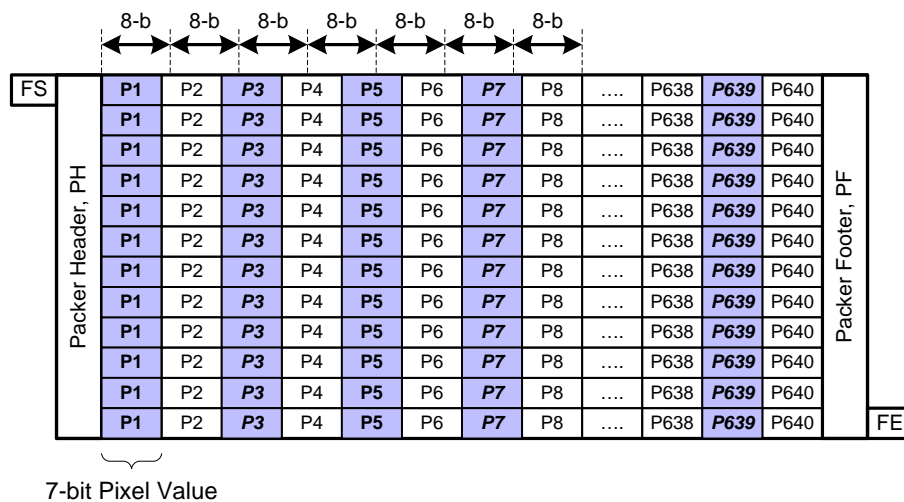1791 **Table 44 RAW20 Packet Data Size Constraints**

Pixels	Bytes	Bits
2	5	40

1792 This sequence is illustrated in *Figure 148* (VGA case).

1793 Bit order in transmission follows the general CSI-2 rule: LSB first.

1794
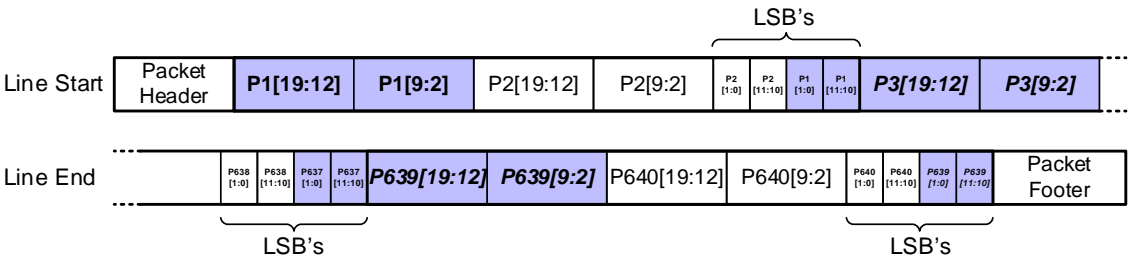
Figure 148 RAW20 Transmission

1795
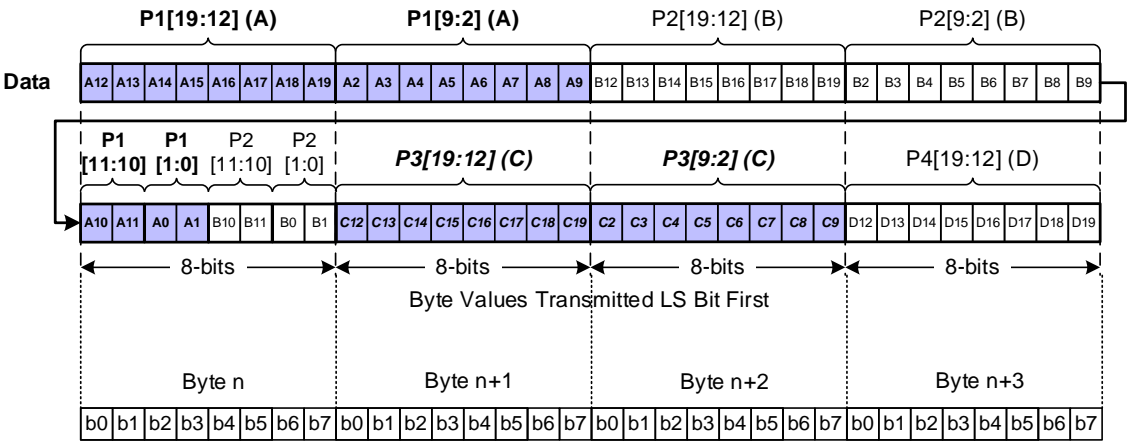
Figure 149 RAW20 Transmission on CSI-2 Bus Bitwise Illustration
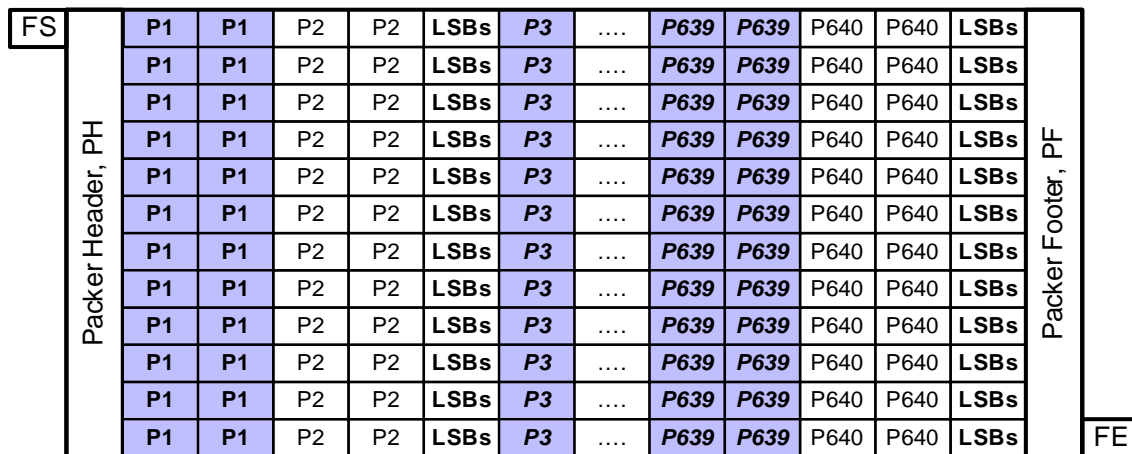
Figure 150 RAW20 Frame Format

11.5   User Defined Data Formats

1797 The User Defined Data Type values shall be used to transmit arbitrary data, such as JPEG and MPEG4
1798 data, over the CSI-2 bus. Data shall be packed so that the data length is divisible by eight bits. If data
1799 padding is required, the padding shall be added before data is presented to the CSI-2 protocol interface.

1800 Bit order in transmission follows the general CSI-2 rule, LSB first.

1801
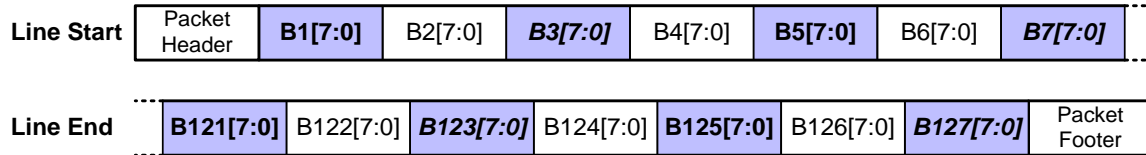
Figure 151 User Defined 8-bit Data (128 Byte Packet)

1802
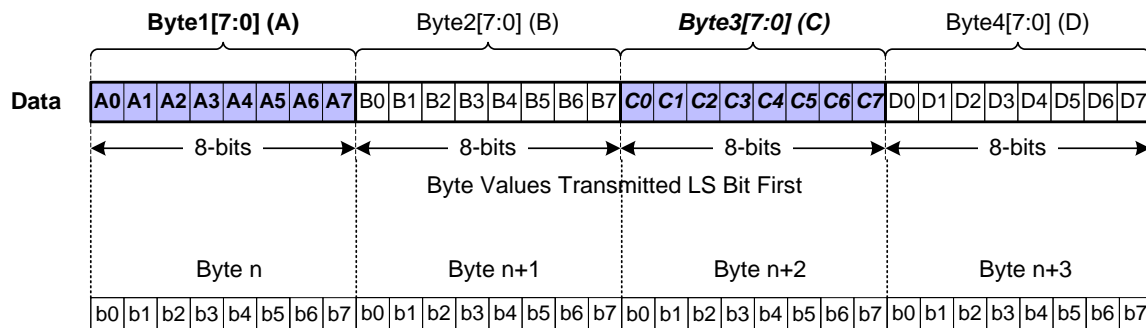
Figure 152 User Defined 8-bit Data Transmission on CSI-2 Bus Bitwise Illustration

1803 The packet data size in bits shall be divisible by eight, i.e. a whole number of bytes shall be transmitted.

1804 For User Defined data:

1805 • The frame is transmitted as a sequence of arbitrary sized packets.

1806 • The packet size may vary from packet to packet.

1807 • The packet spacing may vary between packets.

1808

KEY:
SoT – Start of Transmission   EoT – End of Transmission   LPS – Low Power State
PH – Packet Header   PF – Packet Footer
FS – Frame Start   FE – Frame End
                                      LE – Line End
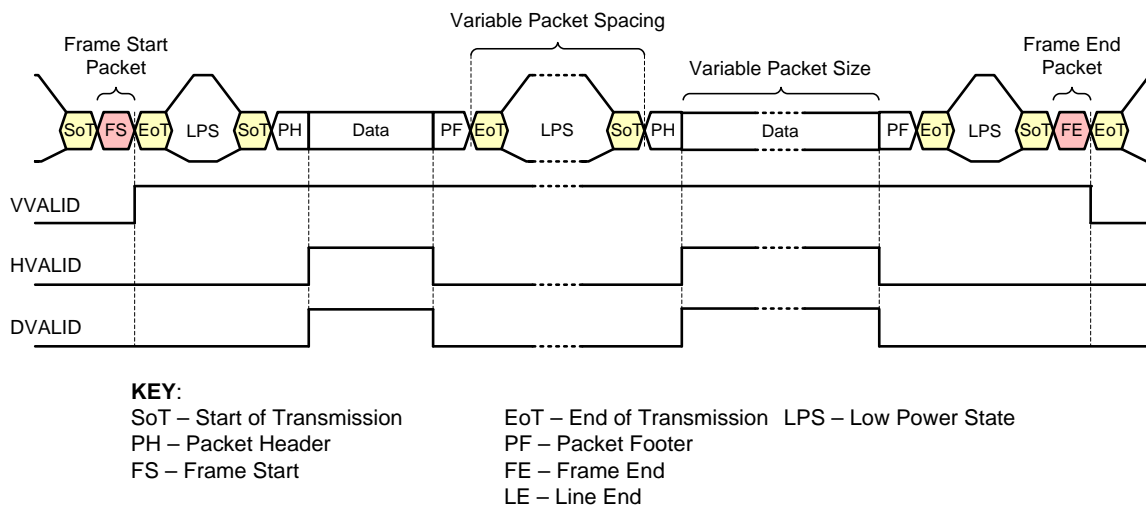
Figure 153 Transmission of User Defined 8-bit Data

1809   Eight different User Defined data type codes are available as shown in *Table 45*.

1810   **Table 45 User Defined 8-bit Data Types**

Data Type	Description
0x30	User Defined 8-bit Data Type 1
0x31	User Defined 8-bit Data Type 2
0x32	User Defined 8-bit Data Type 3
0x33	User Defined 8-bit Data Type 4
0x34	User Defined 8-bit Data Type 5
0x35	User Defined 8-bit Data Type 6
0x36	User Defined 8-bit Data Type 7
0x37	User Defined 8-bit Data Type 8

12   Recommended Memory Storage

1811   This section is informative.

1812   The CSI-2 data protocol requires certain behavior from the receiver connected to the CSI transmitter. The
1813   following sections describe how different data formats should be stored inside the receiver. While
1814   informative, this section is provided to ease application software development by suggesting a common
1815   data storage format among different receivers.

12.1   General/Arbitrary Data Reception

1816   In the generic case and for arbitrary data the first byte of payload data transmitted maps the LS byte of the
1817   32-bit memory word and the fourth byte of payload data transmitted maps to the MS byte of the 32-bit
1818   memory word.

1819   *Figure 154* shows the generic CSI-2 byte to 32-bit memory word mapping rule.
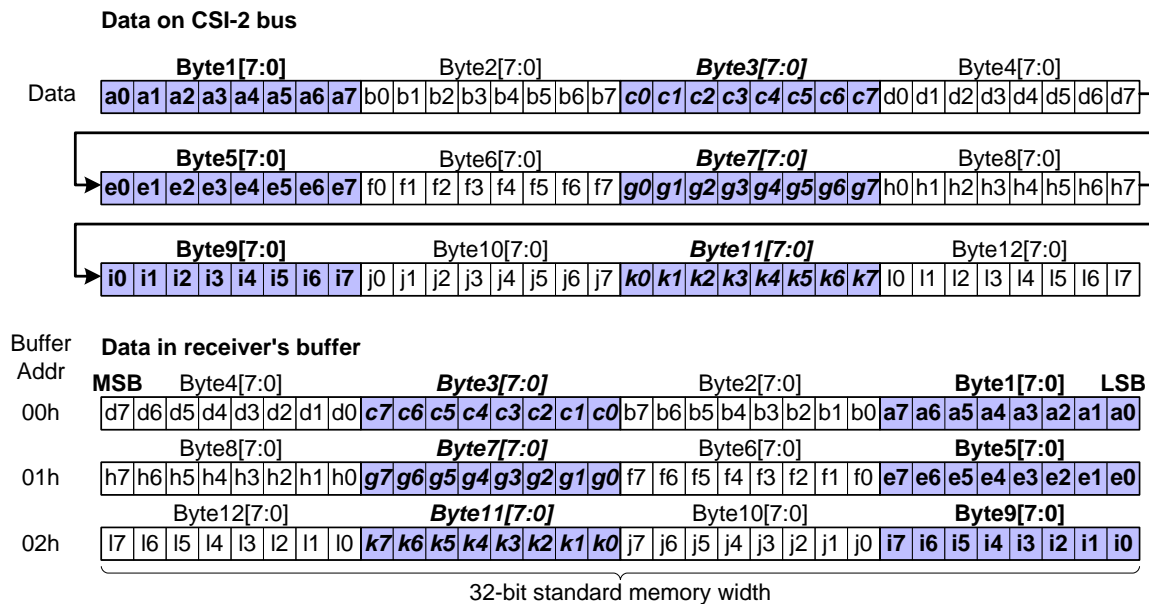
1820
Figure 154 General/Arbitrary Data Reception

12.2 RGB888 Data Reception

1821 The RGB888 data format byte to 32-bit memory word mapping follows the generic CSI-2 rule.

1822
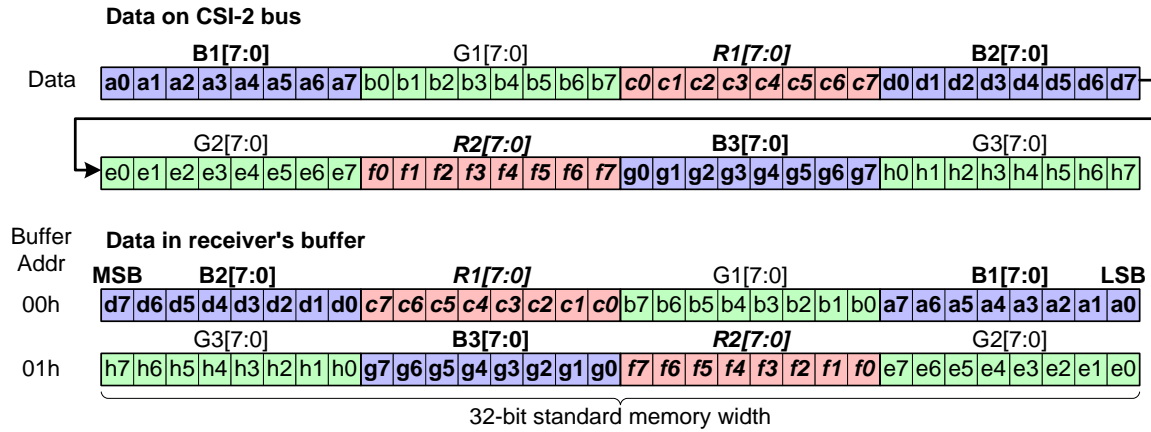
Figure 155 RGB888 Data Format Reception

12.3 RGB666 Data Reception

1823
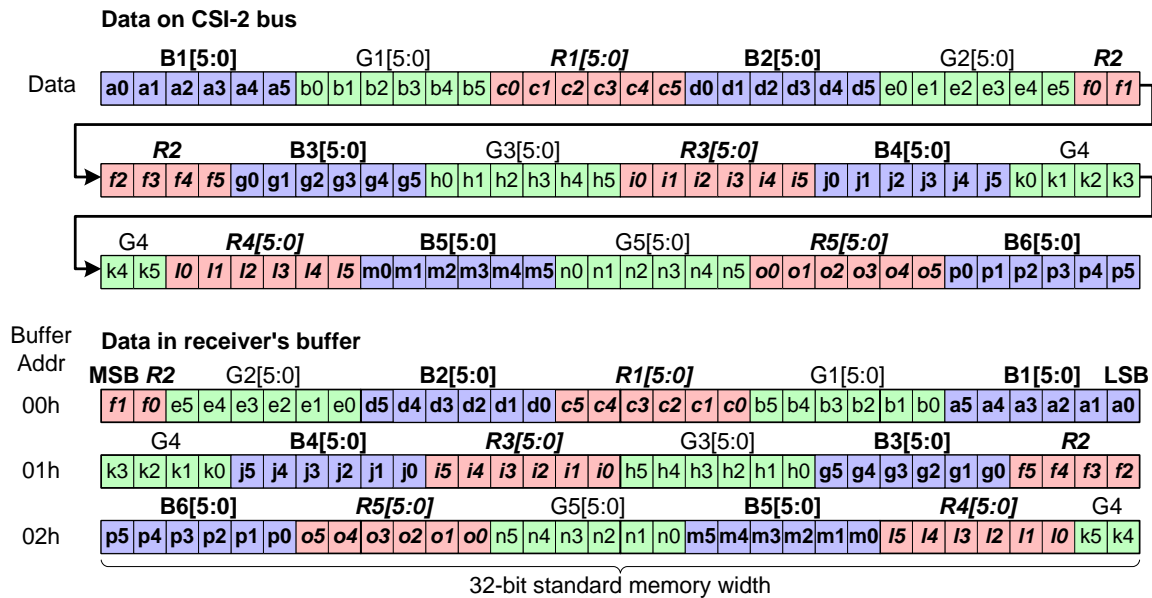
Figure 156 RGB666 Data Format Reception

12.4 RGB565 Data Reception

Data on CSI-2 bus
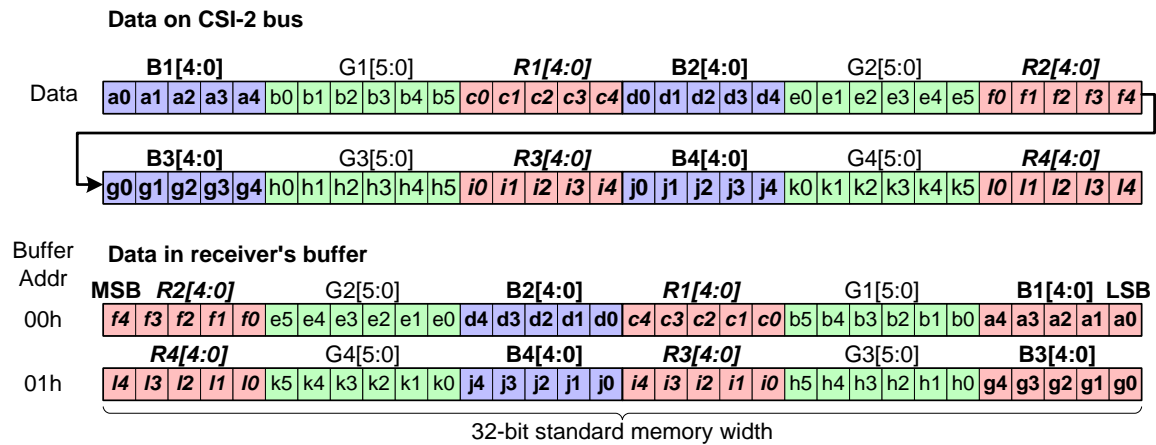
Figure 157 RGB565 Data Format Reception

12.5 RGB555 Data Reception

Data on CSI-2 bus
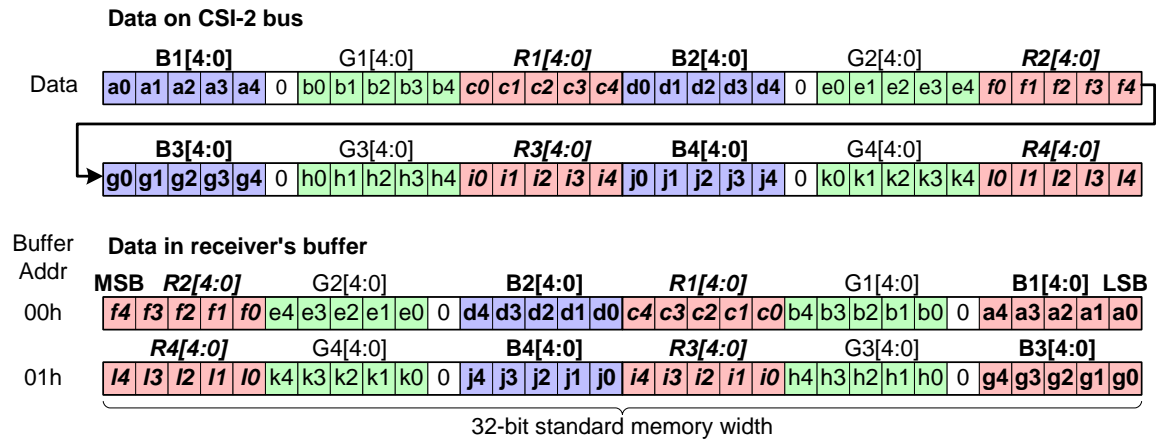
Figure 158 RGB555 Data Format Reception

12.6   RGB444 Data Reception

1826
1827

The RGB444 data format byte to 32-bit memory word mapping has a special transform as shown in *Figure 159*.

1828
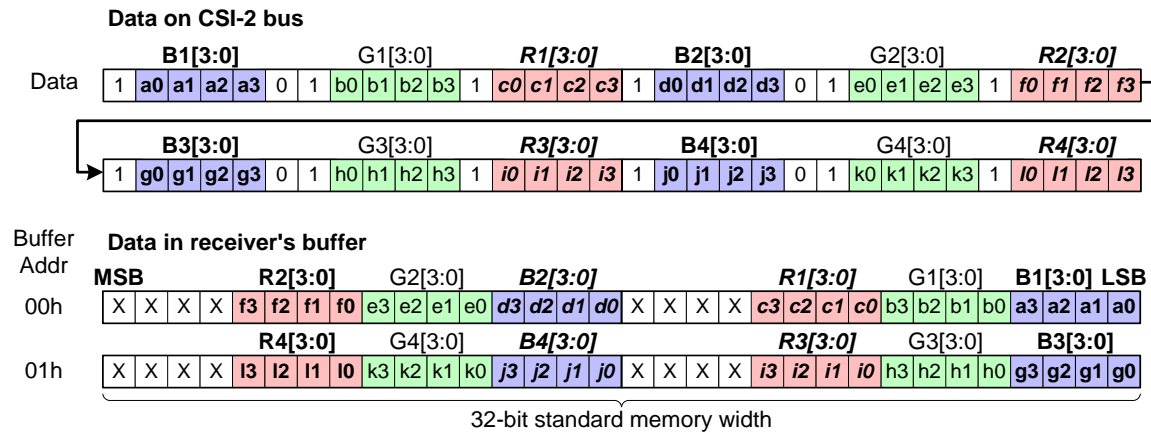
Figure 159 RGB444 Data Format Reception

12.7   YUV422 8-bit Data Reception

1829
1830

The YUV422 8-bit data format the byte to 32-bit memory word mapping does not follow the generic CSI-2 rule.

1831
1832
1833

For YUV422 8-bit data format the first byte of payload data transmitted maps the MS byte of the 32-bit memory word and the fourth byte of payload data transmitted maps to the LS byte of the 32-bit memory word.

1834
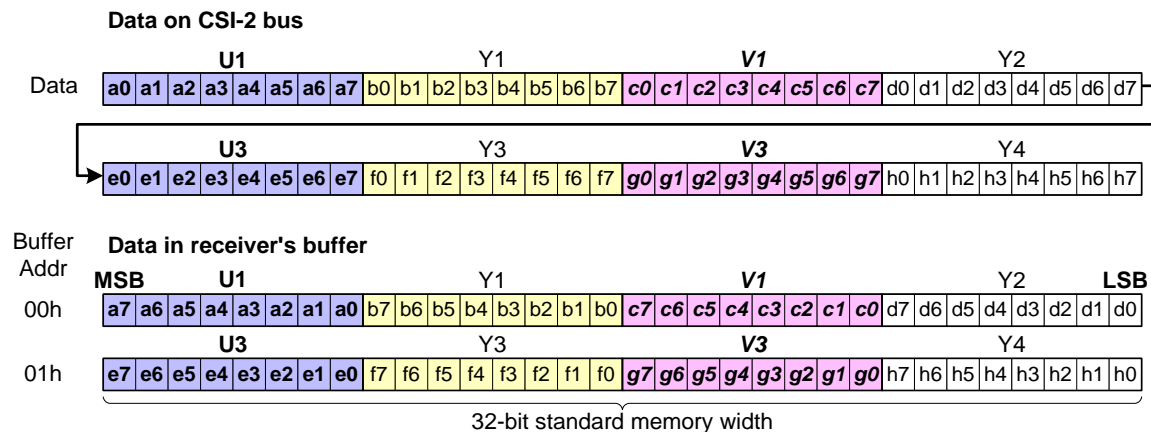
Figure 160 YUV422 8-bit Data Format Reception

12.8   YUV422 10-bit Data Reception

1835   The YUV422 10-bit data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

1836
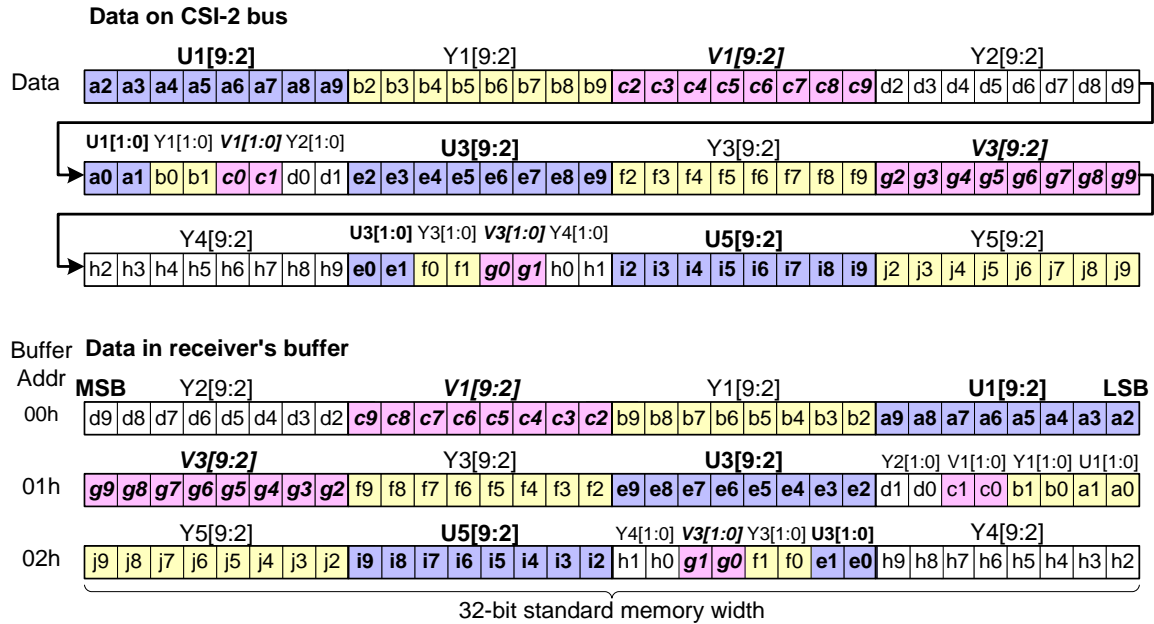
Figure 161 YUV422 10-bit Data Format Reception

12.9   YUV420 8-bit (Legacy) Data Reception

1837   The YUV420 8-bit (legacy) data format the byte to 32-bit memory word mapping does not follow the
1838   generic CSI-2 rule.

1839   For YUV422 8-bit (legacy) data format the first byte of payload data transmitted maps the MS byte of the
1840   32-bit memory word and the fourth byte of payload data transmitted maps to the LS byte of the 32-bit
1841   memory word.

1842

Figure 162 YUV420 8-bit Legacy Data Format Reception

12.10 YUV420 8-bit Data Reception

1843   The YUV420 8-bit data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

1844
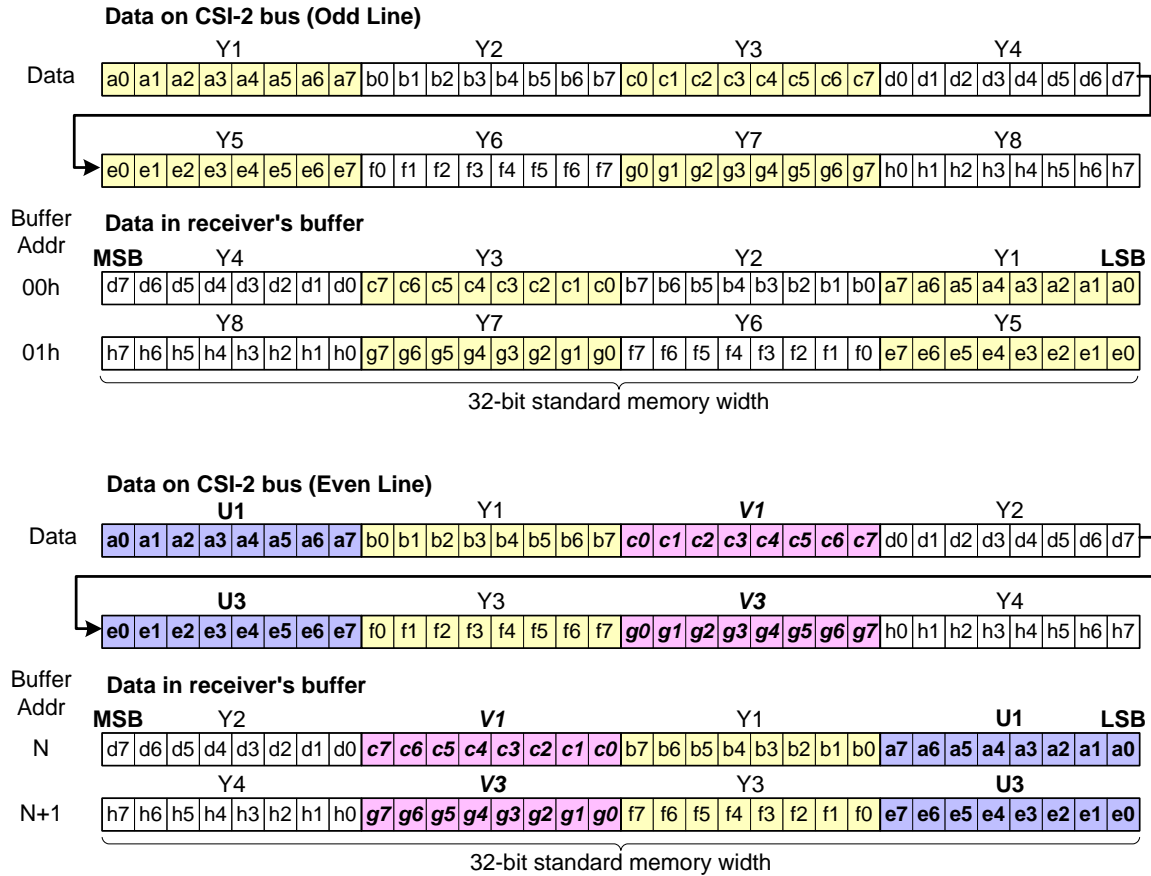
Figure 163 YUV420 8-bit Data Format Reception

12.11 YUV420 10-bit Data Reception

1845 The YUV420 10-bit data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.
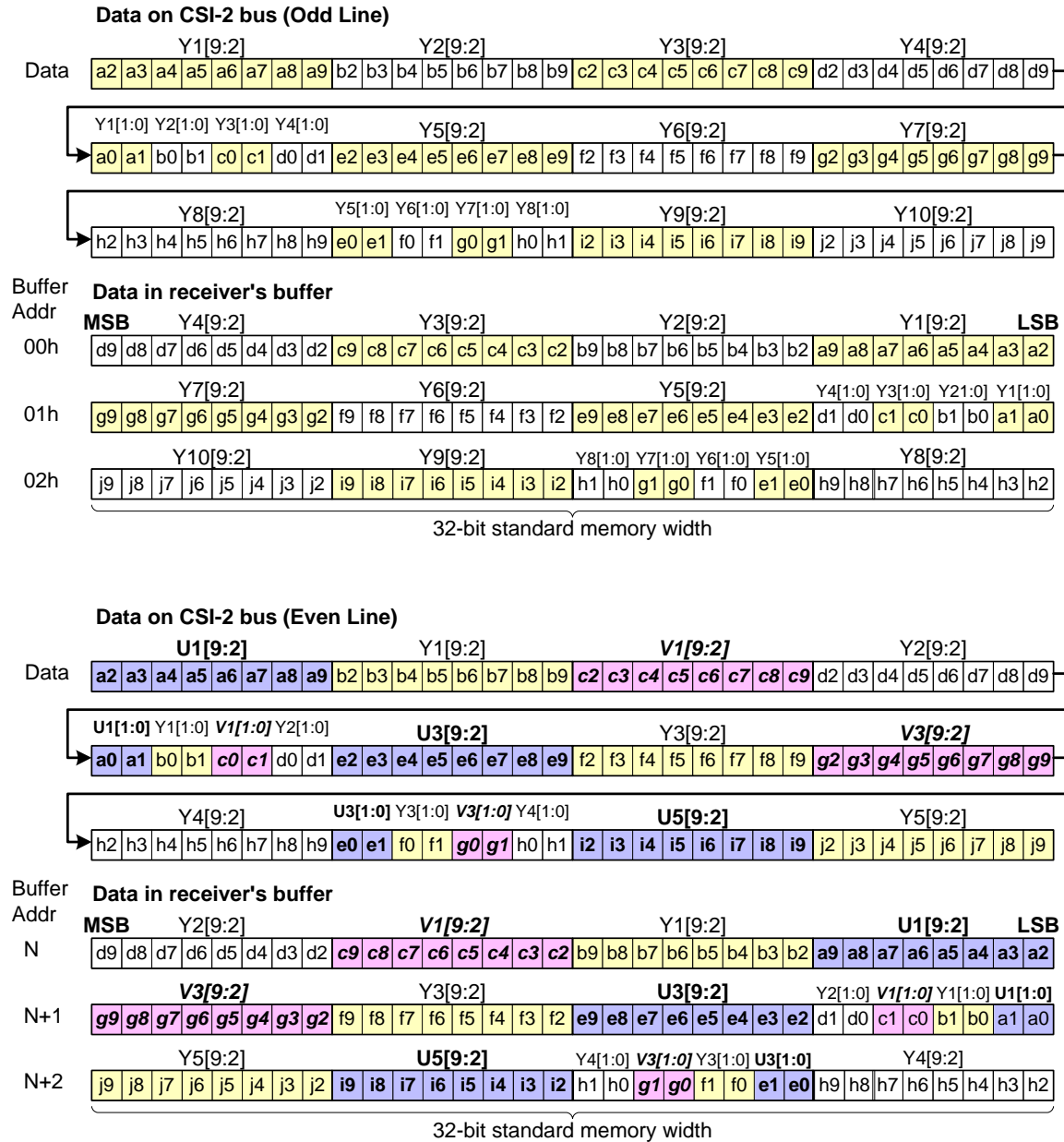
1846

Figure 164 YUV420 10-bit Data Format Reception

12.12 RAW6 Data Reception

Data on CSI-2 bus
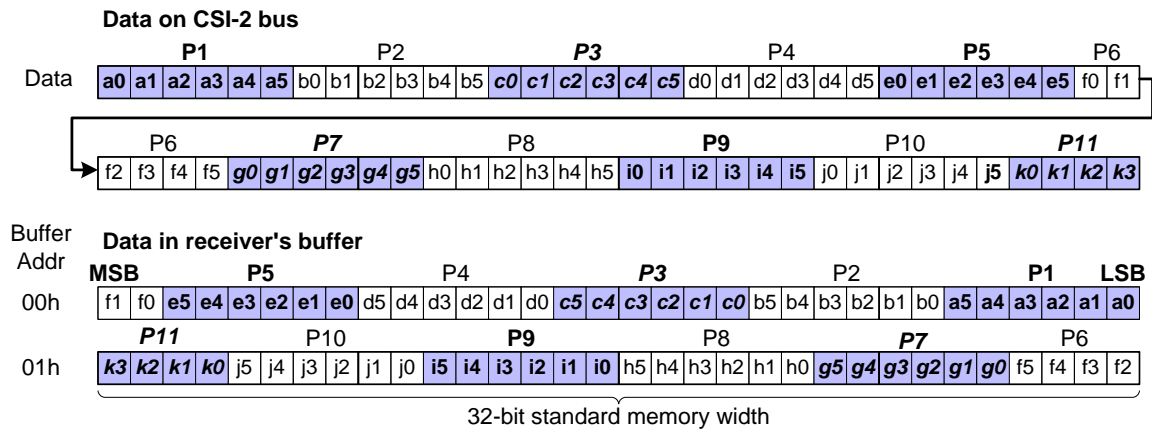
Figure 165 RAW6 Data Format Reception

12.13 RAW7 Data Reception

Data on CSI-2 bus
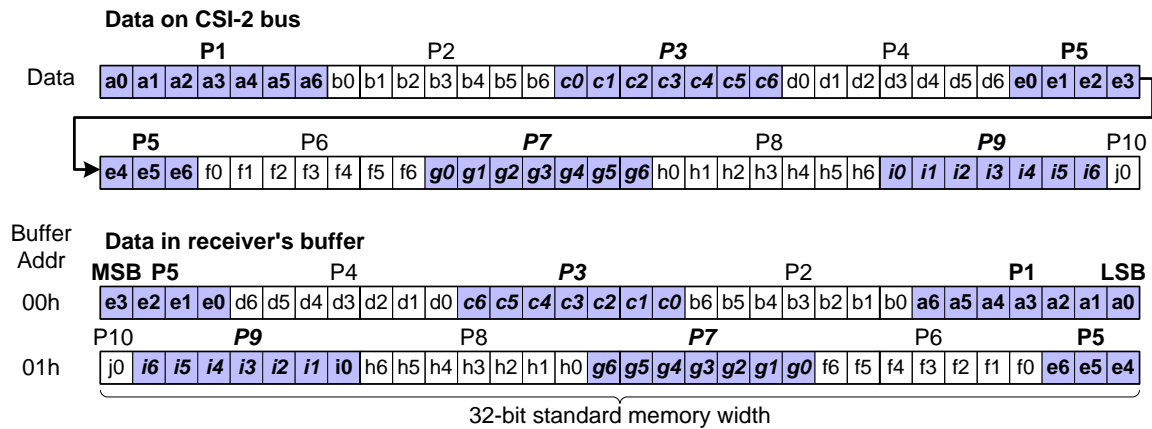
Figure 166 RAW7 Data Format Reception

12.14 RAW8 Data Reception

1849   The RAW8 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

1850
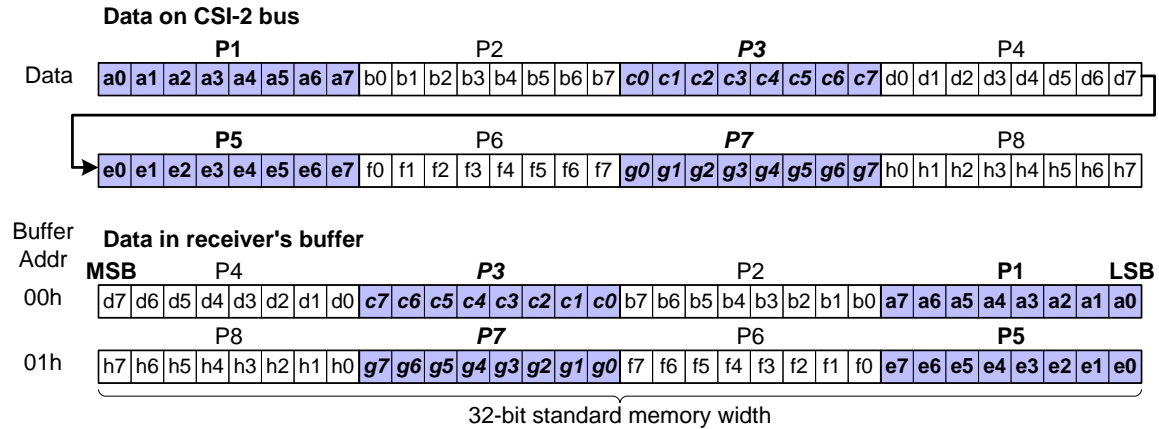
Figure 167 RAW8 Data Format Reception

12.15 RAW10 Data Reception

1851   The RAW10 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

1852
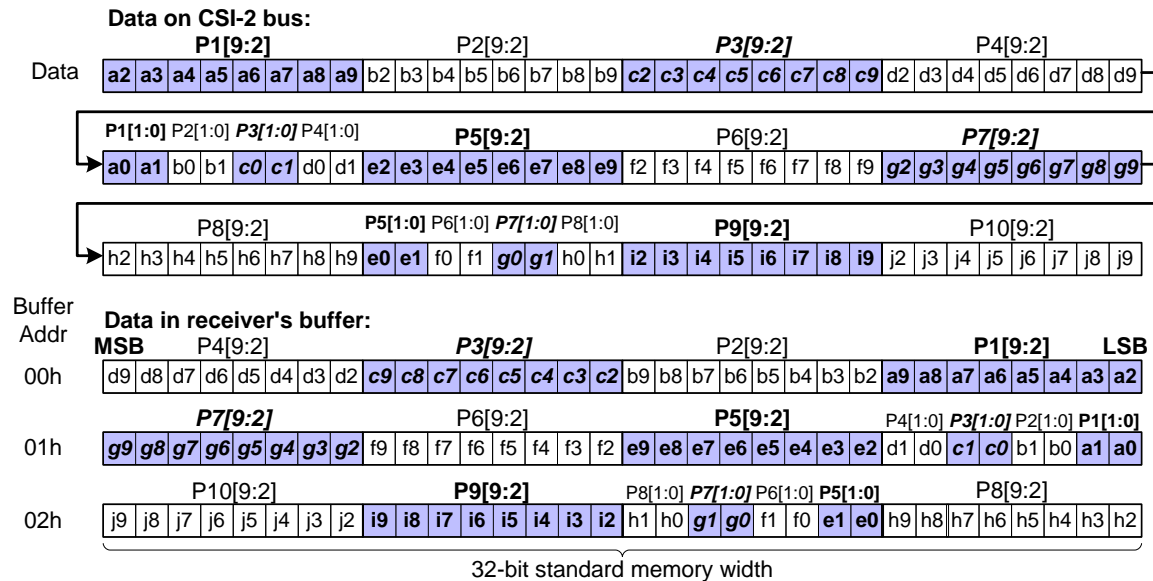
Figure 168 RAW10 Data Format Reception

12.16 RAW12 Data Reception

1853 The RAW12 data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.

1854
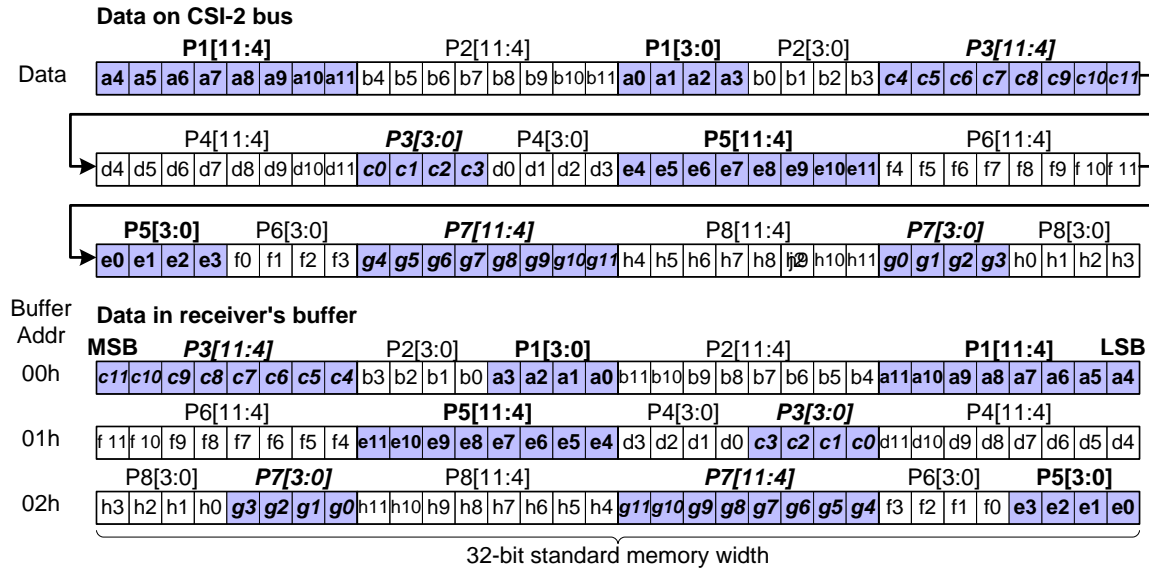
Figure 169 RAW12 Data Format Reception

12.17 RAW14 Data Reception

1855
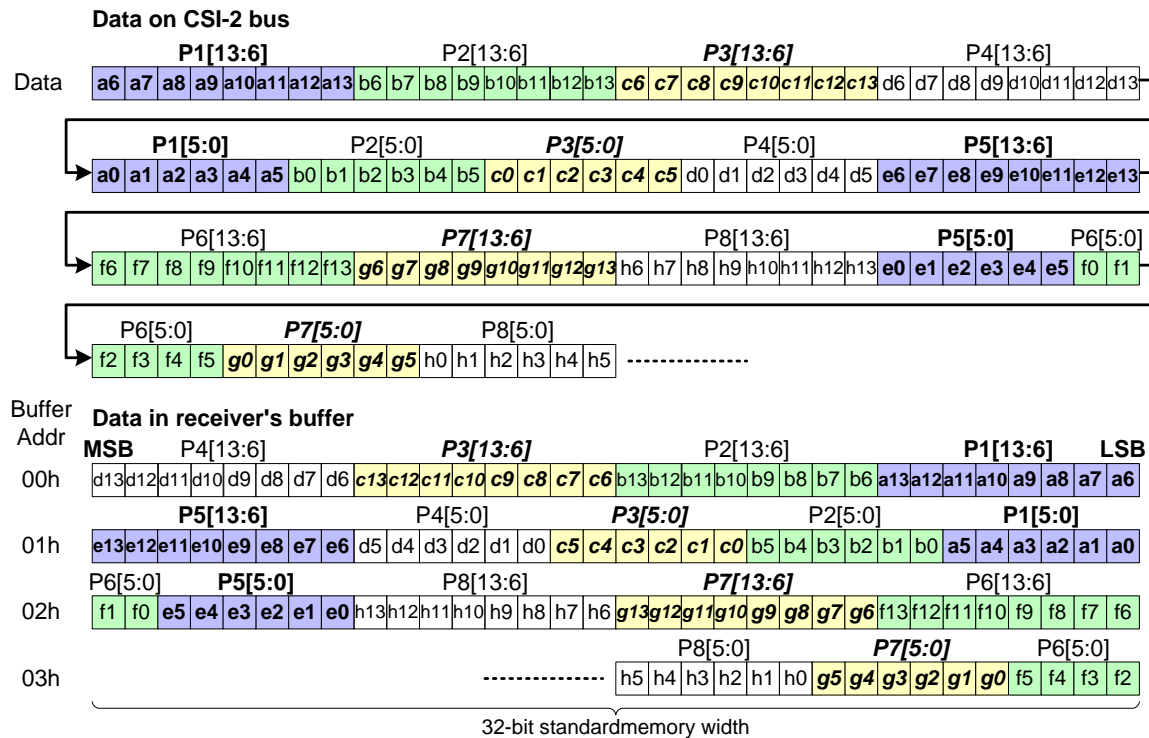
Figure 170 RAW 14 Data Format Reception

12.18 RAW16 Data Reception

1856   The RAW16 data format byte to 32-bit memory word mapping follows the generic CSI-2 rule.

1857
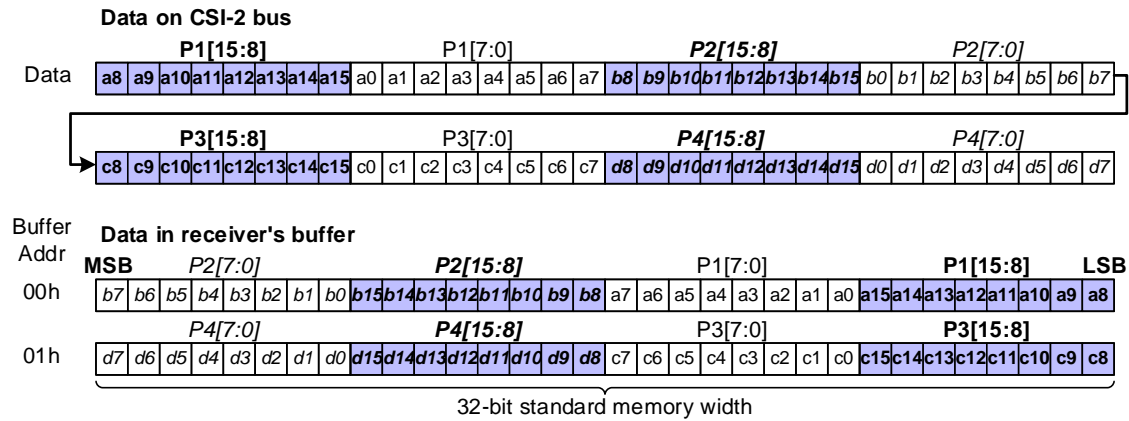
Figure 171 RAW16 Data Format Reception

12.19 RAW20 Data Reception

1858   The RAW20 data format byte to 32-bit memory word mapping follows the generic CSI-2 rule.

1859
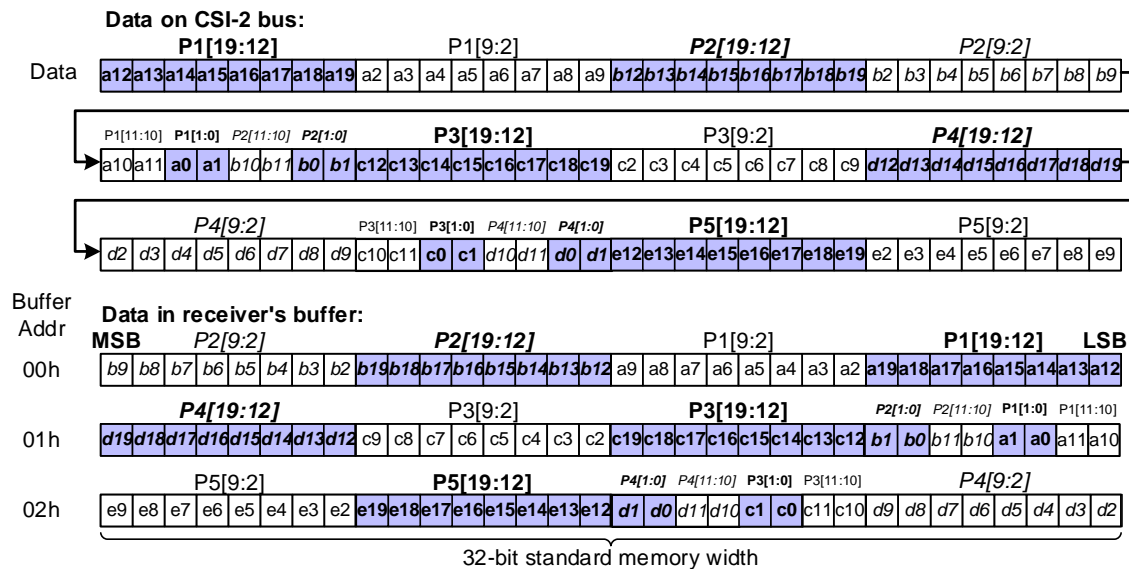
Figure 172 RAW20 Data Format Reception

Annex A JPEG8 Data Format (informative)

A.1   Introduction

This Annex contains an informative example of the transmission of compressed image data format using the arbitrary Data Type values.

JPEG8 has two non-standard extensions:

- Status information (mandatory)
- Embedded Image information e.g. a thumbnail image (optional)

Any non-standard or additional data inside the baseline JPEG data structure has to be removed from JPEG8 data before it is compliant with e.g. standard JPEG image viewers in e.g. a personal computer.

The JPEG8 data flow is illustrated in *Figure 173* and *Figure 174*.
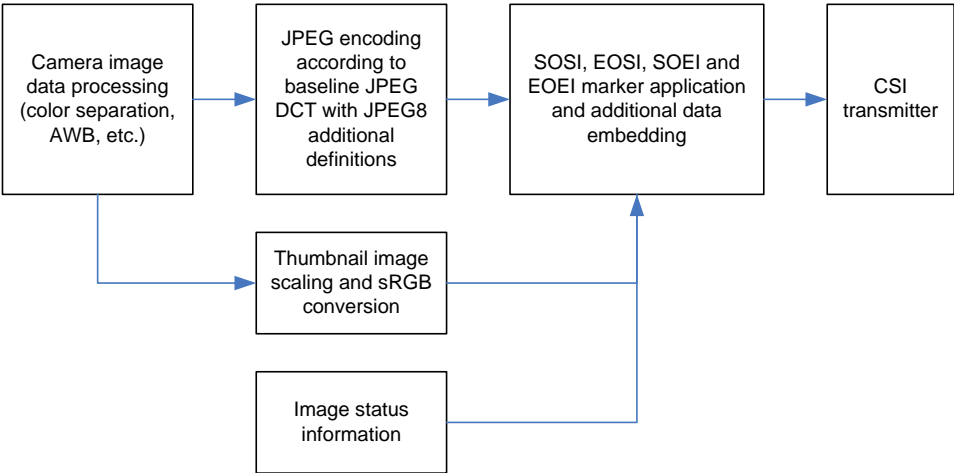
Figure 173 JPEG8 Data Flow in the Encoder
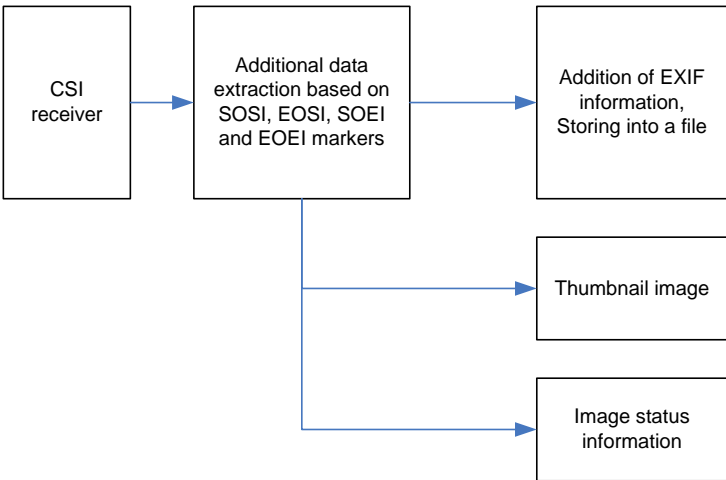
Figure 174 JPEG8 Data Flow in the Decoder

A.2   JPEG Data Definition

1870   The JPEG data generated in camera module is baseline JPEG DCT format defined in ISO/IEC 10918-1,
1871   with following additional definitions or modifications:

1872   • sRGB color space shall be used. The JPEG is generated from YCbCr format after sRGB to YCbCr
1873   conversion.

1874   • The JPEG metadata has to be EXIF compatible, i.e. metadata within application segments has to
1875   be placed in beginning of file, in the order illustrated in **Figure 175**.

1876   • A status line is added in the end of JPEG data as defined in **Section A.3**.

1877   • If needed, an embedded image is interlaced in order which is free of choice as defined in **Section
1878   A.4**.

1879   • Prior to storing into a file, the CSI-2 JPEG data is processed by the data separation process
1880   described in **Section A.1**.

Start of Image (SOI)
JFIF / EXIF Data
Quantization Table (DQT)
Huffman Table (DHT)
Frame Header (SOF)
Scan Header
Compressed Data
End Of Image (EOI)

1881
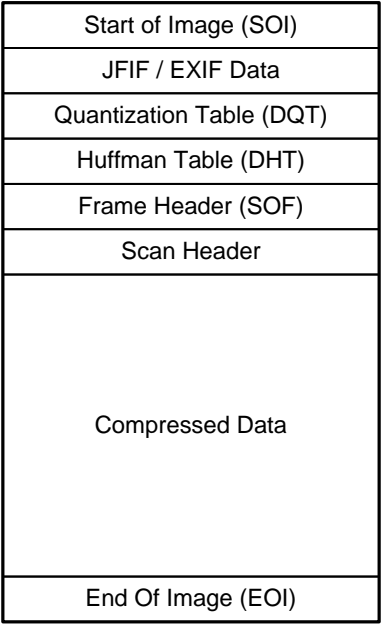
Figure 175 EXIF Compatible Baseline JPEG DCT Format

A.3   Image Status Information

1882   Information of at least the following items has to be stored in the end of the JPEG sequence as illustrated in
1883   *Figure 176*:

1884   • Image exposure time
1885   • Analog & digital gains used
1886   • White balancing gains for each color component
1887   • Camera version number
1888   • Camera register settings
1889   • Image resolution and possible thumbnail resolution

1890   The camera register settings may include a subset of camera's registers. The essential information needed
1891   for JPEG8 image is the information needed for converting the image back to linear space. This is necessary
1892   e.g. for printing service. An example of register settings is following:

1893   • Sample frequency
1894   • Exposure
1895   • Analog and digital gain
1896   • Gamma
1897   • Color gamut conversion matrix
1898   • Contrast
1899   • Brightness
1900   • Pre-gain

1901   The status information content has to be defined in the product specification of each camera module
1902   containing the JPEG8 feature. The format and content is manufacturer specific.

1903   The image status data should be arranged so that each byte is split into two 4-bit nibbles and "1010"
1904   padding sequence is added to MSB, as presented in *Table 46*. This ensures that no JPEG escape sequences
1905   (0xFF 0x00) are present in the status data.

1906   The SOSI and EOSI markers are defined in *Section A.5*.

1907   **Table 46 Status Data Padding**

Data Word	After Padding
D7D6D5D4 D3D2D1D0	1010D7D6D5D4 1010D3D2D1D0

Start of Image (SOI)
JFIF / EXIF Data
Quantization Table (DQT)
Huffman Table (DHT)
Frame Header (SOF)
Scan Header
Compressed Data
End Of Image (EOI)
Start of Status Information (SOSI)
Image Status Information
End of Status Information (EOSI)
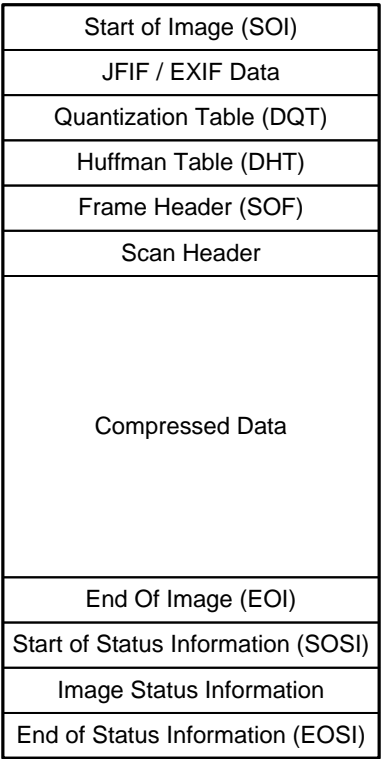
1908

Figure 176 Status Information Field in the End of Baseline JPEG Frame

A.4   Embedded Images

1909 An image may be embedded inside the JPEG data, if needed. The embedded image feature is not
1910 compulsory for each camera module containing the JPEG8 feature. An example of embedded data is a 24-
1911 bit RGB thumbnail image.

1912 The philosophy of embedded / interleaved thumbnail additions is to minimize the needed frame memory.
1913 The EI (Embedded Image) data can be included in any part of the compressed image data segment and in as
1914 many pieces as needed. See *Figure 177*.

1915 Embedded Image data is separated from compressed data by SOEI (Start Of Embedded Image) and EOEI
1916 (End Of Embedded Image) non-standard markers, which are defined in *Section A.5*. The amount of fields
1917 separated by SOEI and EOEI is not limited.

1918 The pixel to byte packing for image data within an EI data field should be as specified for the equivalent
1919 CSI-2 data format. However there is an additional restriction; the embedded image data must not generate
1920 any false JPEG marker sequences (0xFFXX).

1921 The suggested method of preventing false JPEG marker codes from occurring within the embedded image
1922 data it to limit the data range for the pixel values. For example

- 1923 For RGB888 data the suggested way to solve the false synchronization code issue is to constrain
1924 the numerical range of R, G and B values from 1 to 254.

- 1925 For RGB565 data the suggested way to solve the false synchronization code issue is to constrain
1926 the numerical range of G component from 1-62 and R component from 1-30.

1927 Each EI data field is separated by the SOEI / EOEI markers, and has to contain an equal amount bytes and
1928 a complete number of pixels. An EI data field may contain multiple lines or a full frame of image data.

1929 The embedded image data is decoded and removed apart from the JPEG compressed data prior to writing
1930 the JPEG into a file. In the process, EI data fields are appended one after each other, in order of occurrence
1931 in the received JPEG data.

1932
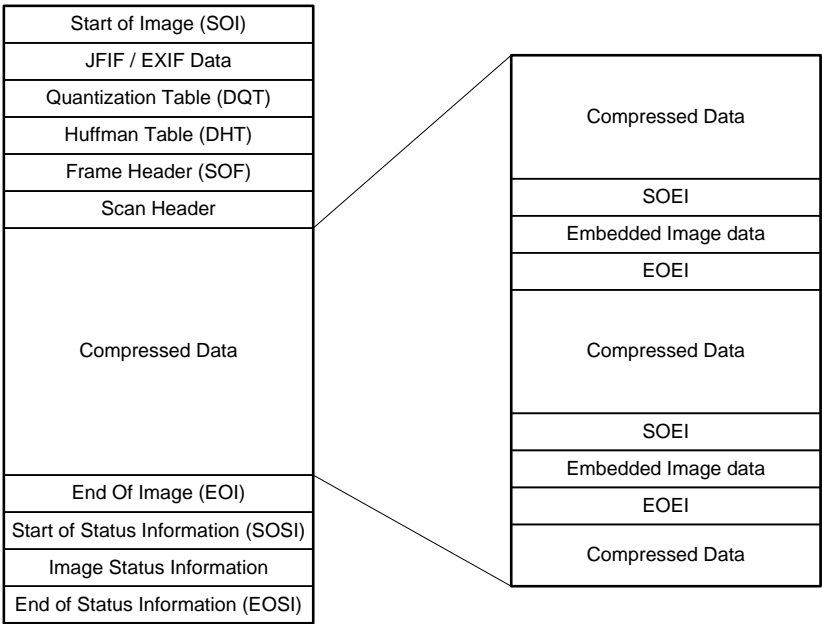
Figure 177 Example of TN Image Embedding Inside the Compressed JPEG Data Block

A.5   JPEG8 Non-standard Markers

1933   JPEG8 uses the reserved JPEG data markers for special purposes, marking the additional segments inside
1934   the data file. These segments are not part of the JPEG, JFIF [0], EXIF [0] or any other specifications;
1935   instead their use is specified in this document in *Section A.3* and *Section A.4*.

1936   The use of the non-standard markers is always internal to a product containing the JPEG8 camera module,
1937   and these markers are always removed from the JPEG data before storing it into a file.

1938   **Table 47 JPEG8 Additional Marker Codes Listing**

Non-standard Marker Symbol	Marker Data Code
SOSI	0xFF 0xBC
EOSI	0xFF 0xBD
SOEI	0xFF 0xBE
EOEI	0xFF 0xBF

A.6   JPEG8 Data Reception

1939   The compressed data format the byte to 32-bit memory word mapping follows the generic CSI-2 rule.
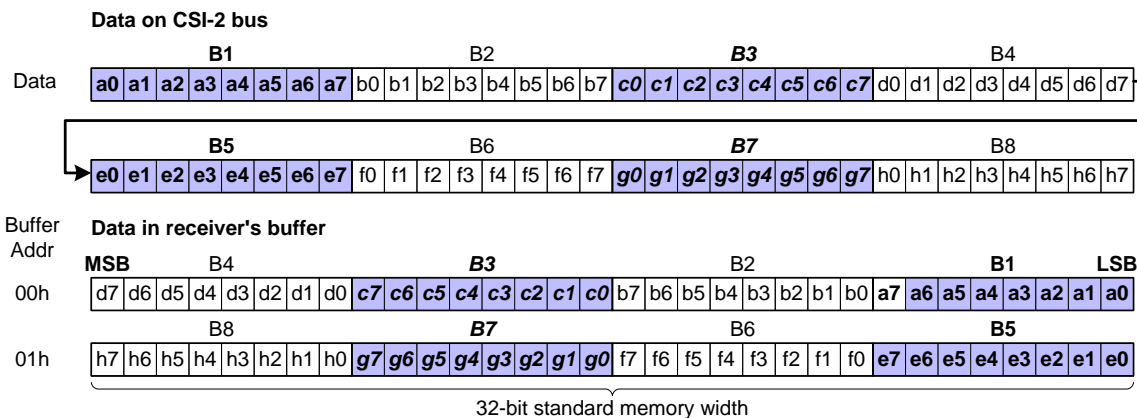
1940   **Figure 178 JPEG8 Data Format Reception**

Annex B CSI-2 Implementation Example (informative)

B.1   Overview

1941 The CSI-2 implementation example assumes that the interface comprises of D-PHY unidirectional Clock
1942 and Data, with forward escape mode and optional deskew functionality. The scope in this implementation
1943 example refers only to the unidirectional data link without any references to the CCI interface, as it can be
1944 seen in *Figure 179*. This implementation example varies from the informative PPI example in *[MIPI01]*.

1945
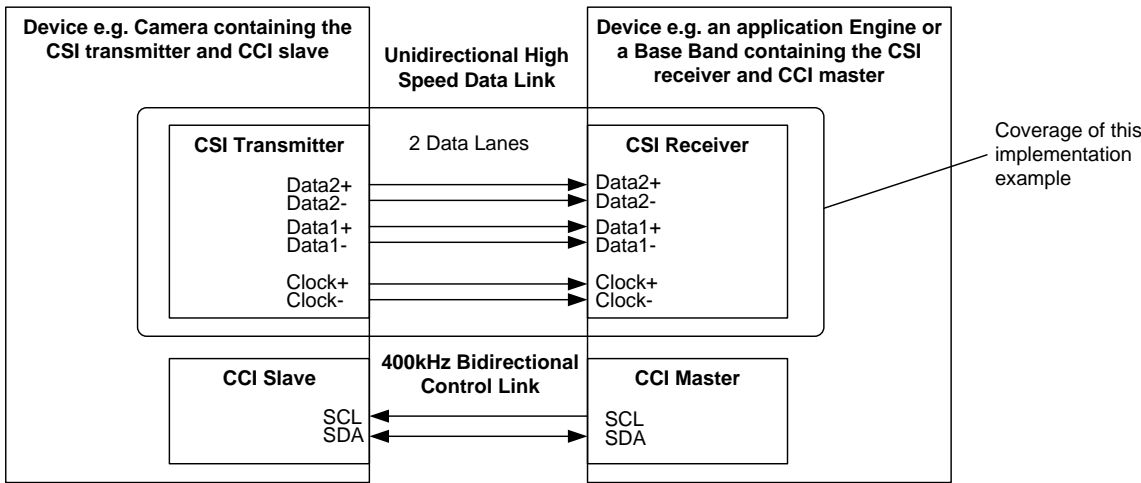
Figure 179 Implementation Example Block Diagram and Coverage

1946 For this implementation example a layered structure is described with the following parts:

1947   • D-PHY implementation details

1948   • Multi lane merger details

1949   • Protocol layer details

1950 This implementation example refers to a RAW8 data type only; hence no packing/unpacking or byte
1951 clock/pixel clock timing will be referenced as for this type of implementation they are not needed.

1952 No error recovery mechanism or error processing details will be presented, as the intent of the document is
1953 to present an implementation from the data flow perspective.

B.2   CSI-2 Transmitter Detailed Block Diagram

1954   Using the layered structure described in the overview the CSI-2 transmitter could have the block diagram in
1955   *Figure 180*.

1956
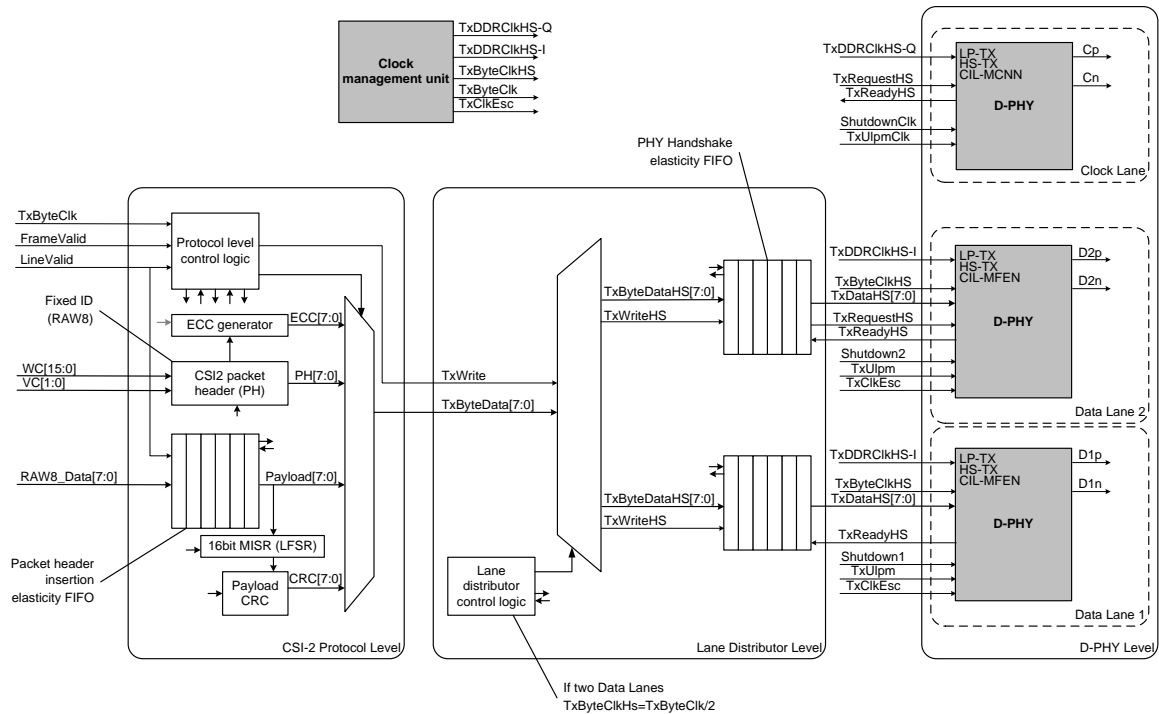
Figure 180 CSI-2 Transmitter Block Diagram

B.3   CSI-2 Receiver Detailed Block Diagram

1957   Using the layered structure described in the overview, the CSI-2 receiver could have the block diagram in
1958   *Figure 181*.

1959
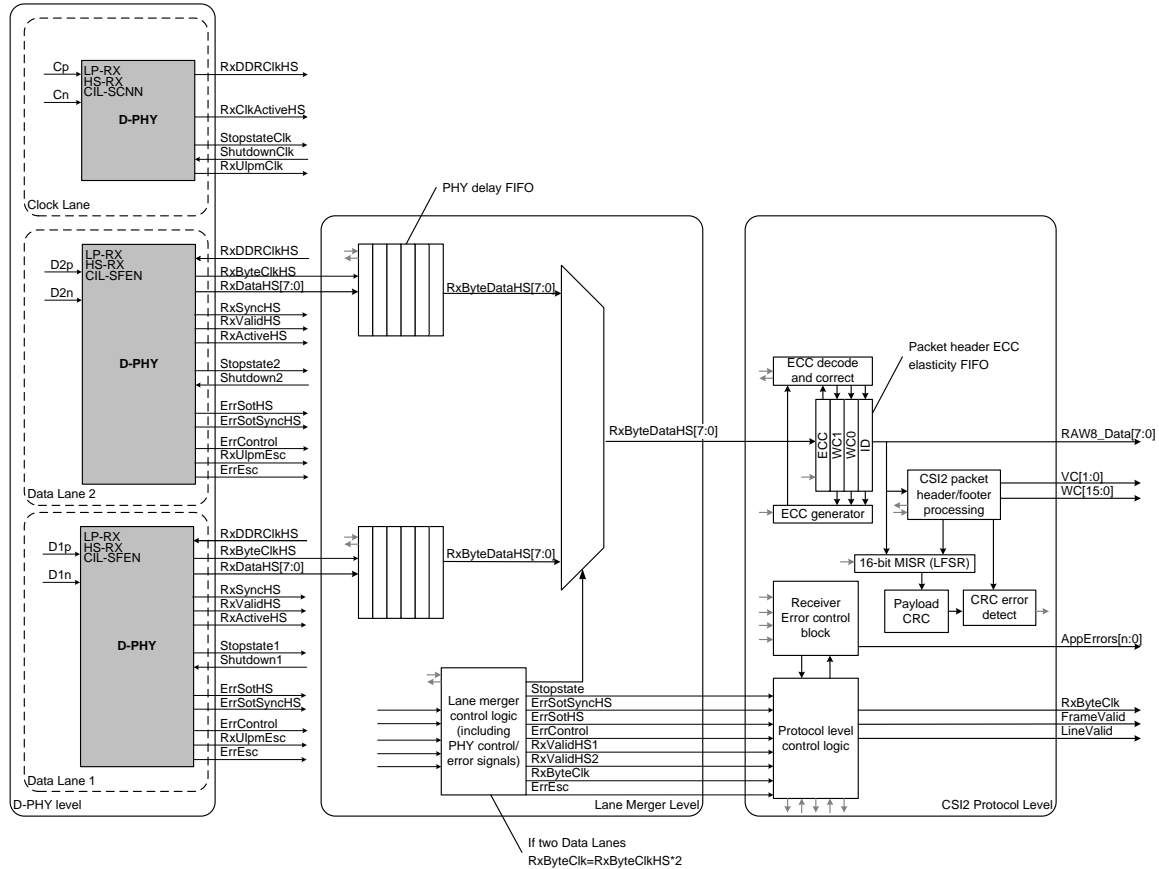
Figure 181 CSI-2 Receiver Block Diagram

B.4   Details on the D-PHY Implementation

1960   The PHY level of implementation has the top level structure as seen in *Figure 182*.

1961
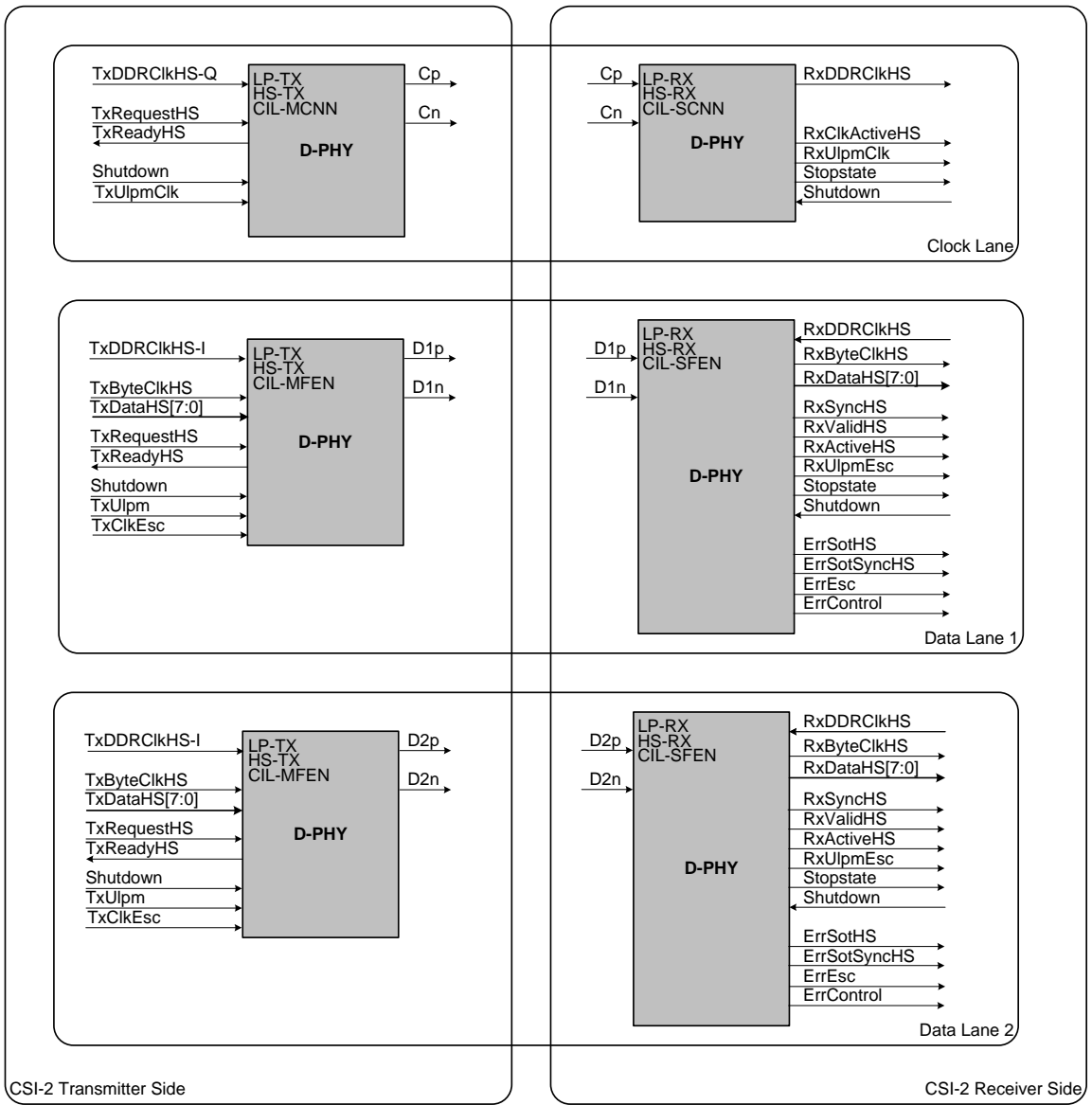
Figure 182 D-PHY Level Block Diagram

1962   The components can be categorized as:

1963   • CSI-2 Transmitter side:

1964   • Clock lane (Transmitter)

1965   • Data1 lane (Transmitter)

1966   • Data2 lane (Transmitter)

1967   • CSI-2 Receiver side:

1968   • Clock lane (Receiver)

1969   • Data1 lane (Receiver)

1970 • Data2 lane (Receiver)

B.4.1   CSI-2 Clock Lane Transmitter

1971 The suggested implementation can be seen in *Figure 183*.

1972
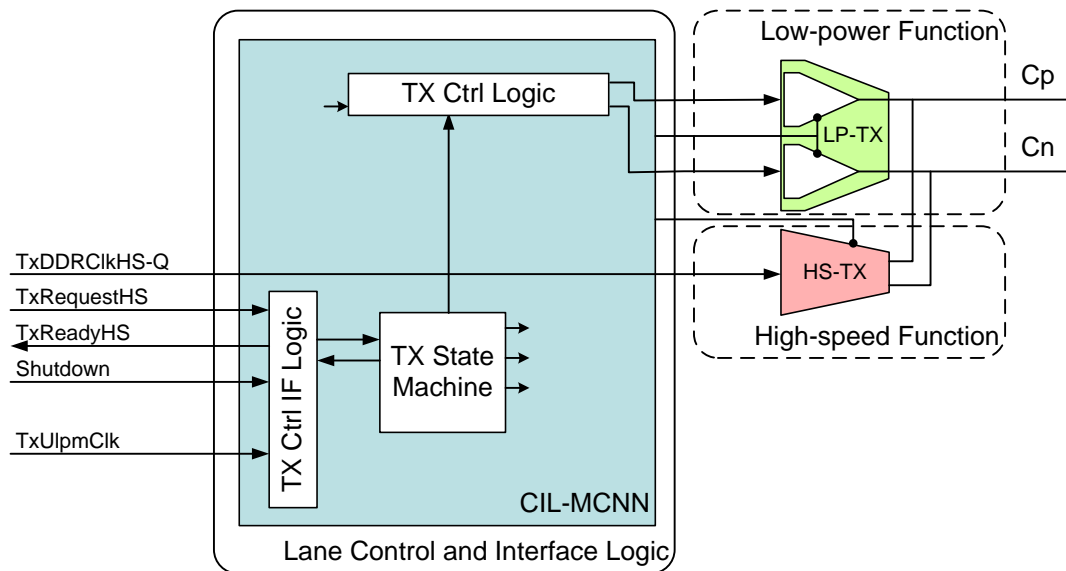
Figure 183 CSI-2 Clock Lane Transmitter

1973 The modular D-PHY components used to build a CSI-2 clock lane transmitter are:

1974 • **LP-TX** for the Low-power function

1975 • **HS-TX** for the High-speed function

1976 • **CIL-MCNN** for the Lane control and interface logic

1977 The PPI interface signals to the CSI-2 clock lane transmitter are:

1978 • **TxDDRClkHS-Q** (Input): High-Speed Transmit DDR Clock (Quadrature).

1979 • **TxRequestHS** (Input): High-Speed Transmit Request. This active high signal causes the lane
1980   module to begin transmitting a high-speed clock.

1981 • **TxReadyHS** (Output): High-Speed Transmit Ready. This active high signal indicates that the
1982   clock lane is transmitting HS clock.

1983 • **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into
1984   "shutdown", disabling all activity. All line drivers, including terminators, are turned off when
1985   Shutdown is asserted. When Shutdown is high, all other PPI inputs are ignored and all PPI outputs
1986   are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on
1987   any clock.

1988 • **TxUlpmClk** (Input): Transmit Ultra Low-Power mode on Clock Lane This active high signal is
1989   asserted to cause a Clock Lane module to enter the Ultra Low-Power mode. The lane module
1990   remains in this mode until TxUlpmClk is de-asserted.

B.4.2　CSI-2 Clock Lane Receiver

1991　The suggested implementation can be seen in *Figure 184*.

1992
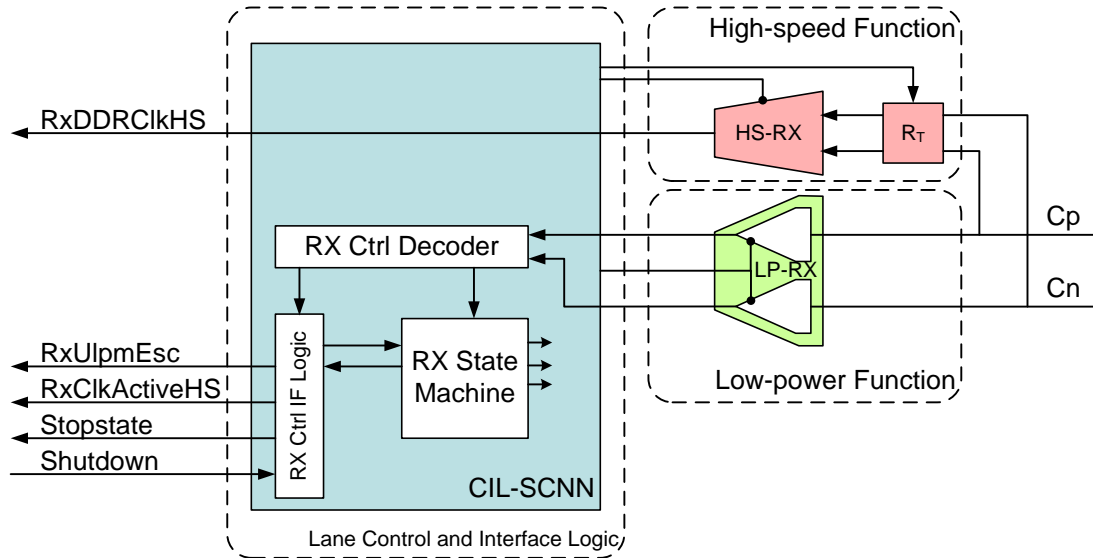
Figure 184 CSI-2 Clock Lane Receiver

1993　The modular D-PHY components used to build a CSI-2 clock lane receiver are:

1994　　• **LP-RX** for the Low-power function

1995　　• **HS-RX** for the High-speed function

1996　　• **CIL-SCNN** for the Lane control and interface logic

1997　The PPI interface signals to the CSI-2 clock lane receiver are:

1998　　• **RxDDRClkHS** (Output): High-Speed Receive DDR Clock used to sample the data in all data
1999　　　lanes.

2000　　• **RxClkActiveHS** (Output): High-Speed Reception Active. This active high signal indicates that the
2001　　　clock lane is receiving valid clock. This signal is asynchronous.

2002　　• **Stopstate** (Output): Lane is in Stop state. This active high signal indicates that the lane module is
2003　　　currently in Stop state. This signal is asynchronous.

2004　　• **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into
2005　　　"shutdown", disabling all activity. All line drivers, including terminators, are turned off when
2006　　　Shutdown is asserted. When Shutdown is high, all PPI outputs are driven to the default inactive
2007　　　state. Shutdown is a level sensitive signal and does not depend on any clock.

2008　　• **RxUlpmEsc** (Output): Escape Ultra Low Power (Receive) mode. This active high signal is
2009　　　asserted to indicate that the lane module has entered the Ultra Low-Power mode. The lane module
2010　　　remains in this mode with RxUlpmEsc asserted until a Stop state is detected on the lane
2011　　　interconnect.

B.4.3   CSI-2 Data Lane Transmitter

2012   The suggested implementation can be seen in *Figure 185*.

Figure 185 CSI-2 Data Lane Transmitter

2014   The modular D-PHY components used to build a CSI-2 data lane transmitter are:

2015   • **LP-TX** for the Low-power function

2016   • **HS-TX** for the High-speed function

2017   • **CIL-MFEN** for the Lane control and interface logic. For optional deskew calibration support, the
2018   data lane transmitter transmits a deskew sequence. The deskew sequence transmission is enabled
2019   by a mechanism out of the scope of this specification.

2020   The PPI interface signals to the CSI-2 data lane transmitter are:

2021   • **TxDDRClkHS-I** (Input): High-Speed Transmit DDR Clock (in-phase).

2022   • **TxByteClkHS** (Input): High-Speed Transmit Byte Clock. This is used to synchronize PPI signals
2023   in the high-speed transmit clock domain. It is recommended that both transmitting data lane
2024   modules share one TxByteClkHS signal. The frequency of TxByteClkHS must be exactly 1/8 the
2025   high-speed bit rate.

2026   • **TxDataHS[7:0]** (Input): High-Speed Transmit Data. Eight bit high-speed data to be transmitted.
2027   The signal connected to TxDataHS[0] is transmitted first. Data is registered on rising edges of
2028   TxByteClkHS.

2029   • **TxRequestHS** (Input): High-Speed Transmit Request. A low-to-high transition on TxRequestHS
2030   causes the lane module to initiate a Start-of-Transmission sequence. A high-to-low transition on
2031   TxRequest causes the lane module to initiate an End-of-Transmission sequence. This active high
2032   signal also indicates that the protocol is driving valid data on TxByteDataHS to be transmitted.
2033   The lane module accepts the data when both TxRequestHS and TxReadyHS are active on the same
2034   rising TxByteClkHS clock edge. The protocol always provides valid transmit data when
2035   TxRequestHS is active. Once asserted, TxRequestHS should remain high until the all the data has
2036   been accepted.

- **TxReadyHS** (Output): High-Speed Transmit Ready. This active high signal indicates that TxDataHS is accepted by the lane module to be serially transmitted. TxReadyHS is valid on rising edges of TxByteClkHS. Valid data has to be provided for the whole duration of active TxReadyHS.
- **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into "shutdown", disabling all activity. All line drivers, including terminators, are turned off when Shutdown is asserted. When Shutdown is high, all other PPI inputs are ignored and all PPI outputs are driven to the default inactive state. Shutdown is a level sensitive signal and does not depend on any clock.
- **TxUlpmEsc** (Input): Escape mode Transmit Ultra Low Power. This active high signal is asserted with TxRequestEsc to cause the lane module to enter the Ultra Low-Power mode. The lane module remains in this mode until TxRequestEsc is de-asserted.
- **TxRequestEsc** (Input): This active high signal, asserted together with TxUlpmEsc is used to request entry into escape mode. Once in escape mode, the lane stays in escape mode until TxRequestEsc is de-asserted. TxRequestEsc is only asserted by the protocol while TxRequestHS is low.
- **TxClkEsc** (Input): Escape mode Transmit Clock. This clock is directly used to generate escape sequences. The period of this clock determines the symbol time for low power signals. It is therefore constrained by the normative part of the *[MIPI01]*.

B.4.4   CSI-2 Data Lane Receiver

2056   The suggested implementation can be seen in *Figure 186*.

2057
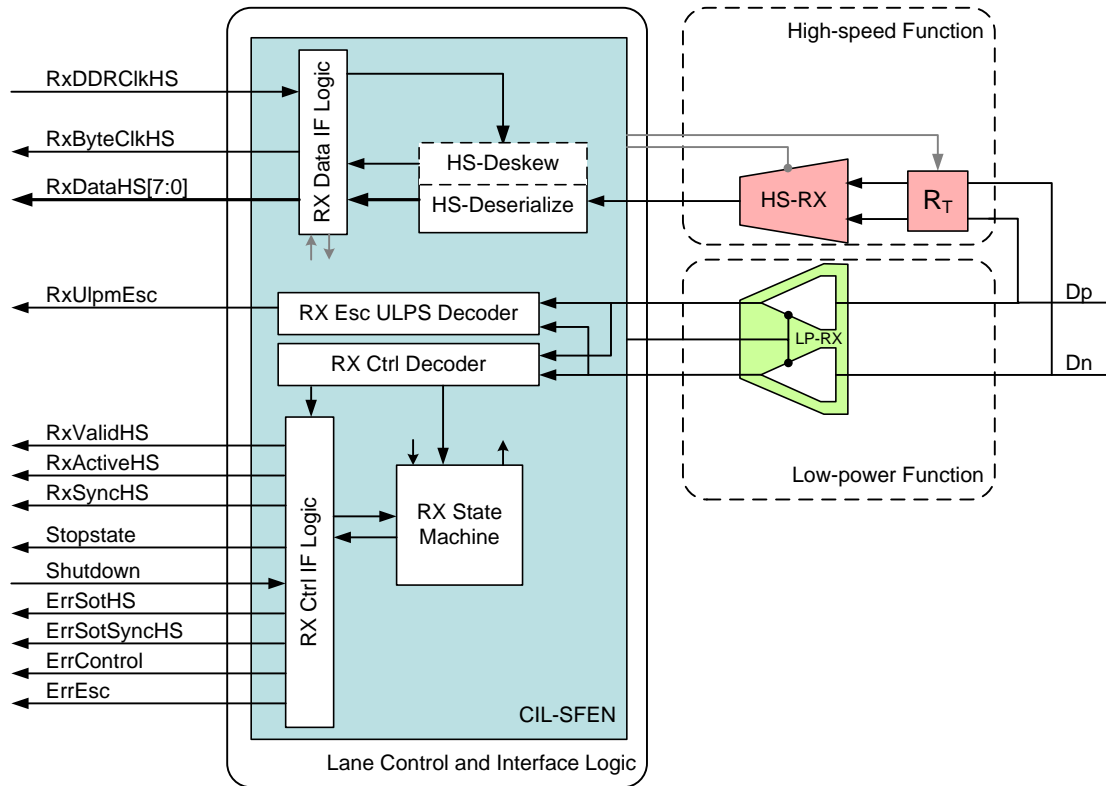
Figure 186 CSI-2 Data Lane Receiver

2058   The modular D-PHY components used to build a CSI-2 data lane receiver are:

2059   • **LP-RX** for the Low-power function

2060   • **HS-RX** for the High-speed function

2061   • **CIL-SFEN** for the Lane control and interface logic. For optional deskew calibration support the
2062   data lane receiver detects a transmitted deskew calibration pattern and performs optimum deskew
2063   of the Data with respect to the RxDDRClkHS Clock.

2064   The PPI interface signals to the CSI-2 data lane receiver are:

2065   • **RxDDRClkHS** (Input): High-Speed Receive DDR Clock used to sample the date in all data lanes.
2066   This signal is supplied by the CSI-2 clock lane receiver.

2067   • **RxByteClkHS** (Output): High-Speed Receive Byte Clock. This signal is used to synchronize
2068   signals in the high-speed receive clock domain. The RxByteClkHS is generated by dividing the
2069   received RxDDRClkHS.

2070   • **RXDataHS[7:0]** (Output): High-Speed Receive Data. Eight bit high-speed data received by the
2071   lane module. The signal connected to RxDataHS[0] was received first. Data is transferred on
2072   rising edges of RxByteClkHS.

2073   • **RxValidHS** (Output): High-Speed Receive Data Valid. This active high signal indicates that the
2074   lane module is driving valid data to the protocol on the RxDataHS output. There is no
2075   "RxReadyHS" signal, and the protocol is expected to capture RxDataHS on every rising edge of
2076   RxByteClkHS where RxValidHS is asserted. There is no provision for the protocol to slow down
2077   ("throttle") the receive data.

2078 • **RxActiveHS** (Output): High-Speed Reception Active. This active high signal indicates that the
2079   lane module is actively receiving a high-speed transmission from the lane interconnect.

2080 • **RxSyncHS** (Output): Receiver Synchronization Observed. This active high signal indicates that
2081   the lane module has seen an appropriate synchronization event. In a typical high-speed
2082   transmission, RxSyncHS is high for one cycle of RxByteClkHS at the beginning of a high-speed
2083   transmission when RxActiveHS is first asserted. This signal missing is signaled using
2084   ErrSotSyncHS.

2085 • **RxUlpmEsc** (Output): Escape Ultra Low Power (Receive) mode. This active high signal is
2086   asserted to indicate that the lane module has entered the Ultra Low-Power mode. The lane module
2087   remains in this mode with RxUlpmEsc asserted until a Stop state is detected on the lane
2088   interconnect.

2089 • **Stopstate** (Output): Lane is in Stop state. This active high signal indicates that the lane module is
2090   currently in Stop state. This signal is asynchronous.

2091 • **Shutdown** (Input): Shutdown Lane Module. This active high signal forces the lane module into
2092   "shutdown", disabling all activity. All line drivers, including terminators, are turned off when
2093   Shutdown is asserted. When Shutdown is high, all PPI outputs are driven to the default inactive
2094   state. Shutdown is a level sensitive signal and does not depend on any clock.

2095 • **ErrSotHS** (Output): Start-of-Transmission (SoT) Error. If the high-speed SoT leader sequence is
2096   corrupted, but in such a way that proper synchronization can still be achieved, this error signal is
2097   asserted for one cycle of RxByteClkHS. This is considered to be a "soft error" in the leader
2098   sequence and confidence in the payload data is reduced.

2099 • **ErrSotSyncHS** (Output): Start-of-Transmission Synchronization Error. If the high-speed SoT
2100   leader sequence is corrupted in a way that proper synchronization cannot be expected, this error is
2101   asserted for one cycle of RxByteClkHS.

2102 • **ErrControl** (Output): Control Error. This signal is asserted when an incorrect line state sequence
2103   is detected.

2104 • **ErrEsc** (Output): Escape Entry Error. If an unrecognized escape entry command is received, this
2105   signal is asserted and remains high until the next change in line state. The only escape entry
2106   command supported by the receiver is the ULPS.

Annex C CSI-2 Recommended Receiver Error Behavior (informative)

C.1 Overview

2107 This section proposes one approach to handling error conditions at the receiving side of a CSI-2 Link.
2108 Although the section is informative and therefore does not affect compliance for CSI-2, the approach is
2109 offered by the MIPI Camera Working Group as a recommended approach. The CSI-2 receiver assumes the
2110 case of a CSI-2 Link comprised of unidirectional Lanes for D-PHY Clock and Data Lanes with Escape
2111 Mode functionality on the Data Lanes and a continuously running clock. This Annex does not discuss other
2112 cases, including those that differ widely in implementation, where the implementer should consider other
2113 potential error situations.

2114 Because of the layered structure of a compliant CSI-2 receiver implementation, the error behavior is
2115 described in a similar way with several "levels" where errors could occur, each requiring some
2116 implementation at the appropriate functional layer of the design:

- *D-PHY Level errors*
  Refers to any PHY related transmission error and is unrelated to the transmission's contents:
  - Start of Transmission (SoT) errors, which can be:
    - Recoverable, if the PHY successfully identifies the Sync code but an error was detected.
    - Unrecoverable, if the PHY does not successfully identify the sync code but does detect a HS transmission.
  - *Control Error,* which signals that the PHY has detected a control sequence that should not be present in this implementation of the Link.
- *Packet Level errors*
  This type of error refers strictly to data integrity of the received Packet Header and payload data:
  - *Packet Header errors*, signaled through the ECC code, that result in:
    - A single bit-error, which can be detected and corrected by the ECC code
    - Two bit-errors in the header, which can be detected but not corrected by the ECC code, resulting in a corrupt header
  - *Packet payload errors*, signaled through the CRC code
- *Protocol Decoding Level errors*
  This type of error refers to errors present in the decoded Packet Header or errors resulting from an incomplete sequence of events:
  - *Frame Sync Error*, caused when a FS could not be successfully paired with a FE on a given virtual channel
  - *Unrecognized ID,* caused by the presence of an unimplemented or unrecognized ID in the header

2139 The proposed methodology for handling errors is signal based, since it offers an easy path to a viable CSI-2
2140 implementation that handles all three error levels. Even so, error handling at the Protocol Decoding Level
2141 should implement sequential behavior using a state machine for proper operation.

C.2   D-PHY Level Error

2142 The recommended behavior for handling this error level covers only those errors generated by the Data
2143 Lane(s), since an implementation can assume that the Clock Lane is running reliably as provided by the
2144 expected BER of the Link, as discussed in **[MIPI01]**. Note that this error handling behavior assumes
2145 unidirectional Data Lanes without escape mode functionality. Considering this, and using the signal names
2146 and descriptions from the **[MIPI01]**, PPI Annex, signal errors at the PHY-Protocol Interface (PPI) level
2147 consist of the following:

2148 • **ErrSotHS:** Start-of-Transmission (SoT) Error. If the high-speed SoT leader sequence is corrupted,
2149   but in such a way that proper synchronization can still be achieved, this error signal is asserted for
2150   one cycle of RxByteClkHS. This is considered to be a "soft error" in the leader sequence and
2151   confidence in the payload data is reduced.

2152 • **ErrSotSyncHS:** Start-of-Transmission Synchronization Error. If the high-speed SoT leader
2153   sequence is corrupted in a way that proper synchronization cannot be expected, this error signal is
2154   asserted for one cycle of RxByteClkHS.

2155 • **ErrControl:** Control Error. This signal is asserted when an incorrect line state sequence is
2156   detected. For example, if a Turn-around request or Escape Mode request is immediately followed
2157   by a Stop state instead of the required Bridge state, this signal is asserted and remains high until
2158   the next change in line state.

2159 The recommended receiver error behavior for this level is:

2160 • **ErrSotHS** should be passed to the Application Layer. Even though the error was detected and
2161   corrected and the Sync mechanism was unaffected, confidence in the data integrity is reduced and
2162   the application should be informed. This signal should be referenced to the corresponding data
2163   packet.

2164 • **ErrSotSyncHS** should be passed to the Protocol Decoding Level, since this is an unrecoverable
2165   error. An unrecoverable type of error should also be signaled to the Application Layer, since the
2166   whole transmission until the first D-PHY Stop state should be ignored if this type of error occurs.

2167 • **ErrControl** should be passed to the Application Layer, since this type of error doesn't normally
2168   occur if the interface is configured to be unidirectional. Even so, the application should be aware
2169   of the error and configure the interface accordingly through other, implementation specific-means
2170   that are out of scope for this specification.

2171 Also, it is recommended that the PPI StopState signal for each implemented Lane should be propagated to
2172 the Application Layer during configuration or initialization to indicate the Lane is ready.

C.3   Packet Level Error

2173 The recommended behavior for this error level covers only errors recognized by decoding the Packet
2174 Header's ECC field and computing the CRC of the data payload.

2175 Decoding and applying the ECC field of the Packet Header should signal the following errors:

2176 • **ErrEccDouble:** Asserted when an ECC syndrome was computed and two bit-errors are detected
2177 in the received Packet Header.

2178 • **ErrEccCorrected:** Asserted when an ECC syndrome was computed and a single bit-error in the
2179 Packet Header was detected and corrected.

2180 • **ErrEccNoError:** Asserted when an ECC syndrome was computed and the result is zero
2181 indicating a Packet Header that is considered to be without errors or has more than two bit-errors.
2182 CSI-2's ECC mechanism cannot detect this type of error.

2183 Also, computing the CRC code over the whole payload of the received packet could generate the following
2184 errors:

2185 • **ErrCrc:** Asserted when the computed CRC code is different than the received CRC code.

2186 • **ErrID:** Asserted when a Packet Header is decoded with an unrecognized or unimplemented data
2187 ID.

2188 The recommended receiver error behavior for this level is:

2189 • **ErrEccDouble** should be passed to the Application Layer since assertion of this signal proves that
2190 the Packet Header information is corrupt, and therefore the WC is not usable, and thus the packet
2191 end cannot be estimated. Commonly, this type of error will be accompanied with an ErrCrc. This
2192 type of error should also be passed to the Protocol Decoding Level, since the whole transmission
2193 until D-PHY Stop state should be ignored.

2194 • **ErrEccCorrected** should be passed to the Application Layer since the application should be
2195 informed that an error had occurred but was corrected, so the received Packet Header was
2196 unaffected, although the confidence in the data integrity is reduced.

2197 • **ErrEccNoError** can be passed to the Protocol Decoding Level to signal the validity of the current
2198 Packet Header.

2199 • **ErrCrc** should be passed to the Protocol Decoding Level to indicate that the packet's payload data
2200 might be corrupt.

2201 • **ErrID** should be passed to the Application Layer to indicate that the data packet is unidentified
2202 and cannot be unpacked by the receiver. This signal should be asserted after the ID has been
2203 identified and de-asserted on the first Frame End (FE) on same virtual channel.

C.4   Protocol Decoding Level Error

The recommended behavior for this error level covers errors caused by decoding the Packet Header information and detecting a sequence that is not allowed by the CSI-2 protocol or a sequence of detected errors by the previous layers. CSI-2 implementers will commonly choose to implement this level of error handling using a state machine that should be paired with the corresponding virtual channel. The state machine should generate at least the following error signals:

- **ErrFrameSync:** Asserted when a Frame End (FE) is not paired with a Frame Start (FS) on the same virtual channel. An ErrSotSyncHS should also generate this error signal.

- **ErrFrameData:** Asserted after a FE when the data payload received between FS and FE contains errors.

The recommended receiver error behavior for this level is:

- **ErrFrameSync** should be passed to the Application Layer with the corresponding virtual channel, since the frame could not be successfully identified. Several error cases on the same virtual channel can be identified for this type of error.
  - If a FS is followed by a second FS on the same virtual channel, the frame corresponding to the first FS is considered in error.
  - If a Packet Level ErrEccDouble was signaled from the Protocol Layer, the whole transmission until the first D-PHY Stop-state should be ignored since it contains no information that can be safely decoded and cannot be qualified with a data valid signal.
  - If a FE is followed by a second FE on the same virtual channel, the frame corresponding to the second FE is considered in error.
  - If an ErrSotSyncHS was signaled from the PHY Layer, the whole transmission until the first D-PHY Stop state should be ignored since it contains no information that can be safely decoded and cannot be qualified with a data valid signal.
- **ErrFrameData**: should be passed to the Application Layer to indicate that the frame contains data errors. This signal should be asserted on any ErrCrc and de-asserted on the first FE.

Annex D CSI-2 Sleep Mode (informative)

D.1   Overview

2229 Since a camera in a mobile terminal spends most of its time in an inactive state, implementers need a way
2230 to put the CSI-2 Link into a low power mode that approaches, or may be as low as, the leakage level. This
2231 section proposes one approach for putting a CSI-2 Link in a "Sleep Mode" (SLM). Although the section is
2232 informative and therefore does not affect compliance for CSI-2, the approach is offered by the MIPI
2233 Camera Working Group as a recommended approach.

2234 This approach relies on an aspect of a D-PHY or C-PHY transmitter's behavior that permits regulators to be
2235 disabled safely when LP-00 (Space state) is on the Link. Accordingly, this will be the output state for a
2236 CSI-2 camera transmitter in SLM.

2237 SLM can be thought of as a three-phase process:

2238 3.   SLM Command Phase. The 'ENTER SLM' command is issued to the TX side only, or to both
2239   sides of the Link.

2240 4.   SLM Entry Phase. The CSI-2 Link has entered, or is entering, the SLM in a controlled or
2241   synchronized manner. This phase is also part of the power-down process.

2242 5.   SLM Exit Phase. The CSI-2 Link has exited the SLM and the interface/device is operational. This
2243   phase is also part of the power-up process.

2244 In general, when in SLM, both sides of the interface will be in ULPS, as defined in *[MIPI01]* or *[MIPI02]*.

D.2   SLM Command Phase

2245 For the first phase, initiation of SLM occurs by a mechanism outside the scope of CSI-2. Of the many
2246 mechanisms available, two examples would be:

2247 1.   An External SLEEP signal input to the CSI-2 transmitter and optionally also to the CSI-2
2248   Receiver. When at logic 0, the CSI-2 Transmitter and the CSI Receiver (if connected) will enter
2249   Sleep mode. When at logic 1, normal operation will take place.

2250 2.   A CCI control command, provided on the I2C control Link, is used to trigger ULPS.

D.3   SLM Entry Phase

2251   For the second phase, consider one option:

2252   Only the TX side enters SLM and propagates the ULPS to the RX side by sending a D-PHY or C-PHY
2253   'ULPS' command on each Lane. In *Figure 187*, only the Data Lane 'ULPS' command is used as an
2254   example. The D-PHY Dp, Dn, and C-PHY Data_A, Data_C are logical signal names and do not imply
2255   specific multiplexing on dual mode (combined D-PHY and C-PHY) implementations.

2256
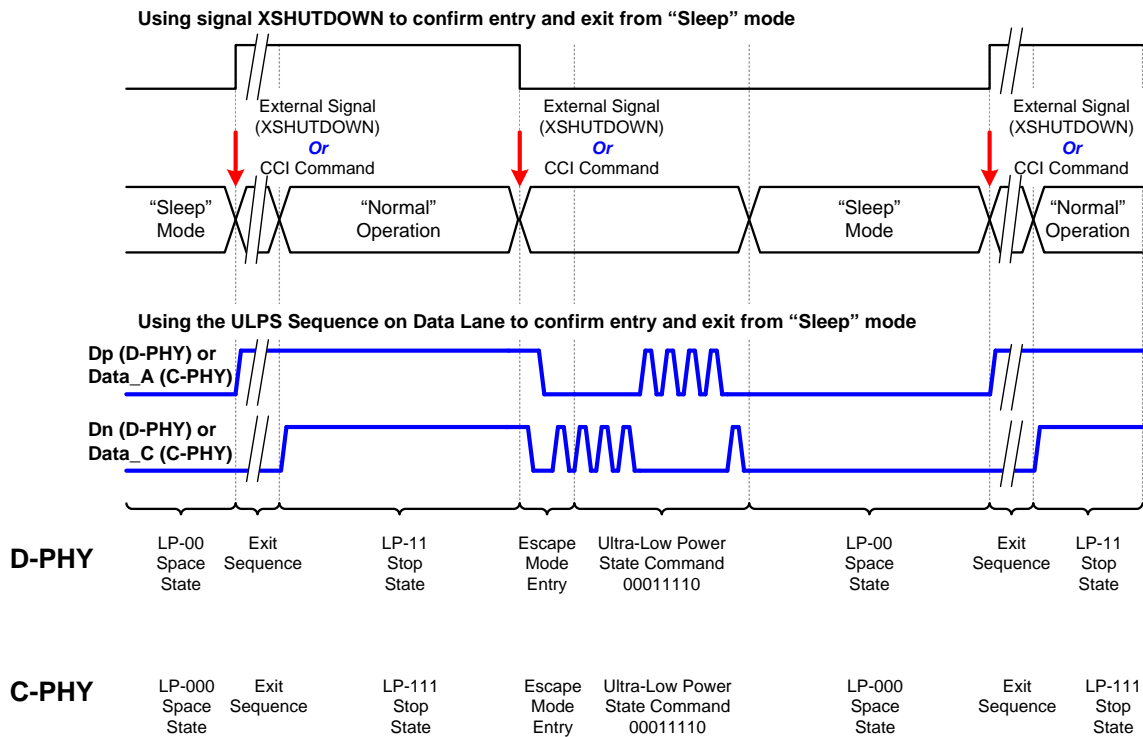
Figure 187 SLM Synchronization

D.4   SLM Exit Phase

2257   For the third phase, three options are presented and assume the camera peripheral is in ULPS or Sleep
2258   mode at power-up:

2259   1.   Use a SLEEP signal to power-up both sides of the interface.

2260   2.   Detect any CCI activity on the I2C control Link, which was in the 00 state ({SCL, SDA}), after
2261   receiving the I2C instruction to enter ULPS command as per *Section D.2*, option 2. Any change on
2262   those lines should wake up the camera peripheral. The drawback of this method is that I2C lines
2263   are used exclusively for control of the camera.

2264   3.   Detect a wake-up sequence on the I2C lines. This sequence, which may vary by implementation,
2265   shall not disturb the I2C interface so that it can be used by other devices. One example sequence
2266   is: StopI2C-StartI2C-StopI2C. See *Section 6* for details on CCI.

2267   A handshake using the 'ULPS' mechanism as described in *[MIPI01]* or *[MIPI02]*, as appropriate, should
2268   be used for powering up the interface.

Copyright © 2005-2018 MIPI Alliance, Inc.

Annex E Data Compression for RAW Data Types (normative)

2269 A CSI-2 implementation using RAW data types may support compression on the interface to reduce the
2270 data bandwidth requirements between the host processor and a camera module. Data compression is not
2271 mandated by this Specification. However, if data compression is used, it shall be implemented as described
2272 in this annex.

2273 Data compression schemes use an X–Y–Z naming convention where X is the number of bits per pixel in
2274 the original image, Y is the encoded (compressed) bits per pixel and Z is the decoded (uncompressed) bits
2275 per pixel.

2276 The following data compression schemes are defined:

2277 • 12-10-12
2278 • 12–8–12
2279 • 12–7–12
2280 • 12–6–12
2281 • 10–8–10
2282 • 10–7–10
2283 • 10–6–10

2284 To identify the type of data on the CSI-2 interface, packets with compressed data shall have a User Defined
2285 Data Type value as indicated in *Table 45*. Note that User Defined data type codes are not reserved for
2286 compressed data types. Therefore, a CSI-2 device shall be able to communicate over the CCI the data
2287 compression scheme represented by a particular User Defined data type code for each scheme supported by
2288 the device. Note that the method to communicate the data compression scheme to Data Type code mapping
2289 is beyond the scope of this document.

2290 The number of bits in a packet shall be a multiple of eight. Therefore, implementations with data
2291 compression schemes that result in each pixel having other than eight encoded bits per pixel shall transfer
2292 the encoded data in a packed pixel format. For example, the 12–7–12 data compression scheme uses a
2293 packed pixel format as described in *Section 11.4.2* except the Data Type value in the Packet Header is a
2294 User Defined data type code.

2295 The data compression schemes in this annex are lossy and designed to encode each line independent of the
2296 other lines in the image.

2297 The following definitions are used in the description of the data compression schemes:

2298 • **Xorig** is the original pixel value
2299 • **Xpred** is the predicted pixel value
2300 • **Xdiff** is the difference value (**Xorig** - **Xpred**)
2301 • **Xenco** is the encoded value
2302 • **Xdeco** is the decoded pixel value

2303 The data compression system consists of encoder, decoder and predictor blocks as shown in *Figure 188*.
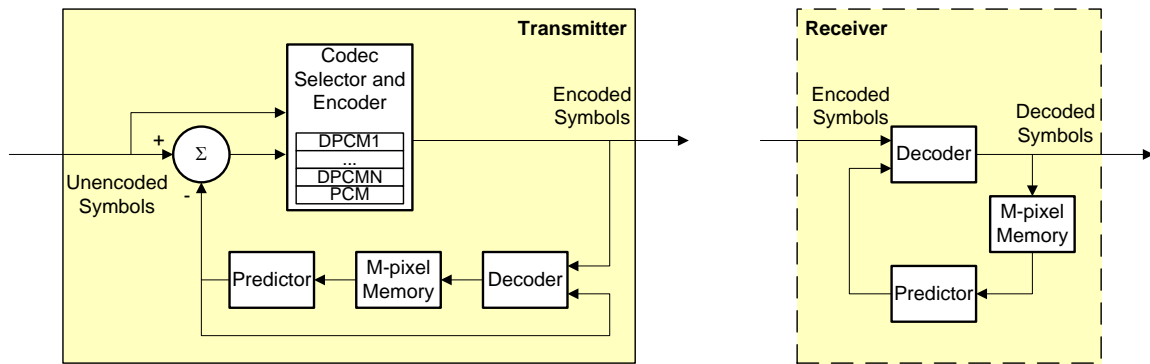
Figure 188 Data Compression System Block Diagram

The encoder uses a simple algorithm to encode the pixel values. A fixed number of pixel values at the beginning of each line are encoded without using prediction. These first few values are used to initialize the predictor block. The remaining pixel values on the line are encoded using prediction.

If the predicted value of the pixel (**Xpred**) is close enough to the original value of the pixel (**Xorig**) (abs(**Xorig - Xpred**) < difference limit), its difference value (**Xdiff**) is quantized using a DPCM codec. Otherwise, **Xorig** is quantized using a PCM codec. The quantized value is combined with a code word describing the codec used to quantize the pixel and the sign bit, if applicable, to create the encoded value (**Xenco**).

E.1 Predictors

In order to have meaningful data transfer, both the transmitter and the receiver need to use the same predictor block.

The order of pixels in a raw image is shown in *Figure 189*.
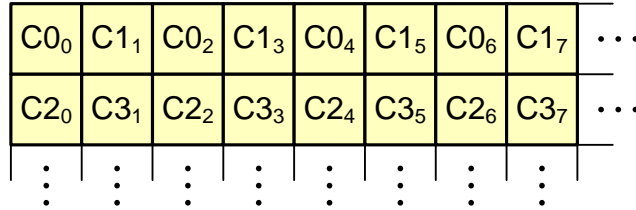
Figure 189 Pixel Order of the Original Image

Figure 190 shows an example of the pixel order with RGB data.
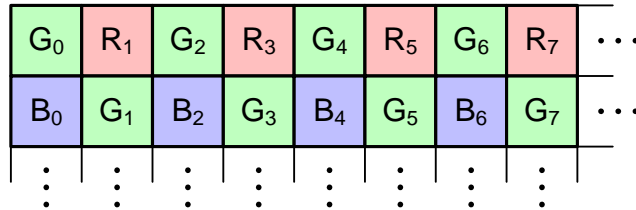
Figure 190 Example Pixel Order of the Original Image

Two predictors are defined for use in the data compression schemes.

Predictor1 uses a very simple algorithm and is intended to minimize processing power and memory size requirements. Typically, this predictor is used when the compression requirements are modest and the original image quality is high. Predictor1 should be used with 10–8–10, 10–7–10, 12-10-12, and 12–8–12 data compression schemes.

The second predictor, Predictor2, is more complex than Predictor1. This predictor provides slightly better prediction than Predictor1 and therefore the decoded image quality can be improved compared to Predictor1. Predictor2 should be used with 10–6–10, 12–7–12, and 12–6–12 data compression schemes.

Both receiver and transmitter shall support Predictor1 for all data compression schemes.

E.1.1 Predictor1

Predictor1 uses only the previous same color component value as the prediction value. Therefore, only a two-pixel deep memory is required.

The first two pixels ($C0_0$, $C1_1$ / $C2_0$, $C3_1$ or as in example G_0, R_1 / B_0, G_1) in a line are encoded without prediction.

The prediction values for the remaining pixels in the line are calculated using the previous same color decoded value, **Xdeco**. Therefore, the predictor equation can be written as follows:

```
Xpred( n ) = Xdeco( n-2 )
```

E.1.2   Predictor2

Predictor2 uses the four previous pixel values, when the prediction value is evaluated. This means that also the other color component values are used, when the prediction value has been defined. The predictor equations can be written as shown in the following formulas.

Predictor2 uses all color components of the four previous pixel values to create the prediction value. Therefore, a four-pixel deep memory is required.

The first pixel ($C0_0$ / $C2_0$, or as in example G_0 / B_0) in a line is coded without prediction.

The second pixel ($C1_1$ / $C3_1$ or as in example R_1 / G_1) in a line is predicted using the previous decoded different color value as a prediction value. The second pixel is predicted with the following equation:

```
Xpred( n ) = Xdeco( n-1 )
```

The third pixel ($C0_2$ / $C2_2$ or as in example G_2 / B_2) in a line is predicted using the previous decoded same color value as a prediction value. The third pixel is predicted with the following equation:

```
Xpred( n ) = Xdeco( n-2 )
```

The fourth pixel ($C1_3$ / $C3_3$ or as in example R_3 / G_3) in a line is predicted using the following equation:

```
if ((Xdeco( n-1 ) <= Xdeco( n-2 ) AND Xdeco( n-2 ) <= Xdeco( n-3 )) OR
   (Xdeco( n-1 ) >= Xdeco( n-2 ) AND Xdeco( n-2 ) >= Xdeco( n-3 ))) then
      Xpred( n ) = Xdeco( n-1 )
else
   Xpred( n ) = Xdeco( n-2 )
endif
```

Other pixels in all lines are predicted using the equation:

```
if ((Xdeco( n-1 ) <= Xdeco( n-2 ) AND Xdeco( n-2 ) <= Xdeco( n-3 )) OR
   (Xdeco( n-1 ) >= Xdeco( n-2 ) AND Xdeco( n-2 ) >= Xdeco( n-3 ))) then
      Xpred( n ) = Xdeco( n-1 )
else if ((Xdeco( n-1 ) <= Xdeco( n-3 ) AND Xdeco( n-2 ) <= Xdeco( n-4 )) OR
   (Xdeco( n-1 ) >= Xdeco( n-3 ) AND Xdeco( n-2 ) >= Xdeco( n-4 ))) then
      Xpred( n ) = Xdeco( n-2 )
else
   Xpred( n ) = (Xdeco( n-2 ) + Xdeco( n-4 ) + 1) / 2
endif
```

E.2   Encoders

2364 There are seven different encoders available, one for each data compression scheme.

2365 For all encoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula
2366 for predicted pixels.

E.2.1   Coder for 10–8–10 Data Compression

2367 The 10–8–10 coder offers a 20% bit rate reduction with very high image quality.

2368 Pixels without prediction are encoded using the following formula:

```
2369    Xenco( n ) = Xorig( n ) / 4
```

2370 To avoid a full-zero encoded value, the following check is performed:

```
2371    if (Xenco( n ) == 0) then
2372       Xenco( n ) = 1
2373    endif
```

2374 Pixels with prediction are encoded using the following formula:

```
2375    if (abs(Xdiff( n )) < 32) then
2376       use DPCM1
2377    else if (abs(Xdiff( n )) < 64) then
2378       use DPCM2
2379    else if (abs(Xdiff( n )) < 128) then
2380       use DPCM3
2381    else
2382       use PCM
2383    endif
```

E.2.1.1   DPCM1 for 10–8–10 Coder

2384 **Xenco**(**n**) has the following format:

```
2385    Xenco( n ) = "00 s xxxxx"
```

2386 where,

```
2387    "00" is the code word
2388    "s" is the sign bit
2389    "xxxxx" is the five bit value field
```

2390 The coder equation is described as follows:

```
2391    if (Xdiff( n ) <= 0) then
2392       sign = 1
2393    else
2394       sign = 0
2395    endif
2396    value = abs(Xdiff( n ))
```

2397 *Note: Zero code has been avoided (0 is sent as -0).*

E.2.1.2   DPCM2 for 10–8–10 Coder

2398   **Xenco**(**n**) has the following format:

2399   `Xenco( n ) = "010 s xxxx"`

2400   where,

2401   `"010"` is the code word
2402   `"s"` is the **sign** bit
2403   `"xxxx"` is the four bit **value** field

2404   The coder equation is described as follows:

2405   `if (`**Xdiff**`( n ) < 0) then`
2406   **sign** `= 1`
2407   `else`
2408   **sign** `= 0`
2409   `endif`
2410   **value** `= (abs(`**Xdiff**`( n )) - 32) / 2`

E.2.1.3   DPCM3 for 10–8–10 Coder

2411   **Xenco**(**n**) has the following format:

2412   `Xenco( n ) = "011 s xxxx"`

2413   where,

2414   `"011"` is the code word
2415   `"s"` is the **sign** bit
2416   `"xxxx"` is the four bit **value** field

2417   The coder equation is described as follows:

2418   `if (`**Xdiff**`( n ) < 0) then`
2419   **sign** `= 1`
2420   `else`
2421   **sign** `= 0`
2422   `endif`
2423   **value** `= (abs(`**Xdiff**`( n )) - 64) / 4`

E.2.1.4   PCM for 10–8–10 Coder

2424   **Xenco**(**n**) has the following format:

2425   `Xenco( n ) = "1 xxxxxxx"`

2426   where,

2427   `"1"` is the code word
2428   the **sign** bit is not used
2429   `"xxxxxxx"` is the seven bit **value** field

2430   The coder equation is described as follows:

2431   **value** `= `**Xorig**`( n ) / 8`

E.2.2   Coder for 10–7–10 Data Compression

2432   The 10–7–10 coder offers 30% bit rate reduction with high image quality.

2433   Pixels without prediction are encoded using the following formula:

2434
```
Xenco( n ) = Xorig( n ) / 8
```

2435   To avoid a full-zero encoded value, the following check is performed:

2436
2437
```
if (Xenco( n ) == 0) then
    Xenco( n ) = 1
```

2438   Pixels with prediction are encoded using the following formula:

2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
```
if (abs(Xdiff( n )) < 8) then
    use DPCM1
else if (abs(Xdiff( n )) < 16) then
    use DPCM2
else if (abs(Xdiff( n )) < 32) then
    use DPCM3
else if (abs(Xdiff( n )) < 160) then
    use DPCM4
else
    use PCM
endif
```

E.2.2.1   DPCM1 for 10–7–10 Coder

2450   **Xenco**(**n**) has the following format:

2451
```
Xenco( n ) = "000 s xxx"
```

2452   where,

2453
2454
2455
```
"000" is the code word
"s" is the sign bit
"xxx" is the three bit value field
```

2456   The coder equation is described as follows:

2457
2458
2459
2460
2461
2462
```
if (Xdiff( n ) <= 0) then
    sign = 1
else
    sign = 0
endif
value = abs(Xdiff( n ))
```

2463   *Note: Zero code has been avoided (0 is sent as -0).*

E.2.2.2   DPCM2 for 10–7–10 Coder

2464   **Xenco**(**n**) has the following format:

2465   `Xenco( n ) = "0010 s xx"`

2466   where,

2467   `"0010" is the code word`
2468   `"s" is the `**`sign`**` bit`
2469   `"xx" is the two bit `**`value`**` field`

2470   The coder equation is described as follows:

2471   `if (`**`Xdiff`**`( n ) < 0) then`
2472   **`sign`**` = 1`
2473   `else`
2474   **`sign`**` = 0`
2475   `endif`
2476   **`value`**` = (abs(`**`Xdiff`**`( n )) – 8) / 2`

E.2.2.3   DPCM3 for 10–7–10 Coder

2477   **Xenco**(**n**) has the following format:

2478   `Xenco( n ) = "0011 s xx"`

2479   where,

2480   `"0011" is the code word`
2481   `"s" is the `**`sign`**` bit`
2482   `"xx" is the two bit `**`value`**` field`

2483   The coder equation is described as follows:

2484   `if (`**`Xdiff`**`( n ) < 0) then`
2485   **`sign`**` = 1`
2486   `else`
2487   **`sign`**` = 0`
2488   `endif`
2489   **`value`**` = (abs(`**`Xdiff`**`( n )) – 16) / 4`

E.2.2.4   DPCM4 for 10–7–10 Coder

2490   **Xenco**(**n**) has the following format:

2491   `Xenco( n ) = "01 s xxxx"`

2492   where,

2493   `"01" is the code word`
2494   `"s" is the `**`sign`**` bit`
2495   `"xxxx" is the four bit `**`value`**` field`

2496   The coder equation is described as follows:

2497   `if (`**`Xdiff`**`( n ) < 0) then`
2498   **`sign`**` = 1`
2499   `else`
2500   **`sign`**` = 0`
2501   `endif`
2502   **`value`**` = (abs(`**`Xdiff`**`( n )) – 32) / 8`

E.2.2.5 PCM for 10–7–10 Coder

2503 **Xenco**(**n**) has the following format:

2504   `Xenco`(`n`) = "1 xxxxxx"

2505 where,

2506   "1" is the code word
2507   the **sign** bit is not used
2508   "xxxxxx" is the six bit **value** field

2509 The coder equation is described as follows:

2510   **value** = **Xorig**(**n**) / 16

E.2.3   Coder for 10–6–10 Data Compression

2511   The 10–6–10 coder offers 40% bit rate reduction with acceptable image quality.

2512   Pixels without prediction are encoded using the following formula:

2513
```
Xenco( n ) = Xorig( n ) / 16
```

2514   To avoid a full-zero encoded value, the following check is performed:

2515
2516
2517
```
if (Xenco( n ) == 0) then
    Xenco( n ) = 1
endif
```

2518   Pixels with prediction are encoded using the following formula:

2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
```
if (abs(Xdiff( n )) < 1) then
    use DPCM1
else if (abs(Xdiff( n )) < 3) then
    use DPCM2
else if (abs(Xdiff( n )) < 11) then
    use DPCM3
else if (abs(Xdiff( n )) < 43) then
    use DPCM4
else if (abs(Xdiff( n )) < 171) then
    use DPCM5
else
    use PCM
endif
```

E.2.3.1   DPCM1 for 10–6–10 Coder

2532   **Xenco**(**n**) has the following format:

2533
```
Xenco( n ) = "00000 s"
```

2534   where,

2535
2536
2537
```
"00000" is the code word
"s" is the sign bit
the value field is not used
```

2538   The coder equation is described as follows:

2539
2540
```
sign = 1
Note: Zero code has been avoided (0 is sent as -0).
```

E.2.3.2   DPCM2 for 10–6–10 Coder

2541   **Xenco**(**n**) has the following format:

2542
```
Xenco( n ) = "00001 s"
```

2543   where,

2544
2545
2546
```
"00001" is the code word
"s" is the sign bit
the value field is not used
```

2547   The coder equation is described as follows:

2548
2549
2550
2551
2552
```
if (Xdiff( n ) < 0) then
    sign = 1
else
    sign = 0
endif
```

E.2.3.3   DPCM3 for 10–6–10 Coder

2553   **Xenco**(**n**) has the following format:

2554   **Xenco**(**n**) = "0001 s x"

2555   where,

2556   "0001" is the code word
2557   "s" is the **sign** bit
2558   "x" is the one bit **value** field

2559   The coder equation is described as follows:

2560   if (**Xdiff**(**n**) < 0) then
2561   **sign** = 1
2562   else
2563   **sign** = 0
2564   **value** = (abs(**Xdiff**(**n**)) – 3) / 4
2565   endif

E.2.3.4   DPCM4 for 10–6–10 Coder

2566   **Xenco**(**n**) has the following format:

2567   **Xenco**(**n**) = "001 s xx"

2568   where,

2569   "001" is the code word
2570   "s" is the **sign** bit
2571   "xx" is the two bit **value** field

2572   The coder equation is described as follows:

2573   if (**Xdiff**(**n**) < 0) then
2574   **sign** = 1
2575   else
2576   **sign** = 0
2577   endif
2578   **value** = (abs(**Xdiff**(**n**)) – 11) / 8

E.2.3.5   DPCM5 for 10–6–10 Coder

2579   **Xenco**(**n**) has the following format:

2580   **Xenco**(**n**) = "01 s xxx"

2581   where,

2582   "01" is the code word
2583   "s" is the **sign** bit
2584   "xxx" is the three bit **value** field

2585   The coder equation is described as follows:

2586   if (**Xdiff**(**n**) < 0) then
2587   **sign** = 1
2588   else
2589   **sign** = 0
2590   endif
2591   **value** = (abs(**Xdiff**(**n**)) – 43) / 16

E.2.3.6 PCM for 10–6–10 Coder

2592 **Xenco**(**n**) has the following format:

2593   `Xenco( n ) = "1 xxxxx"`

2594 where,

2595   `"1" is the code word`
2596   `the `**`sign`**` bit is not used`
2597   `"xxxxx" is the five bit `**`value`**` field`

2598 The coder equation is described as follows:

2599   **`value`**` = `**`Xorig`**`( `**`n`**` ) / 32`

E.2.4　　Coder for 12-10-12 Data Compression

2600　The 12–10–12 coder offers a 16.7% bit rate reduction with very high image quality.

2601　Pixels without prediction are encoded using the following formula:

2602
```
Xenco( n ) = Xorig( n ) / 4
```

2603　To avoid a full-zero encoded value, the following check is performed:

2604
2605
2606
```
if (Xenco( n ) == 0) then
    Xenco( n ) = 1
endif
```

2607　Pixels with prediction are encoded using the following formula:

2608
2609
2610
2611
2612
2613
2614
2615
2616
```
if (abs(Xdiff( n )) < 128) then
    use DPCM1
else if (abs(Xdiff( n )) < 256) then
    use DPCM2
else if (abs(Xdiff( n )) < 512) then
    use DPCM3
else
    use PCM
endif
```

E.2.4.1　　DPCM1 for 12–10–12 Coder

2617　**Xenco**(**n**) has the following format:

2618
```
Xenco( n ) = "00 s xxxxxxx"
```

2619　where,

2620
2621
2622
```
"00" is the code word
"s" is the sign bit
"xxxxxxx" is the seven bit value field
```

2623　The coder equation is described as follows:

2624
2625
2626
2627
2628
2629
```
if (Xdiff( n ) <= 0) then
    sign = 1
else
    sign = 0
endif
value = abs(Xdiff( n ))
```

2630　***Note:***

2631　　*Zero code has been avoided (0 is sent as -0).*

E.2.4.2   DPCM2 for 12–10–12 Coder

2632   **Xenco**(**n**) has the following format:

2633   `Xenco( n ) = "010 s xxxxxx"`

2634   where,

2635   `"010"` is the code word
2636   `"s"` is the **sign** bit
2637   `"xxxxxx"` is the six bit **value** field

2638   The coder equation is described as follows:

```
2639    if (Xdiff( n ) < 0) then
2640        sign = 1
2641    else
2642        sign = 0
2643    endif
2644    value = (abs(Xdiff( n )) - 128) / 2
```

E.2.4.3   DPCM3 for 12–10–12 Coder

2645   **Xenco**(**n**) has the following format:

2646   `Xenco( n ) = "011 s xxxxxx"`

2647   where,

2648   `"011"` is the code word
2649   `"s"` is the **sign** bit
2650   `"xxxxxx"` is the six bit **value** field

2651   The coder equation is described as follows:

```
2652    if (Xdiff( n ) < 0) then
2653        sign = 1
2654    else
2655        sign = 0
2656    endif
2657    value = (abs(Xdiff( n )) - 256) / 4
```

E.2.4.4   PCM for 12–10–12 Coder

2658   **Xenco**(**n**) has the following format:

2659   `Xenco( n ) = "1 xxxxxxxxx"`

2660   where,

2661   `"1"` is the code word
2662   the **sign** bit is not used
2663   `"xxxxxxxxx"` is the nine bit **value** field

2664   The coder equation is described as follows:

2665   `value = Xorig( n ) / 8`

E.2.5   Coder for 12–8–12 Data Compression

2666   The 12–8–12 coder offers 33% bit rate reduction with very high image quality.

2667   Pixels without prediction are encoded using the following formula:

2668
```
Xenco( n ) = Xorig( n ) / 16
```

2669   To avoid a full-zero encoded value, the following check is performed:

2670
2671
2672
```
if (Xenco( n ) == 0) then
    Xenco( n ) = 1
endif
```

2673   Pixels with prediction are encoded using the following formula:

2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
```
if (abs(Xdiff( n )) < 8) then
    use DPCM1
else if (abs(Xdiff( n )) < 40) then
    use DPCM2
else if (abs(Xdiff( n )) < 104) then
    use DPCM3
else if (abs(Xdiff( n )) < 232) then
    use DPCM4
else if (abs(Xdiff( n )) < 360) then
    use DPCM5
else
    use PCM
```

E.2.5.1   DPCM1 for 12–8–12 Coder

2686   **Xenco**(**n**) has the following format:

2687
```
Xenco( n ) = "0000 s xxx"
```

2688   where,

2689
2690
2691
```
"0000" is the code word
"s" is the sign bit
"xxx" is the three bit value field
```

2692   The coder equation is described as follows:

2693
2694
2695
2696
2697
2698
```
if (Xdiff( n ) <= 0) then
    sign = 1
else
    sign = 0
endif
value = abs(Xdiff( n ))
```

2699   *Note: Zero code has been avoided (0 is sent as -0).*

E.2.5.2   DPCM2 for 12–8–12 Coder

2700 **Xenco**(**n**) has the following format:
2701   ```
      Xenco(n) = "011 s xxxx"
      ```
2702  where,
2703      ```
      "011" is the code word
      ```
2704   ```
      "s" is the sign bit
      ```
2705      ```
      "xxxx" is the four bit value field
      ```
2706 The coder equation is described as follows:
2707   ```
      if (Xdiff(n) < 0) then
      ```
2708      ```
          sign = 1
      ```
2709   ```
      else
      ```
2710      ```
          sign = 0
      ```
2711   ```
      endif
      ```
2712      ```
      value = (abs(Xdiff( n )) – 8) / 2
      ```

E.2.5.3   DPCM3 for 12–8–12 Coder

2713 **Xenco**(**n**) has the following format:
2714   ```
      Xenco(n) = "010 s xxxx"
      ```
2715  where,
2716      ```
      "010" is the code word
      ```
2717   ```
      "s" is the sign bit
      ```
2718      ```
      "xxxx" is the four bit value field
      ```
2719 The coder equation is described as follows:
2720   ```
      if (Xdiff(n) < 0) then
      ```
2721      ```
          sign = 1
      ```
2722   ```
      else
      ```
2723      ```
          sign = 0
      ```
2724   ```
      endif
      ```
2725      ```
      value = (abs(Xdiff( n )) – 40) / 4
      ```

E.2.5.4   DPCM4 for 12–8–12 Coder

2726 **Xenco**(**n**) has the following format:
2727   ```
      Xenco(n) = "001 s xxxx"
      ```
2728  where,
2729      ```
      "001" is the code word
      ```
2730   ```
      "s" is the sign bit
      ```
2731      ```
      "xxxx" is the four bit value field
      ```
2732 The coder equation is described as follows:
2733   ```
      if (Xdiff(n) < 0) then
      ```
2734      ```
          sign = 1
      ```
2735   ```
      else
      ```
2736      ```
          sign = 0
      ```
2737   ```
      endif
      ```
2738      ```
      value = (abs(Xdiff( n )) – 104) / 8
      ```

E.2.5.5   DPCM5 for 12–8–12 Coder

2739   **Xenco**(**n**) has the following format:

2740   `Xenco( n ) = "0001 s xxx"`

2741   where,

2742   `"0001" is the code word`
2743   `"s" is the` **sign** `bit`
2744   `"xxx" is the three bit` **value** `field`

2745   The coder equation is described as follows:

2746   `if (`**Xdiff**`( n ) < 0) then`
2747   **sign** `= 1`
2748   `else`
2749   **sign** `= 0`
2750   `endif`
2751   **value** `= (abs(`**Xdiff**`( n )) - 232) / 16`

E.2.5.6   PCM for 12–8–12 Coder

2752   **Xenco**(**n**) has the following format:

2753   `Xenco( n ) = "1 xxxxxxx"`

2754   where,

2755   `"1" is the code word`
2756   `the` **sign** `bit is not used`
2757   `"xxxxxxx" is the seven bit` **value** `field`

2758   The coder equation is described as follows:

2759   **value** `=` **Xorig**`( n ) / 32`

E.2.6   Coder for 12–7–12 Data Compression

2760   The 12–7–12 coder offers 42% bit rate reduction with high image quality.

2761   Pixels without prediction are encoded using the following formula:

2762
```
Xenco( n ) = Xorig( n ) / 32
```

2763   To avoid a full-zero encoded value, the following check is performed:

2764
```
if (Xenco( n ) == 0) then
```
2765
```
    Xenco( n ) = 1
```
2766
```
endif
```

2767   Pixels with prediction are encoded using the following formula:

2768
```
if (abs(Xdiff( n )) < 4) then
```
2769
```
    use DPCM1
```
2770
```
else if (abs(Xdiff( n )) < 12) then
```
2771
```
    use DPCM2
```
2772
```
else if (abs(Xdiff( n )) < 28) then
```
2773
```
    use DPCM3
```
2774
```
else if (abs(Xdiff( n )) < 92) then
```
2775
```
    use DPCM4
```
2776
```
else if (abs(Xdiff( n )) < 220) then
```
2777
```
    use DPCM5
```
2778
```
else if (abs(Xdiff( n )) < 348) then
```
2779
```
    use DPCM6
```
2780
```
else
```
2781
```
    use PCM
```
2782
```
endif
```

E.2.6.1   DPCM1 for 12–7–12 Coder

2783   **Xenco**(**n**) has the following format:

2784
```
Xenco( n ) = "0000 s xx"
```

2785   where,

2786
```
"0000" is the code word
```
2787
```
"s" is the sign bit
```
2788
```
"xx" is the two bit value field
```

2789   The coder equation is described as follows:

2790
```
if (Xdiff( n ) <= 0) then
```
2791
```
    sign = 1
```
2792
```
else
```
2793
```
    sign = 0
```
2794
```
endif
```
2795
```
value = abs(Xdiff( n ))
```

2796   *Note: Zero code has been avoided (0 is sent as -0).*

E.2.6.2   DPCM2 for 12–7–12 Coder

2797 **Xenco**(**n**) has the following format:

2798   `Xenco( n ) = "0001 s xx"`

2799 where,

2800   `"0001" is the code word`
2801   `"s" is the `**`sign`**` bit`
2802   `"xx" is the two bit `**`value`**` field`

2803 The coder equation is described as follows:

2804   `if (`**`Xdiff`**`( n ) < 0) then`
2805   **`sign`**` = 1`
2806   `else`
2807   **`sign`**` = 0`
2808   `endif`
2809   **`value`**` = (abs(`**`Xdiff`**`( n )) - 4) / 2`

E.2.6.3   DPCM3 for 12–7–12 Coder

2810 **Xenco**(**n**) has the following format:

2811   `Xenco( n ) = "0010 s xx"`

2812 where,

2813   `"0010" is the code word`
2814   `"s" is the `**`sign`**` bit`
2815   `"xx" is the two bit `**`value`**` field`

2816 The coder equation is described as follows:

2817   `if (`**`Xdiff`**`( n ) < 0) then`
2818   **`sign`**` = 1`
2819   `else`
2820   **`sign`**` = 0`
2821   `endif`
2822   **`value`**` = (abs(`**`Xdiff`**`( n )) - 12) / 4`

E.2.6.4   DPCM4 for 12–7–12 Coder

2823 **Xenco**(**n**) has the following format:

2824   `Xenco( n ) = "010 s xxx"`

2825 where,

2826   `"010" is the code word`
2827   `"s" is the `**`sign`**` bit`
2828   `"xxx" is the three bit `**`value`**` field`

2829 The coder equation is described as follows:

2830   `if (`**`Xdiff`**`( n ) < 0) then`
2831   **`sign`**` = 1`
2832   `else`
2833   **`sign`**` = 0`
2834   `endif`
2835   **`value`**` = (abs(`**`Xdiff`**`( n )) - 28) / 8`

E.2.6.5   DPCM5 for 12–7–12 Coder

2836   **Xenco**(**n**) has the following format:

2837   `Xenco( n ) = "011 s xxx"`

2838   where,

2839   `"011" is the code word`
2840   `"s" is the` **sign** `bit`
2841   `"xxx" is the three bit` **value** `field`

2842   The coder equation is described as follows:

2843   `if (`**Xdiff**`( n ) < 0) then`
2844   **sign** `= 1`
2845   `else`
2846   **sign** `= 0`
2847   `endif`
2848   **value** `= (abs(`**Xdiff**`( n )) - 92) / 16`

E.2.6.6   DPCM6 for 12–7–12 Coder

2849   **Xenco**(**n**) has the following format:

2850   `Xenco( n ) = "0011 s xx"`

2851   where,

2852   `"0011" is the code word`
2853   `"s" is the` **sign** `bit`
2854   `"xx" is the two bit` **value** `field`

2855   The coder equation is described as follows:

2856   `if (`**Xdiff**`( n ) < 0) then`
2857   **sign** `= 1`
2858   `else`
2859   **sign** `= 0`
2860   `endif`
2861   **value** `= (abs(`**Xdiff**`( n )) - 220) / 32`

E.2.6.7   PCM for 12–7–12 Coder

2862   **Xenco**(**n**) has the following format:

2863   `Xenco( n ) = "1 xxxxxx"`

2864   where,

2865   `"1" is the code word`
2866   `the` **sign** `bit is not used`
2867   `"xxxxxx" is the six bit` **value** `field`

2868   The coder equation is described as follows:

2869   **value** `=` **Xorig**`( n ) / 64`

E.2.7   Coder for 12–6–12 Data Compression

2870   The 12–6–12 coder offers 50% bit rate reduction with acceptable image quality.

2871   Pixels without prediction are encoded using the following formula:

2872
```
Xenco( n ) = Xorig( n ) / 64
```

2873   To avoid a full-zero encoded value, the following check is performed:

2874
2875
2876
```
if (Xenco( n ) == 0) then
    Xenco( n ) = 1
endif
```

2877   Pixels with prediction are encoded using the following formula:

2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
```
if (abs(Xdiff( n )) < 2) then
    use DPCM1
else if (abs(Xdiff( n )) < 10) then
    use DPCM3
else if (abs(Xdiff( n )) < 42) then
    use DPCM4
else if (abs(Xdiff( n )) < 74) then
    use DPCM5
else if (abs(Xdiff( n )) < 202) then
    use DPCM6
else if (abs(Xdiff( n )) < 330) then
    use DPCM7
else
    use PCM
endif
```

2893   *Note: **DPCM2** is not used.*

E.2.7.1   DPCM1 for 12–6–12 Coder

2894   **Xenco**(**n**) has the following format:

2895
```
Xenco( n ) = "0000 s x"
```

2896   where,

2897
2898
2899
```
"0000" is the code word
"s" is the sign bit
"x" is the one bit value field
```

2900   The coder equation is described as follows:

2901
2902
2903
2904
2905
2906
```
if (Xdiff( n ) <= 0) then
    sign = 1
else
    sign = 0
endif
value = abs(Xdiff( n ))
```

2907   *Note: Zero code has been avoided (0 is sent as -0).*

E.2.7.2 DPCM3 for 12–6–12 Coder

2908 **Xenco**(**n**) has the following format:
2909   `Xenco( n ) = "0001 s x"`

2910 where,
2911   `"0001" is the code word`
2912   `"s" is the` **sign** `bit`
2913   `"x" is the one bit` **value** `field`

2914 The coder equation is described as follows:
2915   `if (`**Xdiff**`( n ) < 0) then`
2916   **sign** `= 1`
2917   `else`
2918   **sign** `= 0`
2919   `endif`
2920   **value** `= (abs(`**Xdiff**`( n )) – 2) / 4`

E.2.7.3 DPCM4 for 12–6–12 Coder

2921 **Xenco**(**n**) has the following format:
2922   `Xenco( n ) = "010 s xx"`

2923 where,
2924   `"010" is the code word`
2925   `"s" is the` **sign** `bit`
2926   `"xx" is the two bit` **value** `field`

2927 The coder equation is described as follows:
2928   `if (`**Xdiff**`( n ) < 0) then`
2929   **sign** `= 1`
2930   `else`
2931   **sign** `= 0`
2932   `endif`
2933   **value** `= (abs(`**Xdiff**`( n )) – 10) / 8`

E.2.7.4 DPCM5 for 12–6–12 Coder

2934 **Xenco**(**n**) has the following format:
2935   `Xenco( n ) = "0010 s x"`

2936 where,
2937   `"0010" is the code word`
2938   `"s" is the` **sign** `bit`
2939   `"x" is the one bit` **value** `field`

2940 The coder equation is described as follows:
2941   `if (`**Xdiff**`( n ) < 0) then`
2942   **sign** `= 1`
2943   `else`
2944   **sign** `= 0`
2945   `endif`
2946   **value** `= (abs(`**Xdiff**`( n )) – 42) / 16`

E.2.7.5   DPCM6 for 12–6–12 Coder

2947 **Xenco**(**n**) has the following format:

2948   `Xenco( n ) = "011 s xx"`

2949 where,

2950   `"011" is the code word`
2951   `"s" is the `**`sign`**` bit`
2952   `"xx" is the two bit `**`value`**` field`

2953 The coder equation is described as follows:

2954   `if (`**`Xdiff`**`( n ) < 0) then`
2955   **`sign`**` = 1`
2956   `else`
2957   **`sign`**` = 0`
2958   `endif`
2959   **`value`**` = (abs(`**`Xdiff`**`( n )) - 74) / 32`

E.2.7.6   DPCM7 for 12–6–12 Coder

2960 **Xenco**(**n**) has the following format:

2961   `Xenco( n ) = "0011 s x"`

2962 where,

2963   `"0011" is the code word`
2964   `"s" is the `**`sign`**` bit`
2965   `"x" is the one bit `**`value`**` field`

2966 The coder equation is described as follows:

2967   `if (`**`Xdiff`**`( n ) < 0) then`
2968   **`sign`**` = 1`
2969   `else`
2970   **`sign`**` = 0`
2971   `endif`
2972   **`value`**` = (abs(`**`Xdiff`**`( n )) - 202) / 64`

E.2.7.7   PCM for 12–6–12 Coder

2973 **Xenco**(**n**) has the following format:

2974   `Xenco( n ) = "1 xxxxx"`

2975 where,

2976   `"1" is the code word`
2977   `the `**`sign`**` bit is not used`
2978   `"xxxxx" is the five bit `**`value`**` field`

2979 The coder equation is described as follows:

2980   **`value`**` = `**`Xorig`**`( n ) / 128`

E.3   Decoders

2981   There are six different decoders available, one for each data compression scheme.

2982   For all decoders, the formula used for non-predicted pixels (beginning of lines) is different than the formula
2983   for predicted pixels.

E.3.1   Decoder for 10–8–10 Data Compression

2984   Pixels without prediction are decoded using the following formula:

2985   **Xdeco(n) = 4 * Xenco(n) + 2**

2986   Pixels with prediction are decoded using the following formula:

2987   if (**Xenco(n)** & 0xc0 == 0x00) then
2988   use **DPCM1**
2989   else if (**Xenco(n)** & 0xe0 == 0x40) then
2990   use **DPCM2**
2991   else if (**Xenco(n)** & 0xe0 == 0x60) then
2992   use **DPCM3**
2993   else
2994   use **PCM**
2995   endif

E.3.1.1   DPCM1 for 10–8–10 Decoder

2996   **Xenco(n)** has the following format:

2997   **Xenco(n)** = "00 s xxxxx"

2998   where,

2999   "00" is the code word
3000   "s" is the **sign** bit
3001   "xxxxx" is the five bit **value** field

3002   The decoder equation is described as follows:

3003   **sign** = **Xenco(n)** & 0x20
3004   **value** = **Xenco(n)** & 0x1f
3005   if (**sign** > 0) then
3006   **Xdeco(n) = Xpred(n)** – **value**
3007   else
3008   **Xdeco(n) = Xpred(n)** + **value**
3009   endif

E.3.1.2   DPCM2 for 10–8–10 Decoder

3010 **Xenco**(**n**) has the following format:

3011   `Xenco( ` **n** ` ) = "010 s xxxx"`

3012 where,

3013   `"010" is the code word`
3014   `"s" is the ` **sign** ` bit`
3015   `"xxxx" is the four bit ` **value** ` field`

3016 The decoder equation is described as follows:

3017   **sign** ` = ` **Xenco**`( ` **n** ` ) & 0x10`
3018   **value** ` = 2 * (` **Xenco**`( ` **n** ` ) & 0xf) + 32`
3019   `if (` **sign** ` > 0) then`
3020   **Xdeco**`( ` **n** ` ) = ` **Xpred**`( ` **n** ` ) – ` **value**
3021   `else`
3022   **Xdeco**`( ` **n** ` ) = ` **Xpred**`( ` **n** ` ) + ` **value**
3023   `endif`

E.3.1.3   DPCM3 for 10–8–10 Decoder

3024 **Xenco**(**n**) has the following format:

3025   `Xenco( ` **n** ` ) = "011 s xxxx"`

3026 where,

3027   `"011" is the code word`
3028   `"s" is the ` **sign** ` bit`
3029   `"xxxx" is the four bit ` **value** ` field`

3030 The decoder equation is described as follows:

3031   **sign** ` = ` **Xenco**`( ` **n** ` ) & 0x10`
3032   **value** ` = 4 * (` **Xenco**`( ` **n** ` ) & 0xf) + 64 + 1`
3033   `if (` **sign** ` > 0) then`
3034   **Xdeco**`( ` **n** ` ) = ` **Xpred**`( ` **n** ` ) – ` **value**
3035   `if (` **Xdeco**`( ` **n** ` ) < 0) then`
3036   **Xdeco**`( ` **n** ` ) = 0`
3037   `endif`
3038   `else`
3039   **Xdeco**`( ` **n** ` ) = ` **Xpred**`( ` **n** ` ) + ` **value**
3040   `if (` **Xdeco**`( ` **n** ` ) > 1023) then`
3041   **Xdeco**`( ` **n** ` ) = 1023`
3042   `endif`
3043   `endif`

E.3.1.4   PCM for 10–8–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = "1 xxxxxxx"
```

where,

```
"1" is the code word
the sign bit is not used
"xxxxxxx" is the seven bit value field
```

The codec equation is described as follows:

```
value = 8 * (Xenco( n ) & 0x7f)
if (value > Xpred( n )) then
    Xdeco( n ) = value + 3
endif
else
    Xdeco( n ) = value + 4
endif
```

E.3.2   Decoder for 10–7–10 Data Compression

3058   Pixels without prediction are decoded using the following formula:

3059   `Xdeco( n ) = 8 * Xenco( n ) + 4`

3060   Pixels with prediction are decoded using the following formula:

```
3061      if (Xenco( n ) & 0x70 == 0x00) then
3062         use DPCM1
3063      else if (Xenco( n ) & 0x78 == 0x10) then
3064         use DPCM2
3065      else if (Xenco( n ) & 0x78 == 0x18) then
3066         use DPCM3
3067      else if (Xenco( n ) & 0x60 == 0x20) then
3068         use DPCM4
3069      else
3070         use PCM
3071      endif
```

E.3.2.1   DPCM1 for 10–7–10 Decoder

3072   **Xenco**(**n**) has the following format:

3073   `Xenco( n ) = "000 s xxx"`

3074   where,

```
3075      "000" is the code word
3076      "s" is the sign bit
3077      "xxx" is the three bit value field
```

3078   The codec equation is described as follows:

```
3079      sign = Xenco( n ) & 0x8
3080      value = Xenco( n ) & 0x7
3081      if (sign > 0) then
3082         Xdeco( n ) = Xpred( n ) – value
3083      else
3084         Xdeco( n ) = Xpred( n ) + value
3085      endif
```

E.3.2.2   DPCM2 for 10–7–10 Decoder

3086   **Xenco**(**n**) has the following format:

3087   `Xenco( n ) = "0010 s xx"`

3088   where,

```
3089      "0010" is the code word
3090      "s" is the sign bit
3091      "xx" is the two bit value field
```

3092   The codec equation is described as follows:

```
3093      sign = Xenco( n ) & 0x4
3094      value = 2 * (Xenco( n ) & 0x3) + 8
3095      if (sign > 0) then
3096         Xdeco( n ) = Xpred( n ) – value
3097      else
3098         Xdeco( n ) = Xpred( n ) + value
3099      endif
```

E.3.2.3   DPCM3 for 10–7–10 Decoder

3100 **Xenco**(**n**) has the following format:

3101   `Xenco( n ) = "0011 s xx"`

3102 where,

3103   `"0011" is the code word`
3104   `"s" is the` **sign** `bit`
3105   `"xx" is the two bit` **value** `field`

3106 The codec equation is described as follows:

```
3107      sign = Xenco( n ) & 0x4
3108      value = 4 * (Xenco( n ) & 0x3) + 16 + 1
3109      if (sign > 0) then
3110          Xdeco( n ) = Xpred( n ) – value
3111          if (Xdeco( n ) < 0) then
3112              Xdeco( n ) = 0
3113          endif
3114      else
3115          Xdeco( n ) = Xpred( n ) + value
3116          if (Xdeco( n ) > 1023) then
3117              Xdeco( n ) = 1023
3118          endif
3119      endif
```

E.3.2.4   DPCM4 for 10–7–10 Decoder

3120 **Xenco**(**n**) has the following format:

3121   `Xenco( n ) = "01 s xxxx"`

3122 where,

3123   `"01" is the code word`
3124   `"s" is the` **sign** `bit`
3125   `"xxxx" is the four bit` **value** `field`

3126 The codec equation is described as follows:

```
3127      sign = Xenco( n ) & 0x10
3128      value = 8 * (Xenco( n ) & 0xf) + 32 + 3
3129      if (sign > 0) then
3130          Xdeco( n ) = Xpred( n ) – value
3131          if (Xdeco( n ) < 0) then
3132              Xdeco( n ) = 0
3133          endif
3134      else
3135          Xdeco( n ) = Xpred( n ) + value
3136          if (Xdeco( n ) > 1023) then
3137              Xdeco( n ) = 1023
3138          endif
3139      endif
```

E.3.2.5   PCM for 10–7–10 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = "1 xxxxxx"
```

where,

```
"1" is the code word
the sign bit is not used
"xxxxxx" is the six bit value field
```

The codec equation is described as follows:

```
value = 16 * (Xenco( n ) & 0x3f)
if (value > Xpred( n )) then
    Xdeco( n ) = value + 7
else
    Xdeco( n ) = value + 8
endif
```

E.3.3　　Decoder for 10–6–10 Data Compression

3153　Pixels without prediction are decoded using the following formula:

```
Xdeco( n ) = 16 * Xenco( n ) + 8
```

3155　Pixels with prediction are decoded using the following formula:

```
if (Xenco( n ) & 0x3e == 0x00) then
   use DPCM1
else if (Xenco( n ) & 0x3e == 0x02) then
   use DPCM2
else if (Xenco( n ) & 0x3c == 0x04) then
   use DPCM3
else if (Xenco( n ) & 0x38 == 0x08) then
   use DPCM4
else if (Xenco( n ) & 0x30 == 0x10) then
   use DPCM5
else
   use PCM
endif
```

E.3.3.1　　DPCM1 for 10–6–10 Decoder

3169　**Xenco**(**n**) has the following format:

```
Xenco( n ) = "00000 s"
```

3171　where,

```
"00000" is the code word
"s" is the sign bit
the value field is not used
```

3175　The codec equation is described as follows:

```
Xdeco( n ) = Xpred( n )
```

E.3.3.2　　DPCM2 for 10–6–10 Decoder

3177　**Xenco**(**n**) has the following format:

```
Xenco( n ) = "00001 s"
```

3179　where,

```
"00001" is the code word
"s" is the sign bit
the value field is not used
```

3183　The codec equation is described as follows:

```
sign = Xenco( n ) & 0x1
value = 1
if (sign > 0) then
   Xdeco( n ) = Xpred( n ) – value
else
   Xdeco( n ) = Xpred( n ) + value
endif
```

E.3.3.3   DPCM3 for 10–6–10 Decoder

3191   **Xenco**(**n**) has the following format:

3192
```
Xenco( n ) = "0001 s x"
```

3193   where,

3194
```
"0001" is the code word
```
3195
```
"s" is the sign bit
```
3196
```
"x" is the one bit value field
```

3197   The codec equation is described as follows:

3198
```
sign = Xenco( n ) & 0x2
```
3199
```
value = 4 * (Xenco( n ) & 0x1) + 3 + 1
```
3200
```
if (sign > 0) then
```
3201
```
    Xdeco( n ) = Xpred( n ) - value
```
3202
```
    if (Xdeco( n ) < 0) then
```
3203
```
        Xdeco( n ) = 0
```
3204
```
    endif
```
3205
```
else
```
3206
```
    Xdeco( n ) = Xpred( n ) + value
```
3207
```
    if (Xdeco( n ) > 1023) then
```
3208
```
        Xdeco( n ) = 1023
```
3209
```
    endif
```
3210
```
endif
```

E.3.3.4   DPCM4 for 10–6–10 Decoder

3211   **Xenco**(**n**) has the following format:

3212
```
Xenco( n ) = "001 s xx"
```

3213   where,

3214
```
"001" is the code word
```
3215
```
"s" is the sign bit
```
3216
```
"xx" is the two bit value field
```

3217   The codec equation is described as follows:

3218
```
sign = Xenco( n ) & 0x4
```
3219
```
value = 8 * (Xenco( n ) & 0x3) + 11 + 3
```
3220
```
if (sign > 0) then
```
3221
```
    Xdeco( n ) = Xpred( n ) - value
```
3222
```
    if (Xdeco( n ) < 0) then
```
3223
```
        Xdeco( n ) = 0
```
3224
```
    endif
```
3225
```
else
```
3226
```
    Xdeco( n ) = Xpred( n ) + value
```
3227
```
    if (Xdeco( n ) > 1023) then
```
3228
```
        Xdeco( n ) = 1023
```
3229
```
    endif
```
3230
```
endif
```

E.3.3.5   DPCM5 for 10–6–10 Decoder

3231 **Xenco**(**n**) has the following format:

3232   **Xenco**(**n**) = "01 s xxx"

3233 where,

3234   "01" is the code word
3235   "s" is the **sign** bit
3236   "xxx" is the three bit **value** field

3237 The codec equation is described as follows:

3238   **sign** = **Xenco**(**n**) & 0x8
3239   **value** = 16 * (**Xenco**(**n**) & 0x7) + 43 + 7
3240   if (**sign** > 0) then
3241   **Xdeco**(**n**) = **Xpred**(**n**) – **value**
3242   if (**Xdeco**(**n**) < 0) then
3243   **Xdeco**(**n**) = 0
3244   endif
3245   else
3246   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3247   if (**Xdeco**(**n**) > 1023) then
3248   **Xdeco**(**n**) = 1023
3249   endif
3250   endif

E.3.3.6   PCM for 10–6–10 Decoder

3251 **Xenco**(**n**) has the following format:

3252   **Xenco**(**n**) = "1 xxxxx"

3253 where,

3254   "1" is the code word
3255   the **sign** bit is not used
3256   "xxxxx" is the five bit **value** field

3257 The codec equation is described as follows:

3258   **value** = 32 * (**Xenco**(**n**) & 0x1f)
3259   if (**value** > **Xpred**(**n**)) then
3260   **Xdeco**(**n**) = **value** + 15
3261   else
3262   **Xdeco**(**n**) = **value** + 16
3263   endif

E.3.4 Decoder for 12–10–12 Data Compression

3264   Pixels without prediction are decoded using the following formula:

3265   **Xdeco**(**n**) = 4 * **Xenco**(**n**) + 2

3266   Pixels with prediction are decoded using the following formula:

```
3267    if (Xenco( n ) & 0x300 == 0x000) then
3268        use DPCM1
3269    else if (Xenco( n ) & 0x380 == 0x100) then
3270        use DPCM2
3271    else if (Xenco( n ) & 0x380 == 0x180) then
3272        use DPCM3
3273    else
3274        use PCM
3275    endif
```

E.3.4.1 DPCM1 for 12–10–12 Decoder

3276   **Xenco**(**n**) has the following format:

3277   **Xenco**(**n**) = "00 s xxxxxxx"

3278   where,

3279   "00" is the code word
3280   "s" is the **sign** bit
3281   "xxxxxxx" is the seven bit **value** field

3282   The decoder equation is described as follows:

```
3283    sign = Xenco( n ) & 0x80
3284    value = Xenco( n ) & 0x7f
3285    if (sign > 0) then
3286        Xdeco( n ) = Xpred( n ) – value
3287    else
3288        Xdeco( n ) = Xpred( n ) + value
3289    endif
```

E.3.4.2   DPCM2 for 12–10–12 Decoder

3290 **Xenco**(**n**) has the following format:

3291   `Xenco( n ) = "010 s xxxxxx"`

3292 where,

3293   `"010"` is the code word
3294   `"s"` is the **sign** bit
3295   `"xxxxxx"` is the six bit **value** field

3296 The decoder equation is described as follows:

```
3297     sign = Xenco( n ) & 0x40
3298     value = 2 * (Xenco( n ) & 0x3f) + 128
3299     if (sign > 0) then
3300         Xdeco( n ) = Xpred( n ) – value
3301     else
3302         Xdeco( n ) = Xpred( n ) + value
3303     endif
```

E.3.4.3   DPCM3 for 12–10–12 Decoder

3304 **Xenco**(**n**) has the following format:

3305   `Xenco( n ) = "011 s xxxxxx"`

3306 where,

3307   `"011"` is the code word
3308   `"s"` is the **sign** bit
3309   `"xxxxxx"` is the six bit **value** field

3310 The decoder equation is described as follows:

```
3311     sign = Xenco( n ) & 0x40
3312     value = 4 * (Xenco( n ) & 0x3f) + 256 + 1
3313     if (sign > 0) then
3314         Xdeco( n ) = Xpred( n ) – value
3315         if (Xdeco( n ) < 0) then
3316             Xdeco( n ) = 0
3317         endif
3318     else
3319         Xdeco( n ) = Xpred( n ) + value
3320         if (Xdeco( n ) > 4095) then
3321             Xdeco( n ) = 4095
3322         endif
3323     endif
```

E.3.4.4   PCM for 12–10–12 Decoder

Xenco(**n**) has the following format:

```
Xenco( n ) = "1 xxxxxxxxx"
```

where,

```
"1" is the code word
the sign bit is not used
"xxxxxxxxx" is the nine bit value field
```

The codec equation is described as follows:

```
value = 8 * (Xenco( n ) & 0x1ff)
if (value > Xpred( n )) then
    Xdeco( n ) = value + 3
endif
else
    Xdeco( n ) = value + 4
endif
```

E.3.5   Decoder for 12–8–12 Data Compression

3338   Pixels without prediction are decoded using the following formula:

3339
```
Xdeco( n ) = 16 * Xenco( n ) + 8
```

3340   Pixels with prediction are decoded using the following formula:

3341
```
if (Xenco( n ) & 0xf0 == 0x00) then
    use DPCM1
else if (Xenco( n ) & 0xe0 == 0x60) then
    use DPCM2
else if (Xenco( n ) & 0xe0 == 0x40) then
    use DPCM3
else if (Xenco( n ) & 0xe0 == 0x20) then
    use DPCM4
else if (Xenco( n ) & 0xf0 == 0x10) then
    use DPCM5
else
    use PCM
endif
```

E.3.5.1   DPCM1 for 12–8–12 Decoder

3354   **Xenco**(**n**) has the following format:

3355
```
Xenco( n ) = "0000 s xxx"
```

3356   where,

3357
```
"0000" is the code word
"s" is the sign bit
"xxx" is the three bit value field
```

3360   The codec equation is described as follows:

3361
```
sign = Xenco( n ) & 0x8
value = Xenco( n ) & 0x7
if (sign > 0) then
    Xdeco( n ) = Xpred( n ) – value
else
    Xdeco( n ) = Xpred( n ) + value
endif
```

E.3.5.2   DPCM2 for 12–8–12 Decoder

3368   **Xenco**(**n**) has the following format:

3369
```
Xenco( n ) = "011 s xxxx"
```

3370   where,

3371
```
"011" is the code word
"s" is the sign bit
"xxxx" is the four bit value field
```

3374   The codec equation is described as follows:

3375
```
sign = Xenco( n ) & 0x10
value = 2 * (Xenco( n ) & 0xf) + 8
if (sign > 0) then
    Xdeco( n ) = Xpred( n ) – value
else
    Xdeco( n ) = Xpred( n ) + value
endif
```

E.3.5.3   DPCM3 for 12–8–12 Decoder

3382 **Xenco**(**n**) has the following format:

3383   **Xenco**(**n**) = "010 s xxxx"

3384 where,

3385   "010" is the code word
3386   "s" is the **sign** bit
3387   "xxxx" is the four bit **value** field

3388 The codec equation is described as follows:

```
3389    sign = Xenco( n ) & 0x10
3390    value = 4 * (Xenco( n ) & 0xf) + 40 + 1
3391    if (sign > 0) then
3392        Xdeco( n ) = Xpred( n ) – value
3393        if (Xdeco( n ) < 0) then
3394            Xdeco( n ) = 0
3395        endif
3396    else
3397        Xdeco( n ) = Xpred( n ) + value
3398        if (Xdeco( n ) > 4095) then
3399            Xdeco( n ) = 4095
3400        endif
3401    endif
```

E.3.5.4   DPCM4 for 12–8–12 Decoder

3402 **Xenco**(**n**) has the following format:

3403   **Xenco**(**n**) = "001 s xxxx"

3404 where,

3405   "001" is the code word
3406   "s" is the **sign** bit
3407   "xxxx" is the four bit **value** field

3408 The codec equation is described as follows:

```
3409    sign = Xenco( n ) & 0x10
3410    value = 8 * (Xenco( n ) & 0xf) + 104 + 3
3411    if (sign > 0) then
3412        Xdeco( n ) = Xpred( n ) – value
3413        if (Xdeco( n ) < 0) then
3414            Xdeco( n ) = 0
3415        endif
3416    else
3417        Xdeco( n ) = Xpred( n ) + value
3418        if (Xdeco( n ) > 4095)
3419            Xdeco( n ) = 4095
3420        endif
3421    endif
```

E.3.5.5   DPCM5 for 12–8–12 Decoder

3422   **Xenco**(**n**) has the following format:

3423   `Xenco( n ) = "0001 s xxx"`

3424   where,

3425   `"0001" is the code word`
3426   `"s" is the` **sign** `bit`
3427   `"xxx" is the three bit` **value** `field`

3428   The codec equation is described as follows:

3429   **sign** = **Xenco**(**n**) & 0x8
3430   **value** = 16 * (**Xenco**(**n**) & 0x7) + 232 + 7
3431   if (**sign** > 0) then
3432   **Xdeco**(**n**) = **Xpred**(**n**) – **value**
3433   if (**Xdeco**(**n**) < 0) then
3434   **Xdeco**(**n**) = 0
3435   endif
3436   else
3437   **Xdeco**(**n**) = **Xpred**(**n**) + **value**
3438   if (**Xdeco**(**n**) > 4095) then
3439   **Xdeco**(**n**) = 4095
3440   endif
3441   endif

E.3.5.6   PCM for 12–8–12 Decoder

3442   **Xenco**(**n**) has the following format:

3443   `Xenco( n ) = "1 xxxxxxx"`

3444   where,

3445   `"1" is the code word`
3446   `the` **sign** `bit is not used`
3447   `"xxxxxxx" is the seven bit` **value** `field`

3448   The codec equation is described as follows:

3449   **value** = 32 * (**Xenco**(**n**) & 0x7f)
3450   if (**value** > **Xpred**(**n**)) then
3451   **Xdeco**(**n**) = **value** + 15
3452   else
3453   **Xdeco**(**n**) = **value** + 16
3454   endif

E.3.6   Decoder for 12–7–12 Data Compression

3455   Pixels without prediction are decoded using the following formula:

3456
```
Xdeco( n ) = 32 * Xenco( n ) + 16
```

3457   Pixels with prediction are decoded using the following formula:

```
3458    if (Xenco( n ) & 0x78 == 0x00) then
3459       use DPCM1
3460    else if (Xenco( n ) & 0x78 == 0x08) then
3461       use DPCM2
3462    else if (Xenco( n ) & 0x78 == 0x10) then
3463       use DPCM3
3464    else if (Xenco( n ) & 0x70 == 0x20) then
3465       use DPCM4
3466    else if (Xenco( n ) & 0x70 == 0x30) then
3467       use DPCM5
3468    else if (Xenco( n ) & 0x78 == 0x18) then
3469       use DPCM6
3470    else
3471       use PCM
3472    endif
```

E.3.6.1   DPCM1 for 12–7–12 Decoder

3473   **Xenco(n)** has the following format:

3474
```
Xenco( n ) = "0000 s xx"
```

3475   where,

```
3476    "0000" is the code word
3477    "s" is the sign bit
3478    "xx" is the two bit value field
```

3479   The codec equation is described as follows:

```
3480    sign = Xenco( n ) & 0x4
3481    value = Xenco( n ) & 0x3
3482    if (sign > 0) then
3483       Xdeco( n ) = Xpred( n ) – value
3484    else
3485       Xdeco( n ) = Xpred( n ) + value
3486    endif
```

E.3.6.2   DPCM2 for 12–7–12 Decoder

3487 **Xenco**(**n**) has the following format:

3488   **Xenco(n) = "0001 s xx"**

3489 where,

3490   "0001" is the code word
3491   "s" is the **sign** bit
3492   "xx" is the two bit **value** field

3493 The codec equation is described as follows:

3494   **sign** = **Xenco(n)** & 0x4
3495   **value** = 2 * (**Xenco(n)** & 0x3) + 4
3496   if (**sign** > 0) then
3497   **Xdeco(n)** = **Xpred(n)** – **value**
3498   else
3499   **Xdeco(n)** = **Xpred(n)** + **value**
3500   endif

E.3.6.3   DPCM3 for 12–7–12 Decoder

3501 **Xenco**(**n**) has the following format:

3502   **Xenco(n) = "0010 s xx"**

3503 where,

3504   "0010" is the code word
3505   "s" is the **sign** bit
3506   "xx" is the two bit **value** field

3507 The codec equation is described as follows:

3508   **sign** = **Xenco(n)** & 0x4
3509   **value** = 4 * (**Xenco(n)** & 0x3) + 12 + 1
3510   if (**sign** > 0) then
3511   **Xdeco(n)** = **Xpred(n)** – **value**
3512   if (**Xdeco(n)** < 0) then
3513   **Xdeco(n)** = 0
3514   endif
3515   else
3516   **Xdeco(n)** = **Xpred(n)** + **value**
3517   if (**Xdeco(n)** > 4095) then
3518   **Xdeco(n)** = 4095
3519   endif
3520   endif

Copyright © 2005-2018 MIPI Alliance, Inc.
Confidential

E.3.6.4   DPCM4 for 12–7–12 Decoder

3521 **Xenco**(**n**) has the following format:

3522   `Xenco`(**n**) = "010 s xxx"

3523 where,

3524   "010" is the code word
3525   "s" is the **sign** bit
3526   "xxx" is the three bit **value** field

3527 The codec equation is described as follows:

```
3528     sign = Xenco( n ) & 0x8
3529     value = 8 * (Xenco( n ) & 0x7) + 28 + 3
3530     if (sign > 0) then
3531         Xdeco( n ) = Xpred( n ) – value
3532         if (Xdeco( n ) < 0) then
3533             Xdeco( n ) = 0
3534         endif
3535     else
3536         Xdeco( n ) = Xpred( n ) + value
3537         if (Xdeco( n ) > 4095) then
3538             Xdeco( n ) = 4095
3539         endif
3540     endif
```

E.3.6.5   DPCM5 for 12–7–12 Decoder

3541 **Xenco**(**n**) has the following format:

3542   `Xenco`(**n**) = "011 s xxx"

3543 where,

3544   "011" is the code word
3545   "s" is the **sign** bit
3546   "xxx" is the three bit **value** field

3547 The codec equation is described as follows:

```
3548     sign = Xenco( n ) & 0x8
3549     value = 16 * (Xenco( n ) & 0x7) + 92 + 7
3550     if (sign > 0) then
3551         Xdeco( n ) = Xpred( n ) – value
3552         if (Xdeco( n ) < 0) then
3553             Xdeco( n ) = 0
3554         endif
3555     else
3556         Xdeco( n ) = Xpred( n ) + value
3557         if (Xdeco( n ) > 4095) then
3558             Xdeco( n ) = 4095
3559         endif
3560     endif
```

E.3.6.6 DPCM6 for 12–7–12 Decoder

3561 **Xenco**(**n**) has the following format:

3562   `Xenco( n ) = "0011 s xx"`

3563 where,

3564   `"0011" is the code word`
3565   `"s" is the `**`sign`**` bit`
3566   `"xx" is the two bit `**`value`**` field`

3567 The codec equation is described as follows:

```
3568      sign = Xenco( n ) & 0x4
3569      value = 32 * (Xenco( n ) & 0x3) + 220 + 15
3570      if (sign > 0) then
3571          Xdeco( n ) = Xpred( n ) – value
3572          if (Xdeco( n ) < 0) then
3573              Xdeco( n ) = 0
3574          endif
3575      else
3576          Xdeco( n ) = Xpred( n ) + value
3577          if (Xdeco( n ) > 4095) then
3578              Xdeco( n ) = 4095
3579          endif
3580      endif
```

E.3.6.7 PCM for 12–7–12 Decoder

3581 **Xenco**(**n**) has the following format:

3582   `Xenco( n ) = "1 xxxxxx"`

3583 where,

3584   `"1" is the code word`
3585   `the `**`sign`**` bit is not used`
3586   `"xxxxxx" is the six bit `**`value`**` field`

3587 The codec equation is described as follows:

```
3588      value = 64 * (Xenco( n ) & 0x3f)
3589      if (value > Xpred( n )) then
3590          Xdeco( n ) = value + 31
3591      else
3592          Xdeco( n ) = value + 32
3593      endif
```

E.3.7　　Decoder for 12–6–12 Data Compression

3594　Pixels without prediction are decoded using the following formula:

```
Xdeco( n ) = 64 * Xenco( n ) + 32
```

3596　Pixels with prediction are decoded using the following formula:

```
if (Xenco( n ) & 0x3c == 0x00) then
    use DPCM1
else if (Xenco( n ) & 0x3c == 0x04) then
    use DPCM3
else if (Xenco( n ) & 0x38 == 0x10) then
    use DPCM4
else if (Xenco( n ) & 0x3c == 0x08) then
    use DPCM5
else if (Xenco( n ) & 0x38 == 0x18) then
    use DPCM6
else if (Xenco( n ) & 0x3c == 0x0c) then
    use DPCM7
else
    use PCM
endif
```

3612　*Note: **DPCM2** is not used.*

E.3.7.1　　DPCM1 for 12–6–12 Decoder

3613　**Xenco**(**n**) has the following format:

```
Xenco( n ) = "0000 s x"
```

3615　where,

```
"0000" is the code word
"s" is the sign bit
"x" is the one bit value field
```

3619　The codec equation is described as follows:

```
sign = Xenco( n ) & 0x2
value = Xenco( n ) & 0x1
if (sign > 0) then
    Xdeco( n ) = Xpred( n ) – value
else
    Xdeco( n ) = Xpred( n ) + value
endif
```

E.3.7.2   DPCM3 for 12–6–12 Decoder

3627 **Xenco**(**n**) has the following format:

3628   ```
Xenco(n) = "0001 s x"
```

3629  where,

3630      ```
"0001" is the code word
```
3631   ```
"s" is the sign bit
```
3632      ```
"x" is the one bit value field
```

3633 The codec equation is described as follows:

```
3634      sign = Xenco( n ) & 0x2
3635      value = 4 * (Xenco( n ) & 0x1) + 2 + 1
3636      if (sign > 0) then
3637          Xdeco( n ) = Xpred( n ) – value
3638          if (Xdeco( n ) < 0) then
3639              Xdeco( n ) = 0
3640          endif
3641      else
3642          Xdeco( n ) = Xpred( n ) + value
3643          if (Xdeco( n ) > 4095) then
3644              Xdeco( n ) = 4095
3645          endif
3646      endif
```

E.3.7.3   DPCM4 for 12–6–12 Decoder

3647 **Xenco**(**n**) has the following format:

3648   ```
Xenco(n) = "010 s xx"
```

3649  where,

3650      ```
"010" is the code word
```
3651   ```
"s" is the sign bit
```
3652      ```
"xx" is the two bit value field
```

3653 The codec equation is described as follows:

```
3654      sign = Xenco( n ) & 0x4
3655      value = 8 * (Xenco( n ) & 0x3) + 10 + 3
3656      if (sign > 0) then
3657          Xdeco( n ) = Xpred( n ) – value
3658          if (Xdeco( n ) < 0) then
3659              Xdeco( n ) = 0
3660          endif
3661      else
3662          Xdeco( n ) = Xpred( n ) + value
3663          if (Xdeco( n ) > 4095) then
3664              Xdeco( n ) = 4095
3665          endif
3666      endif
```

E.3.7.4   DPCM5 for 12–6–12 Decoder

3667   **Xenco**(**n**) has the following format:

3668
```
Xenco( n ) = "0010 s x"
```

3669   where,

3670
```
"0010" is the code word
```
3671
```
"s" is the sign bit
```
3672
```
"x" is the one bit value field
```

3673   The codec equation is described as follows:

3674
```
sign = Xenco( n ) & 0x2
```
3675
```
value = 16 * (Xenco( n ) & 0x1) + 42 + 7
```
3676
```
if (sign > 0) then
```
3677
```
    Xdeco( n ) = Xpred( n ) - value
```
3678
```
    if (Xdeco( n ) < 0) then
```
3679
```
        Xdeco( n ) = 0
```
3680
```
    endif
```
3681
```
else
```
3682
```
    Xdeco( n ) = Xpred( n ) + value
```
3683
```
    if (Xdeco( n ) > 4095) then
```
3684
```
        Xdeco( n ) = 4095
```
3685
```
    endif
```
3686
```
endif
```

E.3.7.5   DPCM6 for 12–6–12 Decoder

3687   **Xenco**(**n**) has the following format:

3688
```
Xenco( n ) = "011 s xx"
```

3689   where,

3690
```
"011" is the code word
```
3691
```
"s" is the sign bit
```
3692
```
"xx" is the two bit value field
```

3693   The codec equation is described as follows:

3694
```
sign = Xenco( n ) & 0x4
```
3695
```
value = 32 * (Xenco( n ) & 0x3) + 74 + 15
```
3696
```
if (sign > 0) then
```
3697
```
    Xdeco( n ) = Xpred( n ) - value
```
3698
```
    if (Xdeco( n ) < 0) then
```
3699
```
        Xdeco( n ) = 0
```
3700
```
    endif
```
3701
```
else
```
3702
```
    Xdeco( n ) = Xpred( n ) + value
```
3703
```
    if (Xdeco( n ) > 4095) then
```
3704
```
        Xdeco( n ) = 4095
```
3705
```
    endif
```
3706
```
endif
```

E.3.7.6   DPCM7 for 12–6–12 Decoder

3707   **Xenco(n)** has the following format:

3708   **Xenco(n)** = "0011 s x"

3709   where,

3710   "0011" is the code word
3711   "s" is the **sign** bit
3712   "x" is the one bit **value** field

3713   The codec equation is described as follows:

```
3714       sign = Xenco( n ) & 0x2
3715       value = 64 * (Xenco( n ) & 0x1) + 202 + 31
3716       if (sign > 0) then
3717           Xdeco( n ) = Xpred( n ) - value
3718           if (Xdeco( n ) < 0) then
3719               Xdeco( n ) = 0
3720           endif
3721       else
3722           Xdeco( n ) = Xpred( n ) + value
3723           if (Xdeco( n ) > 4095) then
3724               Xdeco( n ) = 4095
3725           endif
3726       endif
```

E.3.7.7   PCM for 12–6–12 Decoder

3727   **Xenco(n)** has the following format:

3728   **Xenco(n)** = "1 xxxxx"

3729   where,

3730   "1" is the code word
3731   the **sign** bit is not used
3732   "xxxxx" is the five bit **value** field

3733   The codec equation is described as follows:

```
3734       value = 128 * (Xenco( n ) & 0x1f)
3735       if (value > Xpred( n )) then
3736           Xdeco( n ) = value + 63
3737       else
3738           Xdeco( n ) = value + 64
3739       endif
```

Annex F JPEG Interleaving (informative)

3740 This annex illustrates how the standard features of the CSI-2 protocol should be used to interleave
3741 (multiplex) JPEG image data with other types of image data, e.g. RGB565 or YUV422, without requiring a
3742 custom JPEG format such as JPEG8.

3743 The Virtual Channel Identifier and Data Type value in the CSI-2 Packet Header provide simple methods of
3744 interleaving multiple data streams or image data types at the packet level. Interleaving at the packet level
3745 minimizes the amount of buffering required in the system.

3746 The Data Type value in the CSI-2 Packet Header should be used to multiplex different image data types at
3747 the CSI-2 transmitter and de-multiplex the data types at the CSI-2 receiver.

3748 The Virtual Channel Identifier in the CSI-2 Packet Header should be used to multiplex different data
3749 streams (channels) at the CSI-2 transmitter and de-multiplex the streams at the CSI-2 receiver.

3750 The main difference between the two interleaving methods is that images with different Data Type values
3751 within the same Virtual Channel use the same frame and line synchronization information, whereas
3752 multiple Virtual Channels (data streams) each have their own independent frame and line synchronization
3753 information and thus potentially each channel may have different frame rates.

3754 Since the predefined Data Type values represent only YUV, RGB and RAW data types, one of the User
3755 Defined Data Type values should be used to represent JPEG image data.

3756 *Figure 191* illustrates interleaving JPEG image data with YUV422 image data using Data Type values.

3757 *Figure 192* illustrates interleaving JPEG image data with YUV422 image data using both Data Type values
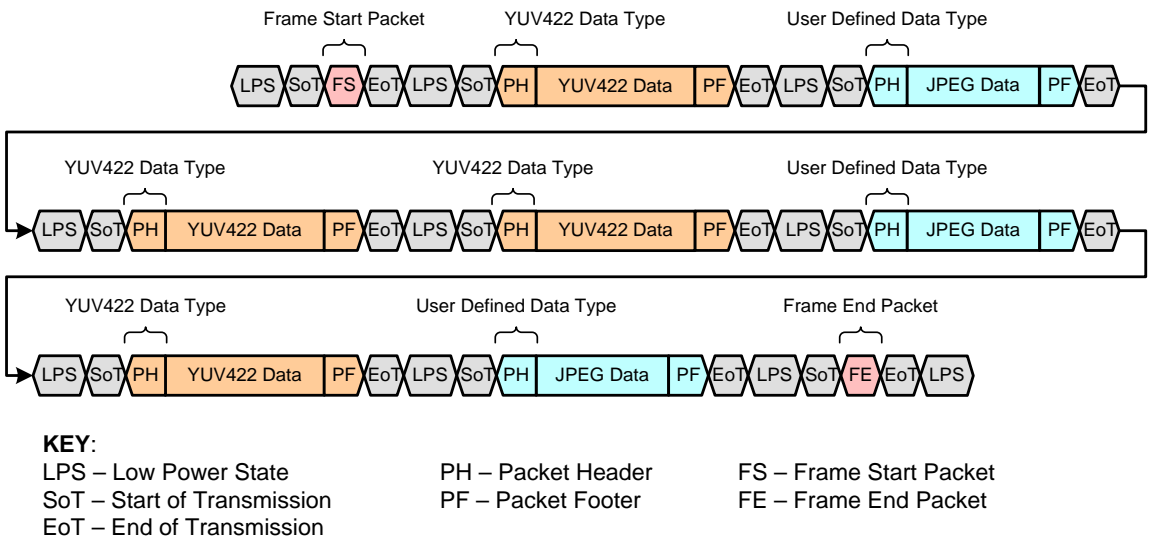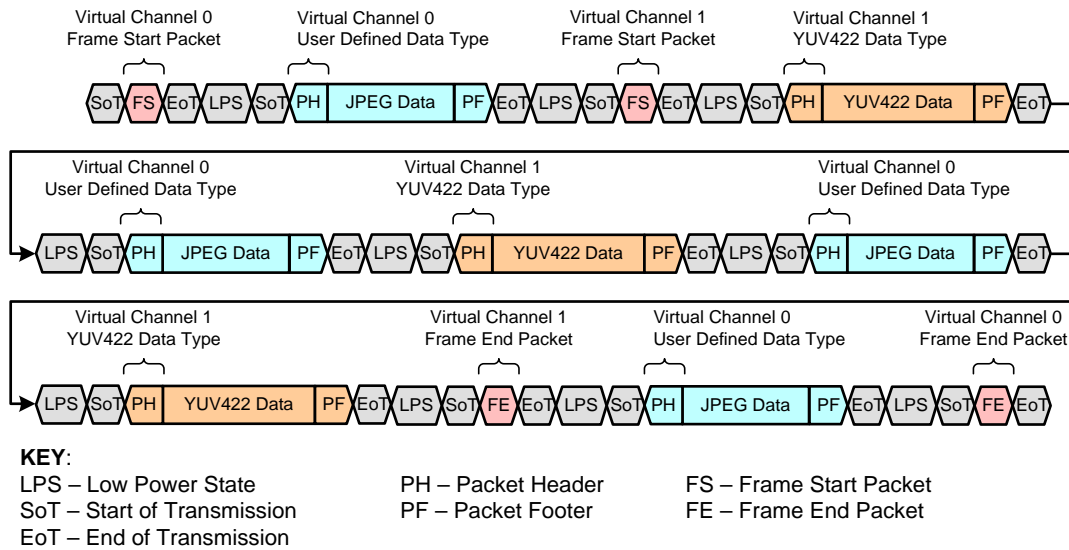3758 and Virtual Channel Identifiers.

3759

Figure 191 Data Type Interleaving: Concurrent JPEG and YUV Image Data

**Figure 192 Virtual Channel Interleaving: Concurrent JPEG and YUV Image Data**

3761 Both *Figure 191* and *Figure 192* can be similarly extended to the interleaving of JPEG image data with
3762 any other type of image data, e.g. RGB565.

3763 *Figure 193* illustrates the use of Virtual Channels to support three different JPEG interleaving usage cases:

3764 • Concurrent JPEG and YUV422 image data.

3765 • Alternating JPEG and YUV422 output - one frame JPEG, then one frame YUV

3766 • Streaming YUV22 with occasional JPEG for still capture

3767 Again, these examples could also represent interleaving JPEG data with any other image data type.
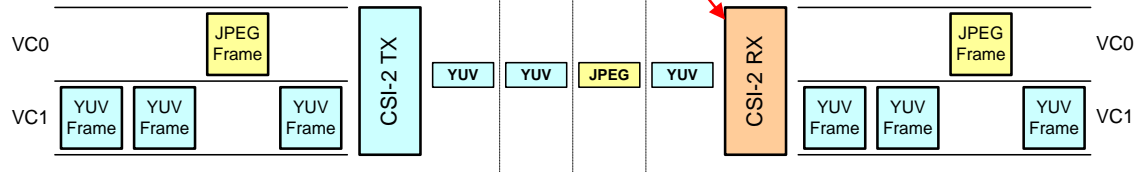
Use Case 1: Concurrent JPEG output with YUV data
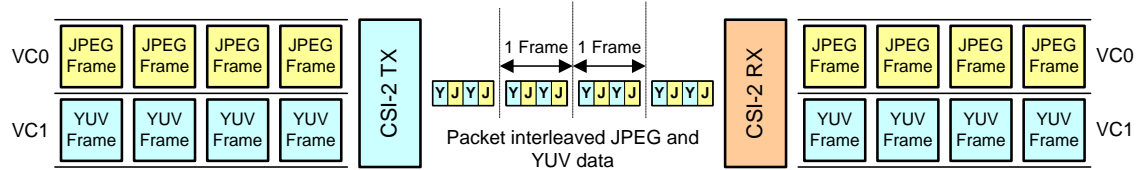
Use Case 2: Alternating JPEG and YUV output – one frame JPEG, then one frame YUV
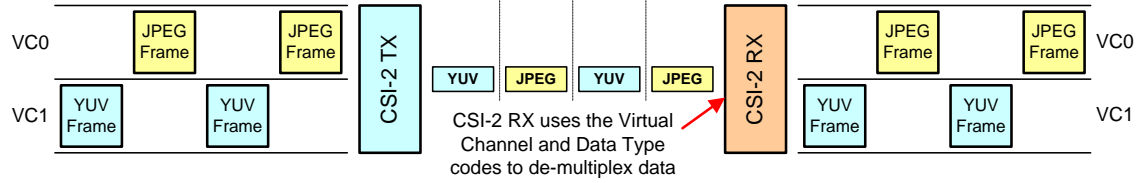
Use Case 3: Streaming YUV with occasional JPEG still capture

3768

Figure 193 Example JPEG and YUV Interleaving Use Cases

This page intentionally left blank

Annex G Scrambler Seeds for Lanes 9 and Above

(See also: *Section 9.12*).

For Links of 9 to 32 Lanes, the Scrambler PRBS registers of Lanes 9 through 32 should be initialized with the initial seed values as listed in *Table 48*.

For Links of more than 32 Lanes, the Scrambler PRBS registers of Lanes 33 and higher shall use the same initial seed value that is used for the Lane number modulo 32. (See *Section 9.12* and *Table 48*.)

Examples:

- Lane 33 shall use the same initial seed value as Lane 1
- Lane 34 shall use the same initial seed value as Lane 2
- Lane 64 shall use the same initial seed value as Lane 32
- Lane 65 shall use the same initial seed value as Lane 1

Table 48 Initial Seed Values for Lanes 9 through 32

Lane	Initial Seed Value
9	0x1818
10	0x1998
11	0x1a59
12	0x1bd8
13	0x1c38
14	0x1db8
15	0x1e78
16	0x1ff8
17	0x0001
18	0x0180
19	0x0240
20	0x03c0
21	0x0420
22	0x05a0
23	0x0660
24	0x07e0
25	0x0810
26	0x0990
27	0x0a51
28	0x0bd0
29	0x0c30
30	0x0db0
31	0x0e70
32	0x0ff0

Note that the binary representation of each initial seed value is symmetrical with respect to the forwards and backwards directions, with the exceptions of Lanes 11, 17, and 27. The initial seed values can be created easily using a Lane index value (i.e., Lane number minus one).

This page intentionally left blank.

3783

Annex H Guidance on CSI-2 Over C-PHY ALP and PPI

H.1   CSI-2 with C-PHY ALP Mode

3784 C-PHY Alternate Low Power (ALP) Mode is an alternative to the legacy LP mode of C-PHY. ALP Mode
3785 uses solely High-Speed signaling with a special state where the signals can cease toggling and collapse to
3786 zero. The legacy LP Mode signaling and escape sequences have equivalent ALP Mode functions so that the
3787 high-voltage low power signaling can be replaced by ALP Mode signaling if that is beneficial in specific
3788 systems. ALP Mode replaces the legacy LP Mode line levels by the transmission of unique code words that
3789 are used only for Lane signaling events. These unique codes are never produced by the 3-Phase mapping
3790 function, so there is never ambiguity in the interpretation of these codes at the receiver.

3791 Reasons to replace the legacy LP mode with equivalent ALP Mode functions are to begin a transitionary
3792 path to the future so that legacy LP mode might someday be eliminated in some devices. Another reason to
3793 choose ALP Mode over Legacy LP mode is to support systems that have long interconnect between the
3794 Master and Slave devices.

H.1.1   Concepts of ALP Mode and Legacy LP Mode

3795 In ALP mode, the conventional LP receivers are not used to detect signaling states. Instead, all
3796 communication is performed using High-Speed signaling levels. The system level functions performed by
3797 ALP signaling are quite similar to the functional behavior of legacy LP mode. The intent of this is to cause
3798 the least amount of disruption to systems that support both ALP Mode and legacy LP mode. *Figure 194*
3799 shows a comparison of a High-Speed data burst with LP Mode versus ALP Mode. The purpose of this
3800 diagram is to show that each of the intervals in the High-Speed data burst with LP mode correspond to
3801 similar intervals in the High-Speed data burst with ALP mode.


3802
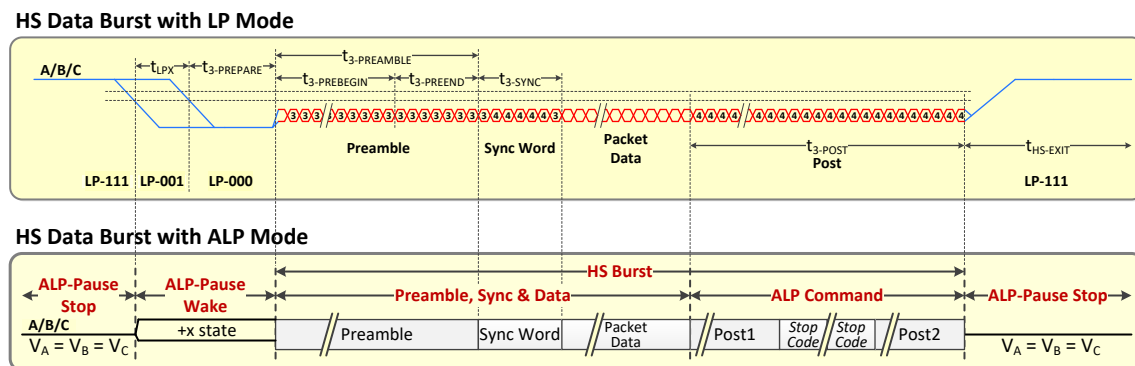
Figure 194 Comparing Data Burst Timing of Legacy LP mode versus ALP Mode

3803 ALP Mode supports the transmission of High-Speed data bursts as well as the transmission of control
3804 sequences that are traditionally transmitted using legacy LP mode Escape Mode sequences. The format of
3805 all ALP mode bursts is like the timing diagram in *Figure 195*.

3806 The burst begins and ends in an ALP-Pause state. There are two types of ALP-Pause: ALP-Pause Stop and
3807 ALP-Pause ULPS. ALP-Pause Stop is analogous to the legacy LP mode Stop state; ALP-Pause ULPS is
3808 analogous to the legacy LP mode ULPS state. The only difference between these two types of ALP-Pause
3809 states is the time allowed to wake up from each, which is the duration of the ALP-Pause Wake interval. The
3810 nominal time allowed to wave from ALP-Pause Stop is 100 ns, which is about the same time as the
3811 duration of the LP-001 and LP-000 states at the beginning of a HS Data Burst using legacy LP mode. The
3812 nominal time to wake from the ALP-Pause ULPS state is 1 msec, which is approximately the time allowed
3813 in legacy LP mode for t_{WAKEUP}. (The time that a transmitter drives a Mark-1 state prior to a Stop state to
3814 initiate an exit from ULPS.) The longer wake-up time from ALP-Pause ULPS compared to ALP-Pause Stop
3815 allows a lower power consumption while in the ALP-Pause ULPS state.

3816 The ALP-Pause Stop and ALP-Pause ULPS line states are defined by the following relationships of the
3817 Line levels: $V_A = V_B = V_C$, and $V_{OD\_AB} = V_{OD\_BC} = V_{OD\_CA} = 0$. Examples of the ALP-Pause and the ALP-
3818 Pause Wake states are illustrated at the beginning and end of the waveform in **Figure 195**. The ALP-Pause
3819 Wake state, which is very long compared to a High-Speed Unit Interval, is detected by the low-power
3820 wake-up receiver. This causes the system to leave one of the ALP-Pause states and to begin receiving a
3821 High-Speed signal.

3822
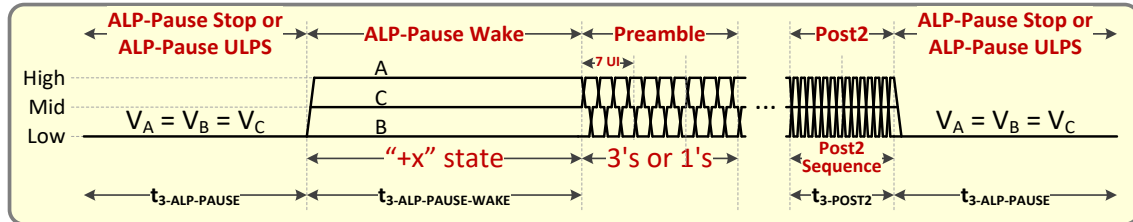
Figure 195 ALP Mode General Burst Format

3823 To minimize power consumption while Lane activity has ceased during one of the ALP-Pause states, a
3824 special low-speed and low-power differential receiver circuit is present, in addition to the three High-Speed
3825 differential receivers for A-B, B-C and C-A. This special low-speed and low-power differential receiver has
3826 a nominal +80 mV offset input threshold voltage that detects the difference in differential levels between
3827 the ALP-Pause state ($V_{OD} = 0$) and ALP-Pause Wake state ($V_{OD} = |V_{OD}|$ Strong). This allows the line signals
3828 to collapse to zero with the 100Ω Z_{ID} termination still connected, and still have a well-defined method to
3829 detect the difference between the ALP-Pause and ALP-Pause Wake line conditions. Collapsing to zero with
3830 the terminations still connected makes it possible for implementations to have very low power consumption
3831 during the ALP-Pause states. The ALP-Pause Wake pulse is very long compared to a High-Speed Unit
3832 Interval so that the wake receiver can be slow and consume very little power compared to the High-Speed
3833 differential receivers.

3834 An example of the differential receiver circuit to support ALP mode is shown in *Figure 196*. Two different
3835 offset receivers are shown for wake from stop versus wake from ULPS, because the power consumption in
3836 the ALP-Pause ULPS state is expected to be lower than in ALP-Pause Stop state. The ALP-Pause Wake
3837 pulse from the ULPS state can be longer than waking from ALP-Pause Stop, so the ALP ULPS receiver can
3838 be slower and consume less power compared to the ALP Stop receiver.

3839
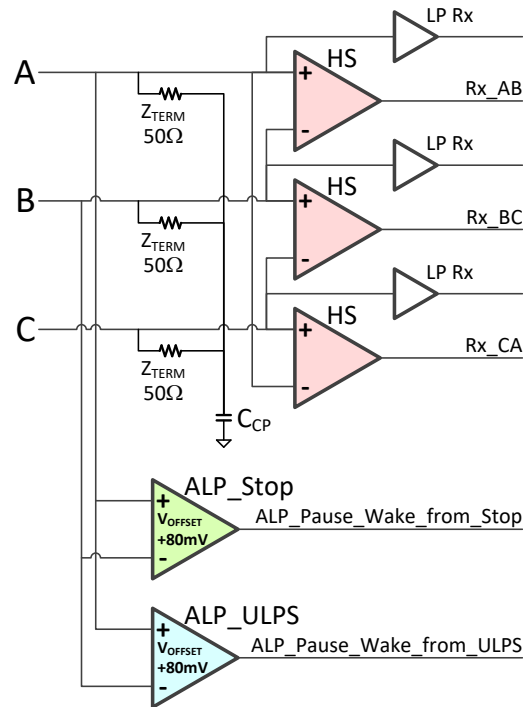
Figure 196 High-Speed and ALP-Pause Wake Receiver Example

3840 The C-PHY specification defines twelve unique 7-symbol ALP Code Words that are the functional
3841 equivalent of the LP pulse sequences of legacy LP mode. In some cases, a single 7-symbol ALP Code Word
3842 can replace the transmission of a long sequence of legacy LP mode pulses, such as for the transmission of
3843 Escape Mode triggers or low-power data transmission. The CSI-2 specification needs only three of these
3844 LP mode pulse sequences to emulate the functionality of legacy LP mode: Stop Code, ULPS Code, and
3845 Post.

3846 Exit from and entry into the ALP-Pause state, which is the functional equivalent of the legacy LP mode
3847 Stop state, requires a special ALP Mode sequence consisting of one or more Stop Codes or ULPS codes
3848 followed by a string of Post codes followed by setting the voltage of all three Lines of a Lane to the same
3849 value.

3850 As illustrated in *Figure 194*, the burst starting sequence of the legacy LP mode consisting of: LP-111, LP-
3851 001, and LP-000 followed by preamble, has a functional equivalent sequence in ALP Mode consisting of:
3852 ALP-Pause Stop followed by ALP Pause Wake followed by preamble. Similarly, the burst ending sequence
3853 of legacy LP mode consisting of Post sequence followed by LP-111, has a functional equivalent sequence
3854 in ALP Mode consisting of: the Post1 field by two or more Stop Codes followed by the Post2 field
3855 followed by ALP-Pause Stop.

H.1.2 Burst Examples Using ALP Mode

Figure 197 shows examples of the three types of High-Speed bursts that can be sent in ALP mode. Many combinations of ALP code sequences are possible, but *Figure 197* shows three sequences that adequately perform the functions necessary to support CSI-2 that are currently performed using legacy LP mode. The ALP state machine from the C-PHY Specification has been highlighted in *Figure 198*, *Figure 199*, and *Figure 200* to show how transmission of these three sequences should occur.

For interop sake, only these three types of sequences are required to support CSI-2. Note that all bursts begin in the same manner with the assertion of ALP-Pause Wake followed by a Preamble. The words that follow the Preamble determine the type of burst that is being transmitted. All bursts end in the same manner with multiple Stop Codes followed by the Post2 field, or multiple ULPS Codes followed by the Post2 field. The Post 1 and Post2 fields are the same as Post (4444444), described in the C-PHY specification for burst transmission using legacy LP mode. The only difference is that the Post1 and Post2 fields are transmitted as a result of signaling over the PPI from the CSI-2 Tx to the C-PHY Tx.

The last ALP code sent in the burst determines whether the system enters the ALP-Pause Stop or the ALP-Pause ULPS state.

High Speed Data Burst
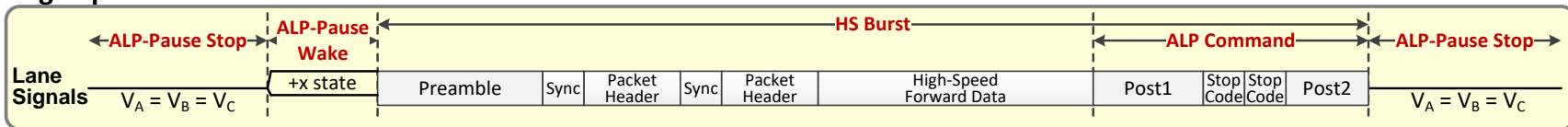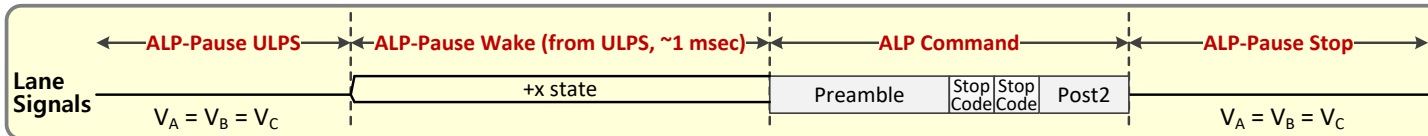

Command to Enter ULPS
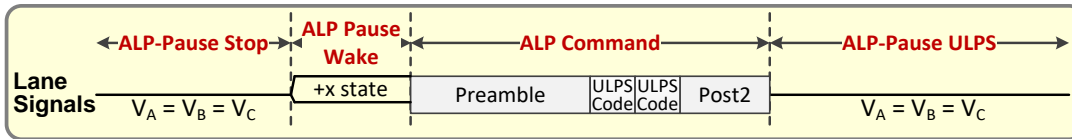
Command to Exit from ULPS

Figure 197 Examples of Bursts to Send High-Speed Data and ALP Commands

3871 *Figure 198* shows the ALP state machine transitions (highlighted in red) necessary to transmit a High-
3872 Speed data burst in ALP mode. States and state transitions that are not used by CSI-2 for any type of burst
3873 are shown using dashed lines. The red highlighted states and transitions indicate the path required to
3874 transmit and receive the High-Speed Data Burst example in *Figure 197*.

3875
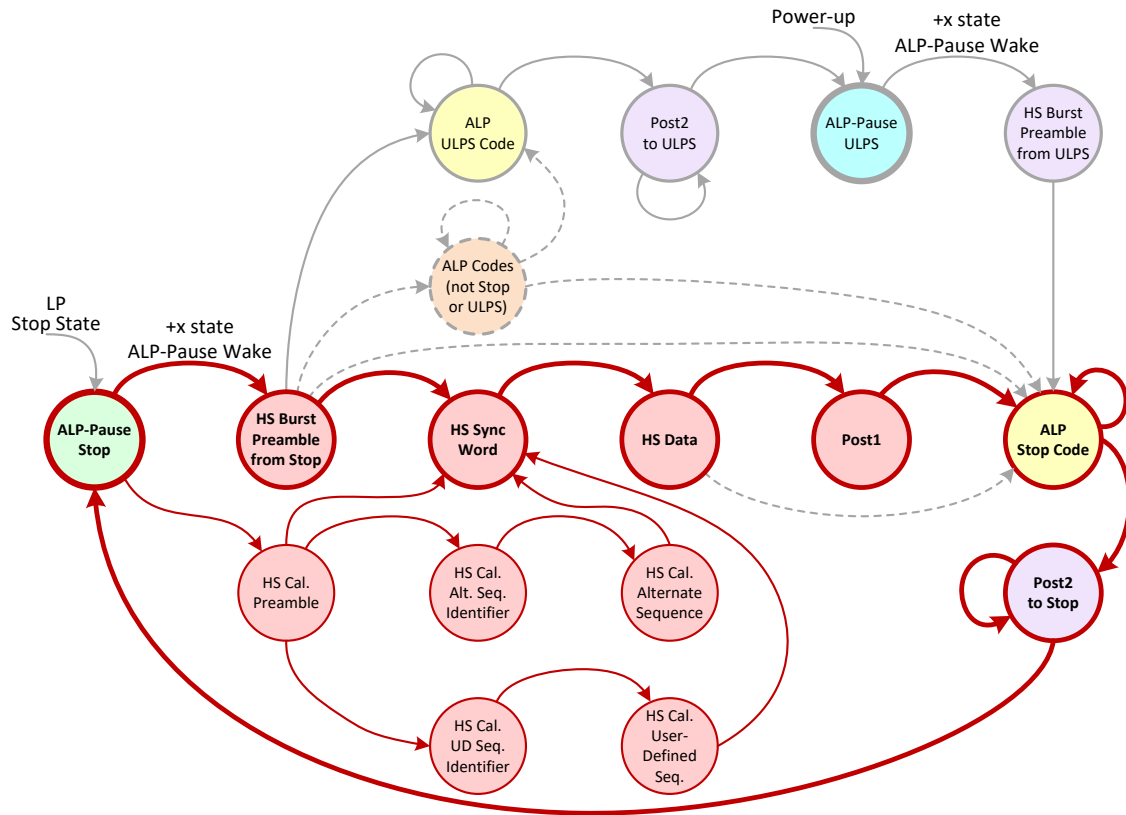
Figure 198 State Transitions for an HS Data Burst

3876 *Figure 199* shows the ALP state machine transitions (highlighted in red) necessary to enter the ALP-Pause
3877 ULPS state.

3878
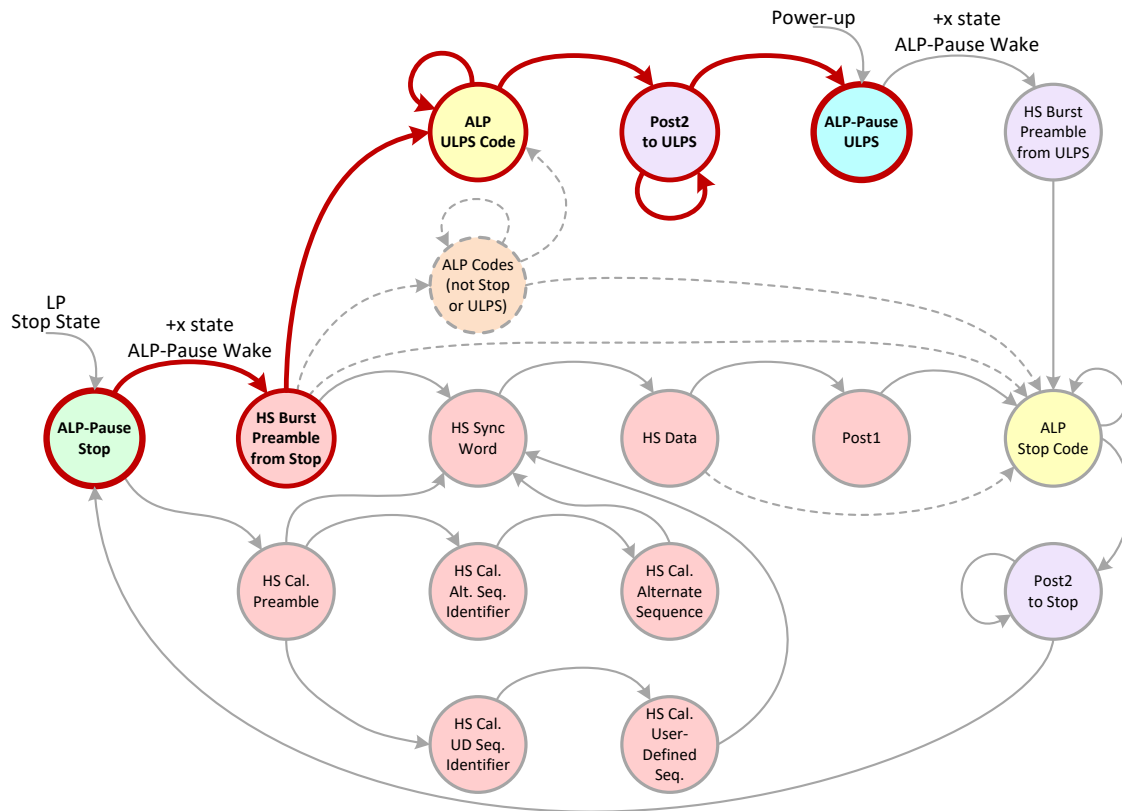
Figure 199 State Transitions to Enter the ULPS State

3879   *Figure 200* shows the ALP state machine transitions (highlighted in red) necessary to enter the ALP-Pause
3880   Stop state.

3881
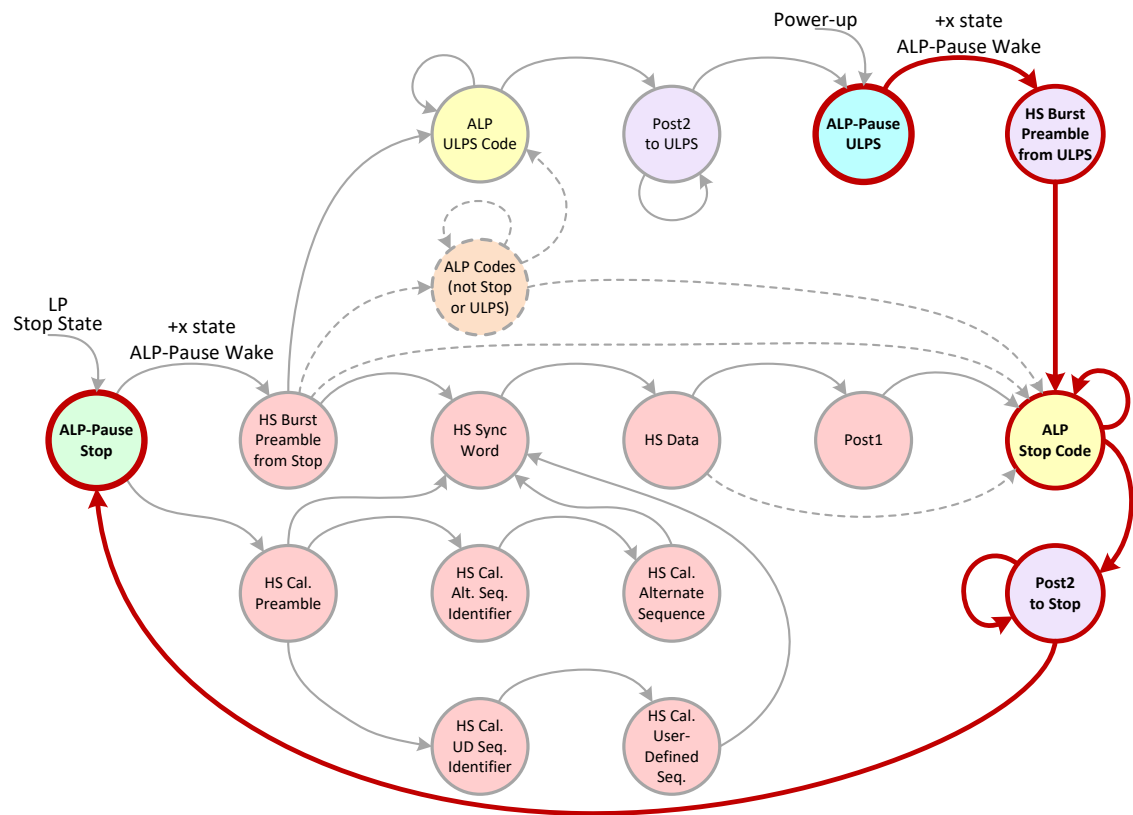
Figure 200 State Transitions to Exit from the ULPS State

3882   *Table 49* describes the 7-symbol codes transmitted in ALP mode. The corresponding LP mode or Escape
3883   mode function is described, where applicable.

3884   **Table 49 ALP Code Definitions used by CSI-2**

ALP Code	Symbol Sequence	PPI ALP Code	Corresponding LP State or Escape Mode Sequence
Stop Code	0244440	0b0000	LP-111 (End of Transmission, or EoT)
ULPS Code	0244441	0b0001	Escape Mode Entry + Ultra-Low Power State (ULPS)
Post1	4444444	0b1011	No equivalent legacy LP mode sequence exists. The CSI-2 TX can cause the Post sequence to be transmitted by sending this code.
Post2			

H.1.3   Transmission and Reception of ALP Commands Through the PPI

3885   In ALP mode there are three types of code words transmitted by the PHY:

- **Data:** Data words received from the CSI-2 Tx are mapped through the C-PHY mapper, encoded,
  and transmitted over the Lane.

- **Sync Words:** The CSI-2 Tx can cause the C-PHY Tx to transmit a Sync Word in place of a data
  word created by the C-PHY mapper. Sync Words can have one of five different values which are
  defined as Sync Types.

- **ALP Codes**: The CSI-2 Tx can cause the C-PHY Tx to transmit a specific ALP code which is one
  of the 7-symbol sequences defined in *Table 49*.

3893   These three different types of code words comprise a high speed burst while in ALP mode. *Figure 201*
3894   highlights the control signals that facilitate the transmission of each of these three different types of code
3895   words.
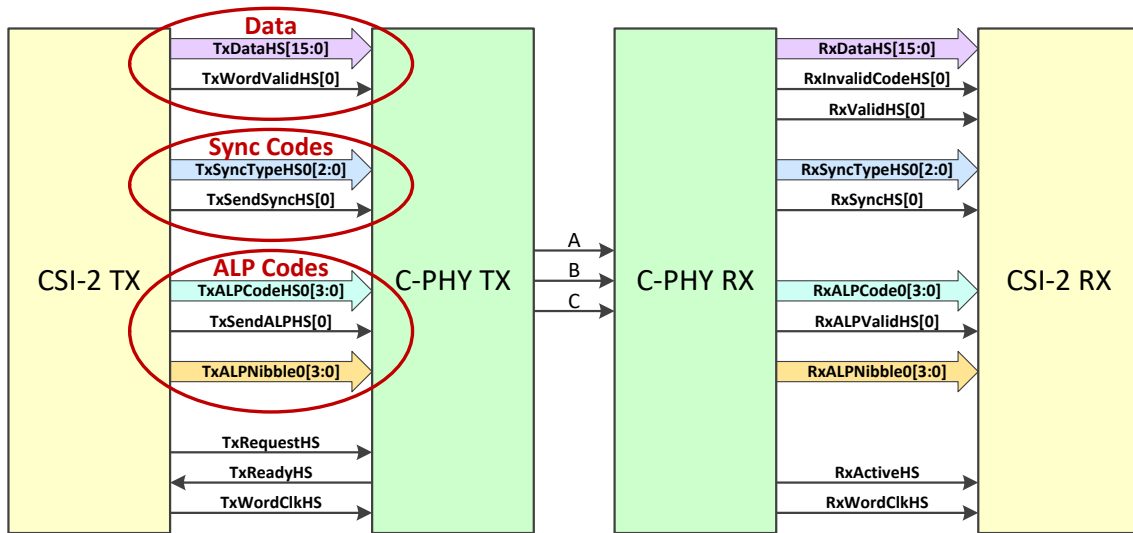
Figure 201 PPI Example: HS Signals for Transmission of Data, Sync and ALP Commands

3897   *Figure 202* and *Figure 203* show examples of PPI signals and the corresponding PHY data for
3898   transmission and reception of high speed data in ALP mode. These figures show additional detail of the
3899   High-Speed Data Burst waveform in *Figure 197*.

3900   The signal TxRequestHS is asserted simultaneously with TxWordValidHS to request that a high speed burst
3901   be transmitted. The PHY will know to send a data burst because TxWordValidHS is asserted early in the
3902   burst timing. This will cause the C-PHY Tx to transmit the first Sync Word at the end of the Preamble. Note
3903   that the first Sync Word is transmitted autonomously by the C-PHY Tx, and has the default Sync Type
3904   value of 3. Subsequent Sync Words transmitted in a burst are sent as a result of asserting the
3905   TxSendSyncHS[0] signal, and the associated Sync Type is defined by the TxSyncTypeHS0[2:0] signals.

3906   The end of burst in the Transmitter functions differently for ALP mode compared to the non-ALP high-
3907   speed mode. In the non-ALP high-speed mode, the end of burst is signaled to the PHY by pulling
3908   TxRequestHS low, as described in Annex A of the C-PHY specification. After TxRequestHS goes low, the
3909   C-PHY Tx will generate the Post sequence of length determined by a PHY configuration parameter that
3910   sets the length of Post.

3911   In ALP mode, the protocol transmit unit generates all fields of the burst after the first sync word, including
3912   the packet headers, data burst, Stop Code, ULPS Code, Post1, and Post2. The burst is ended by pulling
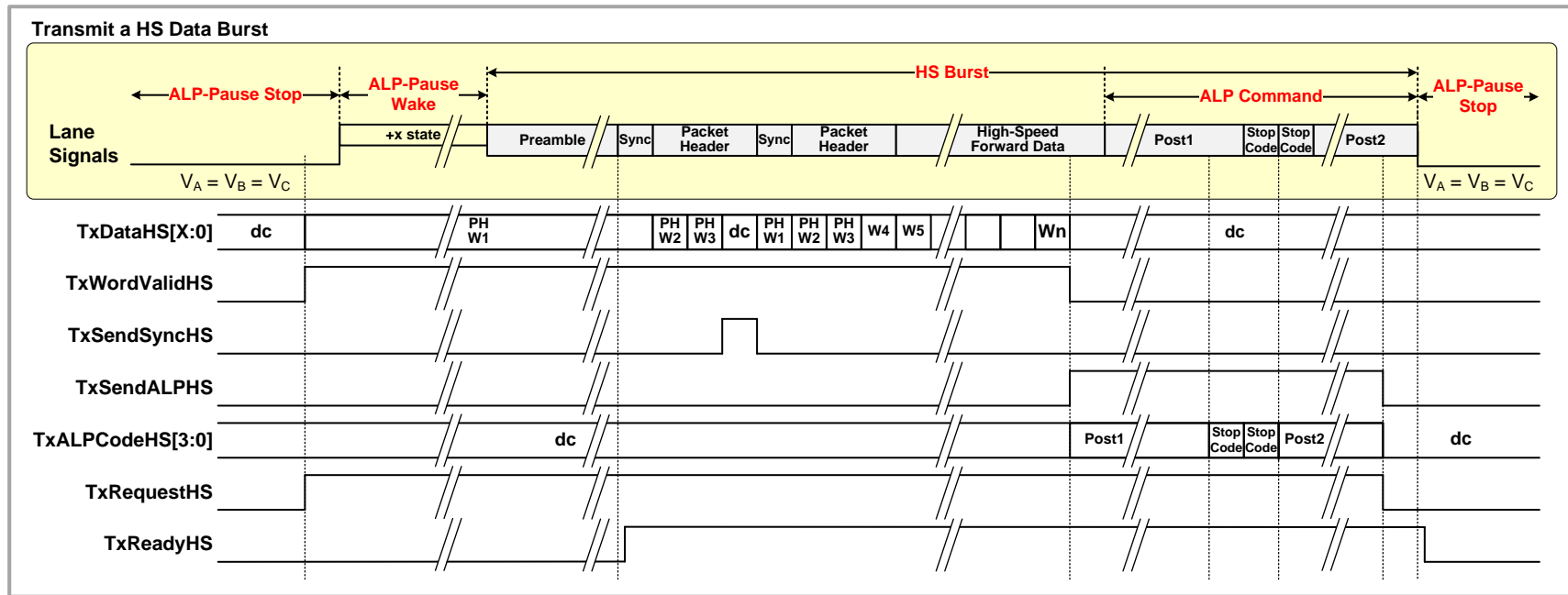3913   TxRequestHS low, and no additional data is transmitted on the Lane after this time.

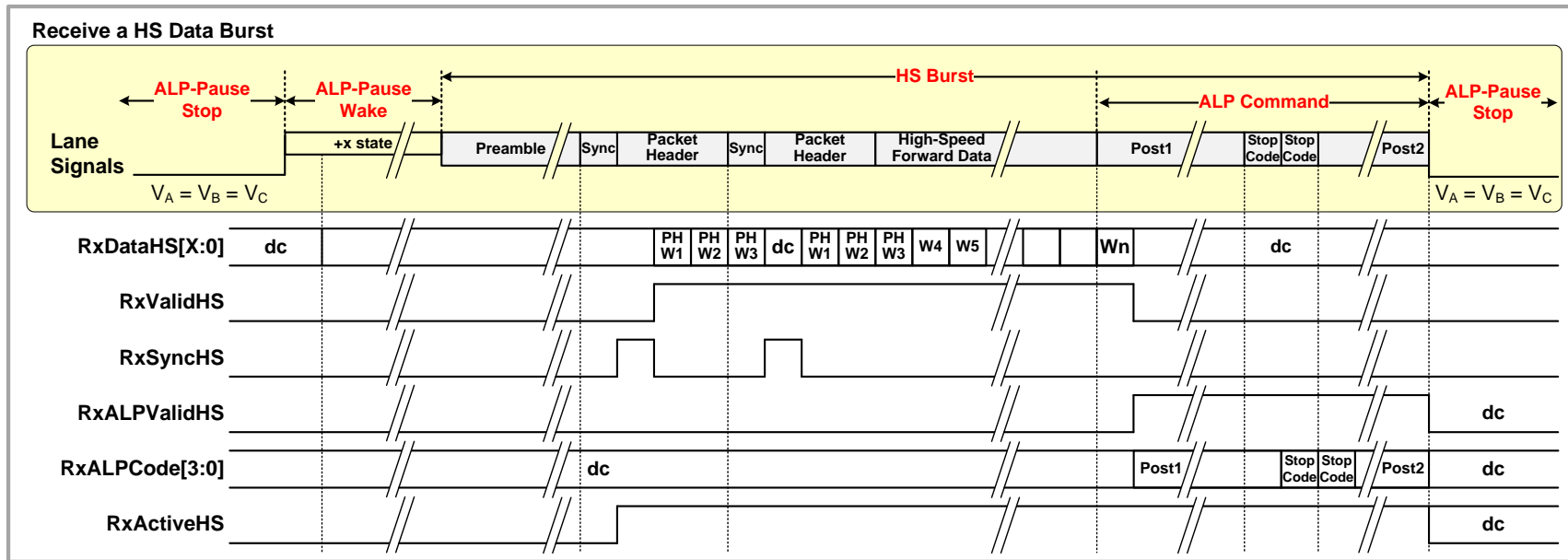**Figure 202 PPI Example Transmit Side Timing for an HS Data Burst**

Figure 203 PPI Example Receive Side Timing for an HS Data Burst

Figure 204, *Figure 205*, *Figure 206*, and *Figure 207* show examples of PPI signals and the corresponding PHY data for transmission and reception ALP Commands to enter into and exit from the ALP-Pause ULPS state in ALP mode. These figures show additional detail of the Command to Enter ULPS and the Command to Exit from ULPS waveforms in *Figure* **197**.

The signal TxRequestHS is asserted simultaneously with TxSendALPHS to request that a high speed burst be transmitted. The PHY will know to send a ALP commands in the burst rather than the Sync Word because TxSendALPHS is asserted early in the burst timing, and TxWordValidHS is not asserted.
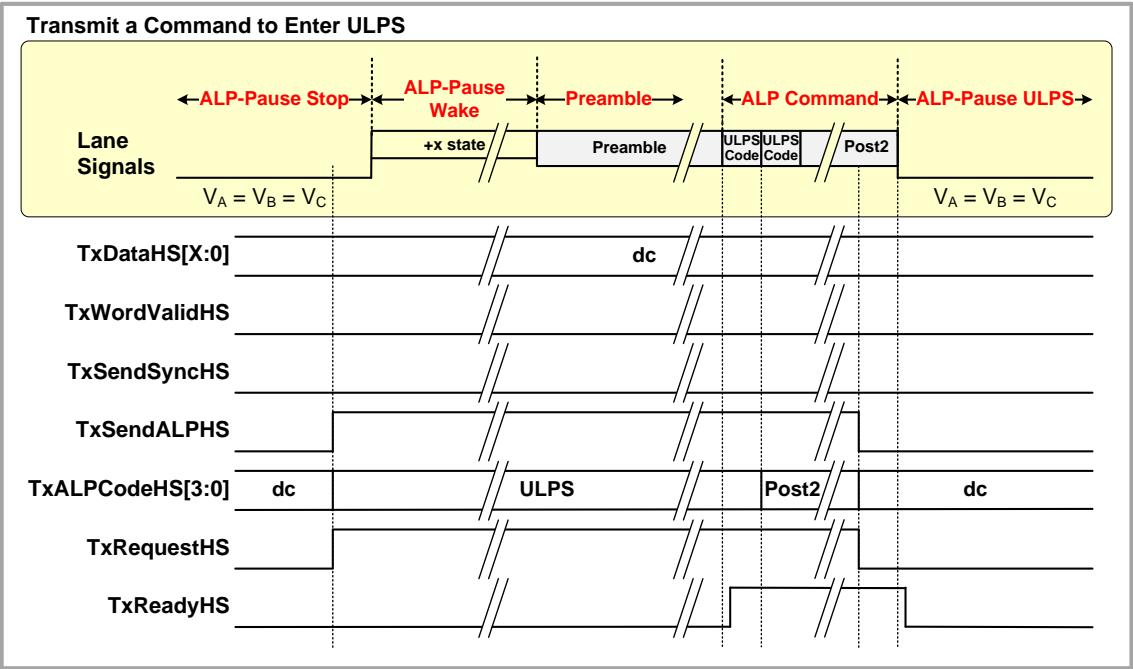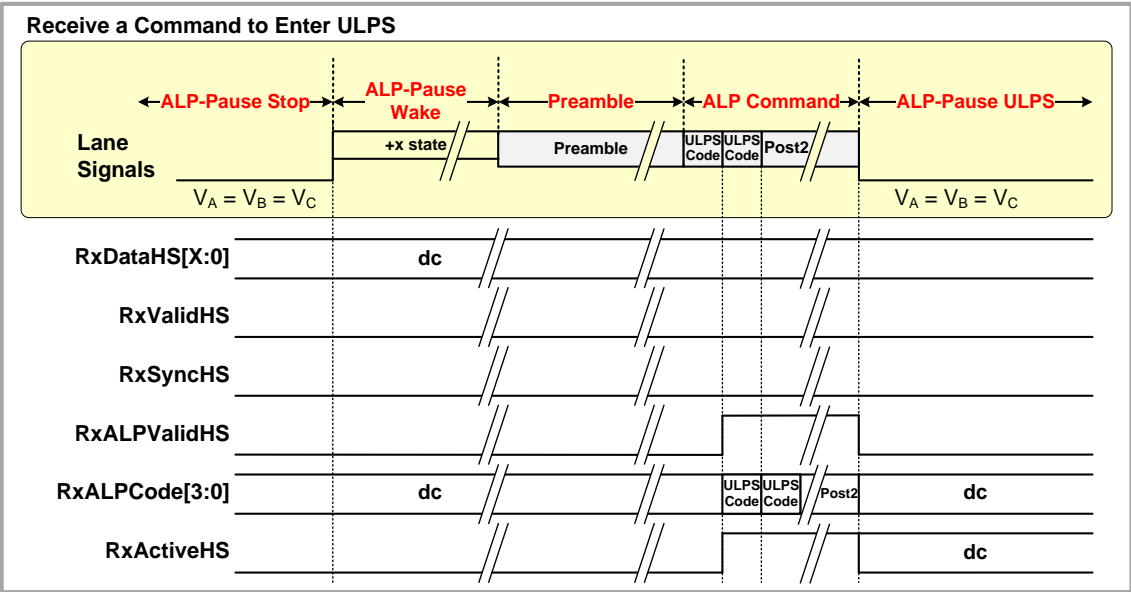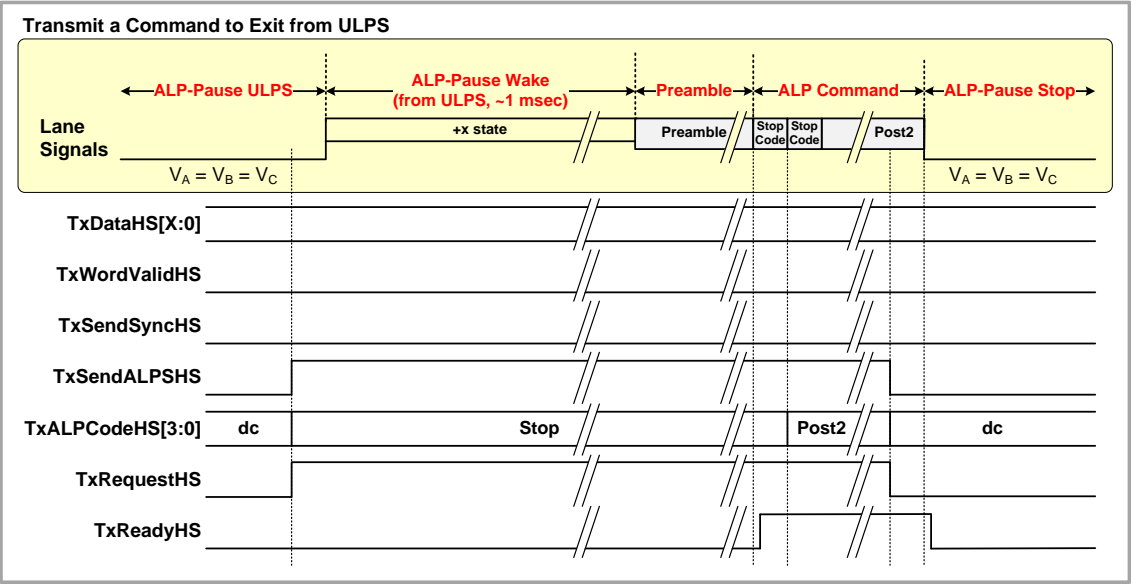

Figure 204 PPI Example Transmit Side Timing to Enter the ULPS State

Figure 205 PPI Example Receive Side Timing to Enter the ULPS State

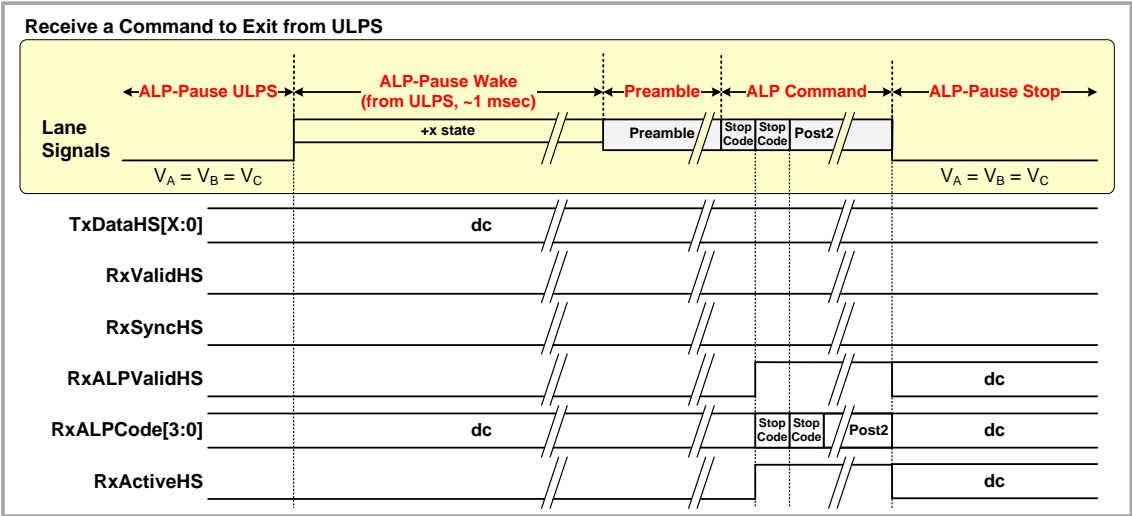**Figure 206 PPI Example Transmit Side Timing to Exit from the ULPS State**

Figure 207 PPI Example Receive Side Timing to Exit from the ULPS State

H.1.4   Multi-Lane Operation Using ALP Mode

Figure 208 and *Figure 209* show examples of three Lanes operating together in a Link in ALP mode. The High-Speed data burst in *Figure 208* begins with identical packet headers (consisting of PH W0, PH W1, and PH W2) transmitted twice on each of the three Lanes. The Packet Headers are followed by packet data (consisting of DW 0 through DW n-1) striped across the three Lanes by the CSI-2 Lane Distribution Function. The burst starts and ends in the manner described in Section H.1.2 above. The example of *Figure 209* showing the command to enter ULPS has identical data on each of the three Lanes.

The example also shows that the assertion of the +x state for ALP-Pause Wake can be staggered in time on each of the lanes. This is shown to highlight a particular implementation where the system designer might prefer to enable the high speed drivers for each of the Lanes at a slightly different time.
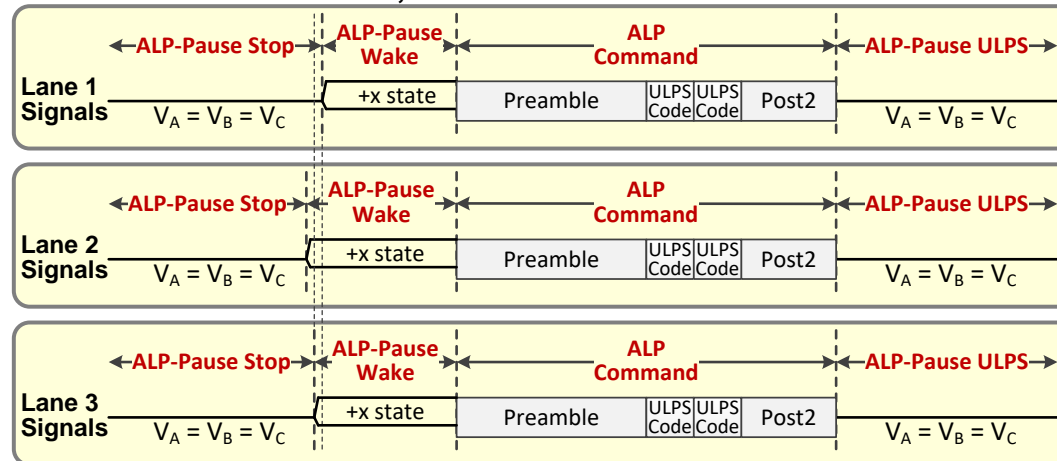
High Speed Data Burst, Three Lanes
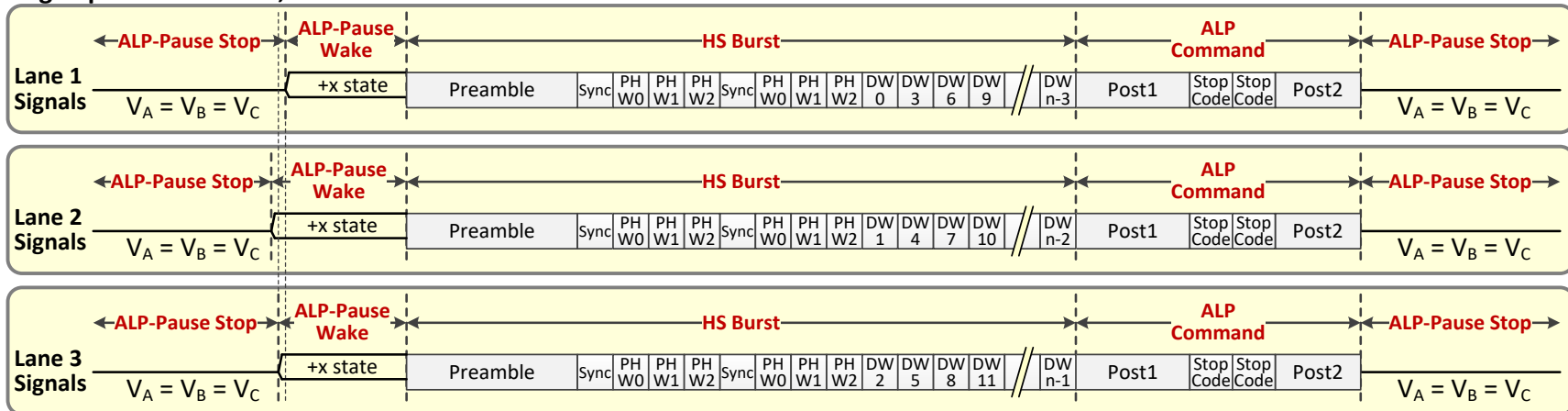
3936

Figure 208 Example Showing a Data Transmission Burst using Three Lanes

Command to Enter ULPS, Three Lanes

3937

Figure 209 Example Showing an ALP Command Burst using Three Lanes

H.1.5   Concurrent LP and ALP Operation

Section 6.4.5.8 of *[MIPI02]* describes the concurrent LP and ALP operation. The system is configured for
LP operation at power-up. During initialization, the system can be configured for LP-only operation, or
ALP-only operation or concurrent LP-ALP operation. It is anticipated that most systems will use a mode bit
or configuration option that causes the system to operate in either LP-only or ALP-only operation.
However, it is also possible to implement the capability for concurrent operation. A burst can begin as LP
and end as ALP, or vice-versa. The method of ending the burst, whether via the transmission of ALP codes
or transmission of LP-111, determines whether the system is in ALP operation or LP operation. This
concept is illustrated in by the state machine in *Figure 210*.
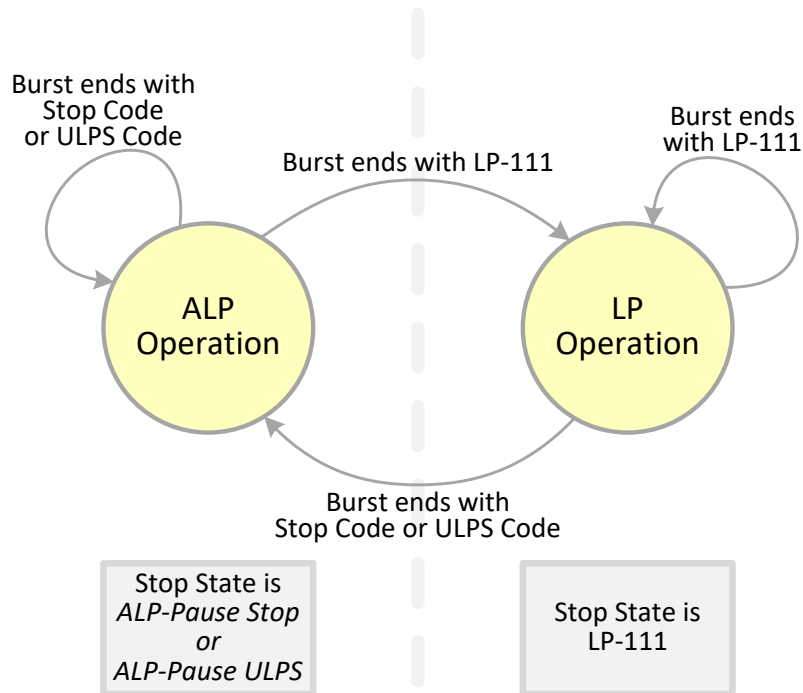
Figure 210 Automatic Selection of ALP Operation or LP Operation

This page intentionally left blank.

Participants

The list below includes those persons who participated in the Working Group that developed this Specification and who consented to appear on this list.

Sakari Ailus, Intel Corporation

Rob Anhofer, MIPI Alliance (staff)

Radha Krishna Atukula, NVIDIA

Chua Teong Rong, Sony Corporation

Gyeonghan Cha, Samsung Electronics, Co.

Zhang Chunrong, Advanced Micro Devices, Inc.

Chris Grigg, MIPI Alliance (staff)

Jason Hawken, Advanced Micro Devices, Inc.

Tom Kopet, ON Semiconductor

Cedric Marta, Synopsys, Inc.

Mikko Muukki, HiSilicon Technologies Co. Ltd.

Manuel Ortiz, Intel Corporation

Prachi Patel , Cadence Design Systems, Inc.

Matthew Ronning, Sony Corporation

Yacov Simhony, Cadence Design Systems, Inc.

Richard Sproul, Cadence Design Systems, Inc.

Hiroo Takahashi, Sony Corporation

Haran Thanigasalam, Intel Corporation

George Wiley, Qualcomm Incorporated.

Past Contributors to v2.0:

Hirofumi Adachi, Teradyne Inc.

Sakari Ailus, Intel Corporation

Rob Anhofer, MIPI Alliance

Radha Krishna Atukula, NVIDIA

Kaberi Banerjee, Lattice Semiconductor Corp.

Thierry Berdah, Cadence Design Systems, Inc.

Thomas Blon, Silicon Line GmbH

Teong Rong Chua, Sony Corporation

Zhang Chunrong, Advanced Micro Devices, Inc.

Tatsuyuki Fukushima, Teradyne Inc.

Karan Galhotra, Synopsys, Inc.

Vaibhav Gupta, Mentor Graphics

Mattias Gustafsson, OmniVision Technologies, Inc.

Mohamed Hafed, Introspect Test Technology Inc.

Will Harris, Advanced Micro Devices, Inc.

Jason Hawken, Advanced Micro Devices, Inc.

Hsieh Chang Ho, MediaTek Inc.

Norihiro Ichimaru, Sony Corporation

Henrik Icking, Intel Corporation

Yukichi Inoue, Teradyne Inc.

Kayoko Ishiwata, Toshiba Corporation

Grant Jennings, Lattice Semiconductor Corp.

Jacob Joseph, NVIDIA

Mrudula Kanuri, NVIDIA

Nadine Kolment, Introspect Test Technology Inc.

Tom Kopet, ON Semiconductor

Thomas Krause, Texas Instruments Incorporated

Yoav Lavi, VLSI Plus Ltd.

Cedric Marta, Synopsys, Inc.

Mikko Muukki, HiSilicon Technologies Co. Ltd.

Raj Kumar Nagpal, Synopsys, Inc.

Amir Naveed, Qualcomm Incorporated

Laurent Pinchart, Google, Inc.

Alex Qiu, NVIDIA

Matthew Ronning, Sony Corporation

Yaron Schwartz, Cadence Design Systems, Inc.

Sho Sengoku, Qualcomm Incorporated

Yacov Simhony, Cadence Design Systems, Inc.

Gaurav Singh, Synopsys, Inc.

Richard Sproul, Cadence Design Systems, Inc.

Tatsuya Sugioka, Sony Corporation

Hiroo Takahashi, Sony Corporation

Haran Thanigasalam, Intel Corporation

Bruno Trematore, Toshiba Corporation

Rick Wietfeldt, Qualcomm Incorporated

George Wiley, Qualcomm Incorporated

Charles Wu, OmniVision Technologies, Inc.

Hirofumi Yoshida, Sony Corporation

MinJun Zhao, HiSilicon Technologies Co. Ltd.

This page intentionally left blank.